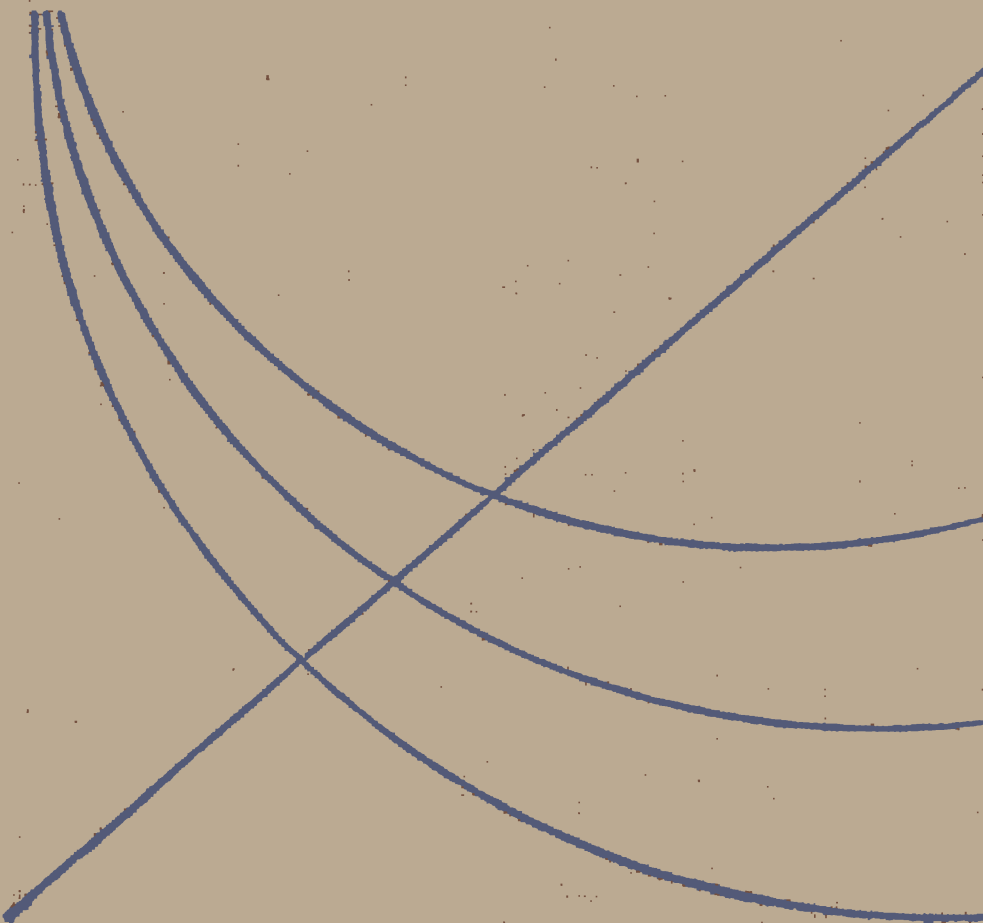


А.А.Пярнпуу

Программирование на современных алгоритмических языках



А. А. ПЯРНПУУ

ПРОГРАММИРОВАНИЕ НА СОВРЕМЕННЫХ АЛГОРИТМИЧЕСКИХ ЯЗЫКАХ

ИЗДАНИЕ ТРЕТЬЕ,
ПЕРЕРАБОТАННОЕ И ДОПОЛНЕННОЕ

*Допущено Государственным комитетом СССР
по народному образованию в качестве учебного пособия
для студентов высших технических учебных заведений*



МОСКВА «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
1990

ББК 22.18

П 99

УДК 519.85 (075.8)

Пиряпуу А. А. Программирование на современных алгоритмических языках: Учеб. пособие для вузов.— 3-е изд., перераб. и доп.— М.: Наука. Гл. ред. физ.-мат. лит., 1990.— 384 с.— ISBN 5-02-614399-5.

Излагаются основы программирования на алгоритмических языках Бейсик, Фортран IV и ПЛ/1. Специальная глава содержит общие сведения о развитии ЭВМ, совершенствовании алгоритмических языков и принципов решения задач на ПЭВМ. Рассмотрены особенности входных языков некоторых широко используемых трансляторов.

2-е изд.— 1983 г.

Для студентов вузов.

Ил. 5. Библиогр. 18 назв.

Рецензент:

кандидат физико-математических наук *И. Е. Педанов*
кафедра вычислительной математики МФТИ

1404000000—072
П 053 (02)-90 177-90
ISBN 5-02-014399-5

© «Наука». Физматлит, 1978;
с изменениями, 1983;
переработанное и допол-
ненное, 1990

ОГЛАВЛЕНИЕ

Предисловие	5
Г л а в а I. Общие сведения	7
§ 1. Электронные вычислительные машины	7
§ 2. Решение задач на ЭВМ	18
§ 3. Понятия алгоритма и программы	20
§ 4. Алгоритмические языки	25
§ 5. Трансляция и входные языки	34
Г л а в а II. Фортран IV	39
§ 1. Элементы языка	39
§ 2. Выражения	46
§ 3. Структура программы	50
§ 4. Операторы описания типа	52
§ 5. Операторы присваивания	57
§ 6. Правила записи операторов	59
§ 7. Операторы перехода	61
§ 8. Условные операторы управления	64
§ 9. Оператор цикла	67
§ 10. Операторы останова и окончания	71
§ 11. Библиотечные подпрограммы	72
§ 12. Оператор-функция	73
§ 13. Подпрограммы-функции	74
§ 14. Подпрограммы	83
§ 15. Оператор входа	89
§ 16. Оператор описания общих блоков	91
§ 17. Оператор присваивания начальных значений	95
§ 18. Подпрограмма данных	97
§ 19. Оператор эквивалентности	99
§ 20. Операторы ввода и вывода	101
§ 21. Оператор формирования списков	113
§ 22. Оператор задания формата	115
§ 23. Классификация операторов и наименований	132
§ 24. Примеры программ	133
Г л а в а III. Бейсик	136
§ 1. Основные конструкции языка	136
§ 2. Данные	138
§ 3. Числовые операции и выражения	143
§ 4. Строковые операции	147
§ 5. Система операторов	148
§ 6. Описание переменных	151
§ 7. Определение функций	152

§ 8.	Размерности массивов	154
§ 9.	Присваивание значений	156
§ 10.	Ввод данных в программу	159
§ 11.	Вывод данных	163
§ 12.	Стандартные функции	170
§ 13.	Система команд	172
§ 14.	Безусловная передача управления	182
§ 15.	Условная передача управления	185
§ 16.	Организация циклов	187
§ 17.	Подпрограммы	192
§ 18.	Операции над матрицами	198
§ 19.	Файлы данных	202
§ 20.	Графическое представление информации	208
§ 21.	Воспроизведение звука	219
§ 22.	Примеры программ	221
Г л а в а IV. ПЛ/1		224
§ 1.	Данные	224
§ 2.	Идентификаторы	228
§ 3.	Операторы	230
§ 4.	Описание данных	232
§ 5.	Выражения	245
§ 6.	Примечания	249
§ 7.	Структура программы	250
§ 8.	Оператор присваивания	250
§ 9.	Операторы управления	252
§ 10.	Блоки	258
§ 11.	Встроенные функции	266
§ 12.	Обработка прерываний	267
§ 13.	Распределение памяти	274
§ 14.	Описатели файла	278
§ 15.	Подготовка файлов к вводу-выводу	282
§ 16.	Ввод и вывод данных	284
§ 17.	Операторы ввода-вывода потока	286
§ 18.	Операторы ввода-вывода записей	296
§ 19.	Пример программы	301
Г л а в а V. Краткие сведения о некоторых входных языках		303
§ 1.	Структура и функции операционной системы	303
§ 2.	Фортран IV ОС ЕС ЭВМ	308
§ 3.	Фортран для ЭВМ БЭСМ-6	313
§ 4.	Фортран ДОС ЕС	322
§ 5.	Фортран-Дубна и Фортран ДОС ЕС	329
§ 6.	Стандарт Фортран 77	333
§ 7.	Бейсик-плюс для СМ ЭВМ	346
§ 8.	Интерпретатор MINIBAS ОС МикроДОС	351
§ 9.	Версия интерпретатора BASIC	354
§ 10.	Подмножество ПЛ/1 ОС ЕС ЭВМ	358
§ 11.	ПЛ/1 ДОС ЕС	360
П р и л о ж е н и е I. Библиотечные подпрограммы Форт- рана IV		364
П р и л о ж е н и е II. Стандартные функции языка Бейсик		368
П р и л о ж е н и е III. Встроенные функции ПЛ/1		370
Список литературы		376
Предметный указатель		377

ПРЕДИСЛОВИЕ

Электронные вычислительные машины (ЭВМ) в настоящее время становятся одним из решающих факторов в развитии современной науки и практически всех отраслей народного хозяйства. Проникновение ЭВМ в различные сферы интеллектуальной деятельности человека обуславливает их широкое применение и использование при решении многих актуальных задач, относящихся к самым различным областям науки и техники. Появление в течение последнего десятилетия новых типов компьютеров — персональных ЭВМ (ПЭВМ) обеспечивает внедрение вычислительной техники буквально на каждое рабочее место. В связи с этим существенно увеличивается количество лиц, прибегающих в своей работе к помощи ЭВМ. Курс программирования и алгоритмических языков слушают студенты и аспиранты, он является обязательным на факультетах повышения квалификации специалистов народного хозяйства, большое число научных работников и инженеров изучает программирование и алгоритмические языки самостоятельно.

В основе данной книги лежит курс лекций, который регулярно читается автором на факультете повышения квалификации Московского авиационного института им. С. Орджоникидзе. Первоначальный вариант этих лекций изложен в книге автора [1].

Затем программа упомянутого курса претерпела структурные изменения, главным образом, вследствие включения в программу языка ПЛ/1 и сосредоточения внимания на ЭВМ серии ЕС. Все указанные изменения нашли отражение в книге [2]. В ней содержится описание алгоритмических языков Алгол-60, Фортран IV, ПЛ/1 и трансляторов с этих языков для ЭВМ БЭСМ-6 и ЕС ЭВМ.

В последние годы в процессе обучения программированию произошли существенные изменения. Появились дисплейные классы и персональные ЭВМ. Одним из популярных языков программирования в режиме диалога с ЭВМ стал алгоритмический язык Бейсик. Естественно, что в курсы лекций по программированию для ЭВМ было включено описание языка Бейсик. Алгоритмический язык Алгол-60 постепенно уступил позиции своему преемнику — языку Паскаль, который из Алгола-60 заимствовал принципы структурирования и форму выражения. В результате Алгол-60 перестал быть составной частью упомянутого выше курса лекций.

В предлагаемой читателю книге сохранена структура современного курса лекций. В ней подвергались редакционной доработке почти все разделы, устранены неточности и опечатки. Новой является гл. III, посвященная языку Бейсик. В этой главе приведено полное описание языка и указаны основные приемы программирования, в том числе в режиме диалога при использовании ПЭВМ.

Глава I является вводной и содержит общие сведения об электронных вычислительных машинах и программировании для них.

Глава II содержит описание языка программирования Фортран IV с учетом дополнительных возможностей. Трансляторы большинства современных ЭВМ в той или иной степени учитывают эти возможности. В то же время материал этой главы не связывается ни с каким конкретным транслятором.

В главе IV изложен язык ПЛ/I с учетом почти всех его особенностей, не включены лишь некоторые наиболее сложные и несущественные для первоначального знакомства с языком элементы (например, средства асинхронного выполнения программы), которые читатель может почерпнуть из других источников.

По мере возможности в книге подчеркиваются особенности каждого из рассматриваемых языков, выделяются их общие свойства, а также принципиальное отличие. Такое сопоставление может помочь программисту сделать выбор в пользу одного из этих языков при решении конкретной задачи. Кроме того, знание языков Фортран IV, Бейсик и ПЛ/I полезно при изучении других современных языков программирования, например Кобола или Алгола-68, Паскаля или языка Си. Изложение языков дано независимо друг от друга, т. е. при чтении книги можно опустить любую из глав II, III или IV.

Глава V посвящена краткой характеристике некоторых широко используемых версий языков для ЭВМ БЭСМ-6, ЕС ЭВМ, серии СМ и ПЭВМ. Назначение этой главы — указать особенности входных языков различных трансляторов. Предполагается, что приводимых здесь сведений будет достаточно при составлении программы для указанных вычислительных машин. Тем не менее перед выходом на ЭВМ будущему программисту необходимо ознакомиться с соответствующими инструкциями данного вычислительного центра.

Как правило, вводимые в книге понятия сопровождаются конкретными примерами. Выполнение упражнений, содержащихся почти в каждом параграфе, должно помочь лучшему усвоению прочитанного материала и служить приобретению практических навыков программирования.

В списке литературы даны названия лишь тех источников, которые вышли в последние годы и могут быть использованы как пособия при изучении языков программирования и знакомстве с трансляторами. В тексте приведено несколько наиболее удачных учебных примеров из этих книг.

Автор признателен всем, кто ему помогал в отборе материала для книги и прочитал рукопись. Их замечания и пожелания учтены в окончательном варианте книги.

А. Пярнлуу

Глава I. ОБЩИЕ СВЕДЕНИЯ

§ 1. Электронные вычислительные машины

В настоящее время трудно назвать все те области человеческой деятельности, успех которых не был бы связан с использованием *электронных вычислительных машин*. Нельзя представить себе сегодня ни одного крупного исследования в области физики без ЭВМ. Исследования космического пространства своими грандиозными достижениями в значительной степени обязаны ЭВМ. Без электронных вычислительных машин немыслимо управление современными технологическими комплексами. Список подобных примеров можно продолжить. Такое широкое распространение ЭВМ объясняется способностью машины выполнять длинные последовательности операций без вмешательства человека и с большой скоростью выдавать точные результаты, хранить большой объем информации.

Рождение ЭВМ диктовалось прежде всего потребностями физики и инженерных наук. Для того чтобы решить задачи бурно развивающейся науки и техники, надо было производить астрономический объем вычислений. Вот и появился электронный арифмометр, делающий тысячи арифметических операций в секунду и предназначенный для автоматической обработки информации.

В машину исходная информация может быть введена в виде физических величин (например, электрических напряжений, пропорциональных действительным измеряемым величинам). Результаты в этом случае обычно выдаются на экран осциллографа или непрерывно выводятся в виде плавной кривой на самопишущие приборы. Схема такой машины строится в соответствии с *математической моделью* явления, т. е. системой уравнений, описывающих изучаемый процесс, поэтому эти машины называются *моделирующими*, или *аналоговыми*, вычислительными машинами (АВМ). Такого типа ЭВМ являются машинами *непрерывного действия*.

Другой тип ЭВМ — *цифровые* вычислительные машины (ЦВМ), или машины *дискретного действия*. В машинах такого типа вся информация представляется в виде цифровых кодов. ЦВМ часто называют *универсальными*, понимая под этим термином широкую сферу их использования. Среди машин этого типа тем не менее выделяют

подтипы миц- и микроЭВМ, предназначенные для вполне определенного круга решаемых задач, и *персональные* электронные вычислительные машины.

В последнее время получили распространение *аналого-цифровые* вычислительные *системы*, в которые входят аналоговая и цифровая вычислительные машины, включенные для *параллельной* работы с помощью устройства связи.

Электронная вычислительная машина, или компьютер, в этой книге понимается как комплекс технических средств, предназначенных для *автоматической обработки* информации, заданной в явном цифровом виде, т. е. предметом данной книги является *программирование* для цифровых вычислительных машин (универсальных ЭВМ). Описание процесса обработки информации, как и сама информация, сообщается машине *программой*. (отсюда и понятие программирование), первоначально записанной на бумаге в виде некоторого текста на *языке*, называемом *алгоритмическим*.

Информация, которая подвергается обработке, программа, промежуточные и окончательные результаты обработки хранятся в *памяти* ЭВМ, или *запоминающем устройстве*. Устройство, в котором выполняется большинство операций, связанных с обработкой информации, в частности арифметические операции над числами, хранящимися в машине, называется *арифметическим* устройством. Для ввода информации в машину служит специальное устройство *ввода*, а для результатов работы машины — устройство *вывода*. Для автоматического управления работой всех устройств ЭВМ в соответствии с заданной программой служит устройство *управления*.

Как правило, ЭВМ обладает несколькими запоминающими устройствами, среди которых имеется по крайней мере одно устройство, предназначенное для приема, хранения и выдачи информации для арифметического устройства, т. е. тесно связано с арифметическим устройством. Такое запоминающее устройство называется *оперативным* (внутренним) запоминающим устройством (ОЗУ) или *оперативной* (внутренней) *памятью*. Наряду с оперативной памятью в состав ЭВМ входит также *постоянное* запоминающее устройство (ПЗУ), содержимое которого можно только считывать. Соответствующая информация в ПЗУ заносится при его изготовлении, сохраняется при включении и выключении ЭВМ и, как правило, не может быть изменена. Остальные запоминающие устройства называют *внешней* (периферийной) *памятью* или внешними запоминающими устройствами (ВЗУ). Объем внешних запоминающих устройств превосходит объем внутренней памяти, но время обращения к ОЗУ значительно меньше, чем к внешней памяти.

Информация из оперативного запоминающего устройства обычно извлекается определенными порциями, поэтому для удобства пользования ОЗУ разбивается на части — *ячейки*, содержащие

вполне конкретное число *разрядов*, куда с помощью *двоичных знаков* (т. е. нулей или единиц) может быть записано *машинное слово*.

Машинное слово представляет собой записанный в ячейке при помощи двоичных знаков код числа или код команды. *Команда* — это инструкция или приказ машине о выполнении некоторой *операции*. Совокупность всех операций, которые может выполнить ЭВМ, называется *системой команд* (машинных кодов) этой машины.

Средства выполнения арифметических и логических операций, средства управления выполнением заданной программой последовательности действий (команд), средства обращения к оперативной памяти и организация обмена информацией между ОЗУ и ВЗУ объединяют единым понятием *процессор* (иногда *центральный процессор*, имея в виду, что ЭВМ может иметь несколько процессоров). Таким образом, процессор составляют арифметическое устройство и устройство управления. В некоторых моделях ЭВМ с процессором конструктивно объединено и оперативное запоминающее устройство, что дает основание считать его составной частью процессора, но логически это два различных типа устройств.

Каждая ЭВМ характеризуется некоторыми основными технико-экономическими показателями, среди которых можно выделить *быстродействие* (некоторое среднее число производимых в секунду операций в центральном процессоре), *объем оперативной памяти* (количество информации, одновременно хранимой во внутренней памяти), *надежность* и *стоимость*. Наиболее важными из них, по крайней мере для пользователя, являются первые два, так как от этих показателей прежде всего зависит эффективность работы ЭВМ. Совершенно очевидно, что универсальные ЭВМ должны обладать достаточным быстродействием и большой памятью, а ЭВМ, имеющие специальное назначение, не обязаны иметь оба эти показателя высокими. Например, в ряде экономических задач, где обрабатываются большие массивы данных при не слишком большом числе операций, важен большой объем оперативной памяти; для машин с большой библиотекой встроенных *подпрограмм*, встречающихся в какой-либо определенной отрасли, важно быстродействие.

В зависимости от уровня инженерно-технических средств, использованных при изготовлении ЭВМ, технико-экономические показатели ЭВМ разных моделей могут быть существенно различными. ЭВМ отличаются друг от друга *элементной базой*, конструктивно-технологическим исполнением, логической организацией, *программным обеспечением*, техническими характеристиками, степенью доступа к ЭВМ со стороны пользователей и др. В процессе развития ЭВМ основной тенденцией всегда было стремление повысить эффективность использования машин, облегчить связь операторов с ЭВМ, стремление упростить подготовку программ решаемых задач, умень-

шить трудоемкость процесса переложения условия задачи (алгоритма) на язык машины.

В зависимости от перечисленных и ряда других характеристик условно принято выделять *поколения* ЭВМ. Поскольку улучшение технико-экономических показателей в значительной степени зависит от используемых для построения ЭВМ электронных схем, то принято считать, что критерием поколения ЭВМ служит их элементная база.

Важной характеристикой ЭВМ является также ее программное обеспечение. Программным (математическим) обеспечением принято считать совокупность программ, которыми снабжена ЭВМ, таких, что позволяют без программирования решать некоторые задачи, выполняют ряд работ при программировании, а также обеспечивают выгодный режим работы машины.

Основным активным элементом ЭВМ *первого* поколения является *электронная лампа*. Появление машин первого поколения относится к пятидесятым годам нашего века. Типичные представители этого поколения из отечественных ЭВМ — это БЭСМ-1, Минск-1, Урал-1, Урал-2, Урал-4, М-1, М-3, БЭСМ-2, Стрела и др. Они были значительных размеров, потребляли большую мощность, имели невысокую надежность работы и слабое программное обеспечение. Быстродействие их не превышало 2—3 тыс. операций в секунду, емкость оперативной памяти — 2 К, или 2048 машинных слов (1 К-1024) длиной 48 двоичных знаков. В 1958 г. появилась машина М-20 с памятью 4 К и быстродействием около 20 тыс. операций в секунду.

В машинах первого поколения были реализованы основные логические принципы построения машин и продемонстрированы возможности цифровой вычислительной техники.

В ЭВМ *второго* поколения элементной базой служат *транзисторы*. Появление полупроводниковых элементов в электронных схемах существенно увеличило емкость оперативной памяти, надежность и быстродействие ЭВМ. Уменьшились размеры, масса и потребляемая мощность.

С появлением машин второго поколения значительно расширилась сфера использования электронной вычислительной техники, главным образом за счет развития программного обеспечения. Появились также специализированные машины, например ЭВМ для решения экономических задач, для управления производственными процессами, системами передачи информации и т. д.

К ЭВМ второго поколения относятся Урал-14, Урал-16, Минск-22, Минск-32, БЭСМ-3, БЭСМ-4, М-220, М-222, БЭСМ-6, МИР-2, Наири и др. Первые четыре из названных использовались главным образом для решения экономических задач, последние три относятся к подтипу мини-ЭВМ, остальные являются универсальными. ЭВМ БЭСМ-4, М-220, М-222 имеют быстродействие порядка 20—

30 тыс. операций в секунду, а оперативную память — соответственно 8 К, 16 К и 32 К.

Среди машин второго поколения особо выделяется БЭСМ-6, обладающая быстродействием около миллиона операций в секунду и оперативной памятью от 32 К до 128 К (в большинстве машин используется два сегмента памяти по 32 К каждый).

Элементная база ЭВМ *третьего* поколения основана на микроэлектронике, и это поколение характеризуется широким применением *интегральных схем*. Интегральная схема — это логически законченный функциональный блок, соответствующий достаточно сложной транзисторной схеме. Размеры такого блока невелики, из них собираются более сложные узлы методом печатного монтажа. Благодаря интегральным схемам удалось существенно улучшить технико-эксплуатационные характеристики ЭВМ. Например, машины третьего поколения по сравнению с машинами второго поколения имеют больший объем оперативной памяти, увеличилось быстродействие, повысилась надежность, а потребляемая мощность, занимаемая площадь и масса уменьшились.

К машинам третьего поколения относятся все серийные ЭВМ Единой системы (ЕС-1010, ЕС-1020, ЕС-1030, ЕС-1040, ЕС-1050, ЕС-1060, ЕС-1066 и несколько промежуточных модификаций — ЕС-1021, ЕС-1033, ЕС-1045, ЕС-1055, ЕС-1061 и др.), агрегатная система средств вычислительной техники, мини- и микроЭВМ Электроника 60, Электроника 100/125, Электроника 79, СМ-3, СМ-4 и др.

ЭВМ ЕС разработаны и промышленно выпускаются совместными усилиями стран — членов СЭВ. Все машины Единой системы по принципам функционирования и взаимодействия основных элементов, по составу периферийного оборудования, которое может быть подключено к центральному процессору, представляют собой программно-совместимые ЭВМ, предназначенные для решения широкого круга научно-технических, экономических задач и использования в автоматизированных системах управления (АСУ).

ЭВМ ЕС-1010 разработана в ВР и имеет быстродействие до 10 тыс. операций в секунду, а объем оперативной памяти — от 8 до 64 Кбайт, *байт* состоит из восьми *бит*, где бит — единица информации, определяемая как количество информации, которое содержится в сообщении, состоящем из одного двоичного знака.

ЕС-1021 создана в ЧСФР. Она имеет некоторые конструктивные особенности, так как проектировалась главным образом как машина для управления технологическими процессами. Быстродействие ее 15 тысяч операций в секунду, объем оперативной памяти — от 16 до 64 Кбайт.

Машины ЕС-1010 и ЕС-1021 являются малыми моделями ЕС ЭВМ (мини-ЭВМ), все остальные — универсальные ЭВМ, причем с возрастанием номера модели, как правило, мощность машины рас-

тет и ее технико-экономические показатели улучшаются. В частности, модели ЕС-1050, ЕС-1060 и ЕС-1066 разработаны в СССР. Среднее быстродействие первой из них 500 тыс. операций в секунду, второй — 1,0—1,3 млн. операций в секунду, а производительность ЭВМ ЕС-1066 достигает более 2 млн. операций в секунду. Объем оперативной памяти этих моделей соответственно: от 256 до 1204 Кбайт, от 2048 до 8192 Кбайт и 8192 Кбайт.

Все ЭВМ третьего поколения, помимо элементной базы, существенно отличаются от ЭВМ второго и первого поколений и другими характеристиками.

Прежде всего, ЭВМ третьего поколения оперируют с произвольной буквенно-цифровой информацией. Единицей адресации памяти является байт, в котором может храниться восьмиразрядный двоичный код, представляющий собой либо две десятичные цифры, либо один алфавитный символ. В соответствии с этим изменилась система команд ЭВМ и появилась возможность каждую ячейку памяти полностью загрузить информацией. По этой же причине объем оперативной памяти ЭВМ третьего поколения указывается не в К машинных 48-разрядных словах, а в К байт.

В машинах третьего поколения машинное слово стандартной длины содержит 4 байта. Рассматриваются также *полуслова* (длина 2 байта) и *двойные слова* (8 байт).

В отличие от ЭВМ первого и частично второго поколения, все основные устройства которых, такие, как центральный процессор и устройства ввода или вывода, работают последовательно, современные машины третьего поколения обладают возможностью параллельной работы устройств. Появились *многопрограммные* ЭВМ, которые (в отличие от однопрограммных машин, где программы выполняются только поочередно) могут одновременно реализовать несколько программ (принцип *мультипрограммирования*) именно за счет организации параллельной работы основных устройств машины.

Принцип мультипрограммирования, совместимость систем программирования, т. е. систем методов и приемов обеспечения удобного и быстрого обмена информацией между человеком и машиной на уровне машинных кодов, высокий уровень технической стандартизации и унификации — все эти качества присущи ЕС ЭВМ.

Следствием реализации принципа мультипрограммирования является возможность работы в режиме *разделения времени* и в режиме *диалога*, при котором несколько пользователей, удаленных от ЭВМ на достаточно большие расстояния, могут общаться с ней непосредственно, оперативно и независимо друг от друга. Взаимодействие удаленных пользователей с ЭВМ осуществляется с помощью периферийной аппаратуры, подсоединенной к каналам связи через стандартную систему сопряжения. Благодаря этому центральный про-

цессор может работать с большим набором разнообразных периферийных устройств (например, экранных пультов управления и др.).

В настоящее время активно строятся ЭВМ четвертого поколения, основанные на применении *больших интегральных схем*, представляющих собой совокупность нескольких десятков электрических цепей, образованных в одном массиве полупроводникового материала (кремниевых пластин) и объединенных внутренними связями в единый функциональный блок. Высокая степень интеграции способствует увеличению плотности компоновки электронной аппаратуры, повышению ее надежности, что ведет к увеличению быстродействия ЭВМ и снижению ее стоимости. Все это оказывает существенное воздействие на логическую структуру (*архитектуру*) ЭВМ и на ее программное обеспечение. Более тесной становится связь структуры машины и ее программного обеспечения, особенно *операционной системы* (ОС) — комплекса служебных программ, которые организуют непрерывную работу машины без вмешательства человека, обеспечивающих доступ к возможностям оборудования и создающих обстановку, в рамках которой программист может создавать, изменять и выполнять свои программы.

ЭВМ четвертого поколения обеспечивают коммунальное использование вычислительных мощностей. Они связаны в единую *вычислительную сеть* и обеспечивают работу в режиме разделения времени. В СССР в настоящее время разработан *многопроцессорный* вычислительный комплекс «Эльбрус». Например, одна из разновидностей этого комплекса Эльбрус 1-КБ имеет быстродействие до 5,5 млн. операций с плавающей запятой, а объем оперативной памяти доведен до 64 Мбайт. Рекордное быстродействие, достигаемое ЭВМ четвертого поколения, составляет несколько десятков миллионов операций в секунду, а объем оперативной памяти — до сотен Мбайт.

Появление *микропроцессоров* в 70-е годы явилось важным революционным этапом в развитии вычислительной техники. Благодаря микропроцессору, представляющему собой центральный блок вычислительной системы, размещенному на одном кристалле кремния размерами несколько миллиметров, стало возможным создание персональной ЭВМ или персонального компьютера. Используемые в ПЭВМ микропроцессоры имеют производительность порядка нескольких сот операций в секунду и работают со словами длиной 16—32 разряда, при этом их стоимость сравнительно низкая. Характерный объем прямо адресуемой памяти для таких микропроцессоров от 1 до 16 Мбайт. Однако доступный объем рабочей памяти, т. е. объем оперативного запоминающего устройства, в наиболее распространенных 16-разрядных ПЭВМ составляет 600—700 Кбайт. В отдельных типах ПЭВМ допускается расширение оперативной памяти до 2—3 Мбайт.

Современный персональный компьютер имеет небольшие размеры, конструктивно состоит из трех основных элементов: *системного блока, клавиатуры и дисплея*, размещаемых на рабочем столе пользователя. К этим основным компонентам добавляется еще *печатающее устройство* (принтер), а в состав системного блока входят средства поддержки коммуникаций для связи с дополнительными устройствами, не относящимися непосредственно к основному комплекту ПЭВМ, например, средства связи с другими ЭВМ.

Объединенные в сети персональные ЭВМ совместно с более мощными вычислительными системами и периферийными устройствами становятся очень эффективным орудием для решения сложнейших задач науки и техники.

ЭВМ пятого поколения базируются не столько на стремлении извлечь дополнительные преимущества из сверхкомпактной упаковки *сверхбольших интегральных схем (СБИС)*, сколько на реализации широкой программы, предусматривающей достижение принципиальных сдвигов во всех областях, связанных с конструированием, производством, техническим обслуживанием и использованием компьютеров. В числе прочих могут быть выделены проблемы создания так называемых *экспертных систем*, которые включают *базы знаний*, обеспечивающие решение сложных задач и принятие решений, а также предусматривают ввод и вывод информации с помощью голоса и графических изображений; проблемы разработки языков сверхвысокого уровня, в том числе функциональных и логических, обеспечивающих совершенные способы программирования; проблемы децентрализованных вычислений с помощью сетей ЭВМ, как больших, находящихся на значительном расстоянии друг от друга, так и миниатюрных микроЭВМ, размещенных на одной печатной плате или даже на одном кристалле полупроводника; проблемы технологии сверхбольших интегральных схем для ЭВМ общего применения и специализированных ЭВМ, обладающих громадной вычислительной мощностью при умеренной стоимости.

Таким образом, основополагающая идея создания ЭВМ пятого поколения состоит в разработке сбалансированной по своей архитектуре *системы компьютеров*, чтобы каждый пользователь такой системы имел возможность (в тот момент, когда и он и система к этому готовы) привести в действие ее огромный вычислительный потенциал.

Для ЭВМ *последующих* поколений по всей видимости элементная база будет основываться на *оптико-электронных* элементах. В этих машинах носители энергии служат не электроны, а фотоны, что значительно повышает скорость передачи сигналов, так что быстроедействие ЭВМ может достигнуть сотен миллионов операций в секунду. Преобразование электрических сигналов в оптические в таких машинах осуществляется лазерами и светонепрелучающими дио-

дами, используются световоды и фотоприемники. Дальнейшее развитие получает начавшийся еще в четвертом поколении процесс образования вычислительных систем, сращивания машин и вычислительных центров с системой связи. Меняется и представление о системе связи, которая в дальнейшем пользователю должна оказывать не только услуги передачи информации, но и ее хранения и обработки.

По прогнозам в следующих поколениях появятся машины с быстрой работой до миллиарда операций в секунду. Эта скорость будет повышаться благодаря параллельной работе всех устройств, в том числе нескольких процессоров. По всей видимости, станет возможным осуществление параллельного преобразования информации типа тей, которая представляется в виде голограмм с помощью систем лазерных элементов. Небезынтересными являются также прогнозы по разработке соответствующих «вычислительных сред» и ряд других, кажущихся сейчас фантастическими, проектов вычислительных машин.

К настоящему времени скорость выполнения операций в центральном процессоре уже достаточно высока, однако полное ее использование сдерживается в определенной степени медленностью периферийных устройств. Правда, современные операционные системы с многими каналами ввода и вывода позволяют частично решить эту проблему. В то же время широкая область использования ЭВМ обеспечивается в значительной степени именно благодаря тому, что разработано много различных периферийных устройств. К таким устройствам, которые необходимы для использования машин независимо от областей их применения, относятся внешние запоминающие устройства на *магнитных барабанах, лентах и дисках*, традиционные устройства ввода и вывода информации через *перфокарты и перфоленты*, *алфавитно-цифровые печатающие устройства (АЦПУ)*, электрифицированные пишущие машинки. Ряд устройств расширяет возможности использования машин в самых различных областях. Это устройства, обеспечивающие дистанционную обработку информации, например абонентские пункты для пользователей; устройства, облегчающие общение человека с машиной, например различные операторские пульта со средствами наглядного отображения алфавитно-цифровой и графической информации на электронно-лучевых трубках — дисплей и др.

Внешние запоминающие устройства используются для создания *архивной памяти*, т. е. для хранения исходной информации, отдельных частей (или всей) выполняемой программы, вспомогательных программ, а также промежуточных и конечных результатов, если объем получаемой информации настолько велик, что не может разместиться в ОЗУ, или полученная в данный момент информация может понадобиться в дальнейшем. Современные ВЗУ

имеют, как правило, емкость памяти, намного превосходящую емкость оперативной памяти ЭВМ. Так, магнитный барабан ЕС-5033 имеет емкость 5,6 Мбайт, скорость записи/считывания 1200 Кбайт/с, среднее время доступа к информации 21 мс; магнитная лента ЕС-5014 имеет емкость 40 Мбайт, скорость записи/считывания 126 Кбайт/с, время разгона и останова ленты 5 мс, номинальная скорость движения ленты 2 м/с. Стационарный пакет дисков ЕС-5051 имеет емкость 100 Мбайт, скорость записи/считывания 83,3 Кбайт/с, среднее время доступа к информации 420 мс, а сменный пакет дисков ЕС-5050 имеет соответственно следующие параметры: 7,25 Мбайт, 156 Кбайт/с, 90 мс. Базовый комплект ВЗУ для ЕС-1050 составляют 4 комплекта магнитных дисков и 16 магнитофонов.

В последнее время появились громадные возможности расширения архивной памяти, предоставляемой пользователю. Это достигается благодаря *видеодискам*, на каждом из которых может быть записана информация емкостью в 10 и более Гбайт. Несколькими сотнями таких дисков могут содержать информацию, равную фондам всех крупнейших библиотек мира. Трудность заключается, однако, в том, как такой огромный объем информации перенести на диски.

Скорость считывания для устройства ввода с перфокарт ЕС-6012 составляет до 500 карт/мин, для устройства ЕС-6013 — до 1200 карт/мин, а для устройства ЕС-6014 — до 2000 карт/мин. Алфавитно-цифровое печатающее устройство АЦПУ-128-6 печатает до 900 строк/мин.

Периферийные устройства ЭВМ постоянно совершенствуются. Так, в небольших вычислительных системах, подобных ПЭВМ, наиболее распространенным типом диска является *гибкий диск*, обычно называемый *дискетой*. Дискеты выпускаются двух размеров: 133 мм и 203 мм по диаметру и представляют собой гибкий пластик в форме диска, покрытый магнитной пленкой и помещенный в твердый пластмассовый конверт для защиты от повреждений. Емкость отдельного дискета колеблется от 90 Кбайт до 1 Мбайт и более в зависимости от применяемого дисководов.

Другим распространенным видом дискового запоминающего устройства является так называемый *винчестерский диск*. Это жесткий диск, на который информация записывается по специальной, винчестерской технологии, позволяющей хранить на одной стороне диска в 100 раз больший объем информации, чем при использовании обычной технологии. Эти диски несъемные и заключены в герметический корпус. Емкость памяти винчестерского диска — десятки Мбайт.

Первоначально ЭВМ использовались только в тех сферах, где было необходимо выполнить большое количество арифметических операций, например при обработке бухгалтерских данных или при расчетах различных вариантов в отраслях техники, в промышленно-

сти. В последнем случае создание, например, нового технологического комплекса требует согласования многих взаимосвязанных, порой противоречивых условий. Машина помогает исследовать много вариантов в зависимости от исходных данных. Сопоставление различных по выбранному параметру (например, минимальным затратам на осуществление проекта) вариантов позволяет выбрать оптимальный. Такой выбор тоже можно поручить машине. В научно-исследовательской работе с помощью ЭВМ могут быть исследованы методы решения математических задач, изучены свойства различных физических, биологических, экономических, социальных систем и других объектов, выявлены зависимости от конкретных факторов и параметров.

Благодаря ЭВМ эффективность работы математиков-вычислителей увеличилась в десятки тысяч раз. Это привело к новому качественному скачку. Стало возможным решать задачи, которые совсем недавно даже не ставились, однако стиль работы сначала во многом напоминал традиционный — как правило, рассматривалась конкретная, хорошо поставленная задача, и специалист-математик строил алгоритм ее решения. В то же время сложные проблемы трудно формализовать, т. е. свести к хорошо поставленной задаче. Кроме того, эффективность алгоритма требует вмешательства человека на различных этапах его реализации. Таким образом, вместо традиционного анализа задачи возникает система, куда входят человек, т. е. сам исследователь, ЭВМ как носитель и средство получения и хранения информации и система программного обеспечения — аппарат, позволяющий исследователю активно вмешиваться в процесс исследования. Такая система — это уже инструмент совершенно нового качества.

В связи с этим качественно изменился характер решения многих старых и новых проблем, таких, как создание *автоматизированных систем управления* (АСУ), моделирование физических процессов, научное прогнозирование (например, ожидаемых экономических ситуаций по данным темпов производства и реализации продукции с учетом возможных действий партнеров и конкурентов) и другие задачи.

Возможности персональных компьютеров позволяют, в частности, создавать *автоматизированные рабочие места* (АРМ) нового поколения. Основными компонентами программного обеспечения такого рода *интеллектуальных рабочих станций* являются функциональные процессоры, предоставляющие возможность удобного использования разных видов информации, интерфейс с пользователем, обеспечивающий эффективный диалоговый режим работы и мощная *база данных*, являющаяся хранилищем как исходной информации, так и получаемой при работе с функциональными процессорами.

И, наконец, ЭВМ используются в так называемых невычислительных приложениях. Это перевод с одного языка на другой, в том числе и *трансляция* (перевод с алгоритмического языка на язык конкретной машины), моделирование работы одной ЭВМ на другой (на этапе проектирования), моделирование *искусственного интеллекта* и др.

§ 2. Решение задач на ЭВМ

При решении любой задачи на ЭВМ предполагается, что некоторая информация подвергается *обработке* по предварительно составленной инструкции, называемой программой. Поэтому под решением задачи на ЭВМ подразумевается гораздо больший круг действий, чем только работа машины.

На первом этапе выбирается общий подход к решению; устанавливается, каким целям должно служить решение задачи и при каких условиях оно будет существовать. Здесь производится разбиение задачи на более мелкие, определяется последовательность их решения и выполняется ряд других действий, т. е. осуществляется общая *постановка задачи*. Например, задача изучения структуры ударной волны в газе требует четкого определения, каковы рассматриваемые параметры газа (плотность, температура и др.), какая будет исследована волна (в аэродинамической трубе или перед сверхзвуковым снарядом в атмосфере Земли) и т. д.

После этого этапа необходимо дать математическое описание задачи. Появление ЭВМ способствовало развитию приложений математики, что привело к математизации различных отраслей науки. Для того чтобы можно было интересующее нас явление подвергнуть математическому анализу, должны быть выполнены некоторые условия, должна существовать математическая теория явления, описывающая его закономерности в виде формул. Такой набор формул называют *математической моделью* явления. Лишь в очень простых и редких случаях математическая модель является в то же время и расчетной схемой, т. е. позволяет по имеющимся исходным данным получить требуемые результаты. Модель определяет искомые величины, как правило, неявно, в виде системы зависимостей, которым эти величины должны удовлетворять, но такая система зависимостей обычно неполна, она оставляет математику большую или меньшую свободу выбора. Выбор надо произвести так, чтобы получить наилучшее значение некоторого критерия. Указание этого критерия также входит в математическую модель.

Когда критерий выбран или написана полная система уравнений, определяющих исследуемое явление, задача становится уже чисто математической — задача математически поставлена. Однако при решении многих практических задач приходится огрублять

и упрощать их постановку, чтобы суметь воспользоваться теми методами решения, которые может предложить современная математика, или же разработать новый метод решения.

За математической постановкой и разработкой метода решения следует *реализация* выбранного метода на ЭВМ. Конечный результат предыдущих постановочных этапов — метод решения задачи — описан математическим языком, т. е. достаточно формально. Программа для ЭВМ — это также формальное описание того же метода. Кажется бы, превратить одно формальное описание в другое — дело несложное и тоже формальное, т. е. может быть выполнено механически. Это верно лишь до некоторой степени. Программист хорошо чувствует, насколько приблизительно и неполно описывают математики методы решения даже чисто вычислительных задач. Это связано с тем, что математик, как любой автор, рассчитывает на известный уровень знаний, опыта и интуиции своего читателя. Все эти качества присущи лишь человеку. Машина обладает тоже подобием знаний и опыта, но в совсем иной форме, чем человек. Для разговора с ЭВМ человек вынужден пользоваться или ее собственным языком, или в лучшем случае языком, значительно более формализованным, а следовательно, и более бедным, чем естественный человеческий язык, которым пользуются математики.

С другой стороны, язык вычислительной машины хотя и формален, но далеко не абстрактен. В языке ЭВМ, являющемся весьма конкретным, должны быть отражены принципиальные особенности машины, например, отсутствие понятия непрерывности, конечность величин, определенные законы машинной арифметики. Так, арифметические операции в вычислительной машине, строго говоря, имеют не так уж много общего с математическими операциями над вещественными числами. ЭВМ выполняет приближенные операции над приближенными представлениями вещественных чисел, поскольку точно представить все вещественные числа в действительном мире конечных вычислительных машин невозможно. Определенные способы приближенного представления чисел и действий с ними встроены в конструкцию ЭВМ.

Итак, этап реализации метода решения содержит в себе исследование математической модели явления и переход от нее к расчетной схеме, включая анализ с оценкой погрешности результатов, собственно программирование — перевод схемы на язык машины, когда формируется последовательность операций, выполняемых машиной, и, наконец, контроль за выполнением программы машиной.

Процесс программирования обычно состоит из двух частей: составления *блок-схемы* программы — графического изображения последовательности действий и написания программы на языке, который ЭВМ понимает непосредственно или с помощью транслятора, т. е. программы-переводчика.

Программу в простейшем случае можно определить как последовательность предложений, написанных на некотором произвольном языке и допускающую однозначность толкования. Например, последовательность предложений

заданы ($f_1=1$, $f_2=2$, $f_3=3$);

сложить (f_1 , f_2 , S_2); сложить (f_3 , S_2 , S_3);

есть программа вычисления суммы $S_3=f_1+f_2+f_3$, написанная на языке, в котором слова «заданы», «сложить», цифры, буквы, скобки, запятая, точка с запятой и знак равенства машиной понимаются однозначно. Вопросы, связанные с составлением блок-схемы программы и уточнения понятия программы, будут рассмотрены в следующем параграфе.

Контроль за выполнением программы машиной включает в себя *отладку* программы и получение результатов. Отладка необходима, поскольку вероятность допустить ошибку при написании программы очень велика. Для обнаружения и устранения ошибок используется сама вычислительная машина. Под отладкой понимается не только устранение ошибок, допущенных при записи программы, но и процесс совершенствования (оптимизации) программы.

Последний этап решения задачи — *обработка* результатов. Продолжительность его зависит от конкретной задачи. Математик будет интерпретировать полученные данные в соответствии с поставленной целью и иногда все делать заново. В обработке результатов, так же как и в их получении, участвует и ЭВМ (рисует графики, выводит данные на телевизионный экран и т. д.).

Таким образом, в результате прохождения всех этапов решения задачи в машине реализуется некоторый процесс. Прототипом такого процесса может быть физическое или биологическое явление, событие в общественной сфере. Реализуемый на ЭВМ процесс может быть назван *вычислительным экспериментом*. При этом ЭВМ не освобождает пользователя от понимания сущности задачи, так как он должен уметь «поставить процесс на машину», что требует глубокого знания особенностей задачи. Это значит, что он должен научиться описывать процесс на языке математики, строить математическую модель.

И, наконец, можно сделать вывод, что программирование — это лишь один из этапов решения задачи на ЭВМ, это лишь составная часть вычислительного эксперимента.

§ 3. Понятия алгоритма и программы

Процесс постановки задачи для вычислительной машины в определенной степени аналогичен постановке задачи человеку, производящему вычисления карандашом на бумаге. Однако если во втором случае это можно успешно сделать средствами обычного разгово-

ного языка с привлечением тех понятий, которые используются в математической литературе, то для ЭВМ словесное описание оказывается громоздким, а главное, недостаточно четким и строгим.

Возникает необходимость в создании аппарата для полного, ясного и однозначного описания вычислительных процессов. Это описание должно содержать формулы, по которым происходит расчет, определять последовательность их применения, условия, при которых используется та или иная формула, а также указывать правила перехода от одной формулы к другой, от одной части вычислительного процесса к другой части.

После того как машиной получено задание, реализация вычислительного процесса внутри ЭВМ происходит автоматически без вмешательства человека. Аппарат для описания процесса должен содержать сведения, как поступить машине при любых обстоятельствах, которые могут возникнуть во время ее работы, т. е. заранее необходимо предусмотреть все возможные ситуации. Машина является аккуратным исполнителем задания, но она не может принять решение, если программист не дал ей соответствующие указания. ЭВМ необходимо указать характер и свойства тех или иных математических объектов — чисел, векторов, матриц и т. д., которые служат исходными данными для решения задачи или используются в ходе вычислений.

Таким образом, при оформлении задания машине возникает необходимость в точном и полном описании определенного вычислительного процесса или любой иной последовательности действий, выполняемых ЭВМ. Конструктивное описание, состоящее из конечного множества правил и определяющее процесс переработки данных, называется операционным правилом обработки или *алгоритмом*. Следовательно, алгоритм задает некоторый свод правил, описывающих процесс, протекающий во времени и определяющий последовательность действий для перехода от исходной ситуации к желаемому новому состоянию. Описание такого свода правил может быть выполнено на обычном языке с помощью математических формул или же специальных символов.

Правила обработки информации для ЭВМ задаются в программе — последовательности предложений, написанных на некотором произвольном, понятном ЭВМ языке, допускающей однозначность толкования и реализующей данный алгоритм, т. е. программа есть запись алгоритма для решения задачи на ЭВМ. Разработка алгоритма для решения любой задачи является наиболее ответственным и важным моментом, так как именно алгоритм определяет ту последовательность действий, которая выполняется машиной. Ошибки, допущенные при записи алгоритма, обычно приводят к неверному ходу вычислительного процесса и, следовательно, к неверному результату. Результат может быть верным, но полу-

ченным не оптимальным путём, если используется алгоритм, не учитывающий индивидуальных особенностей задачи. Основная задача при эффективном использовании ЭВМ — построение хорошего алгоритма.

Основу вычислительного эксперимента составляет последовательность: модель — алгоритм — программа, цикл которого разбивается на описанные выше этапы: построение физической и соответствующей ей математической модели; разработка вычислительного алгоритма в виде программы для ЭВМ; проведение расчетов на ЭВМ; обработка, анализ и интерпретация результатов расчетов, включая сопоставление с физическим экспериментом и, в случае необходимости, уточнение или пересмотр математической модели, т. е. возвращение к первому этапу и повторение цикла вычислительного эксперимента. Таким образом, технологически вычислительный эксперимент состоит из двух фаз: формирования и калибровки моделей, а затем прогноза с помощью выбранных математических моделей.

Рассмотрим построение алгоритма и составление программы на простом примере вычисления значений x , удовлетворяющих квадратному уравнению $ax^2+bx+c=0$.

Для человека, изучившего методы нахождения корней квадратного уравнения, задача понятна при произвольных значениях коэффициентов a, b, c . Для ЭВМ необходимо расписать последовательность всех действий, причем нужно предусмотреть все возможные ситуации, возникающие вследствие тех или иных значений коэффициентов, которые полагаются вещественными, т. е. могут быть положительными, отрицательными или нулевыми.

Существуют различные методы решения этой задачи. В частности, можно пользоваться привычными формулами

$$x_1 = -\frac{b}{2a} + \frac{\sqrt{b^2 - 4ac}}{2a}, \quad x_2 = -\frac{b}{2a} - \frac{\sqrt{b^2 - 4ac}}{2a},$$

которые будут лежать в основе алгоритма, так как они указывают последовательность действий при решении задачи. После подстановки в эти формулы числовых значений a, b, c производятся вычисления и получается решение в виде двух чисел. Остается предусмотреть исключительные ситуации; в противном случае составленная программа не будет полностью описывать вычислительный процесс, т. е. не будет универсальной.

Исключительными могут быть следующие ситуации.

1) $a=0$. Уравнение перестает быть квадратным. Приведенные формулы неприменимы, и единственный корень линейного уравнения $bx+c=0$ вычисляется по формуле $x=-c/b$. Случай $b=0$ (при $a=0$) требует, чтобы при этом и $c=0$, т. е. уравнение вообще отсутствует и задачи как таковой нет. Однако учитывая, что в большинстве практических задач входящие в них параметры суть результат

физических измерений, отбрасывать случай, когда $a=b=0$, не следует. Он может указывать, в частности, на сбой в процессе измерений или особенность системы при некоторой комбинации параметров и т. д.

2) $b^2-4ac \leq 0$. В случае равенства нулю дискриминанта уравнение имеет два совпадающих корня; если же подкоренное выражение отрицательное, то корни комплексные и вычислять квадратный корень следует из значения этого выражения с обратным знаком.

Блок-схема программы, т. е. графическое изображение последовательности действий при реализации алгоритма, составляется из элементов в виде геометрических фигур, имеющих по предварительному соглашению вполне определенный однозначный смысл и обозначающих различные действия. Эти элементы соединяются между собой стрелками, указывающими направления переходов от одних действий (операций) к другим. Будем считать, что элемент в виде ромба означает проверку некоторого условия или сравнение, в виде прямоугольника — вычислительные операции любого вида, операции ввода и вывода данных обозначим фигурами в виде профиля бочонка. Внутри фигур (блоков, блок-схем), как правило, указываются те операции, которые в данном блоке выполняются. Для построения блок-схемы могут использоваться и другие геометрические фигуры.

Наименования величин в программе можно выбирать произвольно. Обычно применяются такие обозначения, которые подсказывают, что данная переменная в действительности означает. Исходя из этого принципа, наименования величин, являющихся коэффициентами a, b, c уравнения, оставим прежними. Для обозначения двух корней введем четыре числа, представляющие собой вещественные ($x1r, x2r$) и мнимые ($x1i, x2i$) части первого и второго корней соответственно. Если корни вещественные, то $x1i=x2i=0$; если корни совпадают, то $x1r=x2r$. Подкоренное выражение понадобится дважды, есть смысл обозначить его одним символом, например D , и вычислить один раз; можно обозначить также \sqrt{D} по той же причине, например, символом S . Еще введем константу k , значение которой полагается равным нулю, если уравнение вырожденное ($a=b=c=0$), и равным единице, если уравнение линейное; в случае квадратного уравнения $k=2$. Значение k будет составной частью решения вместе со значениями $x1r, x2r, x1i, x2i$.

Для указания конкретных операций, выполняемых в блоке, обычно используются математические символы или записи в виде текста. Употребляются также некоторые специальные символы, например символ $:=$ означает, что указанное в блоке значение должно быть заменено новым.

Блок-схема программы для рассматриваемой задачи может быть изображена в таком виде, как показано на рис. 1.

Отметим, что из блоков, обозначенных ромбами, всегда выходит больше одной стрелки, т. е. эти блоки вызывают разветвление программы. Случай $D=0$ целесообразно рассматривать как случай вещественных корней. Во-первых, отпадает третья ветвь в условии $D \geq 0$, во-вторых, точное равенство нулю дискриминанта маловероятно.

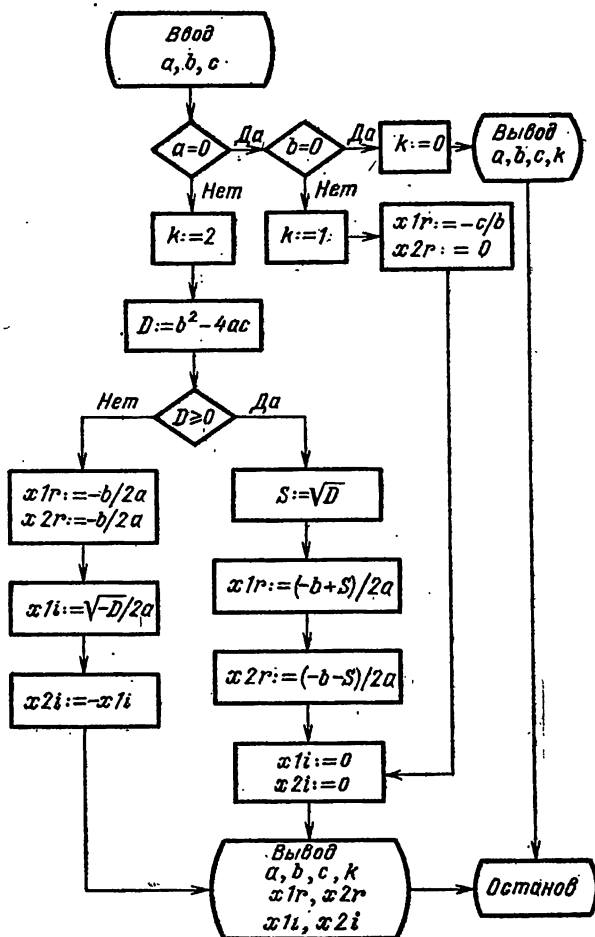


Рис. 1

ятно, поскольку значения a, b, c являются результатами измерений. Ненужная операция вычисления квадратного корня из нуля и двух нулевых значений $x1i, x2i$ в тех редких случаях, когда $D=0$, потребует меньше машинного времени, чем проверка в каждом варианте равенства D нулю.

Одинаковые операции для различных ветвей программы могут быть изображены на блок-схеме одним блоком, т. е. только один раз. В таком случае в один блок входит больше одной стрелки. В рассматриваемом примере по две стрелки направлены в блок вычисления нулевых значений мнимых частей корней в случае вещественных корней и в блок вывода в случаях $k=1$ и $k=2$.

Оформление алгоритма в виде блок-схемы на этом закончено. Остается записать его в виде программы на языке, понятном вычислительной машине. Программа вычисления корней квадратного уравнения с вещественными коэффициентами на языке Фортран будет дана в гл. II, на языке Бейсик в гл. III, а на языке ПЛ/1 — в гл. IV.

Упражнения

1. Составить блок-схему для решения системы уравнений

$$a_{11}x_1 + a_{12}x_2 = b_1, \quad a_{21}x_1 + a_{22}x_2 = b_2$$

с учетом обстоятельства, что система может иметь единственное решение, бесконечное множество решений или вовсе не иметь решения.

2. Дана прямая $y=ax+b$ и окружность $x^2+y^2=r^2$. Исследовать возможности наличия общих точек у этих линий и составить блок-схему вычислений.

3. В n -мерном пространстве заданы векторы x и y с компонентами x_1, x_2, \dots, x_n и y_1, y_2, \dots, y_n . Найти вектор z вида $x+ky$, ортогональный вектору y . Составить блок-схему для решения этой задачи.

§ 4. Алгоритмические языки

Развитие вычислительной техники обусловило появление *языков программирования*. Назначение такого языка — в оснащении набора вычислительных формул дополнительной информацией, которая превращает этот набор в алгоритм. В дальнейшем под языком программирования понимается язык для составления программ, т. е. язык, на котором записывается алгоритм для решения задачи на ЭВМ.

Программирование для ЭВМ первого поколения велось исключительно на *машинном языке*. Машинный язык представляет собой свод правил кодирования в числовом виде определенных действий (в большинстве арифметических). Для всех машин понятна только *двоичная* система счисления, которая, однако, для сокращения записи программистами заменялась *восьмеричной*.

Под *системой счисления* обычно понимается совокупность приемов наименования и обозначения чисел. Обычная система записи чисел представляет собой позиционную *десятичную* систему счисления в соответствии с тем, что от позиции, занимаемой любой из десяти используемых в этой системе цифр, зависит ее числовое зна-

чение. Двоичная система счисления является простейшей, так как в ней используются только две цифры: 0 и 1, а восьмеричная система счисления удобна тем, что основание ее, а именно числовое значение 8, является степенью основания двоичной системы счисления 2. Например, десятичное число 65 в десятичной системе счисления можно представить как

$$6 \times 10^1 + 5 \times 10^0 = 65,$$

в восьмеричной системе как

$$1 \times 8^2 + 0 \times 8^1 + 1 \times 8^0 = 101$$

и в двоичной системе счисления в виде

$$1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 100\ 001.$$

Каждое действие, которое должно быть выполнено ЭВМ, на машинном языке задается в виде команды. Команда — это информация, представленная в форме, позволяющей ввести ее в машину, и определяющая действия ЭВМ в течение некоторого отрезка времени. Таким образом, каждая команда определяет, вообще говоря, некоторую элементарную часть процесса обработки информации, называемую *машинной операцией*. Исходная информация для обработки поставляется, как правило, набором конкретных значений, называемых обычно данными. Исходные данные для выполнения любого действия, в том числе и машинной операции, будем называть *операндами*.

В команде в общем случае должны быть указаны вид действия, место хранения в машине (*адрес*) исходной информации, над которой производится машинная операция, адрес результата, а также следующая команда, которая должна быть выполнена после данной. Для арифметических действий (или операций) исходная информация задается, как правило, в виде двух чисел, следовательно, в команде для нее должны быть указаны два адреса. Таким образом, команда должна содержать код операции, задающий вид выполняемой машинной операции, и четыре адреса: два адреса операндов, адрес результата и адрес следующей команды. Как правило, необходимое число адресов в команде меньше четырех.

Широко распространены ЭВМ с трехадресными командами или трехадресные машины. В них не указывается адрес следующей команды, а автоматически выполняется команда из следующей ячейки памяти (с номером, на единицу большим, который и является адресом следующей команды). Например, если принять для операции сложения код 01, то для сложения двух чисел из ячеек с номерами 2051 и 2052 с результатом, помещаемым в ячейку с номером 2345, в трехадресной машине команда будет выглядеть так:

01 2051 2052 2345

В двухадресной машине обычно отсутствует адрес результата, так что результат помещается либо в ячейку одного из операндов, либо на так называемый регистр результата (сумматор). В одноадресных машинах результат всегда получается на сумматоре, так что для выполнения той же операции сложения двух операндов должны быть предусмотрены три команды: записи одного из операндов на сумматор, сложения содержимого сумматора с операндом, находящимся по определенному адресу, и пересылки результата по заданному адресу, чтобы освободить сумматор для следующей операции.

Составление программы на машинном языке носит характер решения сложной комбинаторной задачи, так как одновременно с составлением команд программисту необходимо также распределять память. *Распределение памяти* машины, т. е. размещение в запоминающих устройствах машины всей информации, относящейся к решению задачи (команд и вспомогательных кодов, исходного числового материала и промежуточных данных, окончательных результатов), неотделимо от составления команд. Команды можно составлять лишь после того, как известны номера ячеек, где будет храниться вся необходимая для этих команд информация. С другой стороны, трудно произвести размещение информации в памяти машины, если неизвестно количество команд программы и количество промежуточных результатов, которые должны одновременно находиться в памяти.

Первым усовершенствованием процесса программирования явилось введение символических адресов, позволившее составление команд и распределение памяти выполнять раздельно. Сущность этого приема заключается в разбиении оперативной памяти машины на массивы, число ячеек в которых заранее не известно, а номера ячеек массива задаются буквенно-числовыми обозначениями типа a_i+1 , a_i+2 , ..., называемыми символическими адресами. Распределение памяти осуществляется приписыванием всем a_i числовых значений уже после составления программы. Последний процесс является чисто механическим и может быть автоматизирован, т. е. присвоение истинных адресов может быть поручено самой вычислительной машине.

Такое усовершенствование процесса программирования вскоре привело к созданию языков символического программирования, или *автокодов*. Такой язык отличается от машинного языка лишь тем, что вместо числовых значений, выражающих код операции команды и ее адреса, используются символические (буквенные) обозначения. Поэтому в первых автокодах существовало взаимно однозначное соответствие между операциями, записанными на языке символического программирования (или кодирования), и командами в машинном языке, на что указывал символ 1 : 1, который записывался по-

сле наименования языка. Например, АВТОКОД 1 : 1 — АВТОматическое КОДИрование один к одному.

Дальнейшее совершенствование автокодов выражалось в появлении дополнительных средств, устанавливающих по обычным правилам порядок действий в арифметических формулах или обеспечивающих в необходимых условиях разветвление вычислительного процесса, циклическое повторение участков программы и другие операции, вытекающие из условия задачи. Так, постепенно автокоды утратили приставку 1 : 1, а их входные языки стали не чисто машинными, а *машинно-ориентированными*. Машинная ориентированность означает, что в основе этих языков продолжала лежать система команд какой-либо конкретной вычислительной машины.

Первые машинно-ориентированные языки в целом были несовершенны. У одних языков описание последовательности вычислений было оторвано от самих формул, другие имели сложную символику, мало наглядную или слишком специализированную, третьи были приспособлены лишь для решения ограниченного круга задач. Основным же недостатком заключался в привязанности языка к данной машине.

С появлением машин второго поколения возникла потребность создания языков, целиком ориентированных на особенности задач и не зависящих от конкретной машины. Это требование усугублялось еще и тем, что ЭВМ разных марок быстро сменяли одна другую или использовались совместно. Символом второго поколения ЭВМ стали *проблемно-ориентированные* языки программирования. Их развитие все в большей степени определялось спецификой задач, а не особенностями машин. На первый план выступило то общее, что было в различных задачах, а это сближало разные языки, созданные в эпоху господства вычислительных задач. Эти языки принято называть формальными алгоритмическими или просто *алгоритмическими* языками.

От формального алгоритмического языка требуется многое. Во-первых, он должен быть наглядным, что может быть достигнуто использованием существующей математической символики и других легко понимаемых изобразительных средств. Во-вторых, гибким, чтобы любой алгоритм мог быть описан без излишнего усложнения, связанного с недостаточностью изобразительных средств. В-третьих, от языка требуется однозначность — запись любого алгоритма, выполненная с соблюдением всех правил языка, не должна допускать различных толкований. В-четвертых, многоступенчатость — сложный алгоритм может быть описан в виде сочетания более простых алгоритмов. И наконец, язык должен быть единым — с одной стороны, число изобразительных средств не должно быть слишком большим, и, с другой стороны, необходимо чтобы одни и те же средства можно было применять для выражения одних и тех же или

родственных понятий в разных (по их назначению) частях алгоритма. Такой язык служит: средством мышления — логическое несовершенство предполагаемого метода решения задачи часто выявляется в процессе записи этого метода средствами алгоритмического языка; средством общения между людьми — описание процесса, выполненное одним человеком, должно быть доступно другим; посредником между человеком и машиной — при этом перевод с алгоритмического языка на язык машины выполняется самой машиной с помощью специальной программы — *транслятора*.

Одним из первых и наиболее удачных языков такого рода стал *Фортран*, разработанный фирмой IBM. В 1954 г. группа американских специалистов в области программирования опубликовала первое сообщение о языке. Название языка происходит от словосочетания FORMulae TRANslation — преобразование формул. Язык Фортран не только просуществовал до наших дней, но и уверенно удерживает первое место в мире по распространенности. Среди причин такого долголетия можно отметить простую структуру как самого Фортрана, так и предназначенных для него трансляторов. Программа на Фортране записывается в виде последовательности предложений, или *операторов* (под оператором понимается описание некоторого преобразования информации), и оформляется по определенным правилам. Эти правила накладывают ограничения, в частности, на форму записи и расположения частей оператора в строке бланка для записи операторов. Программа, записанная на Фортране, представляет собой один или несколько сегментов (подпрограмм) из операторов. Сегмент, управляющий работой всей программы в целом, называется *основной программой*.

Фортран был задуман для использования в сфере научных и инженерно-технических вычислений. Однако на этом языке легко описываются задачи с разветвленной логикой (моделирование производственных процессов, решение игровых ситуаций и т. д.), некоторые экономические задачи и особенно задачи редактирования (составление таблиц, сводок, ведомостей и т. д.).

Модификация языка Фортран, появившаяся в 1958 г., получила название Фортран II и содержала понятия подпрограммы и общих переменных для обеспечения связи между сегментами.

К 1962 г. относится появление языка, известного под названием Фортран IV и ставшего наиболее употребительным в настоящее время. К этому же времени относится и начало деятельности комиссии при Американской Ассоциации Стандартов (ASA), которая выработала (к 1966 г.) два стандарта — языки Фортран и базисный (основной) Фортран (Basic FORTRAN). Эти языки приблизительно соответствуют модификациям IV и II, однако базисный Фортран является подмножеством Фортрана, в то время как Фортран II таковым для Фортрана IV не является.

Язык Фортран до сих пор продолжает развиваться и совершенствоваться, оказывая влияние на создание и развитие других языков. Например, Фортран заложен в основу диалогового языка *Бейсик* (BASIC — Beginner's All-purpose Symbolic Instruction Code — многоцелевой язык символических команд для начинающих, Вычислительный центр Дартмутского колледжа, 1966 г.) и его расширения Бейсик-плюс (BASIC-PLUS, фирма Digital Equipment Corporation, 1975 г.), широко распространенных языков во всех системах с режимом разделения времени, превосходных языков для обучения навыкам использования алгоритмических языков в практике программирования. Эти языки реализованы на ряде отечественных машин, в частности на мини- и микроЭВМ Электроника, СМ и др. и персональных компьютерах. В настоящее время создан новый стандарт — Фортран 77 (см. § 9, гл. V), однако работы над усовершенствованием языка ведутся непрерывно.

Вскоре после создания Фортрана (1957 г.) появился язык *Алгол* (ALGOritmic Language — алгоритмический язык), созданный на основе широкого международного сотрудничества. В 1960 г. было опубликовано официальное сообщение об алгоритмическом языке, названном Алгол-60. Число 60 означает год утверждения языка.

Алгол-60 создавался после разработки и практического применения Фортрана, поэтому характеризуется как введением новых конструкций, так и обобщением понятий, имеющих в Фортране. Например, если в Фортране операторы с функциональной точки зрения подразделяются на исполняемые и неисполняемые, то в Алголе такого деления нет, а роль неисполняемых операторов Фортрана выполняют конструкции, называемые описаниями.

Имеются и другие отличия. Однако общим для Фортрана и Алгола является то обстоятельство, что в основе обоих языков лежит понятие *выражения*, практически совпадающее с понятием математического выражения, использующего лишь алгебраические операции и элементарные функции. Простейшие объекты, из которых составляются выражения, — это целые и приближенные вещественные числа и логические значения.

Алгол повсеместно признан как весьма удобное средство для публикации алгоритмов и для обучения основам программирования.

И Фортран, и Алгол-60 до недавнего времени по праву заслуживали название *универсальных* языков, так как обеспечивали программирование основной массы научно-технических задач (преимущественно вычислительных). Но ни один из этих языков, конечно, не позволял описать все без исключения возникающие задачи. Поэтому примерно в то же время появились алгоритмические языки с другой ориентацией, отвечающие нуждам тех новых направлений

науки и техники, которые стали интенсивно развиваться в последующие годы.

Примером могут служить экономические задачи — задачи учета материальных ценностей, выпущенной продукции, личного состава, финансов и т. д. предприятия или отрасли. Для таких задач основными действиями являются операции ввода и вывода при относительно небольшом количестве несложных вычислений, а также последовательная обработка массивов данных. Описание действий такого рода может быть осуществлено на языке *Кобол* (COMmon Business Oriented Language), предложенным фирмой IBM в 1959 г. Этот язык имел и сохраняет до сих пор заслуженный успех у пользователей.

Задачи обработки символьной информации возникают преимущественно в области научных исследований. Это, например, преобразование формул, решение уравнений (не численное, а в аналитическом виде), анализ и синтез текстов на искусственном или естественном языке (в частности, автоматическое программирование и машинный перевод) и т. п.

Из языков для обработки символьной информации очень популярным, главным образом среди представителей физико-математических наук, является язык *Лисп*, созданный группой исследователей в 1960 г. в Массачусеттском технологическом институте. В этом языке вся находящаяся в обработке информация, в том числе и сама программа, организуется в так называемые списки — последовательности элементов. Элемент может быть первичным (буквенным обозначением или числом) или в свою очередь списком. Так могут возникать сколь угодно сложные структуры.

Другой язык — *Снобол* — применяется в основном для машинного анализа текстов, написанных на естественных языках. В нем основным понятием является строка — произвольная последовательность букв, цифр и других знаков. Главная операция — это поиск в строке части строки, построенной по заданному образцу, и замена этой части другой строкой. Как образ, так и заменяющие его строки состояются из отдельных элементов простого вида. Исход поиска определяет последовательность дальнейших действий. Язык Снобол очень прост для изучения.

Основное достоинство проблемно-ориентированных, машинно-независимых алгоритмических языков в том, что они были построены с максимальным учетом представлений человека если не о существе, то о форме решаемой задачи; с максимальным приближением к той форме, в которой человек привык описывать эти задачи, и с учетом тех логических связей, которые он научился выделять в исследуемых явлениях.

Для Алгола, например, характерно приближение к привычной математической символике и лучшее, чем у предшественников, от-

ражение структуры вычислительной задачи. Фортран же, в отличие от Алгола, ближе к машинному языку, чем к языку человека. Для Лиспа характерно использование аппарата так называемых рекурсивных описаний, широко применяемого в математической логике, в исследованиях по основаниям математики и т. п.

Обилие алгоритмических языков, появившихся в период второго поколения ЭВМ, с одной стороны, во многом объясняется модой, с другой стороны — невозможностью ни одним из предложенных языков удобно описывать все возникавшие задачи. Третье поколение ЭВМ поставило на повестку дня выработку нового подхода к созданию действительно универсального алгоритмического языка.

Одной из попыток такого рода является создание фирмой IBM алгоритмического языка *ПЛ/1* (*Programming Language/1* — язык программирования один). Он основан на языках Фортран и Кобол, ряд изобразительных средств и понятий был почерпнут из Алгола и других языков, в частности языков для обработки символьной информации. В первоначальной версии 1964 г. он назывался *New Programming Language* или *NPL* — новый язык программирования. Затем последовательно было опубликовано несколько версий языка, которые сильно отличались друг от друга, но постепенно язык стабилизировался, и теперь новые публикации отличаются от предыдущих лишь редакционными поправками, устранением неточностей или усовершенствованием отдельных элементов.

Основными элементами программы, написанной на языке *ПЛ/1*, являются операторы, с помощью которых описываются как данные, так и операции их обработки. По аналогии с Фортраном исходная программа представляет собой совокупность основной программы и подпрограмм, имеющих форму *блока*. Понятие блока в *ПЛ/1* базируется на концепциях блока в языке Алгол-60. Таким образом, этот язык построен в целом на базе понятий существующих алгоритмических языков и в их традициях.

Другая попытка связана с дальнейшим развитием Алгола. Группа математиков, членов IFIP (Международной федерации по обработке информации) пошла по линии обобщения и углубления основных понятий в области обработки информации и в 1968 г. опубликовала документ с изложением основ нового универсального алгоритмического языка, получившего название Алгол-68. В этом языке число основных понятий сведено к разумному минимуму с целью добиться высокой изобразительной силы языка, обеспечив свободу сочетания и взаимодействия этих понятий между собой.

Однако и Алгол-68 традиционен, поскольку проявляется стремление обеспечить всех пользователей готовыми средствами для описания их алгоритмов. До сих пор этот подход не мог предотвратить появления все новых специализированных языков. Так, в 1971 г.

был опубликован алгоритмический язык *Паскаль*, названный в честь великого французского ученого XVII века, сумевшего первым в мире изобрести автоматическое устройство, позволяющее складывать числа. Язык Паскаль является преемником Алгола-60, он имеет конструкции, аналогичные существующим в ПЛ/1 и Алголе-68, однако Паскаль более лаконичен. Язык Паскаль почти так же прост, как и Бейсик, однако Паскаль способствует внедрению современной технологии программирования, основанной на постепенном построении программы, состоящей из небольших четко определенных процедур, т. е. последовательно проводятся в жизнь идеи *структурного* программирования. Другой существенной особенностью Паскаля является концепция структуры данных как одного из фундаментальных понятий, лежащих, наряду с понятием алгоритма, в основе программирования.

На основе языка Паскаль в конце 70-х годов был создан язык *Ада*, имеющий очень широкую сферу применения. Язык назван так по имени первой женщины-программиста Ады Лавлейс. Алгоритмический язык Ада претерпел определенные изменения в процессе эволюции и теперь имеет все отличительные признаки языка-стандарта. Это существенно структурированный язык, особенно он подходит для разработки систем реального времени. Однако язык Ада слишком громоздкий, многословный и не предоставляет программисту достаточной свободы.

В отличие от перечисленных языков высокого уровня, появившихся в начале 80-х годов, язык программирования *Си* (название содержит одну латинскую букву С) является языком сравнительно низкого уровня. Но это не значит, что этот язык недостаточно мощный. Язык Си — универсальный язык, тесно связанный с популярной операционной системой UNIX (на языке Си написаны и система UNIX и ее программное обеспечение). Алгоритмический язык Си достаточно полно отражает возможности современных компьютеров, позволяя писать весьма эффективные программы, не прибегая к языкам *Ассемблера*, главным образом за счет простых, последовательных конструкций потоков управления. Предлагаются проверки, циклы, группирование и подпрограммы, но не мультипрограммирование, параллельные операции, синхронизация и сопрограммы — непеременимые атрибуты мощных языков (Ада, ПЛ/1, Алгол-68).

В последнее время проявляется тенденция к созданию так называемых расширяемых универсальных языков. Основная идея такого направления — не избегать специализированных языков-диалектов, а создать общую основу «программистских диалектов».

Расширяемый язык должен располагать средствами для грамматического разбора текстов любого из расширений, чтобы выяснить, какие тексты являются грамматически правильными для данного диалекта и какой структурой они обладают. Такой язык дол-

жен дать будущим его потребителям целый ряд важных возможностей: например, строить семантические модели (т. е. вводить новые термины и описывать выражаемые ими понятия, их взаимосвязь с некоторыми основными, исходными понятиями и с понятиями, введенными ранее) и описывать реализацию расширений — способы их наиболее целесообразного представления с помощью средств, которыми располагают современные ЭВМ. Можно отметить и другие характерные черты расширяемых языков.

Работы по созданию и совершенствованию языков ведутся во многих странах, большое количество исследований выполняется в рамках IFIP.

§ 5. Трансляция и входные языки

Современный алгоритмический язык существенно отличается от языка конкретной машины, поэтому алгоритм, записанный как программа на алгоритмическом языке, не может непосредственно быть воспринятым ЭВМ. Для получения программы, понятной ЭВМ, необходимо перевести запись алгоритма с алгоритмического языка на язык машины. Этот перевод осуществляется самой машиной при помощи специальной программы, написанной в кодах данной машины (или на Автокоде для этой же машины), и называемой транслятором.

Алгоритм для решения любой задачи, записанный на алгоритмическом языке (программа), представляет собой некоторый текст в виде отдельных строк. Как правило, для каждого языка готовятся специальные бланки, на которых и записывается на данном языке исходная программа. Затем вся информация с бланков должна быть введена в машину. Это может быть сделано путем нажатия соответствующих клавиш на *терминале*, где набирается строка символов, которая высвечивается на экране дисплея. После нажатия клавиши «конец строки» строка передается в память ЭВМ. Строки могут быть перенесены и на перфокарты или перфоленту с помощью, например, устройства *подготовки перфокарт* (УПП). Существуют различные типы УПП, однако принцип кодировки исходного текста у всех устройств один и тот же — каждому символу текста соответствует вполне определенный код в восьмеричных цифрах, набиваемый на перфокарты комбинацией отверстий. Как правило, каждой строке текста программы соответствует одна перфокарта.

При помощи устройства ввода перфокарты вводятся в ЭВМ. После ввода исходной программы осуществляется ее трансляция, в результате которой создается программа на машинном языке, т. е. программа, состоящая из команд той машины, на которой будет решаться задача. Эта программа называется *рабочей программой*.

Процесс перевода алгоритма (трансляция) и процесс его выполнения машиной (выполнение рабочей программы) могут сочетаться двумя способами. Первый способ, называемый *компиляцией*, заключается в том, что процесс выполнения алгоритма машиной осуществляется после того, как процесс перевода полностью завершен. Для компиляции характерно, что осуществляющая ее программа-транслятор во время выполнения рабочей программы уже не нужна и потому не находится в оперативной памяти ЭВМ. Тем самым достигается более экономное использование ячеек оперативной памяти.

Второй способ сочетания процесса перевода и процесса выполнения алгоритма — *интерпретация* — предполагает, что отдельные операторы (или другие части исходной программы) сразу после трансляции выполняются, после чего та же процедура совершается над другими операторами и т. д. При интерпретации во время выполнения рабочей программы программа-транслятор находится в оперативной памяти ЭВМ, т. е. требуется дополнительная память для нее. Кроме того, процесс решения задачи замедляется, так как между отдельными этапами выполнения рабочей программы управление передается транслятору. В то же время при интерпретации упрощается задача распределения памяти.

Оба эти способа имеют свои достоинства и недостатки. Очевидно, возможно сочетание компиляции с интерпретацией, заключающееся в том, что компилируется программа, в которую включены специальные коды, называемые *макрокомандами*. Макрокоманда объединяет в рабочей программе определенную группу одиночных машинных команд и интерпретируется при выполнении программы. Другой вариант возможен, если основным является процесс интерпретации, однако получаемые при ней участки программ (модули) сохраняются (если позволяет объем оперативной памяти) и включаются в компилируемую программу вместо соответствующих этим участкам групп кодов, получаемых обычно при повторной интерпретации.

Остановимся несколько подробнее на процессе трансляции. При включении ЭВМ сразу же начинается некоторая записанная в ПЗУ программа *начальной загрузки*. Если ЭВМ имеет диски, то программа начальной загрузки считывает так называемый *нулевой блок* с диска в начало памяти и на него будет передано управление. После этого начинает выполняться та программа, которая была записана в нулевом блоке на диске (программа нулевого блока). Эта программа в свою очередь обычно считывает другую программу, называемую *монитором* операционной системы (программой монитора), с фиксированного места на диске и передает на нее управление. Местоположение монитора на диске и число занимаемых им блоков являются константами в программе нулевого блока. При изменении

расположения или размеров монитора надо изменить и программу нулевого блока. Именно поэтому монитор не считается сразу программой начальной загрузки ПЗУ. Отметим, что программа нулевого блока, в отличие от программы ПЗУ, может быть легко изменена путем записи в нулевой блок новой информации.

Монитор операционной системы — это программа, которая взаимодействует через терминал с человеком, анализирует последовательности нажимаемых человеком клавиш и выполняет соответствующие этим последовательностям действия. В состав этой программы входят файловая система, подпрограммы обработки прерываний и другие необходимые для работы подпрограммы, вместе образующие так называемое *ядро* операционной системы.

Основная функция монитора заключается в *загрузке* (чтении с диска) и выполнении программ в кодах, которые хранятся в именованных файлах на диске. Для выполнения, например, какой-либо программы человек, нажимая клавиши на клавиатуре терминала, должен в каком-то виде указать имя этой программы. Действия монитора состоят в анализе последовательности нажатых человеком клавиш, определении имени файла с данной программой, чтении содержимого этого файла в память и выполнении указанной программы (передаче к ней управления).

Исключительные ситуации и отказы при выполнении программы пользователя обрабатываются программами *обработки прерываний* операционной системы. Эти программы прекращают выполнение программы пользователя, выдают соответствующее сообщение человеку и возобновляют работу монитора. Таким образом, как правило, остановка работы ЭВМ не происходит — и при нормальном, и при непредусмотренном завершении программы пользователя возобновляется работа монитора операционной системы, который опять анализирует нажимаемые человеком клавиши, выполняет соответствующие программы и т. д.

Если программа написана непосредственно в кодах ЭВМ, то из монитора вызывается специальная служебная программа — *редактор программ в кодах*. Этот редактор изображает на экране терминала в символьном виде некоторую часть памяти ЭВМ и анализирует действия программиста. Обычно с помощью редактора программ в кодах человек может ввести, изменить программу, выполнить или записать на диске полученную программу в кодах. Записанную на диск программу в дальнейшем можно выполнять с помощью монитора ОС, не пользуясь редактором программ в кодах.

Если же программа составлена на алгоритмическом языке, то перевод (трансляция) программы с этого языка в коды осуществляет автоматически сама ЭВМ. При этом программист через монитор ОС вызывает служебную программу — *редактор текстов*, с помощью которой может ввести, изменить текст программы и записать его

в файл на диск. После этого управление возвращается в монитор, и программист вызывает другую служебную программу — *компилятор*, указав имя файла с текстом исходной программы и имя файла, в котором надо получить соответствующую программу в кодах.

Реальные компиляторы с языков программирования высокого уровня представляют собой довольно сложные программы. Они превращают исходную программу не сразу в коды, а в некоторую промежуточную форму, называемую *объектным кодом*. После компиляции в такой промежуточный объектный код необходим еще один этап, во время которого из объектных кодов отдельных частей программы собирается вся программа в кодах целиком. На этом же этапе, называемом *редактированием связей* или *сборкой*, происходит подключение стандартных подпрограмм, если они нужны, и преобразование оставшихся имен в адреса.

Таким образом, программист с помощью служебной программы — редактора текстов — записывает исходный текст своей программы в какие-то файлы на диске, с помощью другой программы — компилятора — преобразовывает их в объектные коды, с помощью третьей — редактора связей (сборщика) — получает программу в кодах или *загрузочный файл*. После этого загрузочный файл можно выполнить средствами операционной системы.

Основное достоинство компиляции состоит в том, что в конечном счете выполняется программа в кодах и на этом этапе никакая лишняя работа, непосредственно с выполнением программы не связанная, не совершается. Однако при компиляции имеют место как минимум два представления программы (исходного и в кодах), с которыми должен работать программист, и это, безусловно, недостаток компиляции. Если же что-то изменяется в программе, программисту необходимо провести ряд технических действий (компиляция, сборка, запуск), также отнимающих некоторую часть его интеллектуальных и временных ресурсов.

При интерпретации специальная служебная программа — *интерпретатор* — читает текст исходной программы, анализирует его и тут же выполняет. Обычно интерпретатор может читать исходный текст как из файла на диске, так и из памяти ЭВМ, а чаще всего прямо с клавиатуры терминала по мере ввода его программистом. При интерпретации ЭВМ выполняет саму программу-интерпретатор, а уже программа-интерпретатор анализирует и выполняет программу пользователя. Таким образом, требуемые для выполнения свойства программы многократно определяются по ее тексту, в результате чего при интерпретации программа выполняется на порядок и более медленнее, чем будучи скомпилированной в коды. Однако в случае изменения программы при интерпретации ее можно начать немедленно выполнять, и при этом сообщения об ошибках выдаются в терминах исходной программы.

Очевидно, что компиляция и интерпретация — это в каком-то смысле два крайних подхода. Как уже отмечалось выше, существуют компромиссные решения. Довольно широко распространена компиляция исходной программы не в коды реальной ЭВМ, а в коды некоторой промежуточной воображаемой — *виртуальной* — машины с последующим использованием интерпретатора кодов виртуальной машины для выполнения программы. Поскольку программа в кодах специальным образом подобранной виртуальной машины может быть существенно компактнее программы в кодах реальной ЭВМ, такой подход применяется, например, при нехватке оперативной памяти.

Для отечественных универсальных ЭВМ создано много трансляторов с различных алгоритмических языков. Как правило, каждая марка ЭВМ имеет хотя бы один транслятор. В то же время для машин одного и того же класса создано несколько типов трансляторов с одного и того же языка. Например, существует несколько вариантов трансляторов для ЭВМ БЭСМ-6 с различных языков программирования. Обилие трансляторов объясняется тем, что практически не существует ни одного транслятора, переводящего точно и полностью все элементы формального алгоритмического языка. Такой транслятор с полного языка был бы малоэффективен из-за необходимости учитывать все свойства языка, а также маловероятные ситуации. При переводе он был бы очень громоздким и неоправданно сложным. Поэтому во многих трансляторах предусматривается, как правило, перевод только с неполного языка.

Различие трансляторов может быть вызвано еще и тем, что не на всех вычислительных машинах могут быть в точности реализованы все символы и обозначения, составляющие алфавит *эталонного* языка — общепринятой стандартной *версии* языка. Все языки, которые получают после того, как в эталонном языке произведены некоторые замены, упрощения или, наоборот, расширения, называются конкретными представлениями языка (или *входными* языками). Разумеется, каждый входной язык (и каждая версия языка тоже) имеет свои особенности, которые отражаются на соответствующем трансляторе, отличая его от других трансляторов с этого же алгоритмического языка для данной вычислительной машины. Например, в некоторых трансляторах разрешается использовать буквы русского алфавита, хотя эталонный язык допускает только латинские буквы.

В связи с этим уместно отметить, что перед тем, как использовать данную машину и данный транслятор при решении какой-либо задачи, необходимо ознакомиться с описанием транслятора, его характеристиками и входным языком.

Глава II. ФОРТРАН IV

§ 1. Элементы языка

1. Символы. При записи программ используются символы трех категорий: буквы, цифры, и специальные знаки.

К первой категории относятся 26 прописных букв латинского алфавита:

A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

а также знак денежной единицы, обозначаемый обычно \$ (иногда иначе). Во входных языках для отечественных трансляторов, как правило, разрешается также использовать в качестве букв прописные буквы русского алфавита.

Вторая категория символов — это десять арабских цифр:

0 1 2 3 4 5 6 7 8 9

Специальными знаками языка являются символы:

(пробел),	((открывающая скобка),
= (равно),)	(закрывающая скобка),
+ (плюс),	,	(запятая),
— (минус),	.	(точка),
* (звездочка),	'	(апостроф),
/ (наклонная черта),	&	(коммерческое «и»).

Символ «пробел» служит указателем пропуска позиции при записи программы, а потому не имеет начертания. Для явного указания пробела обычно используется символ `_`, который принят и в этой книге. В некоторых источниках этот символ имеет другое графическое изображение, например `b`.

Набор символов каждого конкретного транслятора может отличаться от приведенного. В частности, в записи комментариев и текстовых констант допускается использование всех литер, которые имеют как устройство ввода, так и принтер.

С помощью перечисленных символов образуются величины, выражения и операторы, которые составляют *элементы языка*. При по-

строении элементов языка используются конструкции, представляющие собой неделимые символы, хотя и образованные из нескольких первичных символов (букв, цифр и специальных знаков). Такие конструкции называются *основными символами*. К ним относятся, в частности, *ключевые* (служебные) слова, используемые при записи операторов.

Ключевыми словами являются основные символы: ASSIGN (присвоить), BACKSPACE (назад), BLOCK DATA (блок данных), CALL (вызвать), COMMON (общий), COMPLEX (комплексный), CONTINUE (продолжать), DATA (данные), DEFINE FILE (определение файла), DIMENSION (размерность), DO (выполнить), END (конец), END FILE (конец файла), ENTRY (вход), EQUIVALENCE (эквивалентность), EXTERNAL (внешний), FIND (найти), FORMAT (формат), FUNCTION (функция), GO TO (перейти к), IF (если), IMPLICIT (неявный), INTEGER (целый), LOGICAL (логический), NAMELIST (список), PAUSE (останов), PRINT (печать), PUNCH (перфорация), READ (читать), REAL (действительный), RETURN (вернуться), REWIND (возврат), STOP (стоп), SUBROUTINE (подпрограмма), TO (к), WRITE (писать).

В различных версиях языка список ключевых слов может быть расширен (или сокращен).

Основными символами являются также следующие конструкции:

- логические константы: .TRUE. (истина), .FALSE. (ложь);
- символ операции возведения в степень: **
- знаки операций отношения: .LT. (меньше, чем), .LE. (не больше), .GT. (больше, чем), .GE. (не меньше), .EQ. (равно), .NE. (не равно);
- знаки логических операций: .NOT. (не), .AND. (и), .OR. (или).

К *величинам* относятся константы, переменные и метки (номера операторов). *Константы* могут быть арифметическими (числовыми или комплексными), логическими, шестнадцатеричными и текстовыми, *переменные* — простыми и с индексами. Переменные с индексами объединяются в массивы. Константы и переменные составляют *данные*.

2. Числовые константы. Для изображения числовых констант (чисел) используются цифры, точка, знаки + и —, а также буквы E и D.

Числа различают двух типов: целые и действительные. *Целое* число — это любая конечная последовательность цифр, перед которой может стоять знак + или —. Отсутствие знака или знак + означают, что число положительное, знак — указывает, что число отрицательное. Примеры целых чисел:

15 +306 —00 —88 0 1938

Правильной дробью называется целое число без знака, перед которым стоит точка, например

.1 .3578 .0064 .0 .120338

Правильная дробь относится к числам *действительного* типа.

Числа действительного типа допускают в записи дробную часть и имеют две формы представления: F и E.

Числом действительного типа в форме F называется либо целое число, за которым поставлена точка, либо правильная дробь, либо последовательность, состоящая из целого числа и правильной дроби; перед всей конструкцией может стоять знак + или —. Отсутствие знака означает, что число положительное. Например, действительными числами в форме F являются

10. 3.14159 +1.5789 0.317
.51 —37.000 —.00561 +.317

Таким образом, наличие точки при написании действительного числа в форме F обязательно. Нетрудно видеть, что форма F отличается от обычной математической записи лишь тем, что вместо десятичной запятой пишется точка.

Форма E представления действительного числа — это аналог записи числа с помощью мантиссы, умноженной на степень десяти (*десятичный порядок*). Десятичный порядок образуется из символа E и следующего за ним целого числа со знаком или без него, содержащего не более двух десятичных разрядов.

Числом действительного типа в форме E называется либо десятичный порядок, либо конструкция, состоящая из мантиссы в виде действительного числа в форме F или целого числа и последующего десятичного порядка. Например, действительными числами в форме E являются

E—01 +0.27E+04 27E2
E2 +.27E4 27.E+2
2700E 27000E—1 27E+02

Первые две записи являются порядками и обозначают действительное число 0., все последующие записи обозначают одно и то же действительное число, равное 2700.

Различаются числа двух видов: стандартной и нестандартной длины. Стандартное машинное слово имеет длину 4 байта (напомним, что 1 байт=8 бит), такова же стандартная длина числа (как целого, так и действительного типов). Максимальным значением целого числа стандартной длины является $2^{31}-1=2147483647$.

Целое число нестандартной длины содержит 2 байта, модуль его находится в пределах от 0 до $2^{16}-1$. Например, целыми константами нестандартной длины являются приведенные выше целые

числа, а целыми константами стандартной длины будут

40001 —56789 +2537648 998877

Действительное число в форме F имеет стандартную длину, если оно содержит не более семи цифр, а в остальных случаях — нестандартную длину. В первом случае число занимает в памяти машины четыре байта, во втором — восемь. Максимальное количество цифр в изображении действительного числа в форме F нестандартной длины равно 16. Например, действительные числа в форме F

123456. 0.38745 —357.482

являются числами стандартной длины, а

12345678. +3.14159624 —1234567.89

— нестандартной длины.

Мантисса действительного числа в форме E, являющегося числом стандартной длины, содержит не более семи цифр. Если в изображении действительного числа в форме E символ E заменить на символ D, то образуется действительное число нестандартной длины (8 байт). Мантисса такого числа может содержать до 16 цифр. Примеры:

12345678.87654321D03 —.01D—13 2700D

Поскольку действительные числа нестандартной длины позволяют получать значения повышенной точности, они называются действительными числами двойной точности. Как следует из вышесказанного, число двойной точности записывается либо как действительное число с точкой, содержащее от 8 до 16 цифр, либо как число, содержащее десятичный порядок с символом D вместо E.

Область абсолютных значений чисел действительного типа имеет границы от $\sim 5.4 \times 10^{-79}$ до $\sim 7.2 \times 10^{+76}$.

3. Комплексные константы. *Комплексная* константа записывается в виде пары двух действительных чисел, окруженных скобками и разделенных запятой. Первое действительное число представляет вещественную часть комплексного числа, второе — мнимую. Пара действительных чисел стандартной длины образует комплексную константу стандартной длины (8 байт), а пара действительных чисел двойной точности — комплексную константу нестандартной длины (16 байт).

Например, комплексное число $12-4.3i$ представляется как (12., —4.3) или (12E, —0.43E+1) или (12., —43E—1) или (.12D2, —43D—1) и т. д. Последняя из приведенных записей является записью комплексного числа $12-4.3i$ двойной точности (нестандартной длины).

4. **Логические константы.** *Логические константы* обозначаются символами .TRUE. и .FALSE., которые являются соответственно логической единицей (истина) и логическим нулем (ложь).

5. **Шестнадцатеричные константы.** *Шестнадцатеричная константа* записывается в виде последовательности, образованной из набора шестнадцатеричных цифр, которой предшествует символ Z.

Шестнадцатеричными цифрами являются символы

0 1 2 3 4 5 6 7 8 9 A B C D E F

Максимальное количество цифр, допустимое в шестнадцатеричной константе, зависит от типа и длины переменной, которой присваивается данное значение, т. е. в памяти машины шестнадцатеричная константа представляется как слово длиной в 1, 2, 4, 8 или 16 байт. Один байт памяти содержит две шестнадцатеричные цифры.

Шестнадцатеричные константы могут быть использованы только как величины, присваиваемые переменным в операторе DATA (§ 17) и в операторах явного описания типа (§ 4).

6. **Текстовые константы.** *Текстовая константа* представляет собой последовательность символов (*строку*). Длина такой последовательности, которая рассматривается как значение некоторой переменной и подвергается обработке аналогично числовой константе, соответствует количеству байт, занятых этой переменной, т. е. 2, 4 или 8 байт. Возможно задание текстовой константы в виде строки символов, заключаемой в апострофы (строчные кавычки) или же строкой символов, которой предшествует конструкция wN (w — целая константа, указывающая количество символов в текстовой константе). В последнем случае количество символов в строке не должно превышать 255. Каждый символ (в том числе пробел) занимает 1 байт.

Для того чтобы использовать символ ' при задании текстовой константы, ограниченной апострофами, этот символ следует указать в строке дважды. Например, строки символов

```
'_FORTRAN' PROGRAM_  
17H_FORTRAN' PROGRAM_
```

задают одну и ту же текстовую константу, которая при выводе на печать имеет вид _FORTRAN' PROGRAM_.

7. **Метки.** *Меткой* (номером оператора) является индивидуальное название, присвоенное оператору программистом. Метка образуется как последовательность цифр, которая, если ее рассматривать как целое число без знака, может принимать значения от 1 до 99999. Впереди стоящие нули у меток игнорируются, так что записи 5, 05, 00005 являются одной и той же меткой. В записи меток допускаются пробелы. Например, одна и та же метка 25 может быть записана любым из следующих способов:

25 2_5 0_2_5

Оператор помечается только одной меткой, по которой осуществляется ссылка на него, два оператора не должны иметь одинаковые метки. Порядок снабжения операторов метками произвольный, в частности метка у оператора может отсутствовать. Таким образом, номер оператора (метка) не является порядковым номером оператора.

8. Переменные. Понятие *переменная* употребляется для обозначения величины, обращение к которой производится через ее наименование и которая может принимать различные значения (а не единственное значение как константа).

Для обозначения переменной служит *идентификатор* — последовательность, состоящая не более чем из шести цифр и букв алфавита, причем первым символом должна быть буква. Пробелы в записи идентификаторов не допускаются. Примеры идентификаторов:

A YZ X12 APOLLO CRAY

Различают два вида переменных: простая переменная и переменная с индексами.

Простая переменная представляет собой величину, принимающую числовые, логические или текстовые значения, и обозначается идентификатором. Так, идентификаторы, приведенные выше, обозначают простые переменные.

Одним идентификатором может быть обозначена группа величин, называемая *массивом*. Каждая отдельная величина является элементом массива — переменной с индексами. У одного элемента массива индексов может быть несколько. Максимальное число индексов равно семи, в ряде версий языка оно сокращено до трех. Индексы у переменной с индексами составляют список индексов. Количество индексов в списке определяет размерность массива.

Записывается переменная с индексами при помощи идентификатора, после которого в скобках следует список индексов. Например, компоненты вектора $x(x_1, x_2, x_3)$ составляют одномерный массив: X(1) X(2) X(3), где X — идентификатор массива, а индексы, являющиеся целыми числами 1, 2, 3, можно рассматривать как номера элементов.

Индексы в списке отделяются друг от друга запятыми. Например, элементом двумерного массива будет запись Y(1, 3). Индексом может служить любое арифметическое выражение (§ 2), не содержащее переменных с индексами и называемое в этом случае индексным выражением (в частности, целая константа или целая переменная).

Значение индексного выражения должно быть целочисленным и больше либо равно 1. Если в результате вычислений значение индексного выражения не есть целое, то отбрасывается дробная часть.

В памяти ЭВМ элементы массива располагаются линейно. При этом первым меняется первый индекс. Например, двумерный массив из четырех элементов в памяти располагается по столбцам

M(1, 1) M(2, 1) M(1, 2) M(2, 2)

Переменные делятся на четыре типа: целые, действительные, комплексные и логические. Каждая переменная может принимать только значения, соответствующие ее типу, однако значения текстовых констант могут принимать переменные любого типа.

Каждому типу переменной соответствует две длины — стандартная и нестандартная. Действительные и комплексные переменные нестандартной длины принимают значения констант двойной точности.

Для указания типа и длины переменных служат операторы описания типа, которые будут рассмотрены в § 4. В то же время для указания типа целых и действительных переменных стандартной длины существует способ *автоматического* объявления типа, состоящий в том, что для обозначения переменной целого типа стандартной длины применяются символы I, J, K, L, M, N, употребляемые в качестве первого символа в идентификаторе переменной. Для обозначения переменной действительного типа стандартной длины используются идентификаторы, первый символ в которых есть любая буква, отличная от перечисленных. Например, идентификаторы

J KOR MA L1MAX2 I836 N55

обозначают переменные целого типа, а

E AB BA X701IK COEFF §55

— переменные действительного типа.

Все элементы одного массива имеют одинаковый тип. Первая буква в наименовании массива имеет тот же смысл, что и в случае наименований простых переменных.

9. Функции. *Функция* записывается в виде идентификатора, за которым следует заключенный в скобки список аргументов, разделенных запятыми. Объекты программы, задающие конкретные значения аргументов функции, называются *фактическими* параметрами. Фактическими параметрами могут быть константы, переменные, выражения, идентификаторы массивов, функций и подпрограмм. Конструкция, состоящая из идентификатора функции и списка фактических параметров, называется обращением к функции или *указателем* функции. Выполнение операций над аргументами функции и получение значения функции осуществляется путем выполнения соответствующего алгоритма при обращении к функции. Значением функции является константа, тип которой зависит от типа функции. Примеры обращений к функции:

EXP(Y) X(T) Z(K, L) F(X)

Упражнения

1. Среди приведенных ниже записей чисел определить те, которые являются: а) целыми числами; б) действительными числами; в) числами двойной точности; г) равными по величине из перечисленных в пп. а), б), в); д) комплексными числами; е) записями чисел, не допускаемыми правилами Фортрана:

152.16E3	152.16D3	+2
—0.891	387.0	2.
+347.D020	100	.1.
—89.1E—02	.891E	0.0000891D
235412	—891.D—03	0.387E3.0
0.235412E+6	891E—3	387E3
(3.4, 0.51E+2)	(7.1D5, 10)	(.6, 60.E)

2. Среди приведенных ниже конструкций выбрать последовательности символов, являющиеся идентификаторами. У остальных указать ошибки в записи:

ARRAY	ALPHA	.LOG.
string	5ALPHA	α
END	HATAHA	BURAN-2

3. Объяснить имеющиеся ошибки в написании приведенных ниже переменных с индексами и указать, какие из этих записей являются правильными:

FLIGHT[S]	B(ITL)	ARRAY(1,1,3,4)
X(K(6))	A(—INDEX)	N(N)
Y(2.)	AREA(2,K)	N(.FALSE.)

4. Указать, какие из приведенных ниже конструкций можно рассматривать как текстовые константы. В остальных найти ошибки:

4H18XYZ	7H1234567	0003HXXX
'1.2/MN"—*'	5HHHHHH	"ABC"
07H—10+135	'FALSE.'	!!!!!!

5. Указать, какие из приведенных конструкций нельзя рассматривать в качестве указателей функции и почему:

A137(M, N)	Q(X,—X)	Y(X, Z)
TIP(2A, B, Z)	R((3., 4.))	ART(X, Y)
\$(MONEY)	C(A.GT.B)	AB(A, B)

§ 2. Выражения

Под *выражением* понимается строка символов языка, удовлетворяющая определенным правилам и служащая для вычисления арифметического или логического значений. Выражение образуется из констант, переменных и указателей функций, называемых операндами, и знаков операций. Выражение в свою очередь может быть использовано как операнд, если оно заключено в скобки. В простейшем случае выражение состоит только из переменной или константы.

Операции подразделяются на арифметические, логические и операции отношения. В зависимости от типа значения выражения определяются арифметические и логические выражения.

1. Арифметическое выражение. Арифметическое выражение представляет собой имеющую математический смысл конструкцию из констант, переменных, обращений к функциям, знаков арифметических операций и скобок. Арифметические операции определяются символами

$+$ (сложение), $*$ (умножение),
 $-$ (вычитание), $/$ (деление),
 $**$ (возведение в степень).

При построении выражений два знака арифметических операций не должны стоять рядом. Например, вместо $A*-B$ следует писать $A*(-B)$ или $-B*A$. Примеры арифметических выражений:

$A \quad K+L \quad (X+Y)*Z(5)-(X-Y)/Z(1)$
 3.14 $A**B \quad X**2/Y**2+X*Y$

Последовательность выполнения арифметических операций следующая: сначала — возведения в степень, затем — умножения и деления и в последнюю очередь — сложения и вычитания. Порядок выполнения операций может быть изменен скобками по обычным правилам. Если в качестве операнда в арифметическое выражение входит указатель функции, то значение функции вычисляется в первую очередь.

Операции одного ранга выполняются последовательно слева направо. Исключение составляет операция возведения в степень, которая выполняется справа налево. Так, выражение $L**M**N$ понимается как $L**(M**N)$.

Если операнды, входящие в выражение, одного типа и одной длины, то вычисляемое значение (результат) имеет тот же тип и ту же длину. Если типы операндов смешанные, то всему выражению приписывается более сложный тип из типов операндов: целые значения преобразуются в действительные, а действительные — в комплексные; при этом результат имеет наибольшую длину из длин всех операндов. Так, результат имеет двойную точность, если в выражение входит хотя бы одна величина двойной точности.

Например, результат выражения $I**J$ будет целый стандартной длины, если переменные I, J обе имеют стандартную длину, результат выражения $5/2$ будет целый и равный 2, а выражение $5./2$ — действительного типа со значением 2,5; результат выражения $X+3$ действительный, если X действительного типа, комплексный, если X — комплексная величина, двойной точности, если двойную точность имеет X .

Операция возведения в целую степень выполняется как многократное умножение, а возведение в действительную степень —

через догарифмы. Следовательно, комплексную величину можно возводить в целую степень, но нельзя в действительную и двойной точности. Показатель степени не может быть комплексной величиной. Результат операции $A**R$ не определен также в случае, когда $A < 0$, а R — действительного типа.

2. Логическое выражение. Логическое выражение — это по определенным правилам составленная конструкция из логических констант, логических переменных, отношений, символов логических операций и скобок.

Логические переменные — это переменные, тип которых определяется с помощью соответствующего описательного оператора как логический. Переменные логического типа принимают только два значения: значения логических констант, `.TRUE.` и `.FALSE.`

Операции отношения обозначаются следующими комбинациями букв с окаймляющими с обеих сторон точками:

`.GT.` `.GE.` `.EQ.` `.NE.` `.LE.` `.LT.`

Смысл этих операций может быть выражен математическими символами (соответственно):

$> \geq = \neq \leq <$

Каждый символ операции отношения объединяет два операнда, представляющих собой арифметические выражения произвольного типа и длины, за исключением выражений комплексного типа. В версии Фортран-63 это ограничение снято; при этом из арифметического выражения комплексного типа для сравнения используется только вещественная часть. Результат сравнения, т. е. результат операции отношения, является логическим значением. Например:

`2..GT.B (A+B).LE.AB D(1).LT.1. A*B.NE.B-C`

Логические операции определены для операндов логического типа, обозначаются символами

`.NOT.` `.AND.` `.OR.`

и имеют соответственно следующий смысл: отрицание, логическое умножение (конъюнкция), логическое сложение (дизъюнкция).

С помощью символов `.AND.` и `.OR.` соединяются два операнда, а операция `.NOT.` относится к одному операнду. Например:

`P.AND.Q .NOT.L R.OR..FALSE.`

Здесь `L`, `P`, `Q`, `R` — переменные логического типа.

Результатом логической операции `.NOT.` является логическое значение `.TRUE.` в том случае, когда операнд имеет значение `.FALSE.`, и `.FALSE.`, когда операнд имеет значение `.TRUE.`

Результат операции `.AND.` есть логическое значение `.TRUE.`, если оба операнда имеют значение `.TRUE.`, и `.FALSE.` — во всех других случаях.

И наконец, когда оба операнда в логической операции .OR. имеют значение .FALSE., результатом этой операции будет значение .FALSE. . Во всех остальных случаях результат есть логическое значение .TRUE. .

При построении логического выражения с помощью логических операций два символа логических операций не должны следовать непосредственно друг за другом, за исключением случая, когда вторым является символ операции отрицания, например:

L1.AND.L2.OR.L3.AND..NOT.L1

где L1, L2, L3 являются логическими выражениями.

Таким образом, в логическом выражении могут содержаться знаки арифметических операций, символы операций отношения и логических операций. Порядок действий определяется скобками, а при отсутствии скобок сначала выполняются арифметические операции, затем — операции отношения и в последнюю очередь — логические операции.}

Внутри первого уровня иерархии соблюдается рассмотренное выше правило рангов. Операции отношения являются все одноранговыми. Последовательность выполнения логических операций следующая: отрицание, логическое умножение, логическое сложение. Одноранговые операции выполняются последовательно слева направо. В следующем примере цифрами под символами операций указана очередность выполнения действий:

A.GT.D**K.AND..NOT.L.OR.N
 2 1 4 3 5

Упражнения

1. Составить арифметические выражения, соответствующие математическим формулам:

$$a + \frac{b+c}{a}, \quad (x+y)^{\frac{kx}{ly}}, \quad (a^2+b^2)^{3/2},$$

$$a + \frac{b}{c+a}, \quad \left(\frac{q_1}{q_2}\right)^{\frac{y}{y-1}}, \quad 4\left(\frac{a^2}{b^2}\right)^{1/3},$$

$$x \frac{x^2-y^2}{x^2+y^2}, \quad a^{b^{a+b}}, \quad (2\pi+x)(4\pi-y)^3.$$

2. Определить, соответствуют ли приведенные в правом столбце выражения на Фортране стоящим слева математическим выражениям:

$$a^{b^2}$$

A ** B ** 2

$$ab^2$$

A * B ** 2

$$\frac{ab}{x+y}$$

A * B / (X + Y)

$$\left(\frac{a+b+c+\pi}{2d}\right)^2$$

(A + B + C + 3.1416) / (2 * D) ** 2

3. Определить типы и длины значений следующих арифметических выражений:

$$\begin{aligned} &(A**2+N)*B+3.0*N+L \\ &((K-L**3)*K+1)*A+2-4*K \\ &-(X+Y*R)+((Z+Y)/X+K**L+R)**2 \\ &(C/(H*X-R)+R/H)*G+1D07 \end{aligned}$$

и типы, длины и значения следующих арифметических выражений:

$$\begin{aligned} &X+(A*B-X)*3.1+X/A \\ &(N**2-A1*C+4.1)*A1-6.3*C/A1+B1 \end{aligned}$$

где N, L — целые переменные нестандартной длины; K, I — целые переменные стандартной длины; A, B, A1, B1 — действительные переменные стандартной длины; X, Y, R, Z — действительные переменные нестандартной длины; C, H, G — комплексные переменные стандартной длины.

В двух последних выражениях переменные X, A, B, N, A1, C, B1 имеют соответственно значения: $-3.76D+1$, 0.06, 0.0002E2, -7 , 4.02, (5.1, 0.06), 105.06E—4

4. Для каких целочисленных значений K, заключенных между -6 и $+5$, логическое выражение

$$(K+4.LE.0.OR.K.GT.-4).AND.(K.LE.3.OR.K-4.GT.0)$$

имеет значение .FALSE.? Можно ли это выражение написать без скобок?

5. Являются ли следующие записи конструкциями языка Фортран? Если да, то как они называются?

$$\begin{array}{lll} E(K) & L(1, 2, 3) & X**2+Y**2 \\ XYZ & (A, B)**2 & F(L)*D(I, J, K) \\ Y(Z) & OPERAT & A.AND..NOT.B.OR.C \\ E(X) & TRUE & .TRUE. \end{array}$$

6. Определить значения следующих логических выражений:

$$a) .NOT.A.AND.B.OR.X.GE.Y+1.0$$

где X и Y — переменные действительного типа, значения которых соответственно равны 64.02E02 и 6002.; A и B — переменные логического типа, значения которых соответственно равны .TRUE. и .FALSE.;

$$b) X*Y/(Z-X).NE.X.OR..NOT.(F.AND.A.OR.F).AND..NOT.F.OR.A$$

если значения действительных переменных X, Y, Z соответственно равны -4.6 , 3.7, $-16.2E-1$, а логических переменных F и A — соответственно .TRUE. и .FALSE. .

§ 3. Структура программы

Программа составляется из операторов. *Оператор* — это основной элемент языка, представляющий собой предписание, приводящее к однозначному выполнению определенных действий, Действия

ЭВМ могут заключаться в выполнении операций вычисления или передачи управления или же в закреплении вполне определенных свойств данных и отдельных элементов программы, в размещении массивов или совмещении рабочих полей в памяти. Операторы языка, записанные на бланке программирования (или листе бумаги) по установленным правилам в соответствии с алгоритмом решаемой задачи, в совокупности образуют *программу*. Для выполнения на вычислительной машине программа заносится в память путем набора на клавиатуре в виде строк всех операторов либо при помощи перфокарт, на которых предварительно кодируются операторы.

В зависимости от характера действий различают операторы исполняемые и неисполняемые. *Неисполняемый* оператор является указанием транслятору и не переводится транслятором в команды машины. В соответствии с *исполняемыми* операторами программы транслятор создает в кодах машины рабочую программу, по которой затем производятся заданные действия (например, вычисления).

Операторы могут быть подразделены на пять основных видов: описательные операторы, операторы присваивания, операторы управления, операторы ввода и вывода и подпрограммы.

Подпрограмма представляет собой последовательность операторов и комментариев, которая по своему смысловому значению рассматривается как один оператор. Под *комментарием* понимается текст, используемый для пояснения действий, выполняемых в программе.

Записывается оператор с помощью ключевого слова или заголовка оператора, после которого следует тело оператора:

M_KC_T

где M — метка, KC — ключевое слово, T — тело оператора. Исключение составляют арифметический и логический операторы присваивания и оператор определения оператор-функции, запись которых начинается с идентификатора с последующим символом $=$ (знак присваивания), после чего следует выражение. Тело оператора представляет собой список, элементы которого и разделители между ними в каждом конкретном случае зависят от назначения оператора.

Последовательность операторов и комментариев, не содержащая операторов BLOCK DATA, FUNCTION, SUBROUTINE, называется *основной программой*.

Программной единицей (или сегментом программы) будем называть основную программу или подпрограмму.

Программа всегда состоит из основной программы и может включать одну или несколько подпрограмм.

Выполнение программы начинается с первого исполняемого оператора основной программы. К программной единице можно обра-

щаться как из основной программы, так и из других подпрограмм. Программная единица, из которой осуществляется обращение к другой подпрограмме или к функции, называется *вызывающей программой*.

§ 4. Операторы описания типа

Описательные операторы определяют свойства переменных, массивов и функций и содержат информацию, необходимую для размещения величин в памяти. С помощью описательных операторов можно задавать начальные значения данных. В состав описательных операторов входят операторы описания типа.

1. Неявное объявление типа. Определение типа величины (переменной, массива или функции) с помощью первой буквы ее идентификатора является неявным определением типа и называется автоматическим объявлением типа или объявлением типа по предварительному соглашению. Автоматическое объявление типа допустимо только при задании типа целой или действительной величины стандартной длины.

Для описания всех величин, идентификаторы которых начинаются с заданной буквы или с любой буквы из заданной группы подряд идущих в латинском алфавите букв, служит оператор *неявного* описания типа, имеющий вид

IMPLICIT ТИП *S1(A1), ТИП *S2(A2),...

где ТИП — это любой из основных символов COMPLEX, INTEGER, LOGICAL, REAL; *S1, *S2,... — указатели длины (числа байтов); A1, A2,... — списки, элементами которых являются отдельные буквы или конструкция из двух букв, между которыми стоит символ — (минус, играющий роль тире); разделителем между элементами в списке используется символ , (запятая).

Символ COMPLEX используется для описания комплексного типа, INTEGER — для описания целого типа, LOGICAL — для описания логического типа, REAL — для описания действительного типа.

В качестве указателя длины используются символы *1, *2, *4, *8 или *16. Указатель длины *1 применяется для задания нестандартной длины величин логического типа, указатель длины *2 — для задания нестандартной длины величин целого типа, *4 — стандартной длины величин логического, целого и действительного типов, *8 — нестандартной длины величин действительного типа и стандартной длины величин комплексного типа, *16 — нестандартной длины величин комплексного типа. Если указатель длины опущен, то предполагается задание стандартной длины.

Элемент списка оператора IMPLICIT указывает либо букву, с которой начинается идентификатор величины, либо начальную и конечную буквы последовательности букв, расположенных в алфавитном порядке, каждая из которых может являться первой буквой описываемого идентификатора. Например, два оператора

```
IMPLICIT INTEGER*2(N, P), INTEGER (Q—T)
IMPLICIT REAL (A—C, F—I), REAL*8(D)
```

описывают все величины, идентификаторы которых начинаются с букв N и P, как величины целого типа нестандартной длины (2 байта), величины, имена которых начинаются с букв Q, R, S, T,— как целые величины стандартной длины, величины с идентификаторами, начинающимися с букв A, B, C, F, G, H, I,— как действительные величины и, наконец, величины, у которых первая буква идентификатора есть D,— двойной точности.

Оператор неявного описания типа задает тип каждой величины по тому же принципу, что и при автоматическом объявлении типа, однако в качестве начальных букв идентификаторов величин того или иного типа выбираются буквы по усмотрению программиста. Возможно описание величин всех типов как стандартной, так и нестандартной длины. Оператор IMPLICIT, кроме того, отменяет тип и длину, установленные при автоматическом объявлении типа, т. е. приоритет отдается оператору неявного описания типа.

2. Явное объявление типа. Оператор *явного* описания типа объявляет тип величин не по первой букве идентификатора, а по всему наименованию. Ключевым словом такого оператора является один из символов:

COMPLEX INTEGER LOGICAL REAL

после которого следует список идентификаторов. Элементами списка могут быть идентификаторы, являющиеся именами простых переменных, массивов или функций, а также идентификаторы массивов с указанием верхних границ изменения индексов. Верхние границы изменения индексов записываются в виде целых чисел (через запятую) в скобках сразу после идентификатора массива и представляют собой максимальные значения соответствующих индексов. Количество верхних границ указывает размерность массива.

Элементы списка идентификаторов отделяются один от другого запятыми. Примеры операторов явного описания типа, определяющие величины стандартной длины:

```
INTEGER V, X(6), K, VALUE, Y1, Z, Z1
REAL MIV, J, MAXI, FORINT(10, 10)
LOGICAL MIMA, L1, X1, F(6, 6, 6)
COMPLEX C1, C2, C, M1, P(20, 10)
```

Указатель длины слова, отводимого под определяемые переменные, может быть включен в оператор явного описания типа в виде конструкции *S. В этом случае оператор будет выглядеть так:

ТИП *S A1 *S1(K1), A2 *S2(K2),...

где ТИП — это одно из четырех перечисленных выше ключевых слов; *S — общий указатель длины; A1, A2,... — идентификаторы величин; *S1, *S2,... — указатели длины величин A1, A2, ...; K1, K2, ... — списки верхних границ, если описываемая величина является массивом (списки K1, K2, могут и отсутствовать, даже если A1, A2, ... — идентификаторы массивов).

Если указатель длины для конкретной величины отсутствует, то ее длина описывается общим указателем длины *S. Если же и общий указатель длины отсутствует, то длина всех величин в списке оператора явного описания типа принимается стандартной, и оператор имеет простейший вид. Так, оператор

INTEGER *2 B, C(5, 5), Q

описывает простые переменные B и Q и двумерный массив из 25 элементов как величины целого типа нестандартной длины, а в операторе

REAL *8 A, B, C *4, D, E *4, F *4(2, 2)

переменные A, B, D являются переменными двойной точности, а C, E и элементы массива F имеют стандартную длину.

В операторе явного описания типа одновременно с объявлением типа и длины величины ей может быть присвоено *начальное значение*. В этом случае оператор явного описания типа имеет вид

ТИП *S A1 *S1(K1) /X1/, A2 *S2(K2)/X2/,...

где сохранен смысл ранее использованных обозначений, а X1, X2, ... — списки значений.

Начальные значения могут быть присвоены только переменным и массивам. Начальное значение переменной задается константой, начальное значение массива — списком констант, разделяемых запятыми. В списке значений разрешено использование *коэффициента кратности* (повторителя) в виде конструкции J*, где J — целая константа. Повторитель указывает, сколько раз должно быть повторено следующее за символом J* значение.

Количество констант в списке значений должно быть равно длине массива, а сами константы иметь тип, определенный символом ТИП и указателем длины *S (или *S1, *S2,...). Например, оператор

REAL *8 A, B/3./, F *4(2, 2)/5., 2*3., 6E—01/

приводит к тому, что переменным A и B отводятся поля по 8 байт, при этом A не получает значения, а начальное значение B будет рав-

но 3.0; массив F расположится на четырех полях по 4 байта каждое, и элементы получат начальные значения 5.0, 3.0, 3.0, 0.6 соответственно. Здесь 2* — коэффициент кратности.

В одной программе может содержаться несколько операторов описания типа как неявных, так и явных. Оператор явного описания типа имеет приоритет над автоматическим объявлением типа и над оператором неявного определения типа. Это означает, что оператор явного задания типа отменяет тип и длину, установленные по предварительному соглашению или оператором IMPLICIT. Например, если в программе содержится оператор

INTEGER P2

то он объявляет переменную P2 стандартной целой величиной, несмотря на наличие, допустим, такого оператора:

IMPLICIT INTEGER *2(N, P)

который все идентификаторы, начинающиеся с букв N и P, связывает с целыми величинами нестандартной длины.

3. Оператор задания размеров массивов. Если оператором явного описания типа определяется массив, однако список верхних границ изменения индексов в нем не указывается, т. е. отсутствует указание размеров массива, то соответствующее задание структуры массива должно содержаться либо в операторе COMMON (§ 16), либо в операторе задания *размеров* массивов, записываемого в виде

DIMENSION M1(K1), M2(K2),...

где M1, M2,... — идентификаторы массивов; K1, K2,... — списки верхних границ изменения индексов описываемых массивов.

В одной программе оператор DIMENSION может встретиться любое число раз. Примеры операторов задания размеров массивов;

DIMENSION Y(10, 30), Z(5, 10, 15)

DIMENSION A(20, 20), B(20, 20), X(100)

В первом примере описываются два массива: двумерный Y с 10 строками и 30 столбцами и трехмерный Z (размеры 5×10×15), во втором — два массива A и B одинаковой структуры и одномерный массив X, содержащий 100 элементов.

Использование в программе переменных с индексами без указания размеров массива недопустимо. Размеры массива должны объявляться лишь однажды одним из трех способов: оператором явного описания типа, оператором DIMENSION или оператором COMMON. Например, описание матрицы действительных значений

$$\begin{pmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \end{pmatrix}$$

может быть выполнено оператором

REAL L(2, 2)

или же с помощью двух операторов:

REAL L

DIMENSION L(2, 2)

В качестве верхних границ используются главным образом целые константы. Список верхних границ может состоять из переменных целого типа только в одном случае — когда идентификатор данного элемента из списка массивов и его размеры появляются в списке формальных параметров функции или подпрограммы (§ 13). В этом случае размеры могут быть определены посредством значений фактических параметров, задаваемых при обращении к этой функции или подпрограмме. Максимальный размер, который может иметь каждый данный массив, определяется соответствующим оператором описания массива в основной программе.

Операторы описания типа, как и оператор задания размеров массивов, являются неисполняемыми и в соответствии со структурой программы должны быть расположены в программной единице до первого исполняемого оператора.

Упражнения

1. Первыми операторами программы являются:

REAL MANT, INDEX, LIP(10)

INTEGER STEP, X10

COMPLEX C1, C2

REAL *8 BLOCK, PR1X, D, F

LOGICAL L

Определить тип следующих переменных, встречающихся в этой программе:

A	CONT	D	L	P
ARRAY	C1	F	LIM	PR1X
ALPHA	C	INDEX	LIP(5)	STEP
BLOCK	C2	INTER	M	X10

2. Описать в виде массива матрицу целых чисел:

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \end{pmatrix}.$$

3. В программе содержатся операторы:

IMPLICIT INTEGER *2(C, F), REAL *4(L, P)

REAL *8 FA(4)/-0.2D3, 2* -7.6D3, 0.72D-2/

REAL *8 IP *4(2, 3)/1.25, -3.14E2, 4*.6E3/, PA

Определить типы и длины величин:

IP, I2, FA, FB, LI, PI, L17, CA, F2, PA, P2, X2, Y,

используемых в программе, а также количество элементов в массивах, встречающихся в этой же программе.

§ 5. Операторы присваивания

Операторы *присваивания* обеспечивают переменные значениями. Указанные значения могут быть результатами выражений, т. е. результатами определенных действий над операндами, входящими в выражения, или же значениями меток.

Различают два вида операторов присваивания: первый — арифметический и логический операторы присваивания и второй — оператор присваивания метки.

1. Операторы присваивания первого вида. В общем случае оператор присваивания первого вида записывается следующим образом:

$$V=E$$

где V — переменная (простая или элемент массива), E — выражение, символ $=$ называется знаком присваивания.

При выполнении оператора присваивания вначале вычисляется значение выражения, стоящего справа от знака присваивания, а затем оно присваивается величине, записанной слева от символа $=$.

Оператор присваивания первого вида называется арифметическим оператором присваивания, если E — арифметическое выражение, и логическим оператором присваивания, если E — логическое выражение. Левая и правая части логического оператора присваивания относятся к логическому типу, а арифметического — к одному из арифметических типов (целому, действительному или комплексному).

При выполнении оператора присваивания старое значение переменной V теряется, а операнды выражения E сохраняют свои значения. Например:

```
B=X**2-Y**2
Z(I)=Z(I+10)+1.
L=.FALSE.
N=N-1
```

В последнем примере текущее значение переменной N уменьшается на единицу. Предполагается, что в третьем примере переменная L имеет логический тип.

Значения всех переменных в правой части оператора присваивания должны быть определены.

Если тип и длина правой части арифметического оператора присваивания отличаются соответственно от типа и длины его левой части, то значение выражения, стоящего справа, автоматически преобразуется к типу величины в левой части. При этом соответственно может измениться значение переменной, так как преобразования в операторе $V=E$ выполняются по следующим правилам:

— если V — целого типа, а E — действительного, то значением V будет целая часть E ;

— если V — целого типа, а E — комплексного, то значением V будет целая часть вещественной компоненты E ;

— если V — действительного {типа, а E — комплексного, то в качестве значения V берется вещественная компонента E ;

— если V — комплексного типа, а E — целого или действительного типов, то вещественной компонентой V будет значение E (преобразованное в действительный тип, если E было целым), а мнимой компонентой — нуль.

Например, если A, B, D, E, F имеют действительный тип, I, J, K, M — целый тип, а C — комплексный тип и $A=2.0, B=3.0, J=9, K=-2$, то операторы

$$I=B/A \quad D=J/K \quad M=(2.0, 3.0)$$
$$F=B/A \quad C=K+J \quad E=(2.0, 3.0)$$

дают соответственно результаты: 1, 1.5, —4., (7., 0.), 2, 2.

2. Оператор присваивания метки. Оператором присваивания второго вида или оператором присваивания метки называется конструкция

ASSIGN n TO M

где n — метка оператора, M — простая переменная целого типа. При выполнении этого оператора переменная M принимает значение метки n . Например, после выполнения оператора

ASSIGN 27 TO K

переменная K получает значение метки, равное 27, и только после этого она может быть использована в программе, а именно, в присваиваемом операторе перехода (§ 7).

Изменение значения переменной, указанной в операторе **ASSIGN**, осуществляется новым оператором присваивания метки.

Переменная, получившая значение метки, не может быть использована как переменная, имеющая числовое значение, до тех пор, пока она его не получит, например, в арифметическом операторе присваивания.

Упражнения

1. Описать в виде операторов присваивания вычислительные операции для нахождения действительных корней X_1 и X_2 квадратного уравнения $AX^2+BX+C=0$, используя для вычисления квадратного корня прием возведения в степень $1/2$.

2. Указать, какие из приведенных ниже операторов присваивания не содержат ошибок:

$$K=C+(A*B+C)-4.76$$
$$Z=X-(T**Z-Z)/X+T-R.GT.,TRUE.$$
$$L=A.LE.B.OR.L$$
$$ASSIGN\ 070\ TO\ F$$
$$ASSIGN\ M\ TO\ L$$

3. Составить операторы присваивания переменной Z значений следующих математических выражений:

$$\begin{array}{ll} x=y, & \frac{x}{x+y} - 2z \neq 2x \wedge x > y, \\ x+y \leq 3.15, & \neg(x \vee y) \wedge x \vee \neg y, \\ x^{y^z}, & x=y \vee z \neq y \wedge x+y > 2z, \\ \frac{x+y}{x^2+y^2} - \frac{z}{x+y}, & \neg(a \vee \neg b) \vee a \wedge b. \end{array}$$

Каков может быть тип переменной Z в каждом из получаемых операторов присваивания, а также тип остальных переменных в правой части?

§ 6. Правила записи операторов

Операторы в программе выполняются последовательно в порядке их записи. Этот порядок может быть изменен только операторами управления. Операторы записываются в виде строк. Для одного оператора отводится, как правило, одна строка.

Длина оператора зависит от сложности входящих в него операндов, в частности от длины арифметического или логического выражений. Она ограничивается форматом бланка, на котором пишутся операторы. Бланком может служить обычный лист бумаги. Однако пользоваться заранее размеченным бланком — значит облегчить как написание и чтение программы, так и ее ввод в машину.

Одна строка бланка соответствует одной 80-колоночной перфокарте, при этом каждая из 80 позиций строки соответствует одной колонке перфокарты. Возможный вариант бланка для записи операторов приведен на рис. 2.

Шапка бланка содержит необходимые минимальные сведения о программе. Сюда же внесены и позиции 73—80, служащие для записи шифра (наименования) программы. Шифр является общим для всех строк.

Позиции 7—72 используются для записи оператора. Каждый символ (в том числе пробел) занимает на бланке одну позицию. В записи операторов (но не чисел и идентификаторов) допускаются пробелы. Например, оператор присваивания

$$X = A**2 - B**2$$

занимает на бланке (рис. 2) 11 позиций, начиная с седьмой (подряд или с произвольным числом пробелов).

Если оператор не помещается в одну строку, то разрешается часть оператора перенести на следующую строку. Перенос можно делать в любом месте строки. Признаком того, что последующая строка является продолжением предыдущей, служит любой отличный от пробела и нуля символ в шестой, помеченной на рис. 2 буквой П,

Задача	Дата.	Страница.	Лист
Автомат			80
Номер			
1	58	7	10
2	15	20	25
3	30	35	40
4	45	50	55
5	60	65	70
6	75	80	85
7	90	95	100
8	105	110	115
9	120	125	130
10	135	140	145
11	150	155	160
12	165	170	175
13	180	185	190
14	195	200	205
15	210	215	220
16	225	230	235
17	240	245	250
18	255	260	265
19	270	275	280
20	285	290	295
21	300	305	310
22	315	320	325
23	330	335	340
24	345	350	355
25	360	365	370
26	375	380	385
27	390	395	400
28	405	410	415
29	420	425	430
30	435	440	445
31	450	455	460
32	465	470	475
33	480	485	490
34	495	500	505
35	510	515	520
36	525	530	535
37	540	545	550
38	555	560	565
39	570	575	580
40	585	590	595
41	600	605	610
42	615	620	625
43	630	635	640
44	645	650	655
45	660	665	670
46	675	680	685
47	690	695	700
48	705	710	715
49	720	725	730
50	735	740	745
51	750	755	760
52	765	770	775
53	780	785	790
54	795	800	805
55	810	815	820
56	825	830	835
57	840	845	850
58	855	860	865
59	870	875	880
60	885	890	895
61	900	905	910
62	915	920	925
63	930	935	940
64	945	950	955
65	960	965	970
66	975	980	985
67	990	995	1000

Рис. 2.

позиции строки. В строке продолжения в позициях с первой по пятую должны содержаться пробелы. Например, можно писать оператор в виде, указанном на рис. 2 в третьей и четвертой заполненных строках. Символом продолжения здесь использована буква R.

Максимальное количество строк, которое можно занимать под один оператор, равно 20.

Первая позиция бланка на рис. 2 помечена буквой C. Если в какой-либо строке в первой позиции написать символ C, то строка не воспринимается транслятором как оператор. Она обычным образом вводится в машину и печатается при выводе программы. Такая строка является пояснительным текстом и называется *комментарием*. В тексте комментария, начинающегося непосредственно за буквой C (со второй позиции строки), разрешается использовать все символы, которые имеются на устройствах ввода и вывода конкретной машины.

Позиции 2—5 используются под метки операторов, для этой цели разрешается использовать и первую позицию. В начальной строке оператора шестая позиция должна быть либо пустой, либо содержать цифру 0.

Во многих входных языках, если операторы короткие, разрешается в одну строку писать несколько операторов. В этом случае между операторами ставится определенный разделитель, обычно не являющийся символом Фортрана и зависящий от транслятора. Так, запись строки, приведенной на рис. 2 последней, допустима для Фортрана ЭВМ БЭСМ-6; здесь разделителем служит символ \diamond (ромбик). В такой строке помеченным может быть только первый оператор. Следовательно, в приведенном примере метка 26 относится к оператору $C=A*B$.

§ 7. Операторы перехода

Порядок выполнения операторов в программе определяют операторы *управления*, частным случаем которых являются операторы *перехода*. Выполнение программы начинается всегда с исполняемого оператора, встретившегося в записи программы первым, а затем последовательно выполняются все остальные операторы, пока некоторый оператор управления, например оператор перехода, не нарушит эту последовательность.

Различают три вида операторов перехода: оператор безусловного перехода; вычисляемый оператор GO TO, или оператор перехода по вычислению; присваиваемый оператор GO TO, или оператор перехода по предписанию (по назначению).

1. Оператор безусловного перехода, Записывается в следующем виде:

GO TO N

где N — метка исполняемого оператора. Этот оператор обеспечивает передачу управления на оператор с меткой N . Например, наличие в программе оператора

GO TO 347

означает, что следующим должен выполняться оператор, имеющий метку 347. После этого оператора выполняется оператор, стоящий за ним (а не за оператором перехода) и так последовательно до тех пор, пока не встретится другой оператор управления.

2. Вычисляемый оператор перехода. Этот оператор записывается в следующей форме:

GO TO (N), M

где N — список меток, содержащий любое количество меток исполняемых операторов, а M является простой целой переменной. Номера операторов в списке меток разделяются запятыми.

Вычисляемый оператор GO TO производит передачу управления оператору с меткой, имеющей в списке меток порядковый номер, равный текущему значению M . Значение M определяется оператором присваивания или ввода до выполнения оператора GO TO (т. е. вычисляется, отсюда и название — вычисляемый оператор). Например, если в программе содержится оператор

GO TO (7, 9, 1020, 346), L

то управление будет передано оператору с меткой 7, если $L=1$, оператору с меткой 9, если $L=2$, и т. д.

В списке меток номера операторов могут повторяться. Если к моменту выполнения вычисляемого оператора перехода переменная M принимает значение, превосходящее количество элементов в списке меток, то данный оператор передает управление непосредственно стоящему за ним оператору.

3. Присваиваемый оператор перехода. Этот оператор имеет вид

GO TO M , (N)

где N — список меток, состоящий из произвольного количества элементов, а M — целая переменная без индексов. Номера операторов в списке меток могут повторяться.

К моменту выполнения оператора GO TO значение M должно быть определено предшествующим оператором ASSIGN и равняться одному из номеров операторов, включенных в список меток.

Между операторами ASSIGN и присваиваемым GO TO может стоять любое количество операторов.

После выполнения оператора ASSIGN любой последующий присваиваемый оператор перехода, использующий переменную M , будет передавать управление на исполняемый оператор с меткой,

значение которой присвоено М. Например, из двух операторов

ASSIGN 15 TO K

GO TO K, (4, 13, 16, 15, 100)

первый присваивает переменной К значение метки 15, а второй передает управление оператору с этой меткой.

Вычисляемый оператор перехода удобно использовать в том случае, когда из нескольких формул необходимо выбрать одну определенную формулу для расчета. Оператор перехода по предписанию используется главным образом в случае необходимости организации обращения к одному и тому же фрагменту из разных меет программы. В этом случае фрагмент записывают только один раз, а передачи управления на него организуют с помощью безусловных операторов перехода. Места возврата устанавливаются заранее с помощью операторов присваивания метки. Повторяющийся фрагмент должен оканчиваться присваиваемым оператором GO TO.

Пр и м е р. Вычислить значения выражения $y = x^2 + P_l(x)$, где $P_l(x)$ — полином Лежандра; $l = 0, 1, 2, 3, 4$;

$$P_0(x) = 1, P_1(x) = x, P_2(x) = (3x^2 - 1)/2,$$

$$P_3(x) = (5x^3 - 3x)/2, P_4(x) = (35x^4 - 30x^2 + 3)/8.$$

Фрагмент программы вычисления значения y может быть оформлен так:

```
6 M=L+1
  GO TO (8, 10, 12, 14, 16), M
8 P=1.
  GO TO 17
10 P=X
  GO TO 17
12 P=(3*X**2-1.)/2
  GO TO 17
14 P=(5.*X**3-3.*X)/2
  GO TO 17
16 P=(35.*X**4-30*X**2+3.)/8
17 Y=X**2+P
```

Здесь предполагается, что значения L и X определяются предшествующим данному участком программы, а вывод на печать значений Y и четырехкратный переход на оператор с меткой 6 — последующим участком.

Практически вычисляемый и присваиваемый варианты оператора GO TO взаимозаменяемы.

Расположение меток в присваиваемом операторе GO TO не имеет значения, в то время как в вычисляемом операторе перехода оно является решающим.

Упражнения

1. Составить последовательность операторов для вычисления значения y , если

$$y = \begin{cases} ax^2 + bx + c & \text{при } k=1, \\ dx^3 + ex + f & \text{при } k=2, \\ gx^2 + hx + i & \text{при } k=3. \end{cases}$$

2. Указать метку оператора, которому будет передано управление в результате выполнения операторов:

60 ASSIGN 5 TO M

GO TO 6

61 ASSIGN 64 TO M

6 ASSIGN 50 TO M

ASSIGN 63 TO M

GO TO M, (5, 37, 50, 63, 64, 77)

3. Написать операторы, осуществляющие вычисление площади одной из геометрических фигур: квадрата, прямоугольника, прямоугольного треугольника и круга, если сторона квадрата равна A , стороны прямоугольника или катеты треугольника — B и C , радиус круга равен R , а выбор той или иной фигуры осуществляется с помощью переменной K в присваиваемом операторе GO TO.

§ 8. Условные операторы управления

Условными принято называть операторы управления, ключевым словом которых является символ IF. Различают два вида условных операторов: арифметический (арифметический оператор IF) и логический (логический оператор IF).

1. Арифметический условный оператор. Арифметический оператор IF используется для изменения последовательности выполнения операторов в программе после проверки заданного условия, содержащегося в записи оператора. Оператор имеет вид

IF (E) N1, N2, N3

где E — арифметическое выражение целого или действительного типа; $N1, N2, N3$ — метки операторов.

Действие этого оператора заключается в вычислении значения арифметического выражения E и последующей передаче управления на один из операторов, помеченных метками $N1, N2, N3$. Если вычисленное значение выражения $E > 0$, то осуществляется переход к оператору, помеченному меткой $N1$; если $E = 0$, то управление будет передано оператору с меткой $N2$; если же $E < 0$, то осуществится переход к оператору, имеющему метку $N3$.

В языке Фортран-63 в арифметическом операторе IF допускаются выражения комплексного типа. В этом случае переход осуществляется аналогично в зависимости от значения вещественной части выражения E .

Пр и м е р. Пусть по ходу выполнения программы требуется вычислить y как функцию x , если значение x задано, по формулам

$$y = \begin{cases} 0.6x + 0.8 & \text{при } x \leq 3.9, \\ 0.7x + 1.0 & \text{при } x > 3.9. \end{cases}$$

Фрагмент программы, соответствующий вычислению значения y , может быть оформлен так (здесь и далее текст не на языке Фортран, служащий для пояснения примеров, будет заключен в угловые скобки $\langle \rangle$):

```

      IF(X-3.9) 40, 40, 30
40    Y=.6*X+.8
      GO TO 45
30    Y=.7*X+1.
45    ⟨продолжение программы⟩

```

Метки в арифметическом условном операторе могут совпадать. Если оператор IF имеет метку, то она не должна совпадать с N1, N2 или N3.

С помощью оператора IF возможно неоднократное повторение одного оператора или целой группы операторов. Например, фрагмент программы для вычисления факториала числа N , т. е. произведения $1 \cdot 2 \cdot \dots \cdot N$, можно оформить, используя арифметический оператор IF:

```

      NF=1
      K=1
1     NF=NF*K
      K=K+1
      IF (N-K) 2, 1, 1
2     ⟨продолжение программы⟩

```

Здесь результат обозначен идентификатором NF. Группа из трех операторов, первый из которых имеет номер 1, повторяется N раз.

2. Логический условный оператор. Логический оператор IF имеет вид

IF (L) S

где L — логическое выражение, S — любой исполняемый оператор, отличный от оператора цикла и логического условного оператора.

Выполнение логического условного оператора заключается в вычислении значения выражения L и в передаче управления либо оператору S, если значение выражения L есть .TRUE., либо оператору, непосредственно следующему за данным оператором, если значение выражения L есть .FALSE.. Например:

```

      IF (A*B.LT.4.) GO TO 11
      IF(.NOT.(X.LT.26.5)) R=X*Y-12.4

```

Пример 1. Два варианта записи

10. $Z = X/Y$ $Z = X/Y$

Пример 2. Программа для вычисления корней квадратного трехчлена ax^2+bx+c в соответствии с блок-схемой, приведенной в § 3 гл. 1, может быть записана в следующем виде:

Использованные в этом примере операторы STOP и END будут рассмотрены в § 10, операторы READ и PRINT — в § 20, а оператор FORMAT — в § 22.

Упражнения

1. Составить последовательность операторов для вычисления значений компонент векторов $a(a_1, a_2, \dots, a_m)$ и $b(b_1, b_2, \dots, b_n)$ по формулам

$$a_i = \begin{cases} x_i, & \text{если } x_i \geq 0, \\ -1, & \text{если } x_i < 0, \end{cases} \quad b_j = \begin{cases} x_j, & \text{если } x_j < 0, \\ 1, & \text{если } x_j \geq 0, \end{cases}$$

где $x(x_1, x_2, \dots, x_k)$ — вектор, причем $m \leq k$, $n \leq k$.

2. Составить последовательность операторов для вычисления значений элементов матрицы $X(x_{ij})$, где $x_{ij} = y_{ij}$, если $i=j$, и $x_{ij}=0$, если $i \neq j$, а матрица $Y(y_{ij})$ задана ($i=1, 2, \dots, n$; $j=1, 2, \dots, n$).

3. Определить, какой из операторов получит управление после выполнения следующих условных операторов:

а) IF(A+X*B.GT.0.) IF(A+2.) 3, 4, 5

если $A=-2.0$, $B=4.25$, $X=1.5$;

б) IF(X+Y) 33, 2, 7

в) IF(A.OR.B.AND.C) IF(X-Y) 37, 21, 72

если $A=.TRUE.$, $B=C=.FALSE.$, $X=1.6$, $Y=1.9$.

4. Написать программу, которая присваивает каждому элементу массива M значение, равное порядковому номеру элемента в массиве. Так, значение $M(1)$ есть 1, $M(2)$ есть 2 и т. д. Массив содержит 280 элементов.

5. Написать операторы, осуществляющие проверку принадлежности значения X заданному отрезку $[A, B]$. В случае положительного ответа вычислить $Y = AX^2 + BX + C$, в противном случае присвоить Y значение, равное X^2 .

6. Вычислить F_x и F_y по формулам $F_x = 2xy - y^2$, $F_y = x^2 - 2xy$, если выполнено условие $x > y$. При $x \leq y$ вычислить F_x по второй формуле, а F_y — по первой.

7. Вычислить последовательные значения F как функции переменной x по формуле $F = F_0 + F_1 \frac{x}{11} + F_2 \frac{x^3}{31} + F_3 \frac{x^5}{51}$, если x изменяется с шагом 0.01 от значения $x=0$ до $x=5$, а $F_0=1$, $F_1=0.5$, $F_2=0.3$, $F_3=0.1$.

§ 9. Оператор цикла

При программировании циклических вычислительных алгоритмов могут быть использованы условные операторы. Пример вычисления значения $N!$, приведенный в предыдущем параграфе, является иллюстрацией такой возможности.

Для этих же целей служит специальный оператор *цикла* (или оператор DO), являющийся наиболее сложным и мощным из числа операторов управления. Он записывается в виде

DO M I=N1, N2, N

где M — метка, I — простая переменная целого типа — параметр или управляющая переменная цикла, $N1$ — нижняя граница пара-

метра цикла, $N2$ — верхняя граница, N — величина шага, с которым параметр I пробегает значения от нижней границы до верхней. $N1$, $N2$, N — целые константы или целые переменные без индексов (возможно, со знаками).

Конструкция $I=N1, N2, N$ называется заголовком цикла. В заголовке цикла величину N можно не указывать, в этом случае автоматически величина шага принимается равной 1.

Операторы, следующие за оператором DO , включая тот, который помечен указанной меткой M , образуют область цикла или область действия оператора DO . Последний оператор области цикла называется концевым оператором цикла или концом цикла. Соответственно начало цикла — это первый оператор после DO .

Оператор цикла вызывает многократное исполнение операторов, составляющих область цикла. При этом первоначально параметр цикла принимает заданное начальное значение $N1$ и выполняются операторы области цикла. Затем к значению параметра цикла прибавляется шаг N , и если новое значение параметра цикла не превосходит верхнюю границу $N2$, то снова выполняются операторы области цикла, и т. д. При значении I , равном $N2$, оператор цикла выполняется. Если же $N2$ не кратно N , то последнее исполнение цикла производится при ближайшем, но не превосходящем $N2$ значении параметра.

Если $N > 0$, то должно быть $N1 < N2$, если же $N < 0$, то, естественно, $N1 > N2$.

Пример 1. Вычисление $N!$ может быть выполнено в следующем цикле:

```
NF=1
DO 2 K=1,N
2 NF=NF*K
```

Пример 2. Фрагмент программы вычисления суммы квадратов 40 чисел выглядит так:

```
DIMENSION X(40)
<ввод(X)>
S=0
DO 7 I=1, 40
7 S=S+X(I)**2
```

В этом примере (как и в примере 1) цикл состоит только из одного оператора, т. е. совпадают начало и конец цикла. Тем не менее в заголовке цикла метку конца цикла необходимо указать.

Параметр цикла может использоваться внутри цикла не только как управляющая переменная. В частности, параметр цикла может выступать как индекс массива. Вообще говоря, параметр цикла может быть использован в любом операторе внутри цикла, однако он

не должен подвергаться изменению, помимо естественного изменения, определяемого заголовком цикла.

Внутри цикла не допускается также изменение значений N1, N2, N, если эти величины заданы не константами, а переменными.

Пример 3. В цикле

DO 8 I=200, 400, 5

X=I

X=X/100

.....

8 <конец цикла>

значениям I от 200 до 400 с шагом 5 ставятся в соответствие значения X от 2.0 до 4.0 с шагом 0.05. Тем самым организуется цикл, в котором шаг не является целой величиной.

После последнего исполнения цикла управление передается оператору, следующему за оператором с меткой M. Выход из цикла в этом случае называется нормальным, а значение параметра I не определено.

Выход из цикла при значении параметра, не достигшем верхней границы, называется специальным. Специальный выход из цикла обеспечивается операторами GO TO или IF, передающими управление на оператор, не принадлежащий данному циклу. Текущее значение параметра цикла здесь сохраняется и может быть использовано в последующих операторах.

Первый оператор в области цикла должен быть исполняемым. Последним оператором цикла не может быть оператор GO TO, арифметический оператор IF, другой оператор DO, операторы STOP (§ 10) и RETURN (§ 13).

Концом цикла может быть логический оператор IF(L) S, в состав которого не входят операторы, которым запрещено быть последними в операторе цикла.

Если возникает такая ситуация, что цикл заканчивается одним из запрещенных операторов, в него необходимо добавить оператор *продолжения*

CONTINUE

Это неисполняемый оператор, он отмечает конец цикла и тем самым обязан иметь метку, указанную в операторе DO. Нахождение оператора CONTINUE в любом другом месте программы не оказывает влияния на последовательность выполнения операторов, т. е. этот оператор пропускается.

Пример 4. Найти в массиве IX первый элемент, значение которого равно 2. Программа может быть оформлена так:

DIMENSION IX(100)

<код(IX)>

```

DO 5 I=1, 100
  IF (IX(I)—2) 5, 10, 5
5 CONTINUE
10 <продолжение программы>

```

В этом примере оператор CONTINUE, во-первых, обеспечивает проверку следующего элемента массива, если данный оказался не равным 2, во-вторых, обеспечивает выход из цикла, если $I=100$, а ни один из элементов не равен 2. Если найдется элемент, равный 2, то осуществится специальный выход из цикла, и значение I будет соответствовать порядковому номеру элемента массива.

Разрешается использование циклов в цикле. В этом случае область внутреннего DO не должна выходить за область внешнего DO. Допустим общий конец обоих операторов.

Пример 5. Переписать таблицу T размером 10×1000 в таблицу S можно по следующей программе:

```

DIMENSION T(10, 1000), S(10, 1000)
<ввод(T)>
DO 1 I=1, 10
  DO 1 J=1, 1000
1 S(I, J)=T(I, J)

```

Комбинацию двух циклов, удовлетворяющую правилу вхождения циклов, называют *вложенными* циклами. Понятие вложенных циклов обобщается на случай трех и более циклов; при этом количество вложений в общем случае не ограничивается.

Два вложенных цикла должны иметь в качестве параметров различные переменные. Циклы же, выполняемые последовательно, свободны от этого ограничения.

Запрещено передавать управление (операторами GO TO или IF) извне в область действия оператора цикла. Можно передавать управление извне только на оператор DO.

В случае вложенных циклов разрешается передавать управление из области действия внутреннего цикла в область действия внешнего цикла, но запрещается обратное. Разрешается также в ходе выполнения оператора DO передавать управление внешнему по отношению к данному циклу участку программы (например, к подпрограмме) с последующим возвратом в область действия того же оператора цикла.

Упражнения

1. Написать программу транспонирования квадратной матрицы размера 12×12 .
2. Составить программу, подсчитывающую число отрицательных, нулевых и положительных элементов массива $T(38)$.
3. В двумерном массиве поменять местами строки с номерами I и K .

4. Вычислить одномерный массив Y значений функции $y(x) = x^3/(5x^2+3)^2$ для аргумента X , изменяющегося следующим образом: от 0 до 1 с шагом 0.1, от 1 до 10 с шагом 1, от 10 до 50 с шагом 5, а от 50 до 100 с шагом 10. Вместе со значением $Y(1)$ фиксировать номер 1.
5. Выполнить упражнение 1 из § 8, используя оператор цикла.
6. Составить последовательность операторов для упражнения 2 из § 8, используя оператор цикла.

§ 10. Операторы останова и окончания

1. Оператор полного останова. Этот оператор имеет вид
STOP N

где N — целая константа без знака, содержащая не более пяти цифр. Константа в записи оператора может и отсутствовать.

Оператор *полного останова* прекращает выполнение программы и печатает символ STOP и константу. Если в записи оператора отсутствует константа N , то ничего не печатается.

По оператору STOP задача снимается со счета и возобновить работу программы можно только с самого ее начала. Управление передается операционной системе для перехода к следующей программе.

2. Оператор условного останова. Форма записи этого оператора
PAUSE N

где константа N имеет тот же вид и смысл, что и в операторе STOP. Константа может отсутствовать или заменяться комментарием в виде текста, заключенного в апострофы:

PAUSE 'ТЕКСТ'

Оператор *условного останова* аналогичен оператору STOP. Отличие лишь в том, что после условного останова печатается либо символ PAUSE с константой или комментарием (без кавычек), либо просто символ PAUSE, если в записи оператора отсутствуют и константа и комментарий, и программа переходит в состояние ожидания. Продолжить работу программы, начиная с оператора, следующего за оператором PAUSE, можно вмешательством с пульта машины или с клавиатуры терминала. Этот оператор служит для контроля работы программы при отладке.

Операторы STOP и PAUSE останавливают только исполнение рабочей программы, но не влияют на работу транслятора.

3. Оператор окончания. Вид оператора

END

Оператор *окончания* отмечает конец программной единицы. Роль END заключается в том, что он указывает транслятору конец

текста данной программной единицы. Это неисполняемый оператор. Оператор END должен быть последним оператором в исходной программе.

§ 11. Библиотечные подпрограммы

Под *стандартными функциями*, или библиотечными подпрограммами, подразумеваются программы вычисления элементарных функций (например, \sin , \cos и др.), а также программы для некоторых часто встречающихся алгоритмов. Стандартные функции подразделяются на два вида: встроенные и внешние, различающиеся реализацией процесса вычисления при обращении к ним. Встроенные функции не могут служить фактическими параметрами, указываемыми в операторе внешних подпрограмм.

Записывается стандартная функция в виде указателя функции, т. е. с помощью идентификатора (с соблюдением обычных правил формирования идентификатора) и стоящего за ним аргумента (аргументов) в скобках. Стандартные функции используются в выражениях в качестве операндов как обычные переменные. Не разрешается только употребление идентификаторов стандартных функций в левой части оператора присваивания.

Аргументом стандартной функции может быть произвольное арифметическое выражение. Это выражение, в частности, может в свою очередь содержать библиотечную подпрограмму. Например:

$$Y = \cos(X + Z)$$

Такой оператор находит значения переменных X и Z , вычисляет их сумму, затем извлекает из библиотеки подпрограмму с наименованием \cos (вычисление косинуса), вычисляет значение косинуса соответствующего аргумента и присваивает полученное значение переменной Y . После этого выполняется очередной оператор программы.

Для стандартных функций тип результата устанавливается заранее. В случае целых или действительных функций сохранено правило первой буквы в наименовании при определении типа. Для результата комплексного типа и двойной точности в качестве первой буквы в наименовании стандартных функций обычно употребляются буквы C и D соответственно.

Количество стандартных функций для различных версий языка, как правило, различное. Так, в первоначальной версии Фортран IV их число равнялось 66, а в базисном Фортране — только 45. В настоящее время число библиотечных подпрограмм (например, для ДЭС ЭВМ) доведено до 88. Список идентификаторов и назначение стандартных функций в каждом конкретном случае следует уточнять по описанию используемого входного языка.

§ 12. Оператор-функция

В каждой программе могут встречаться выражения, зависящие от параметров и используемые многократно. Обращение к значениям таких выражений может быть осуществлено с помощью указателя функции, если их оформить (описать) в виде *оператор-функции*.

Общий вид оператор-функции:

$$F(A)=E$$

где F — наименование оператор-функции, E — выражение, арифметическое или логическое, не содержащее переменных с индексами, A — список формальных параметров; элементы списка разделяются запятыми.

Список формальных параметров состоит из идентификаторов простых переменных. В списке не допускается повторение одинаковых элементов, однако одни и те же идентификаторы могут использоваться в качестве формальных параметров нескольких оператор-функций, между собой никак не связанных. Формальные параметры обычно используются в выражении E . Например, конструкция

$$R(X, Y)=X**2+Y**2$$

является определением оператор-функции $R(X, Y)$. Сам по себе этот оператор не исполняется, и поэтому все оператор-функции должны быть помещены перед первым исполняемым оператором программной единицы. Согласно правилу первой буквы тип этой оператор-функции действительный. Однако тип может быть задан особо описательными операторами или оператором `IMPLICIT`. Так, группа операторов

`LOGICAL V, P`

`V(A, B, P)=A.LT.B.AND.P`

задает оператор-функцию $V(A, B, P)$ логического типа (A, B — действительные переменные).

Выполнение оператор-функции происходит после обращения к оператор-функции через ее указатель, представляющий собой идентификатор оператор-функции со списком фактических параметров. В качестве фактических параметров допускаются выражения. Между формальными и фактическими параметрами должно соблюдаться соответствие в количестве, типах и порядке следования.

В процессе обращения к оператор-функции сначала вычисляются значения всех фактических параметров. Рассчитанные значения присваиваются соответствующим формальным параметрам. Затем вычисляется значение оператор-функции. Оно используется далее в том выражении, из которого происходило обращение к данной оператор-функции. Оператор-функции могут использоваться

только в тех программных единицах, где они определены. Например, если в начале программы содержится оператор-функция $R(X, Y)$, то внутри программы оператор

$$T=R(A, B)-R(C, D)$$

вызовет вычисление значения оператор-функции R (дважды) согласно описанию при указанных фактических параметрах (A и B , C и D соответственно).

Правая часть оператор-функции (выражение E) может содержать не только формальные параметры, но и другие переменные, а также обращения к стандартным функциям, ранее определенным оператор-функциям и подпрограммам типа **FUNCTION** (§ 13). В выражение E не должно входить обращение к этой же самой оператор-функции. Так, определение оператор-функции в виде

$$Q(Z)=Z+Q(X)$$

является недопустимым. Однако обращение

$$S=R(A, B)+R(R(C, D), F)$$

является правильной конструкцией. Здесь сначала вычисляется значение выражения $R(C, D)$, затем вновь происходит обращение к этой же оператор-функции с новыми фактическими параметрами, второй из которых есть F . Полученное значение является вторым слагаемым при получении значения переменной S . Первое слагаемое вычисляется однократным обращением к оператор-функции R .

Наименования формальных параметров оператор-функции не должны появляться в операторах **COMMON**, **DATA**, **DIMENSION**, **EQUIVALENCE** и **EXTERNAL**. Эти операторы будут рассмотрены ниже.

Упражнения

1. Оформить упражнение 3 из предыдущего параграфа в виде оператор-функции.

2. Написать оператор-функции для вычисления: 1) площади круга радиуса R ; 2) объема шара радиуса R .

§ 13. Подпрограммы-функции

Рассмотренные в предыдущем параграфе оператор-функции при всех своих положительных качествах имеют ограниченное применение вследствие того, что состоят из единственного оператора — оператора присваивания, в котором вычисляется значение лишь одного выражения, обеспечивая получение, таким образом, одного значения. Так, например, нельзя составить оператор-функцию для вычисления факториала и для других достаточно простых алгоритмов.

Подпрограмма-функция, или подпрограмма типа FUNCTION, отличается от оператор-функции двумя особенностями:

— для определения подпрограммы-функции в ее описании может быть использовано любое количество операторов, число выходных величин также произвольно;

— отладка и трансляция подпрограммы-функции может осуществляться как совместно с основной программой, так и отдельно от нее.

Таким образом, подпрограмма-функция — это самостоятельная программная единица. Она выполняется после обращения к ней из другой программной единицы. При определении подпрограммы типа FUNCTION используются формальные параметры; информация, необходимая при выполнении подпрограммы-функции, поставляется фактическими параметрами. Возможность автономной трансляции (т. е. отдельно от других программных единиц) предоставляет значительные удобства при отладке программ.

Описание подпрограммы-функции составляется следующим образом:

```
ТИП FUNCTION F *S(A)
  {любое количество операторов}
RETURN
END
```

Первый из написанных здесь операторов называется оператором начальной строки подпрограммы-функции, где F — идентификатор, являющийся наименованием подпрограммы-функции, ТИП — указатель типа, A — список формальных параметров, *S — указатель длины подпрограммы-функции.

На месте указателя типа могут использоваться символы COMPLEX, INTEGER, LOGICAL, REAL или же он может быть опущен. Если в качестве указателя типа использован один из перечисленных символов, то он определяет тип подпрограммы-функции. Определение типа подпрограммы-функции может быть выполнено и без указателя типа в операторе начальной строки, а по первой букве идентификатора или использованием соответствующего оператора описания типа внутри подпрограммы-функции. Указатель длины в операторе начальной строки также может быть опущен.

Список формальных параметров A состоит из наименований простых переменных, массивов, других подпрограмм-функций или подпрограмм. Элементы списка разделяются запятыми. Список должен содержать хотя бы один элемент.

Формальные параметры служат только для указания типа и длины, количества и порядка записи фактических параметров, которые ставятся им в соответствие при каждом обращении к подпро-

грамме. Тип формальных параметров может быть задан неявно или установлен операторами описания типа, следующими за оператором начальной строки подпрограммы-функции. Если формальным параметром является идентификатор массива, то он должен быть описан внутри подпрограммы-функции.

Наименования формальных параметров имеют силу только внутри подпрограммы-функции, а за ее пределами эти же наименования могут иметь другой смысл.

Список формальных параметров может содержать наименования переменных, значения которых вычисляются вместе со значением F.

После оператора начальной строки, операторов описания типа и операторов задания размеров массивов (если таковые имеются) следуют другие неисполняемые и исполняемые операторы. В неисполняемых операторах не должно упоминаться наименование F подпрограммы-функции, которое входит в оператор начальной строки. Это наименование должно встретиться хотя бы один раз как левая часть оператора присваивания или как элемент списка оператора ввода либо как фактический параметр в операторе вызова подпрограммы (§ 14), и по крайней мере один из этих операторов должен выполняться.

Внутри подпрограммы-функции не могут находиться операторы, определяющие другие подпрограммы-функции или подпрограммы, а также операторы BLOCK DATA (§ 18) и останова. Операторы определения оператор-функций, если они имеются, должны располагаться до первого исполняемого оператора. Ни один из элементов списка формальных параметров не должен входить в операторы COMMON, DATA, EQUIVALENCE или EXTERNAL внутри данной подпрограммы-функции.

Обращение к подпрограмме-функции осуществляется использованием ее указателя в качестве операнда в выражениях, содержащихся в другой программной единице. Указателем подпрограммы-функции, точно так же как и в случае стандартных функций и оператор-функций, является конструкция, состоящая из идентификатора подпрограммы-функции и списка фактических параметров (в скобках).

Между формальными и фактическими параметрами должно быть соблюдено соответствие в количестве, типе и порядке следования. При обращении к подпрограмме-функции в общем случае формальные параметры заменяются соответствующими фактическими и выполняются операторы, содержащиеся в определении подпрограммы-функции между оператором начальной строки и оператором *возврата* RETURN. С помощью оператора RETURN осуществляется возврат из подпрограммы в вызывающую программу. Операторов RETURN в подпрограмме-функции может быть несколько.

В подпрограмме-функции должен содержаться только один оператор END, который указывает конец подпрограммы-функции и всегда является последним оператором в определении подпрограммы типа FUNCTION.

При вычислении значений выражений, в которых содержится обращение к подпрограмме-функции, вместо указателя подпрограммы-функции фигурирует значение, которое вычисляется в результате выполнения операторов этой подпрограммы-функции при заданных значениях фактических параметров. Таким образом, операторы, содержащиеся в описании подпрограммы-функции, выполняются во всех случаях, если только в вызывающей программе используется указатель этой подпрограммы-функции.

Пример 1. Вычисление $N!$ можно оформить в виде следующей подпрограммы-функции:

```
FUNCTION NF(N)
  NF=1
  DO 2 K=1, N
2 NF=NF*K
  RETURN
END
```

Теперь может быть получено значение трехчлена $(N!)^2 + N! + N$, если в вызывающей программе написать оператор

$$L = NF(N) ** 2 + NF(N) + N$$

где L — идентификатор результата. Отметим, что в этом примере формальный и фактический параметры совпадают по написанию, тем не менее это разные величины.

Вычисление L осуществляется в такой последовательности: после знака присваивания по идентификатору NF управление будет передано к подпрограмме-функции с таким же наименованием; формальный параметр в описании заменяется фактическим и выполняются все операторы, содержащиеся в описании; по оператору RETURN происходит возврат в вызывающую программу в оператор, из которого было произведено обращение к подпрограмме-функции, т. е. в то же место после знака присваивания, только теперь здесь уже будет использовано значение $NF(N)$, которое возводится в квадрат; после знака сложения вновь происходит обращение к подпрограмме-функции; затем значение $NF(N)$, которое здесь вычислялось заново, прибавляется к уже известному квадрату этого значения; в последнюю очередь к полученной сумме прибавляется значение N.

Очевидно, вычисление L будет экономнее, если вместо одного оператора в вызывающей программе написать два:

```
M = NF(N)
L = M ** 2 + M + N
```

т. е. благодаря введению дополнительной переменной и дополнительного оператора удалось избавиться от второго обращения к подпрограмме.

Отметим, что значение L может быть вычислено и в подпрограмме. Например, в следующей подпрограмме-функции одновременно вычисляются значения NF и L :

```
FUNCTION NFL(N, L)
  NFL=1
  DO 2 K=1, N
2  NFL=NFL*K
  L=NFL**2+NFL+N
  RETURN
END
```

Если тип подпрограммы-функции задается явно, то идентификатор подпрограммы-функции должен быть описан также в вызывающей программе. Например, если подпрограмма-функция имеет вид

```
REAL FUNCTION INT(X)
INT=SIN(2./X)—cos(2./X)
RETURN
END
```

то в вызывающей программе следует указать оператор

```
REAL INT
```

Отметим, что подпрограмма-функция $INT(X)$ может начинаться также с операторов

```
FUNCTION INT(X)
REAL INT
```

или

```
REAL FUNCTION INT *4(X)
```

или

```
FUNCTION INT(X)
REAL *4 INT
```

Среди операторов, содержащихся в определении подпрограммы-функции, не допускается оператор, который непосредственно или косвенно обращался бы к ней самой.

При обращении к подпрограмме-функции в качестве фактического параметра могут быть использованы: переменная, элемент массива, массив, выражение, стандартная функция или подпрограмма-функция, подпрограмма.

Если формальным параметром подпрограммы-функции является идентификатор переменной, то фактическим параметром может быть простая переменная, элемент массива или выражение (в частности, константа или указатель функции). Формальному параметру,

являющемуся идентификатором массива, соответствует фактический параметр в виде элемента массива или идентификатора массива. Если формальным параметром подпрограммы-функции является идентификатор подпрограммы-функции, то фактическим параметром может быть идентификатор стандартной функции или идентификатор подпрограммы-функции. Формальному параметру, являющемуся идентификатором подпрограммы, соответствует в качестве фактического параметра идентификатор подпрограммы.

При выполнении подпрограммы-функции замена формальных параметров фактическими осуществляется двумя способами — по наименованию и по значению.

В первом случае перед началом выполнения подпрограммы формальный параметр во всех операторах подпрограммы-функции заменяется на соответствующий фактический параметр и все действия, предусмотренные над формальным параметром, в действительности будут выполняться над соответствующим фактическим параметром. Замена формальных параметров фактическими осуществляется по наименованию, если соответствующий формальный параметр является либо идентификатором массива, либо идентификатором подпрограммы-функции или подпрограммы.

Во втором случае перед началом выполнения подпрограммы-функции значение фактического параметра присваивается соответствующему формальному параметру. После выполнения операторов подпрограммы-функции фактический параметр получает значение соответствующего формального параметра. Замена формальных параметров осуществляется по значению, если формальный параметр является идентификатором простой переменной.

Пример 2. В подпрограмме-функции

```
FUNCTION F(X, Y, Z)
A=X+Y
Z=A
Y=A**2
F=A**2+A+X
RETURN
END
```

формальные параметры являются простыми переменными. Следовательно, в качестве фактических параметров при обращении к F могут быть использованы простые переменные, элементы массива и выражения, т. е. возможны обращения:

```
R1=R+P*F(S, T, U)
R2=R+P*F(S**2+SIN(C), COS(C)*3.14, U)
R3=R+P*F(D(5), D(K), D(7))
```

Во всех этих обращениях операторы подпрограммы-функции F(X, Y, Z) выполняются над формальными параметрами X, Y, Z,

получившими значения соответствующих фактических параметров. Первый и второй фактические параметры являются входными, так как их значения используются в вычислениях внутри подпрограммы-функции. Третий параметр получает значение в результате выполнения подпрограммы-функции, и он называется выходным. Значение, которое имел этот параметр до выполнения подпрограммы функции, после выхода из нее будет потеряно. По смыслу подпрограммы-функции на месте третьего фактического параметра не должно стоять выражение.

Что касается третьего обращения к этой подпрограмме-функции, то массив D должен быть определен в вызывающей программной единице и все переменные, входящие в индексное выражение K, должны получить значения там же. Следует учитывать, что значение индексного выражения не может изменяться внутри подпрограммы-функции.

Если в качестве формального и фактического параметров выступают идентификаторы массивов, то формальный параметр должен быть описан внутри подпрограммы-функции, а фактический параметр — в вызывающей программной единице. Таким образом, этими описаниями определяются формальный и фактический массивы. Формальный массив при обращении заменяется фактическим массивом: первый элемент формального массива заменяется первым элементом фактического массива и т. д. Размерности формального и фактического массивов могут быть различными.

Пример 3. Найти произведение элементов на главной диагонали матрицы. Подпрограмма-функция для этой задачи может быть записана так:

```
FUNCTION DP(A, N)
  DIMENSION A(10, 10)
  DP=A(1, 1)
  DO 5 I=2,N
5  DP=DP*A(I, I)
  RETURN
END
```

Значение фактического параметра, соответствующего формальному параметру N в этой подпрограмме, не должно превосходить 10, однако, исходя из требования совпадения диагональных элементов формального и фактического массивов, данная подпрограмма-функция может быть использована для обработки двумерных квадратных матриц с числом строк и столбцов равным десяти. Например, в вызывающей программе могут содержаться операторы

```
DIMENSION X(10, 10)
R=DP(X, 10)
```

В массивах, которые описываются в подпрограммах-функциях, т. е. в формальных массивах, допускаются переменные или регулируемые размеры. Это означает, что при описании такого формального массива список верхних границ изменения индексов содержит простые переменные целого типа, которые и называются переменными размерами. Каждый из таких размеров должен входить либо в список формальных параметров подпрограммы-функции, либо в одну из общих областей (§ 16). К моменту обращения к подпрограмме-функции переменные размеры должны получить конкретные значения, которые передаются в первом случае через фактические параметры указателей функций, а во втором — через переменные, входящие в состав общих областей. Здесь рассматривается первая возможность.

Пример 4. Подпрограмму функцию из предыдущего примера можно переписать так:

```

FUNCTION DP(A, N)
  DIMENSION A(N, N)
  DP=A(1, 1)
  DO 5 I=2, N
5  DP=DP*A(I, 1)
  RETURN
END

```

Здесь идентификатор массива **A** и переменный размер **N** являются формальными параметрами. В таком варианте не накладываются ограничения на размеры обрабатываемых матриц, так что из одной вызывающей программы может быть обращение $R=DP(X, 8)$, а из другой — обращение $R=DP(Y, 100)$, если, конечно, в вызывающих программных единицах содержатся соответственно операторы

DIMENSION X(8) и DIMENSION Y(100)

В случае использования в качестве фактического параметра элемента массива, когда формальным параметром является идентификатор массива, формальный и фактический массивы должны быть описаны обычным образом. Формальный массив заменяется в процессе обращения к подпрограмме-функции фактическим массивом, причем первый элемент формального массива ставится в соответствие элементу массива, заданному фактическим параметром. Это определяет соответствие между остальными элементами массивов. Необходимо, чтобы каждому элементу формального массива соответствовал элемент фактического массива.

Пример 5. Пусть два первых оператора некоторой подпрограммы-функции записаны в виде

```

FUNCTION P(A)
  DIMENSION A(5)

```

а обращение к ней выполняется оператором $R=P(B(3))$ из программной единицы, в которой описан фактический массив DIMENSION B(7). Тогда в процессе обращения к подпрограмме-функции P устанавливается соответствие между элементом массива A(1) и элементом B(3), A(2) и B(4), ..., A(5) и B(7).

Если формальным параметром подпрограммы-функции является идентификатор подпрограммы-функции или подпрограммы, а в качестве соответствующего фактического параметра выступает идентификатор внешней стандартной функции, подпрограммы-функции или подпрограммы, то замена формального параметра производится по наименованию, а идентификатор такого фактического параметра должен быть указан в списке идентификаторов оператора *внешних подпрограмм* в вызывающей программной единице. Этот оператор относится к неисполняемым операторам и имеет вид

EXTERNAL LI

где LI — список идентификаторов (наименований стандартных функций подпрограмм-функций и подпрограмм); элементы списка разделяются запятыми. Например:

EXTERNAL PF, COS, ALPHA, EXP

Оператор внешних подпрограмм указывает, что идентификатор, входящий в список LI, является наименованием фактического параметра, представляющего собой подпрограмму или функцию, а не переменную. Оператор EXTERNAL должен стоять перед первым исполняемым оператором в вызывающей программе.

Отметим, что наименование оператор-функции не должно появляться в списке идентификаторов оператора EXTERNAL. В то же время, если фактическим параметром оператор-функции является наименование другой функции или подпрограммы, то это наименование должно быть упомянуто в операторе внешних подпрограмм в той программной единице, в которой содержится данная оператор-функция.

Пример 6. Подпрограмма-функция имеет вид

```
FUNCTION H(A, P)
H=P(A)
RETURN
END
```

В программной единице, из которой осуществляется обращение к подпрограмме-функции H, например, такое:

$B=D+H(AF, PF)$

должен содержаться оператор EXTERNAL PF, где PF — фактический параметр, являющийся наименованием некоторой опреде-

ленной подпрограммы-функции. Формальный параметр А при указанном обращении заменяется на фактический АF, а значение Н вычисляется как PF(AF).

Если формальный параметр подпрограммы-функции является простой переменной, то замена его фактическим параметром может быть выполнена по наименованию. Для этого формальный параметр следует заключить с обеих сторон в символы /. Например, если оператор начальной строки в примере 2 переписать в виде

FUNCTION F(X, Y, /Z/)

то при обращении к этой подпрограмме-функции оператором

R1=R+P*F(S, T, U)

при вычислении F будут использованы текущие значения фактических параметров S и T и наименование третьего параметра U.

Упражнения

1. Составить подпрограмму-функцию для суммирования элементов трехмерного массива.

2. Составить подпрограмму типа FUNCTION для вычисления дисперсии элементов одномерного массива А: $D = \frac{1}{N} \sum_{i=1}^N \bar{A}_i^2 - M^2$,

где $M = \frac{1}{N} \sum_{i=1}^N A_i$ — математическое ожидание. Значение М также включить в число выходных параметров.

3. Составить подпрограмму-функцию для вычисления по формуле

$$y_n(x, y, h) = y + \frac{h}{6} (K_1 + 2K_2 + 2K_3 + K_4),$$

где $K_1 = f(x, y)$; $K_2 = f\left(x + \frac{h}{2}, y + \frac{h}{2} K_1\right)$; $K_3 = f\left(x + \frac{h}{2}, y + \frac{h}{2} K_2\right)$; $K_4 = f(x+h, y+hK_3)$, а $f(x, y)$ есть оператор-функция для вычисления значения выражения $2y/x + x^2 e^x$.

4. Разработать подпрограмму типа FUNCTION для нахождения корня уравнения четвертой степени. В качестве аргументов подпрограммы включить коэффициенты уравнения, начальное приближение, точность решения, наименование корня и максимально допустимое количество итераций.

§ 14. Подпрограммы

Подпрограмма (называемая также подпрограммой типа SUBROUTINE или подпрограммой-процедурой) является самостоятельной автономной программной единицей и имеет большое сходство с подпрограммой-функцией. Однако подпрограмма обладает рядом особых свойств, расширяющих возможности ее использования при программировании. Наиболее важное различие между под-

программой и подпрограммой-функцией состоит в том, что с идентификатором подпрограммы не связывается никакое значение и обращение к подпрограмме осуществляется специальным оператором обращения, называемым оператором вызова подпрограммы.

Описание подпрограммы составляется следующим образом:

```
SUBROUTINE S(A)
  (любое количество операторов)
RETURN
END
```

Первый оператор в описании подпрограммы называется оператором начальной строки или оператором-заголовком подпрограммы. В нем S — наименование (идентификатор) подпрограммы, A — список формальных параметров.

Список формальных параметров состоит из элементов, разделяемых запятыми. В качестве формального параметра могут быть использованы простая переменная, идентификатор массива, подпрограммы-функции или подпрограммы, символ * (звездочка). Формальный параметр, являющийся простой переменной, может быть заключен с обеих сторон в символы /. Список формальных параметров (вместе с окаймляющими его скобками) у подпрограммы может отсутствовать.

Тип и длина формальных параметров задаются либо неявно, либо устанавливаются операторами описания типа, следующими за оператором-заголовком подпрограммы. Если формальным параметром является массив, то в подпрограмме должен содержаться описательный оператор, в котором задаются размеры массива. Формальные параметры не могут получать начальные значения, а также содержаться в операторах COMMON, EQUIVALENCE или EXTERNAL.

Среди исполняемых и неисполняемых операторов подпрограммы (между оператором начальной строки и единственным, но обязательным оператором окончания END) не должны находиться операторы BLOCK DATA, FUNCTION, SUBROUTINE, STOP и PAUSE.

Подпрограммы могут содержать внутри себя операторы CALL (см. ниже), вызывающие другие подпрограммы, и любые операторы, выполняющие обращения к подпрограммам типа FUNCTION, библиотечным подпрограммам и описанным в самой подпрограмме оператор-функциям.

Наименование подпрограммы не может появиться внутри подпрограммы в операторе присваивания, в операторах ввода или вывода или же в качестве фактического параметра в операторе вызова подпрограммы. Значения, получаемые в результате работы подпрограммы, присваиваются идентификаторам формальных параметров.

Оператор RETURN обеспечивает возврат из подпрограммы в программную единицу, из которой было произведено обращение к данной подпрограмме. Допускается использование оператора возврата с параметром в виде

RETURN I

где I — простая переменная целого типа стандартной длины или целая константа без знака. Укажем, что использование оператора RETURN в основной программе равносильно оператору STOP.

Обращение к подпрограмме выполняется из другой программной единицы оператором *вызова подпрограммы*. Этот оператор имеет вид

CALL S(AF)

где S — идентификатор подпрограммы, AF — список фактических параметров. В случае подпрограммы без параметров оператор вызова подпрограммы также не содержит параметров.

Оператор CALL передает управление подпрограмме, и подпрограмма будет выполняться до тех пор, пока в ней не встретится оператор RETURN. Если оператор CALL является последним оператором в цикле DO, то после возврата продолжится выполнение цикла.

Подпрограммы могут вызываться как из основной программы, так и из других подпрограмм, но никакая подпрограмма не может обращаться прямо или косвенно к вызвавшей ее программной единице. Например, две подпрограммы не могут обращаться друг к другу. Следовательно, подпрограмма не должна содержать обращения к самой себе ни непосредственно, ни через другие подпрограммы.

Оператор вызова подпрограммы для подпрограмм с параметрами обеспечивает замену формальных параметров подпрограммы фактическими. Обязательно должно выполняться количественное соответствие между формальными и фактическими параметрами, а также соответствие по порядку следования их в списках A и AF. Формальные и фактические параметры, имеющие тип, т. е. параметры, отличные от наименований подпрограмм и символа *, должны быть согласованы по типу и длине.

Пример 1. Подпрограмма определения наибольшего по абсолютной величине элемента в заданной строке двумерного массива имеет вид

```
SUBROUTINE EM(A, I, XM, J, N)
C   A—ИМЯ МАССИВА,
C   XM—НАИБОЛЬШИЙ ЭЛЕМЕНТ,
C   N—РАЗМЕР МАССИВА,
C   I—НОМЕР СТРОКИ,
C   J—НОМЕР СТОЛБЦА,
C   A, I, N—ВХОДНЫЕ ПАРАМЕТРЫ,
C   XM, J—ВЫХОДНЫЕ ПАРАМЕТРЫ
```

```

DIMENSION A(N, N)
XM=ABS(A(I, 1))
J=1
DO 9 K=2, N
IF(ABS(A(I, K))-XM) 9, 9, 7
7 XM=ABS(A(I, K))
J=K
9 CONTINUE
RETURN
END,

```

Нахождение наибольшего по абсолютной величине элемента YB в строке M массива ARRAY(8, 8), описанного, например, в основной программе, можно теперь осуществить, включив в основную программу оператор

```
CALL EM(ARRAY, M, YB, L, 8)
```

где буквой L обозначена переменная, которой присваивается номер столбца. Этот оператор вызовет подпрограмму с наименованием EM, произведет замену формальных параметров фактическими и осуществит передачу управления подпрограмме EM. После выполнения подпрограммы оператором RETURN управление передается оператору, следующему в основной программе за оператором вызова подпрограммы. Полученные значения фактических параметров (выходных данных) можно использовать в основной программе до следующего обращения к данной подпрограмме.

В качестве фактического параметра в операторе вызова подпрограммы могут быть использованы: переменная, элемент массива, массив, выражение (в частности, константа, переменная или указатель функции), стандартная функция или подпрограмма-функция, подпрограмма, конструкция вида &N (N — метка оператора).

Если формальным параметром является переменная, принимающая значение в результате выполнения подпрограммы, то соответствующим ему фактическим параметром может быть только переменная, в остальных случаях фактическим параметром, соответствующим формальному параметру, являющемуся переменной, может быть выражение. Фактическим параметром может быть идентификатор массива или переменная с индексами (элемент массива), если соответствующий формальный параметр есть идентификатор массива.

Если формальным параметром подпрограммы является идентификатор подпрограммы-функции, то фактическим параметром может быть идентификатор стандартной функции или наименование подпрограммы-функции. Формальному параметру, являющемуся идентификатором подпрограммы, соответствует в качестве фактического параметра наименование подпрограммы. Фактический параметр в

виде конструкции &N соответствует формальному параметру, обозначаемому символом *. Если в списке формальных параметров подпрограммы содержится символ *, то внутри подпрограммы используется оператор возврата с параметром, т. е. RETURN I, где значение I является порядковым номером того символа * среди всех таких символов (звездочек) в списке формальных параметров данной подпрограммы (считая слева направо), которому соответствует в списке фактических параметров метка оператора возврата в вызывающую программу.

Пример 2. Пусть вызывающая программа содержит операторы

```
10 CALL TAK(A, B, &60, C, &80)
30 Y=A+B
   GO TO 90
60 Y=A+C
   GO TO 90
80 Y=B+C
90 <продолжение программы>
```

а подпрограмма имеет вид

```
      SUBROUTINE TAK(X, Y, *, Z, *)
      <операторы>
      IF(P) 2, 3, 4
2     RETURN
3     RETURN 1
4     RETURN 2
      END
```

Очевидно, оператор с меткой 10 вызовет подпрограмму TAK. Если при выполнении оператора IF в подпрограмме окажется, что $P < 0$, то осуществится возврат в вызывающую программу на оператор, стоящий за оператором CALL и имеющий метку 30, т. е. обычным путем. Если $P = 0$, то оператор с меткой 3 передаст управление тому оператору, метка которого как фактический параметр соответствует первому символу *, т. е. оператору с меткой 60. Если $P > 0$, то возврат осуществится на оператор с меткой 80.

При выполнении подпрограммы замена формальных параметров фактическими параметрами, как и при выполнении подпрограммы-функции, осуществляется двумя способами — по наименованию и по значению. Замена формальных параметров выполняется по наименованию, если формальный параметр окаймлен с обеих сторон символами /, либо является идентификатором массива, подпрограммы-функции или подпрограммы, и по значению, если формальный параметр является идентификатором переменной и не заключен в символы /, либо является символом *.

Связь между фактическими и формальными параметрами в подпрограмме устанавливается по тем же правилам, которые были рассмотрены в предыдущем параграфе для подпрограммы-функции, в частности, в формальных массивах допускаются переменные размеры.

Пример 3. Подпрограмма сложения [матриц может быть представлена в виде

```
SUBROUTINE MAT(X, Y, Z, M, N)
  DIMENSION X(M, N), Y(M, N), Z(M, N)
  DO 2 I=1, M
    DO 2 J=1, N
      2 Z(I, J)=X(I, J)+Y(I, J)
  RETURN
END
```

Если теперь основная программа содержит описание массивов

```
DIMENSION A(10, 10), B(10, 10), C(10, 10)
```

то она может вызывать подпрограмму MAT для сложения любых матриц с числом элементов в пределах 10×10 , например:

```
CALL MAT(A, B, C, 14, 6) или CALL MAT(A, B, C, 8, 12)
```

Идентификаторы фактических параметров, являющихся наименованиями внешних стандартных функций, подпрограмм-функций или подпрограмм, должны быть указаны в операторе EXTERNAL в вызывающей программной единице.

Упражнения

1. Составить подпрограмму для задачи определения возможности построения треугольника из отрезков, длины которых заданы значениями переменных A, B, C.

2. Написать подпрограмму типа SUBROUTINE, с помощью которой можно было бы по известным значениям переменных A, B, X и L вычислить

$$R = \cos(2\pi X + A) e^{BX}, \quad T = \left(\frac{A+BX}{2} \right)^{L-1} - \left(\frac{A-BX}{3} \right)^{L+2},$$

$$S = \sqrt{A+BX+X^L}, \quad U = (A^2+B^2)^{3/2} + 4X \left(\frac{A^2}{B^2} \right)^{L/2}.$$

3. Составить программу вычисления определителя матрицы третьего порядка (т. е. матрицы с тремя строками и тремя столбцами).

4. Используя программу из предыдущего упражнения как подпрограмму, написать программу вычисления определителя матрицы четвертого порядка разложением по строке или столбцу.

§ 15. Оператор входа

При обращении к подпрограмме-функции или к подпрограмме стандартным способом, рассмотренным в предыдущих параграфах, управление передается оператору начальной строки соответствующей программной единицы, и выполняются последовательно все операторы, пока не встретится в этой последовательности оператор возврата. В ряде случаев такой стандартный вход в подпрограмму является нежелательным, и возникает необходимость входа в данную программную единицу не с самого ее начала. Для организации дополнительных входов в подпрограмму-функцию или подпрограмму используются операторы *входа*.

Оператор входа записывается в виде

ENTRY FS (A)

где FS — наименование входа (идентификатор), A — список формальных параметров.

Все операторы входа, а их в одной подпрограмме-функции или подпрограмме может быть несколько, располагаются среди операторов данной программной единицы в любом месте, кроме области действия оператора цикла. Перед оператором ENTRY не должна стоять метка. Оператор входа может находиться и в основной программе.

В вызывающей программе обращение к подпрограмме-функции или подпрограмме по наименованию входа выполняется по тем же правилам, что и обращение стандартным способом, т. е. по указателю функции, где идентификатором служит наименование входа, или же оператором вызова подпрограммы, где также на месте имени подпрограммы указывается идентификатор входа.

Если обращение к подпрограмме осуществляется по идентификатору, записанному в операторе ENTRY, то управление передается первому исполняемому оператору, следующему за этим оператором. При последовательном выполнении операторов, среди которых имеется оператор входа, последний, как неисполняемый оператор, пропускается.

Пр и м е р 1. Пусть подпрограмма-функция имеет вид

```
INTEGER FUNCTION F(K, L)
2  F=K-L
  RETURN
  ENTRY F1(K, L)
5  IF(L.LT.K)GO TO 2
3  F=K+L
  RETURN
END
```

Если из основной программы обращение к данной подпрограмме-функции осуществляется, например, так:

$$I = J + F(5 * M, N - 2 * M)$$

то после выполнения оператора с меткой 2 происходит возврат в вызывающую программу.

Если обращение имеет вид

$$I = J + F1(5 * M, N - 2 * M)$$

то вычисление значения функции выполнится по оператору либо с меткой 3, либо с меткой 2 в зависимости от условия в операторе с меткой 5. Второй вариант входа из основной программы в подпрограмму-функцию обеспечивается включением в ее описание оператора ENTRY.

Наименование дополнительного входа в подпрограмме-функции должно быть согласовано с наименованием подпрограммы-функции по типу. Поэтому в рассматриваемом примере, поскольку функция F имеет целый тип, в вызывающей программе должен содержаться оператор описания типа INTEGER, в список которого включены идентификаторы F и F1.

Список параметров в операторе CALL или в обращении к подпрограмме-функции должен быть согласован со списком параметров в соответствующем операторе ENTRY, если обращение выполняется по наименованию входа. В то же время списки формальных параметров в операторе начальной строки и в операторах входа могут быть различными.

Пример 2. Если имеется подпрограмма

```
SUBROUTINE PR(X, Y, P)
  IF(Y) 5, 3, 3
3  X = -X
5  Y = X + Y
  ENTRY PT(Y, P)
  P = 2 * Y ** 2
  RETURN
END
```

а в основной программе содержатся операторы

```
X = 8.0
Y = 3.
CALL PR(X, Y, Z)
CALL PT(4., P)
```

то результатом выполнения первого оператора CALL будет значение Z=50.0, результатом второго — значение P=32.0.

Упражнения

1. Написать подпрограмму вычисления определителя матриц вплоть до четвертого входа, предусмотрев исключение выполнения «лишних» операторов для матриц меньшего порядка (второго или третьего).

2. Составить подпрограмму-функцию для упражнения 3 из § 7, используя операторы входа. Написать операторы основной программы, которые осуществляют выбор фигуры и обращение к подпрограмме-функции.

§ 16. Оператор описания общих блоков

В предыдущих параграфах было показано, что передавать информацию из одной программной единицы в другую можно при помощи замены формальных параметров фактическими в подпрограммах-функциях и подпрограммах. Для этих же целей используются общие области памяти или общие блоки. *Общим блоком* называется группа величин, обозначенных идентификаторами простых переменных или идентификаторами массивов (с указанием верхних границ изменения индексов или без этого) и размещаемая на участке оперативной памяти, к которой возможно обращение из нескольких программных единиц. Входящие в общий блок величины, называемые обычно элементами общего блока, разделяются запятыми. Общий блок может быть помечен идентификатором, который окаймляется символом / с обеих сторон. Например, общий блок, содержащий в качестве элементов переменные X, Y, Z и помеченный идентификатором B, обозначается как /B/ X, Y, Z.

Для определения общих блоков служит оператор *описания общих блоков*, записываемый в виде

COMMON P

где в списке P общих блоков должен стоять хотя бы один общий блок.

При описании общего блока идентификатор может отсутствовать. В этом случае соответствующий общий блок является непомеченным. Непомеченный общий блок определяется либо опусканием идентификатора блока и связанных с ним символов / (если он встречается в начале оператора COMMON), либо наличием двух последовательных символов / перед группой идентификаторов непомеченного блока. Примеры использования оператора описания общих блоков:

COMMON A, B, C, D

COMMON //A, B, C, D

COMMON /X/ P, Q, /Y/ R, S//T, W, V

В первых двух примерах указан один и тот же непомеченный общий блок, в третьем примере первый блок помечен идентифика-

тором X, второй — идентификатором Y, а третий не помечен. Отметим, что в списке общих блоков не используется в качестве разделителя между различными общими блоками символ , (запятая).

Если в общем блоке содержится идентификатор массива без индексов, то значения индексов должны определяться оператором DIMENSION или оператором описания типа в той же программной единице. Если в операторе COMMON указан идентификатор массива со списком верхних границ, то отпадает необходимость дополнительного описания этого массива. Так, в последнем примере идентификаторы P, Q, R, S, T, W, V могут относиться как к простым переменным, так и к массивам, а в общих блоках, определяемых оператором

```
COMMON /B1/ A(100), B(100) /B2/ C(10), D(8, 10)
```

A, B, C и D являются идентификаторами массивов.

Структура общего блока определяется порядком следования в нем элементов, а длина — количеством и типом идентификаторов, перечисленных в списке данного блока. Например, оператор

```
COMMON /A/ P(4), Q(4), C(2)
```

содержит общий блок длиной 48 байт, если P и Q — действительные, а C — комплексный массивы.

Идентификатор общего блока используется только для опознавания блока в процессе трансляции, так что этим же идентификатором может быть обозначена любая другая величина в программной единице, в частности элемент общего блока. Один и тот же идентификатор общего блока в операторе описания общих блоков может встречаться несколько раз. В этом случае совокупность элементов, записанных после всех совпадающих имен, определяет один общий блок. Например, операторы

```
COMMON X, Y /A/ U, V, W //Z, C(5), /A/ A(6)  
COMMON X, Y, Z, C(5) /A/ U, V, W, A(6)
```

определяют совпадающие общие блоки: непомеченный — длиной 8 элементов — и помеченный идентификатором A — длиной 9 элементов.

Оператор COMMON неисполняемый, и он должен находиться среди неисполняемых операторов, предшествующих первому исполняемому оператору в программной единице. В программной единице может содержаться любое количество операторов описания общих областей, однако элемент из одного общего блока не должен появляться в другом общем блоке.

Оператор общих блоков обеспечивает доступ к одной и той же области памяти, соответствующей данному общему блоку, из всех программных единиц, содержащих оператор COMMON. Каждая

общая область памяти выделяется именем общего блока независимо от того, какими идентификаторами обозначены элементы этого общего блока в различных программных единицах.

Пример 1. Если некоторая переменная *X* обозначает один и тот же объект в подпрограмме и в основной программе, то в обеих программных единицах следует написать оператор

COMMON X

Пример 2. Пусть в основной программе содержатся переменные *A*, *B*, *C*, а в подпрограмме соответствующие объекты обозначены *X*, *Y*, *Z*. В этом случае в основной программе следует написать оператор

COMMON /S/ A, B, C

и в подпрограмме — оператор

COMMON /S/X, Y, Z

Длины одинаково помеченных общих блоков во всех программных единицах должны совпадать, а непомеченные общие блоки, определенные в разных программных единицах, могут иметь различные длины.

Пример 3. Основная программа содержит оператор

COMMON A(3), B, C

а в подпрограмме используются величины *A(3)* и *B*. В этом случае в подпрограмме можно написать оператор

COMMON A(3), B

и длины непомеченных общих блоков в основной программе и подпрограмме будут различны.

Если же общий блок в основной программе является помеченным, например, имеет вид

COMMON /T/A(3), B, C

или, хотя и является непомеченным, но в подпрограмме используется величина *Z*, соответствующая величине *C* в основной программе, а величина *B* не используется, то длины общих блоков должны совпадать. В этих случаях в подпрограмме должен содержаться один из следующих операторов (соответственно):

COMMON /T/ A(3), B, Z или **COMMON A(3), B, Z**

В первом из них элемент *Z* в подпрограмме не используется, он вводится, чтобы не нарушить длину общего блока */T/*. Во втором операторе элемент *B* введен для того, чтобы не было нарушено соответствие величин *C* и *Z* в непомеченном общем блоке,

Оператор COMMON не допускает описания формальных параметров, для которых соответствие фактическим параметрам устанавливается обращением к подпрограммам-функциям или с помощью оператора CALL. Следовательно, все используемые в подпрограмме переменные должны быть упомянуты либо в списке ее формальных параметров, либо в списке общих блоков, т. е. в операторе COMMON (за исключением переменных, появляющихся в левой части оператора присваивания в подпрограмме).

В ряде случаев, в частности, когда подпрограмма вызывается лишь из одной программной единицы, например основной программы, можно переменную перевести из списка формальных параметров в элементы какого-либо общего блока для основной программы и подпрограммы.

Пр и м е р 4. Подпрограмма переписывания чисел из одного массива в другой может быть оформлена так:

```
SUBROUTINE C(A, B, N)
  DIMENSION A(N), B(N)
  DO 1 I=1, N
1  B(I)=A(I)
  RETURN
END
```

Формальный параметр N, обозначающий размер массива, можно включить в список общих блоков, если в основной программе будет оператор COMMON N.

Соответствующий вариант подпрограммы примет вид

```
SUBROUTINE C1(A, B)
  DIMENSION A(N), B(N)
  COMMON N
  DO 1 I=1, N
1  B(I)=A(I)
  RETURN
END
```

Если же размеры массивов в основной программе не изменяются, то подпрограмма может вообще не иметь формальных параметров:

```
SUBROUTINE C2
  COMMON A(N), B(N), N
  DO 1 I=1, N
1  B(I)=A(I)
  RETURN
END
```

В этом случае в основной программе должен быть оператор COMMON A, B, N, а также заданы размеры массивов A, B и оператор вызова подпрограммы в виде CALL C,

Общие блоки используются не только для передачи информации из одной программной единицы в другую, но и для целей экономии памяти машины; когда вводится общий блок для того, чтобы одну и ту же область памяти сделать доступной нескольким программным единицам, хотя и нет необходимости в обмене информацией между этими программными единицами. Поэтому сопоставляемые переменные в операторах COMMON, принадлежащих различным программным единицам, могут иметь любой тип (целый, действительный, комплексный, логический) и любую длину (стандартную, нестандартную), лишь бы только совпадали длины (в байтах) для всех одноименных (помеченных) общих блоков. Например, в основной программе могут содержаться операторы

```
COMMON /X/ A(2), B, D(3)
REAL *8 A
COMPLEX D
```

а в подпрограмме —

```
COMMON /X/ P(2), L(6), M, N, C(2)
INTEGER L*2, M, N
COMPLEX C
```

так что область памяти /X/ длиной 44 байта используется обеими программными единицами.

Упражнения

1. Основная программа обрабатывает массивы T1 (10) и T2 (5,5), действительные переменные A, B, C, D, E, F и целые K, L, M, N; она обращается к трем подпрограммам. В первой из них участвует массив T1, действительные переменные A, C, E и все целые переменные; во второй — массив T2, все действительные переменные и целые L, M; третья подпрограмма использует оба массива, действительные переменные C, D, E, F и целые K, L, M. Написать операторы COMMON для всех программных единиц. Каково оптимальное число общих блоков?

2. Написать программу вычисления корней квадратного уравнения $AX^2 + BX + C = 0$ с использованием подпрограмм нахождения каждого из корней. Идентификаторы корней уравнения оформить как общие переменные подпрограммы и основной программы.

§ 17. Оператор присваивания начальных значений

Перед выполнением программы простым переменным и элементам массива могут быть присвоены начальные значения неисполняемым оператором

```
DATA A /R/, B /S/, ...
```

который называется оператором *присваивания начальных значений*.

Здесь A, B, ... — списки, содержащие простые переменные, переменные с индексами (индексы должны быть целыми константами)

или наименования массивов; R, S, ... — списки констант (числовых, логических, текстовых или шестнадцатеричных). Любой константе может предшествовать конструкция вида J*, J* называемая коэффициентом кратности (повторителем), где J — целая константа, отличная от нуля. Повторитель J* указывает число переменных из предыдущего списка, которым должно быть присвоено данное значение. Переменная из одного списка не должна появиться в другом списке.

Элементы списка разделяются запятыми. Количество элементов в одном списке переменных должно совпадать с числом элементов в соответствующем списке констант. Тип значения должен совпадать с типом элемента, которому присваивается значение. Значения, задаваемые текстовыми и шестнадцатеричными константами, можно присвоить элементам любого типа.

Пример 1. Оператор

```
DATA K1 /1/, K2, T(1,2) /2, 3.0/, X, Y /2* 1.5/,  
*L /.TRUE./, C, D /'TEXT', 4HTEST/, LB /2, 3, 2*1, 2*4,  
*6, 5/
```

присваивает переменным K1 и K2 соответственно значения 1 и 2, элементу массива T(1, 2) — значение 3.0, обоим переменным X и Y — значение 1.5, логической переменной L — значение .TRUE., а переменным C и D — значения текстовых констант TEXT и TEST. Элементы массива LB получают соответственно значения 2, 3, 1, 1, 4, 4, 6, 5. Так как здесь оператор DATA занимает три строки, вторая и третья строки начинаются с символа продолжения.

Все описательные операторы, определяющие переменные, используемые в операторе DATA, должны предшествовать оператору присваивания начальных значений. Так, в предыдущем примере предполагается, что массив LB(8) предварительно описан.

Переменные получают начальные значения с помощью оператора DATA перед выполнением программы независимо от того, в каком месте программы он расположен, однако оператор присваивания начальных значений должен находиться в программе после всех других неисполняемых операторов.

Значения переменных, определенные оператором DATA, могут в процессе выполнения программы изменяться, например, операторами присваивания или ввода, но не оператором DATA. В списке переменных оператора DATA не должны содержаться формальные параметры подпрограмм-функций и подпрограмм, а также элементы из общих блоков.

Пример 2. Операторы

```
DATA A, B, C(5) /2.5, 2* 3.7/  
DATA A /2.5/, B /3.7/, C(5) /3.7/
```

являются эквивалентными и присваивают переменным А и В соответственно значения 2.5 и 3.7, значение 3.7 получает также элемент массива С(5).

Пр и м е р 3. Если заданы операторы

```
DIMENSION X(5), Y(8)
```

```
DATA X, Y /1.1, 2.1, 5* 3.1, 4.1, 5.1, 4* 6.1/
```

то элементы массива X получают начальные значения: 1.1, 2.1, 3.1, 3.1, 3.1, а элементы массива Y — следующие начальные значения: 3.1, 3.1, 4.1, 5.1, 6.1, 6.1, 6.1, 6.1.

Упражнения

1. В программе содержатся операторы

```
REAL A, B(10), C, D(5, 5), E
```

```
DATA A, B, C, D, E /5* 2.8, 3.2, 3* 1.6, 6*2.9, 15* 0., 8* 1.0/
```

Определить начальные значения всех описанных действительных величин.

2. Составить операторы, в которых были бы описаны типы величин А, В, С, D, E, F, H, если А, С — целые стандартной длины, В, D, E — действительные нестандартной длины, F, H — комплексные стандартной длины, и всем величинам присвоены посредством оператора DATA начальные значения, в том числе вещественной и мнимой частям комплексных величин, равные единице.

§ 18. Подпрограмма данных

Для присваивания начальных значений элементам из помеченного общего блока используется специальная подпрограмма, называемая *подпрограммой данных*.

Структура подпрограммы такова:

```
BLOCK DATA
```

```
. . . . .
```

```
END
```

где между операторами BLOCK DATA и END могут стоять операторы описания типа, операторы DIMENSION, COMMON, EQUIVALENCE, IMPLICIT и оператор присваивания начальных значений. Никакие операторы, кроме перечисленных, не должны содержаться в подпрограмме BLOCK DATA. Подпрограмма данных должна содержать хотя бы один оператор COMMON. С помощью одной подпрограммы данных можно присвоить начальные значения нескольким помеченным блокам. Нельзя присвоить начальные значения одному и тому же помеченному блоку с помощью нескольких подпрограмм данных. В подпрограмме данных в соответствующем операторе COMMON должны быть перечислены все элементы общего блока — как те, которым присваиваются начальные значения.

ния внутри подпрограммы данных (операторами описания типа или оператором DATA), так и те элементы, которые не получают начальных значений.

Пример.

```
BLOCK DATA
DIMENSION M(2, 2)
COMMON /P/ A, M, B, D, C
REAL M, D, B
INTEGER A, C /1/
DATA M(1, 2), D /2.3, 4.12/, A /0/
END
```

С помощью этой подпрограммы начальные значения получают только переменные A, C, D, M(1, 2), являющиеся элементами общего блока /P/.

Подпрограмма BLOCK DATA автономно не транслируется, а подключается к той программной единице, в которой указаны содержащиеся в подпрограмме данных общие блоки. Если такой программной единицей является подпрограмма, то переменные из общего блока определены с момента обращения к подпрограмме до выхода из нее по оператору RETURN.

Таким образом, задание начальных значений переменным может быть осуществлено тремя способами: операторами явного описания типа, оператором DATA и подпрограммой BLOCK DATA. Второй способ от первого отличается лишь тем, что в операторе DATA не определяется тип, а третий способ пригоден только для помеченных блоков COMMON. Задание начальных значений элементам непомеченного общего блока не предусмотрено.

Упражнения

1. Оформить подпрограмму из упражнения 2 § 14 как отдельную программу, предусмотрев задание начальных значений.

2. Пусть задана следующая подпрограмма данных:

```
BLOCK DATA
INTEGER A(2)
REAL K
LOGICAL L, M
DIMENSION B(3)
COMPLEX C
COMMON /BA/ A, K, L, M, /BB/ B, C, D(2)
DATA A /2* 15/, K /7.5/, L, M /. FALSE.,
*.TRUE./, B /2*1.5, 2.5/, C /(1.2, 1.2)/, D /1.5, 1.5/
END
```

Указать, какие начальные значения будут иметь входящие в нее величины.

§ 19. Оператор эквивалентности

В тех случаях, когда в различных частях одной и той же программной единицы используются величины, которые имеют отношение только к одной части этой программной единицы, можно достичь экономии памяти при программировании, если для запоминания указанных величин использовать одну и ту же область памяти. Выделение одного и того же места в памяти для размещения нескольких величин осуществляется оператором *эквивалентности*, имеющим вид

EQUIVALENCE Q

где Q — список групп эквивалентности. Каждая группа эквивалентности представляет заключенный в скобки список, элементы которого являются простыми переменными или переменными с индексами, размещаемыми в одной и той же области памяти. Элементы в обоих списках разделяются запятыми, количество их в группе эквивалентности должно быть не меньше двух,

Пример 1. Оператор

EQUIVALENCE (K, L, M), (A, B)

указывает, что значения переменных K, L, M помещаются в одной и той же области памяти, а переменных A и B — в другой, т. е. тождественными объявляются переменные K, L и M, и A и B.

Если в группе эквивалентности стоит переменная с индексами, то её список индексов может содержать только константы. Одновременно с объявлением эквивалентными элементов массивов, указанных в группе эквивалентности, эквивалентность распространяется на все предшествующие и последующие элементы массивов. Например, оператор

EQUIVALENCE (X(10), Y(2), Z(3))

устанавливает эквивалентность между следующими тройками переменных с индексами:

X(9)	Y(1)	Z(2)
X(10)	Y(2)	Z(3)
X(11)	Y(3)	Z(4)
...

образующими в памяти некоторую область эквивалентности.

Обычно оператор EQUIVALENCE используется в тех случаях, когда два или более массивов могут занимать одно и то же место памяти, при этом структура и длина массивов могут не совпадать.

При установлении эквивалентности необходимо учитывать типы переменных и массивов. При помощи оператора эквивалентности нельзя расчленять массивы на отдельные части, а также устанавливать соответствие между элементами одного и того же массива,

Если переменная, занимающая два слова (по 4 байта каждое), эквивалентна переменной, занимающей одно слово, то последняя будет совмещена с первым словом первой переменной.

П р и м е р 2. Если в программной единице заданы операторы

```
DIMENSION A(2, 4), B(2, 3, 2), C(6)
REAL *8 U, V(5), T
COMPLEX R, S(3)
EQUIVALENCE (A(1, 1), B(1, 2, 1), U),
*(S(1), V(1), C(1)), (R, T, J)
```

то в памяти образуются три области эквивалентности. Первая из них содержит в последовательных ячейках (длиной 4 байта) следующие элементы: A(1, 1), B(1, 2, 1), старшие разряды U; A(2, 1), B(2, 2, 1), младшие разряды U; A(1, 2), B(1, 3, 1); ...; A(2, 4), B(2, 2, 2); вторая: вещественная часть S(1), старшие разряды V(1), C(1); мнимая часть S(1), младшие разряды V(1), C(2); ...; мнимая часть S(3), младшие разряды V(3), C(6); третья: вещественная часть R, старшие разряды T, J; мнимая часть R, младшие разряды T.

Оператор EQUIVALENCE является неисполняемым и должен находиться среди неисполняемых операторов, предшествующих первому исполняемому оператору в программе или подпрограмме.

Не допускается установление эквивалентности между элементами общих блоков (прямо или косвенно). Если две переменные или два элемента массива совмещены по памяти вследствие действия оператора EQUIVALENCE, идентификаторы этих переменных или массивов не могут оба одновременно появляться в операторах COMMON в той же самой программной единице.

Одна и та же переменная может встретиться и в операторе COMMON и в операторе EQUIVALENCE. В этом случае сначала будет этой переменной отведено место в общем блоке, а уже потом установлено соответствие по оператору эквивалентности. Например, операторы

```
DIMENSION D(3)
COMMON A, B, C
EQUIVALENCE (B, D(1))
```

расположат переменные в памяти следующим образом:

```
  A      B      C
  D(1)   D(2)   D(3)
```

тем самым расширяя общий блок размещением новых элементов после последнего элемента блока.

Не разрешается задавать оператор эквивалентности, который потребовал бы выход за начало общего блока, т. е. запрещено расширять общий блок путем размещения новых элементов до первого

элемента блока. Так, в приведенном примере на месте D(1) нельзя писать D(3), поскольку это приведет к следующему расположению переменных:

	A	B	C
D(1)	D(2)	D(3)	

Упражнения

1. Написать оператор, предусматривающий размещение в одном и том же месте памяти 70 первых элементов массива M и 70 последних элементов массива N. Массивы M и N двумерные и состоят каждый из 100 элементов (10×10).

2. Определить значения элементов массива X, которые они получают в результате выполнения следующей программы:

```

REAL X(10)/2*—1.8, 2.75, 7*0.12/,
  *Y(6)/16.1, 3*—11.8,—17.1,—7.9/, Z(6)
EQUIVALENCE (X(5), Z(1))
DO 6 K=1,6
6  Z(K)=Y(K)
STOP
END

```

§ 20. Операторы ввода и вывода

1. Понятие файла. Операторы *ввода* и *вывода* предназначены для обмена информацией между оперативной памятью вычислительной машины и внешними запоминающими устройствами. Внешними устройствами являются: устройство ввода перфокарт, магнитофон, магнитный барабан, устройство с дисками, алфавитно-цифровое печатающее устройство (АЦПУ), выходной перфоратор и др. Вводимая или выводимая информация представляется в виде последовательностей *записей* и носит название *файла*. Запись в свою очередь является последовательностью символов.

Каждому файлу присваивается условный номер, который может быть использован в операторах ввода или вывода в виде целого числа без знака или переменной целого типа. В качестве файла, как правило, рассматривают последовательность данных на совокупности перфокарт, на бумажной ленте, на диске, барабане, на магнитной ленте и т. д. Так, совокупность перфокарт (массив перфокарт) представляет собой файл, в котором записью является отдельная перфокарта, содержащая не более 80 символов. Устройство печати рассматривается как файл, в котором под записью понимается одна печатная строка на бумажной ленте. Совокупность определенного количества строк образует страницу. Количество позиций в строке и число строк в странице определяются устройством печати (для АЦПУ-128 запись может иметь максимальную длину 128 символов, в странице 60 строк). Перфокарту, строку принтера, а также

строку на экране терминала принято называть единицей записи, подчеркивая тем самым, что запись для файлов на указанных носителях имеет всегда фиксированную длину. Количество реально занятых позиций может быть меньше предельного числа, но не больше.

Различают файлы последовательного доступа и прямого доступа.

Файл *последовательного доступа* — это файл, в котором определены начальная и конечная записи, а относительно любой другой записи определены понятия «текущая запись», «предыдущая запись» и «последующая запись». Файл последовательного доступа используется в тех случаях, когда записи передаются в файл в порядке их расположения.

Файл *прямого доступа* — это файл, в котором нумерация записей имеет определяющее значение. Файл прямого доступа используется в тех случаях, когда необходимо выполнить передачу отдельных записей файла в произвольном порядке. Каждый из файлов прямого доступа должен быть описан в программе один раз с помощью оператора описания файлов.

Во время передачи информация может быть преобразована; например, числовые значения преобразованы в двоичную (при вводе) или десятичную (при выводе на принтер) формы. Обмен информацией в ее машинном (двоичном) представлении называется бесформатным обменом. Например, занесение программы или результатов выполнения программы на магнитные ленты, диски, барабаны в тех случаях, когда эта информация будет повторно использована, обычно представляет собой бесформатный обмен. Обмен информацией с преобразованием в соответствии со спецификациями оператора FORMAT (§ 22) называется форматным обменом. Примером форматного обмена может служить вывод данных на АЦПУ — здесь информация из двоичного представления преобразуется в десятичную форму, или ввод числового материала с перфокарт (с клавиатуры), где он набит (набран) в десятичном представлении, и при вводе преобразуется в двоичную форму представления.

Операторы ввода и вывода указывают характер обмена информацией (чтение, запись, печать и т. д.), номер файла, с которым происходит обмен (номер файла, как правило, содержит информацию о типе устройства, на котором организован файл, о так называемом логическом номере устройства), и список величин, которые участвуют в обмене. Кроме того, для организации ввода или вывода в определенных случаях необходимо указать тип передаваемых величин и связанную с этим информацию о преобразовании (формат данных и их расположение в файле).

2. Оператор описания файлов. Оператор *описания файлов* используется для описания файлов прямого доступа и имеет вид

DEFINE FILE N(M, H, F, K)

где наборов символов $N(M, H, F, K)$ в списке оператора может быть произвольное число (в этом случае один набор от другого отделяется запятой). Каждый набор $N(M, H, F, K)$ определяет свой файл. Здесь N — номер файла, M — целое число без знака, определяющее максимальное число записей этого файла (записей может быть на самом деле меньше M , но не больше), H — целое число без знака, определяющее максимальный размер каждой записи. Числовое значение H указывается либо в байтах, либо в машинных словах; при этом поддерживается соотношение: одно слово равно четырем байтам. Что принимается за единицу измерения, указывается параметром F , на месте которого может стоять один из трех символов: E , L или U .

Если F есть E , то размеры записей файлов указываются в байтах, и обращение к файлу такого типа осуществляется операторами ввода или вывода только с форматами (см. пп. 7 и 8); в этом случае говорят, что все записи являются форматными. Если F есть U , то размеры H записей файлов измеряются количеством слов, и обращение к файлам выполняется операторами ввода и вывода только без форматов (пп. 5 и 6); записи являются бесформатными. И, наконец, если F есть L , значение H указывается в байтах, а обращение к файлу такого типа осуществляется операторами ввода и вывода как с форматами, так и без форматов (имеет место смесь форматных и бесформатных записей).

Символ K — простая переменная целого типа. Эта переменная автоматически принимает значение, равное номеру записи, которая непосредственно следует за последней из обработанных (т. е. введенных или выведенных) записей. После выполнения оператора поиска элемента файла переменная принимает значение, равное номеру найденной записи. Переменная K — номер записи — не может использоваться в программе в каком-либо другом смысле.

В одной программе может содержаться несколько операторов описания файлов, и все они должны находиться среди неисполняемых операторов до первого исполняемого оператора программы. Например, оператор

DEFINE FILE 9(128, 8, U, J), 14(66, 50, E, I), 5(30, 48, L, N)

определяет три файла с номерами 9, 14 и 5: первый — это 128 бесформатных записей, каждая запись содержит 8 слов, переменная J будет принимать значения 2, 3, ..., 129 после обработки записей соответственно с номерами 1, 2, ..., 128; второй содержит 66 форматных записей, каждая из которых состоит не более чем из 50 байт, переменная I будет принимать после обработки первой записи значение 2, после обработки второй записи — значение 3 и т. д. до значения 67; файл, имеющий номер 5, состоит из 30 записей, каждая длиной не более 48 байт, переменная N будет принимать значения 2, 3, ..., 31 после обработки соответствующих записей с номерами

1, 2, ..., 30. Символ U в описании первого файла указывает, что для передачи информации должны использоваться операторы ввода или вывода только без форматов. Символ E в описании файла с номером 14 указывает, что для обработки записей этого файла должны использоваться только операторы ввода или вывода с форматами. Записи файла с номером 5 могут передаваться операторами: ввода или вывода с форматами и без форматов, на что указывает символ L в описании.

3. Список ввода-вывода. С операторами ввода и вывода связано понятие *списка ввода-вывода*, который определяет участки оперативной памяти, используемые для ввода или вывода информации. Иными словами, в списке ввода-вывода указываются величины, участвующие в обмене информацией между оперативной памятью и внешними устройствами, и порядок передачи значений этих величин.

Список ввода-вывода имеет вид

A1, A2, ..., AN

где элементами списка могут быть простые переменные и переменные с индексами, наименования массивов и *неявные циклы*.

Под неявным циклом понимается конструкция вида

(B1, B2, ..., BK, I=N1, N2, N3)

где элементы B1, B2, ..., BK — либо переменные (простые или с индексами), либо в свою очередь неявные циклы, I=N1, N2, N3 — заголовок цикла, в котором: I — целая переменная, используемая в индексных выражениях переменных B1, B2, ...; N1, N2, N3 — целые константы или простые переменные целого типа; если N3 имеет значение, равное +1, то элемент N3 в записи заголовка цикла может быть опущен. Например, неявный цикл может иметь вид

((R(I, J, K), I=L1, L2, L3), J=M1, M2, M3), K=N1, N2, N3)

Эта конструкция осуществляет перечисление элементов массива R в соответствии с порядком выполнения вложенных циклов:

DO N K=N1, N2, N3

DO N J=M1, M2, M3

DO N I=L1, L2, L3

где N — метка конца цикла.

Примеры списков ввода-вывода:

X, Y, K, H(J), N(3, 5)

BA, S, (Z(I), I=1, 20), E(J+1)

((A(I, J), I=1, 3), J=1, 5)

((A(I, J), J=1, 5), I=1, 3)

(X(J), Y(J), J=1, 30)

(I, P(I), I=1, 100)

Если в качестве элемента списка ввода-вывода указана переменная, то это означает, что необходимо выполнить ввод или вывод значения этой переменной. Если в качестве элемента списка указан идентификатор массива, то это означает, что надо выполнить ввод или вывод всех элементов этого массива в том порядке, в котором эти элементы размещены в массиве. Если в качестве элемента списка ввода-вывода указан неявный цикл, то это означает, что необходимо передать значения всех переменных, указанных в неявном цикле, организовав порядок их следования в соответствии с циклическим изменением параметров циклов.

Элементы списка ввода-вывода обрабатываются последовательно слева направо. В одном списке могут содержаться элементы разных типов.

В первом из приведенных примеров последовательно перечисляются идентификаторы X, Y, K, которые могут быть как наименованиями простых переменных, так и массивов, элемент массива N с индексом J, элемент массива N с индексами 3 и 5.

Во втором примере третьим элементом списка является неявный цикл для элементов массива Z с индексами от 1 до 20.

В третьем примере перечисляются элементы двумерного массива по столбцам (первым меняется первый индекс), а в четвертом примере эти же элементы перечисляются по строкам (первым меняется второй индекс J).

В пятом примере попарно перечисляются элементы массивов X и Y: X(1), Y(1), X(2), Y(2), ..., а в последнем примере указан элемент списка, который используется обычно в операторе вывода. В неявный цикл здесь включена простая переменная, являющаяся параметром цикла. В этом списке перечисляются попарно параметр цикла и соответствующий ему элемент массива: 1, P(1), 2, P(2), ...

4. Операторы управления файлами. Операторы *управления файлами* используются при работе с магнитными лентами и дисками. К этим операторам относятся: оператор подвода файла, оператор возврата, оператор конца файла и оператор поиска записи файла. Первые три из них применяются для файлов на ленте и для файлов последовательного доступа на диске, последний — только для файлов прямого доступа на диске.

Оператор *подвода файла* имеет вид

REWIND N1

где N1 — номер файла. Этот оператор устанавливает файл с номером N1 в начальное состояние, т. е. подготавливает первую запись этого файла к работе с операторами ввода и вывода.

Оператор *возврата* записывается в виде

BACKSPACE N1

где N1 — номер файла. Этот оператор при своем выполнении осуществляет переход от текущей записи файла с номером N1 к его предыдущей записи, т. е. к обработке подводится предыдущая, участвовавшая в работе запись. Если текущей записью при этом была первая запись файла, то оператор возврата не выполняет никакого действия. Например, если магнитная лента уже установлена в начальное положение, ни оператор REWIND, ни оператор BACKSPACE не выполняются.

Оператор *конца файла* имеет вид

END FILE N1

и записывает признак (метку) конца файла с номером N1. Выполнение этого оператора заключается в преобразовании текущей записи файла в его конечную запись, т. е. файл с номером N1 заканчивается данной текущей записью.

Оператор *поиска записи файла* имеет вид

FIND (N1'R)

где N1 — номер файла, R — номер записи этого файла. В результате выполнения оператора FIND запись файла N1 с номером R подготавливается к работе с операторами ввода и вывода, т. е. устанавливается в положение, когда обработка этой записи осуществляется за минимальное время.

5. Операторы ввода без формата. Ввод без формата (или бесформатный ввод) используется тогда, когда записи файла содержат данные, представленные в двоичной форме, т. е. преобразование представления информации не требуется.

Оператор *ввода последовательного доступа без формата* имеет вид

READ (N1) L

где N1 — номер файла, L — список ввода. Этот оператор осуществляет ввод текущей записи файла с номером N1 в оперативную память, определенную списком L. Например, оператор

READ (4) X, Y

вводит из текущей записи файла с номером 4 в область оперативной памяти, определяемой переменными X и Y, два значения.

Если в рассматриваемом операторе отсутствует список ввода, т. е. оператор записан в форме

READ (N1)

то действие оператора заключается в переходе от текущей записи файла к следующей его записи. Эта разновидность оператора ввода последовательного доступа без формата употребляется в том случае, когда считываемая информация не используется, но есть необ-

ходимость перехода к новой записи (например, можно перемотать участок магнитной ленты, не снимая с него информацию).

Оператор *ввода прямого доступа без формата* имеет вид

READ (N1'R) L

где N1 — номер файла, R — номер записи этого файла, L — список ввода. Под R понимается выражение, задающее относительный номер или местоположение требуемой записи (разумеется, в пределах максимального числа записей M, определенного оператором DEFINE FILE).

Выполнение этого оператора сводится к вводу записи с номером R файла N1 в область оперативной памяти, которая определена списком ввода L. Например, оператор

READ (6'7) X, Y

вводит запись с номером 7 файла 6 в область оперативной памяти, определяемую переменными X и Y, два значения.

6. Операторы вывода без формата. Промежуточный вывод данных на ленты и диски в тех случаях, когда выведенные данные через некоторое время вновь будут при выполнении программы использованы в оперативной памяти, целесообразно организовать без преобразования формы представления. Для этих целей используются два оператора: оператор *вывода последовательного доступа без формата* и оператор *вывода прямого доступа без формата*.

Первый из них имеет вид

WRITE (N1) L

где N1 — номер файла, L — список вывода. Этот оператор осуществляет выборку данных из оперативной памяти, определенных списком L, и, не изменяя формы представления, образует текущую запись файла, который имеет номер N1. Например, оператор

WRITE (5) A, B

есть оператор последовательного доступа без формата, который передает значения переменных A и B в файл с номером 5 без преобразования, образуя очередную запись этого файла.

Оператор вывода прямого доступа без формата имеет вид

WRITE (N1'R) L

где N1 — номер файла, R — номер записи этого файла, L — список вывода. Этот оператор выводит данные, определенные списком L, из оперативной памяти и образует запись с номером R файла с номером N1. Например, оператор

WRITE (7'9) A, B

выводит значения переменных A и B в запись с номером 9 файла с номером 7 без преобразования.

7. Операторы ввода с форматом. Выполнение операторов ввода и вывода с форматом управляется оператором задания формата (§ 22), указывающего вид и структуру передаваемых величин.

Оператор *ввода последовательного доступа с форматом* обычно имеет вид

READ (N1, N2) L

где N1 — номер файла, N2 — метка оператора FORMAT, L — список ввода.

С помощью этого оператора последовательно вводятся записи файла с номером N1 в те места оперативной памяти, которые определяются элементами списка L. При этом информация преобразуется в соответствии со списком спецификаций оператора FORMAT с меткой N2. Оператор задания формата с меткой N2 указывает также количество вводимых записей и их структуру.

Пример 1. Оператор

READ (N1, N2) A, B, C, M

с файла, имеющего номер N1, например с перфокарт, вводит четыре числа; значение первого из них присваивается переменной A, второго — B и т. д.

В элементе списка, заданном неявным циклом, на месте параметров циклов (нижней границы, верхней границы и шага) могут стоять целые переменные, значения которых задаются самим оператором READ.

Пример 2. Оператор

READ (N1, N2) K, (X(I), I=K, 200, 4)

сначала вводит значение K, затем элемент массива X(K), далее элемент X(K+4) и т. д.

Пример 3. Оператор

READ (N1, N2) (X(I), Y(I), I=1, 6)

вводит элементы двух массивов X и Y в такой последовательности: X(1), Y(1), X(2), Y(2), X(3), ...

Если же структура файла с номером N1 такова, что в нем расположены сначала элементы одного массива, затем другого, то соответствующий оператор имеет вид

READ (N1, N2) (X(I), I=1, 6), (Y(I), I=1, 6)

Оператор ввода с форматом может быть записан в виде

READ (N1, N2, END=N3, ERR=N4) L

где N3, N4 — метки операторов; параметры END=N3 и ERR=N4 могут задаваться в любом порядке и любой из них (либо оба) может быть опущен, N3 — метка оператора, на который передается управление после окончания ввода, если нет совпадения по коли-

честву переменных в списке L и констант на вводе, N4 — метка оператора, получающего управление, если при вводе будет обнаружена ошибка. Например, в операторе ввода

```
READ (5, 71, ERR=101, END=26) A, B, C, D
```

после окончания ввода данных файла с номером 5 в соответствии с оператором FORMAT, имеющим метку 71, предусмотрена передача управления на оператор с меткой 26. Если при вводе будет зафиксирована ошибка, управление передается на оператор с меткой 101.

Поскольку совокупность перфокарт представляет собой файл, где записью является каждая отдельная перфокарта, при вводе перфокарт номер соответствующего файла есть логический номер устройства ввода с перфокарт и в этом случае оператор ввода может быть записан проще:

```
READ N2, L
```

где N2 и L имеют тот же смысл, что и в общем случае. Например, запись

```
READ 18, (A(I), B(I), C(I), I=1, 10)
```

представляет собой оператор ввода с перфокарт значений элементов массивов A, B и C в соответствии с оператором задания формата с меткой 18.

Оператор ввода может употребляться и без списка ввода в виде

```
READ (N1, N2)
```

где N1 — номер файла, N2 — метка оператора FORMAT. Эта разновидность оператора ввода используется тогда, когда вся считываемая информация идет для пополнения текста, заданного в самом операторе FORMAT с меткой N2.

Оператор *ввода прямого доступа с форматом* имеет вид

```
READ (N1'R, N2) L
```

где N1 — номер файла, R — номер записи этого файла, N2 — метка оператора задания формата, L — список ввода. Этот оператор осуществляет чтение данных из одной или нескольких записей файла с номером N1, начиная с записи, имеющей номер R, преобразование их в двоичную форму представления и размещение в областях оперативной памяти, заданных списком ввода L. Информация о количестве считываемых записей и о структуре записей содержится в операторе FORMAT с меткой N2. Например, оператор

```
READ (9'J, 12) S
```

обеспечивает чтение значения переменной S из записи J файла с номером 9 согласно формату, заданному в операторе с меткой 12.

Значение J к моменту выполнения оператора READ должно быть определено.

8. Операторы вывода с форматом. Различаются операторы вывода последовательного доступа с форматом и операторы вывода прямого доступа с форматом.

Оператор *вывода последовательного доступа с форматом* в общем случае записывается в виде

WRITE (N1, N2) L

где N1 — номер файла, N2 — метка оператора задания формата, L — список вывода. Этот оператор осуществляет последовательный вывод данных из участков оперативной памяти, определенных списком L. Преобразование данных и их расположение в файле с номером N1 выполняется в соответствии со спецификациями оператора FORMAT, имеющего метку N2. Например, оператор

WRITE (1, 22) I, J, A, B, C, (X(N), N=1, 10)

выводит 15 величин: два целых и три действительных числа и 10 значений элементов массива X. Если номер файла 1 есть номер файла печати (т. е. логический номер АЦПУ), то указанные данные будут напечатаны (с предварительным преобразованием в десятичную форму представления). Количество записей (строк) и структура строк определяются оператором задания формата с меткой 22.

Оператор

WRITE (1, N2) (I, A(I), I=1, 100)

выводит элементы одномерного массива A с соответствующими номерами, т. е. последовательность значений: 1, A(1), 2, A(2), 3, A(3), 4, ...

Вывод через печатающее устройство может быть осуществлен также оператором

PRINT N2, B

Например, оператор печати элементов массива A со своими номерами будет выглядеть так:

PRINT N2, (I, A(I), I=1, 100)

Для передачи информации на выходной перфоратор служит оператор

PUNCH N2, B

Если вся выводимая информация содержится в операторе FORMAT, то вывод осуществляется оператором WRITE без списка вывода

WRITE (N1, N2)

Смысл обозначений N2, L в операторах PRINT и PUNCH и обозначений N1, N2 в последнем операторе WRITE совпадает с принятым ранее.

Оператор *вывода прямого доступа с форматом* имеет вид

WRITE (N1'R, N2) L

где, как и выше, N1 — номер файла, R — номер записи, N2 — метка оператора задания формата, L — список вывода.

Этот оператор осуществляет выборку данных из областей оперативной памяти, заданных списком вывода L, преобразование и размещение их в одной или нескольких записях файла N1, начиная с записи с номером R. Информация о количестве записей и о форме представления данных в записях содержится в операторе задания формата с меткой N2.

Пример 4. Пусть задан двумерный массив X(10, 30). Требуется записать его на магнитную ленту так, чтобы каждая строка массива образовала отдельную запись.

Для выполнения указанной операции можно воспользоваться операторами

```
DO 1 I=1, 10
1 WRITE (7) X(I, J), J=1, 30)
```

где принято, что файл на магнитной ленте имеет номер 7.

Чтобы прочитать записанную информацию, необходимо установить магнитную ленту в начало или же осуществить возврат ленты назад на определенное количество записей. Так, если требуется прочитать значения элементов шестой строки массива и присвоить их элементам некоторого другого массива Y(30), то эти действия могут выполнить операторы

```
DO 3 I=1,5
3 BACKSPACE 7
READ (7) (Y(I), I=1, 30)
```

где первые два оператора обеспечивают необходимый возврат на пять записей.

Возможен другой путь решения этой задачи. Так, запись строк массива X(10, 30) можно произвести вместе с соответствующими номерами строк:

```
DO 1 I=1, 10
1 WRITE (7) I, (X(I, J), J=1, 30)
REWIND 7
```

Запись теперь представляет собой набор значений, первое из которых есть номер строки, остальные — элементы данной строки. Последний оператор этой группы устанавливает магнитную ленту в начальное положение. Чтение шестой строки матрицы в таком

варианте выполняется операторами

```
4 READ (7) I
  IF (I.NE.6) GO TO 4
  BACKSPACE 7
  READ (7), I, (Y(J), J=1, 30)
```

в которых после считывания первого элемента I очередной строки его значение сравнивается с номером разыскиваемой (шестой) строки. Если прочитанное значение не совпадает с номером необходимой строки (т. е. I не равно 6), то нужно перейти к чтению первого элемента следующей записи (строки). Если же значение I совпадает с заданным номером, то необходимо считывать остальные элементы строки, которые составят массив данных $Y(30)$. При этом предварительно необходимо выполнить оператор возврата `BACKSPACE 7`, так как после считывания любой текущей записи или ее части к операции считывания будет подготовлено начало следующей записи (к считывающим головкам подводится начало следующей записи).

Упражнения

1. Описать три файла с номерами 3, 13 и 23 для произвольных записей длиной до 6 слов в количестве до 62 записей в каждом файле,

2. Составить список ввода-вывода для передачи значений элементов двух массивов $A(20, 40)$ и $B(40, 20)$ в такой последовательности: первая строка массива A , первый столбец массива B , вторая строка массива A , второй столбец массива B и т. д. с указанием каждый раз перед обработкой очередной строки или очередного столбца соответствующего номера строки или столбца.

3. Составить программу вывода значений элементов массива $M(16)$ в файл 5 последовательного доступа, содержащий пять записей, так, чтобы эти записи содержали соответственно элементы массива с номерами:

- 1, 3, 5, 7, 9, 11, 13, 15
- 2, 4, 6, 8, 10, 12, 14, 16
- 1, 2, 4, 5, 7, 8, 10, 11, 13, 14
- 3, 6, 9, 12, 15
- 1, 2, 5, 6, 9, 10, 13, 14

4. Первая из 25 записей файла прямого доступа с номером 6 содержит числа:

1.8, -3.7, 5.2, 6.7, -1.7, -0.8, 2.0;

вторая запись состоит из чисел:

4.7, -3.1, 9.2, -8.0, -0.8,

Эти числа имеют стандартную длину, относятся к действительному типу и представлены в двоичной форме. Определить значения переменных X, Y, A, B, C , которые они получают в результате выполнения следующей программы:

```
REAL A, B, C, X, Y
DEFINE FILE 6(25, 120, L, K)
```

```

READ (6'2) A, B, C, X, Y
READ (6'1) X, Y
STOP
END

```

§ 21. Оператор формирования списков

Операторами ввода и вывода последовательного доступа могут быть обработаны значения без перечисления их имен в списке ввода-вывода, что создает определенные удобства для оперативной смены значений переменных, используемых в программе. В этом случае переменные и массивы должны быть описаны неисполняемым оператором *формирования списков*, имеющим вид

```
NAMELIST P
```

где P — список блоков наименований. Каждый блок наименований в списке имеет вид

```
/X/ A1, A2, ..., AN
```

Здесь X — идентификатор блока, A1, A2, ..., AN — переменные или наименования массивов. Разделителем двух блоков является первый из символов /, которыми окаймляется наименование следующего блока.

Наименование блока может содержаться только в одном операторе NAMELIST и не должно появляться где-либо в программе, кроме как в операторах ввода и вывода, в которых отсутствует список. Идентификатор переменной или массива может принадлежать одному или нескольким блокам.

В операторах READ и WRITE не задается ни список ввода-вывода, ни метка оператора FORMAT. Вместо этого указывается идентификатор блока наименований оператора NAMELIST, т. е. операторы ввода и вывода в этом случае соответственно имеют вид

```

READ (N1, X)
WRITE (N1, X)

```

где N1 — номер файла, X — идентификатор блока.

Во вводном файле записываются наименования переменных и их значения в форме

```
A1=K
```

где A1 — простая переменная или элемент массива, K — значение (константа). Форма записи массивов следующая:

```
A2=J1* K1, J2*K2, ...
```

где A2 — идентификатор массива, K1, K2, ... — значения (константы), J1, J2 — целые константы (коэффициенты кратности), указывающие, сколько раз должно быть повторено следующее за

коэффициентом кратности значение. Коэффициенты кратности могут и отсутствовать.

Файл начинается с записи &X, где X — идентификатор блока, и заканчивается записью &END; при этом первая позиция (в строке или на перфокарте) игнорируется, т. е. символ & должен содержаться во второй позиции.

Между этими записями располагаются записи, в которых задаются значения переменных и массивов указанным выше способом. Количество элементов в записях может быть любым, в частности, не обязательно, чтобы все элементы блока наименований были перечислены и получили значения, но все элементы записей должны принадлежать заданному блоку наименований. Между отдельными элементами ставится разделитель, (запятая). Каждая запись может содержать одну или несколько переменных, в записи допускаются пробелы. Порядок следования переменных в записях несуществен.

Пр и м е р. Пусть в программе содержатся операторы

```
DIMENSION S(5)  
NAMELIST /N/ A, S, M
```

а в устройство ввода заложен массив перфокарт, содержащий следующую информацию:

```
&N  
M=5; A=3.45, S=1.1, 2*1.5, 1.9, 2.1  
&END
```

Здесь каждая строка соответствует содержимому одной перфокарты. Тогда оператор

```
READ (3, N)
```

осуществит ввод (если 3 — номер устройства ввода перфокарт) с указанных перфокарт семи значений и присвоение этих значений элементам блока /N/. Все данные вводятся в таком виде (формате), какой был задан на вводном устройстве. При выводе данных сохраняется тот же самый основной формат, что они имели при вводе.

В выводной файл включаются идентификаторы и значений всех переменных и элементов массивов, принадлежащих заданному в операторе WRITE блоку. Порядок следования их при выводе соответствует порядку следования наименований в блоке. Так, если в процессе работы программы значения элементов блока /N/ из предыдущего примера не изменились, то оператор вывода

```
WRITE (1, N)
```

сформирует выводной файл с номером 1 в виде

```
└─&N  
└─A=3.45, S=1.1, 1.5, 1.5, 1.9, 2.1, M=5  
└─&END
```

Упражнения

1. Написать программу, обеспечивающую ввод значений элементов матриц $A(3, 3)$ и $B(3, 3)$, вычисление элементов матрицы $C(3, 3)$, являющейся суммой матриц A и B (элементы матрицы C получаются по алгоритму $c_{ij} = a_{ij} + b_{ij}$), и вывод значений элементов матрицы C , если значения элементов матрицы A набиты на одной перфокарте следующим образом:

$A = 37., 81., 25., 44., 76., 59., 62., 98.$

а значения элементов матрицы B — на другой перфокарте:

$B = 77.2, 2* 66., 99., 4* 44., 55.$

2. На перфокарте набита следующая информация:

$X = 5, Y = 2, Z = 4, B = 2.5, A = 1.5, C = 0.5$

Составить операторы, с помощью которых на выводном устройстве может быть получена перфокарта, содержащая следующую информацию:

$A = 1.5, B = 2.5, C = 0.5, X = 5, Y = 2, Z = 4$

Какую информацию следует предварительно ввести в машину?

§ 22. Оператор задания формата

Обычно операторы ввода и вывода используются совместно с оператором *задания формата*, который обеспечивает необходимые преобразования и редактирование при обмене информацией между оперативной памятью и внешними устройствами.

Оператор задания формата имеет вид

$N2 \text{ FORMAT } (S)$

где $N2$ — метка оператора (эта метка указывается в соответствующем операторе ввода или вывода), S — список спецификаций.

Оператор FORMAT является неисполняемым, он может располагаться в любом месте программы.

1. **Список спецификаций.** *Список спецификаций* S состоит из кодов (спецификаций) оператора FORMAT , разделяемых запятыми или символами / (наклонная черта). Так, список S может иметь вид

$S1, S2, \dots / \dots, N(SM, \dots), JSQ, \dots / \dots$

где $S1, S2, \dots$ — коды оператора FORMAT ; N, J — коэффициенты кратности (целые константы без знака), указывающие число повторений следующих за ними повторяемых групп спецификаций. Например, $3SQ$ означает SQ, SQ, SQ , а конструкция $2(SM, SN)$ указывает, что группа спецификаций SM, SN повторяется дважды, т. е. имеем SM, SN, SM, SN .

Внутри одной пары скобок, составляющих повторяемую группу спецификаций первого уровня, могут в свою очередь содержаться повторяемые группы спецификаций, которые в этом случае состав-

ляют повторяемые группы спецификаций второго уровня. Более высокие уровни повторяемых групп спецификаций не допускаются.

Элементы в списке ввода-вывода оператора, содержащего метку оператора задания формата, преобразуются из внешнего (относительно оперативной памяти) представления во внутреннее или из внутреннего представления во внешнее согласно кодам (спецификациям) в списке S. Символ / в списке означает начало новой записи при обработке списка ввода-вывода.

В общем случае спецификация может быть представлена в виде

$Cw.d$

где C — индекс спецификации, w и d — целые константы без знака.

Константа w указывает длину поля, отведенного для данного значения в записи, т. е. количество символов в значении; константа d — это количество знаков в дробной части числовых значений, т. е. количество десятичных цифр справа от точки внутри поля (за исключением спецификации G).

Все спецификации по своему назначению могут быть разбиты на две группы: к первой группе относятся спецификации, предназначенные для описания структуры значений вводимых или выводимых величин; ко второй группе относятся спецификации, предназначенные для управления вводом и выводом или для редактирования. Индексами спецификаций первой группы являются символы

F, E, D, I, L, G, Z, A,

которые называются индексами спецификаций *преобразования*. Индексы спецификаций второй группы — это символы

', H, X, T,

которые называются индексами *управляющих редакционных* спецификаций.

Индекс спецификации обязан содержаться в коде, константы w и d , а также точка в некоторых спецификациях отсутствуют.

Установление соответствия между переменными в списке ввода-вывода и спецификациями в операторе FORMAT осуществляется с учетом управляющих редакционных спецификаций согласно их следованию в списке слева направо.

Каждой спецификации преобразования, содержащейся в списке спецификаций, соответствует один элемент в списке ввода-вывода. Исключение составляет элемент комплексного типа, которому в списке спецификаций соответствуют два кода оператора FORMAT с индексами спецификаций F, E или D.

Для редакционных управляющих спецификаций не существует соответствующего элемента в списке ввода-вывода. Эти спецификации предназначены для управления вводом и выводом или для редактирования записей соответствующего файла,

Таким образом, выполнение операторов ввода или вывода вместе с соответствующим оператором FORMAT начинается с перебора спецификаций слева направо с учетом коэффициентов кратности и скобок. Если выбранная очередная спецификация относится к первой группе, то из списка ввода-вывода выбирается соответствующий элемент. Значение этого элемента (переменной) преобразуется согласно спецификации и передается либо в поле записи (при выводе), либо в оперативную память машины (при вводе). Если выбранная очередная спецификация относится ко второй группе, то выполняются необходимые (редакционные) действия без обращения к списку ввода-вывода.

Если список ввода-вывода будет исчерпан раньше, чем список спецификаций, то перебор спецификаций прекратится либо при обнаружении в списке спецификаций элемента первой группы, либо при исчерпании списка спецификаций. После этого выполнение оператора ввода или вывода будет закончено.

Если список спецификаций будет исчерпан раньше, чем список ввода-вывода, то перебор элементов списка спецификаций будет повторен, начиная с первого элемента последней группы спецификаций первого уровня, иными словами — с самой правой открывающей скобки первого уровня. Повторный перебор элементов списка спецификаций, не содержащего повторяемых групп спецификаций, начинается с первого элемента.

Переход на повторный перебор списка спецификаций сопровождается переходом к новой (следующей по порядку) записи файла. Переход к новой записи происходит и тогда, когда в процессе перебора элементов списка спецификаций встречается символ /.

Действие оператора задания формата в последующих примерах рассматривается совместно с операторами ввода или вывода. Поэтому каждая спецификация преобразования будет рассмотрена как при вводе, так и при выводе. Для большей наглядности примеров в дальнейшем предполагается, что ввод осуществляется с перфокарт, а выводится информация на АЦПУ, т. е. номер файла в примерах — это либо логический номер устройства ввода с перфокарт, либо логический номер печатающего устройства.

2. Спецификации преобразования. *F-спецификация* служит для описания формы представления значений переменных действительного типа и имеет вид

$N F w . d$
где коэффициент кратности N может отсутствовать.

В в о д. Преобразование, указываемое спецификацией F , несет число, содержащее w символов, на соответствующее поле оперативной памяти. При этом производится перекодировка числа из десятичного представления в двоичное. Например, оператор $N2 \text{ FORMAT } (5F8.3)$

указывает, что с одной перфокарты вводится 5 чисел, каждое из которых занимает на перфокарте поле, состоящее из 8 колонок. Если точка на перфокарте не набита или набита перед третьей цифрой, считая справа, то, согласно спецификации F, она будет представлена после ввода именно перед третьим символом, считая справа. Если точка набита на поле из w символов в любой другой позиции, то задание точки F-спецификацией независимо от значения d отменяется, и вводимое число содержит точку в указанной на перфокарте позиции. Например, число 1.37296E5 может быть введено при любом d , но всегда будет записано в память машины как 1.37296×10^5 .

Если в спецификации значение d отсутствует, то оно принимается равным нулю.

Индекс спецификации F указывает, в какой форме будет записано данное действительное число в памяти машины, и не накладывает никаких ограничений на форму представления этого числа в вводимой записи (на перфокарте), за исключением одного — с поля ввода считывается w символов. Например, оператор

N2 FORMAT (F5.3, F6.2, F7.4)

совместно с оператором ввода предполагает ввод с одной перфокарты трех чисел соответственно с полей длиной в 5, 6 и 7 символов. Если нет соответствия между указанным форматом и длиной полей на перфокарте, возможны ошибки при вводе.

С одной перфокарты считывается только такое количество символов, которое указано в коде для данной единицы записи, т. е. $N \times w$ или Σw символов (N — коэффициент кратности). Содержимое остальных колонок не воспринимается. Так, в предпоследнем примере считывается информация с $5 \times 8 = 40$ колонок, а в последнем примере — с $5 + 6 + 7 = 18$ колонок.

Если необходимо ввести массив данных, который не помещается на одной перфокарте, оператор FORMAT, написанный для одной записи, используется повторно, пока не будет введен весь массив. Например,

```
READ (3, 2) (X(I), I=1, 100)
2 FORMAT (16F5.1)
```

Здесь с одной перфокарты (если 3 — логический номер устройства ввода перфокарт) вводится 16 чисел, каждое из 5 символов. Всего оператор READ осуществляет ввод содержимого $100/16 + 1 = 6 + 1 = 7$ перфокарт. При этом с последней перфокарты будет прочитана только информация с $4 \times 5 = 20$ колонок, т. е. четыре числа, хотя набито может быть и больше.

Примеры ввода при F-спецификации:

— с поля, на котором записано число 763.9325, при спецификации F8.4 запишется результат в виде 763.9325; здесь в поле присутствуют дробная и целая части и точка;

— с поля 25793 при спецификации F5.7 запишется в память машины число .0025793, так как в поле нет дробной части и точки и введенное число преобразуется к виду 25793×10^{-7} ;

— с поля .43625 при спецификации F6.d (d — любое) запишется .43625, так как в поле содержится точка, и поэтому значение d не оказывает влияния; целой части в поле нет;

— с поля +245.15E—03 при спецификации F11.2 запишется результат .24515; здесь положение точки определяется заданием ее десятичным порядком на перфокарте.

В ы в о д. Поле, на которое выводится число, занимает w позиций в записи (например, в строке). Значение величины заносится в запись в виде десятичного числа со знаком с фиксированной точкой и d дробными и максимум $w-d-2$ целыми разрядами.

Если число положительное, то знак + не печатается. Если длина поля недостаточна для помещения туда выводимого числа, то оно заполняется символами * (звездочка). Если поле содержит больше позиций, чем требуется для записи числа, то избыточные позиции слева заполняются пробелами.

П р и м е р 1. Операторы

```
PRINT 5, X, Y, Z
```

```
5 FORMAT (F8.5)
```

осуществляют печать трех строк. В каждой строке содержится одно число: в первой — значение X, во второй — Y, в третьей — Z. Каждое число занимает поле из восьми позиций, из них одна позиция отводится под точку, одна — под знак, пять — под дробные десятичные разряды.

П р и м е р 2. Операторы

```
WRITE (1, 1) (X(I), I=1, 1000)
```

```
1 FORMAT (5F9.5)
```

выводят на печать (если 1 — номер АЦПУ) 200 строк, в каждой по 5 значений элементов массива X.

Максимальное число выводимых в запись символов ($N \times w$ или Σw) не должно превышать допустимое число символов записи для данного файла.

П р и м е р 3. Пусть значение переменной A равно +12.475; операторы

```
PRINT 2, A
```

```
2 FORMAT (F8.3)
```

вызовут печатание результата в виде `12.475`, в котором слева содержится два пробела, а знак + не печатается.

Пр и м е р 4. Значение переменной А равно —12.475; в результате действия операторов

PRINT 5, А

5 FORMAT (F6.3)

будет напечатано ***** (шесть звездочек), так как на поле длиной в шесть символов отсутствует место для знака минус.

Е-спецификация, как и *F-спецификация*, служит для ввода и вывода действительных значений. Код ее имеет вид

NEw.d

В в о д. Сохраняются все правила, перечисленные при рассмотрении *F-спецификации*. Число, записанное на поле (на перфокарте), не обязано содержать все части, важно лишь, чтобы заннмаемое числом поле не превосходило значения w в коде Е.

Примеры ввода при *Е-спецификации*:

— с поля +126.43Е—03 при спецификации Е11.2 в память запишется значение .12643; здесь в поле присутствуют все элементы записи числа;

— с поля —12.437629Е+1 при спецификации Е13.6 запишется значение, равное —124.37629;

— с поля 3698Е+04 при спецификации Е9.10 запишется значение .003698; здесь в поле нет дробной части, поэтому введенное число будет $3698 \times 10^{+4} \times 10^{-10}$ (d равно 10);

— с поля 127.256 при спецификации Е7.3 введется значение 127.256, хотя в поле и нет десятичного порядка;

— с поля —.0002736Е5 при спецификации Е11.7 введется значение, равное —27.36; здесь целая часть в поле состоит только из знака минус;

— если поле содержит только пробелы, то при любой спецификации Еw.d введется значение 0.;

— с поля 1Е1 при спецификации Е3.0 (d равно 0) введется значение 10. ($1 \times 10^{+1} \times 10^{-9}$);

— с поля, содержащего только десятичный порядок, независимо от его значения и значения d записывается всегда нуль; так, поле Е+07 при любой спецификации интерпретируется как 0.

В ы в о д. Поле в записи состоит из w позиций. Выводимое число представляется в виде

$\pm 0.\underbrace{aa \dots a}_{d \text{ позиций}} E \pm aa$

где знак + обычно заменяется пробелом, разряды мантиссы после точки содержат d позиций; буквой а обозначены позиции, занимаемые цифрами.

Длина поля должна быть достаточной, чтобы в нем уместились значащие цифры, знаки, точка и десятичный порядок, поэтому w

должно быть больше или равно $d+7$. При недостаточной длине поле заполняется звездочками. Если размер поля больше количества символов в выводимом числе, то левые избыточные позиции поля заполняются пробелами. Например, если выводятся два элемента массива X со значениями —16.245 и 1052.61 соответственно с помощью операторов

```
WRITE (1, 1) (X(I), I=1, 2)
1 FORMAT (F10.4)
```

то будут напечатаны две строки:

```
□□—16.2450
□ 1 052.6100
```

За неимением значащих цифр в младших разрядах после точки печатаются нули. Если бы был задан код F9.4, на месте второго числа (во второй строке) напечатались бы звездочки, так как отсутствует позиция для знака +, хотя он и не печатается.

Если организовать вывод операторами

```
WRITE (1, 3) (X(I), I=1, 2)
3 FORMAT (E 13.5)
```

то будет напечатано:

```
□—0.16245E+02
□□0.10526E+04
```

Хотя длины поля достаточно для вывода указанных значений (перед первым числом остается один, перед вторым — два пробела), во втором числе потерялся один младший разряд, так как значение d равно в коде 5, а число значащих цифр равно 6.

D-спецификация определяет преобразование, которое применяется в случае значений двойной точности. Код ее имеет вид

NDw.d

Правила использования D-спецификации полностью совпадают с правилами, рассмотренными для E-спецификации, за исключением того, что при выводе на поле вывода символ E заменяется символом D. Элементы списка ввода-вывода соответствующего оператора ввода или вывода должны иметь двойную точность.

I-спецификация служит для описания формы представления значений переменных целого типа и имеет вид

NIw

Эта спецификация используется только для ввода и вывода целых десятичных чисел, поэтому соответствующий элемент списка ввода-вывода должен быть целой величиной.

В в о д. В поле могут присутствовать цифры и пробелы, а также знаки + и —, которые должны предшествовать первой цифре.

Например, операторы

```
READ 5, K, L, M
5 FORMAT (I6, 2I10)
```

считывают с перфокарты содержимое трех полей: одного — длиной в 6 колонок и двух следующих — по 10 колонок в каждом.

В ы в о д. Целое число занимает поле из w позиций, причем одна позиция всегда отводится для знака. Если в поле вывода позиций больше, чем требуется для записи числа, избыточные позиции заполняются пробелами слева; если длина поля недостаточна для записи, все поле заполняется символами *. Например, если в результате вычислений получены значения I, J, M, равные соответственно +17356, -376, +2180, то после вывода операторами

```
PRINT 8, I, J, M
8 FORMAT (I10, I5, I5)
```

будет напечатано:

□	□	□	□	□	17356	□	-376	□	2180
↑					↑		↑		↑
1					10		15		20

позиции

L-спецификация используется при вводе и выводе логических значений. Она записывается в виде

NLw

В в о д. С перфокарты вводится содержимое поля из w символов. Если на этом поле первый, отличный от пробела символ есть буква T, то в область оперативной памяти, определяемой соответствующим элементом логического типа списка ввода, заносится значение .TRUE.; если же первый символ есть буква F или все w позиций заняты пробелами, в память заносится логическое значение .FALSE. . Таким образом, в поле логического значения при вводе должна присутствовать последовательность символов, в которой в качестве первого символа, отличного от пробела, записана буква T или F (или все символы на поле — пробелы).

В ы в о д. Поле вывода имеет длину w символов, где в последней позиции печатается символ T или F в зависимости от того, имеет ли соответствующий элемент (логического типа) списка вывода значение соответственно .TRUE. или .FALSE. . Например, в результате выполнения операторов

```
WRITE (1, 4) A, B, C, D, E
4 FORMAT (5L3)
```

будет напечатана строка

□□T□□T□□F□□T□□F

если логические переменные A, B, и D имели значение .TRUE. , а C, E — значение .FALSE. ,

G-спецификация используется при передаче целых, действительных, комплексных и логических значений. Эта спецификация записывается в виде

NGw.d

Поле, занимаемое значением, состоит из w позиций.

В в о д. Для целых и логических значений G-спецификация эквивалентна соответственно I- и L-спецификациям, т. е. вместо Iw и Lw может стоять в списке спецификаций конструкция Gw. При вводе действительных значений спецификация Gw.d эквивалентна коду Fw.d. Смысл константы d тот же, что и выше. В случае целых и логических значений наличие d в G-спецификации игнорируется.

В ы в о д. Если десятичный порядок r преобразуемого действительного числа, представленного в форме E с нормализованной мантиссой, меньше нуля или больше d, то значение выводится в форме E. Если же $0 \leq r \leq d$, то число выводится в форме F с r цифрами в целой части и d-r цифрами в дробной части числа; при этом четыре правые позиции заполняются пробелами. Таким образом, при выводе константа d указывает количество значащих цифр в поле.

Например, используя спецификацию G10.3, исходное число 0.555×10^{-1} будет выведено в виде `□0.555E-01`, число 0.555×10^0 имеет вид `□0.555□□□□`, число 0.555×10^3 — вид `□□555.□□□□`, а число 0.55×10^4 будет представлено как `□0.555E+04`.

П р и м е р 5. Предположим, что имеются операторы

WRITE (1, 3) J, X, Y, D, C, L

3 FORMAT (G4, 2G9.2, G10.7, 2G5.2, G2)

а целая переменная J получила значение 126, действительные переменные X и Y имеют соответственно значения 236.71 и 11.86, D — переменная двойной точности со значением 3.1415972, C — комплексное число (1.2, 7.3), L — логическая переменная со значением .FALSE. .

Результатом действия этих операторов будет печать строки

`□126□0.23E+03□□11.□□□□□3.141597□1.2□□7.3□□F`

Z-спецификация служит для задания шестнадцатеричных чисел, код ее имеет вид

NZw

В списке оператора ввода или вывода должны быть переменные целого или действительного типов или двойной точности. Считается, что значениями этих величин могут быть последовательности шестнадцатеричных цифр. Количество l шестнадцатеричных цифр в значении переменной определяется типом переменной из расчета по две цифры на один байт.

В в о д. Если $w > l$, то вводимое значение усекается слева, т. е. теряются левые (старшие) разряды. При $w < l$ справа добавляется соответствующее количество ($w - l$) шестнадцатеричных нулей.

В ы в о д. При $w > l$ в поле вывода слева добавляются пробелы. В случае $w < l$ усекаются левые цифры.

A-спецификация используется для задания текстовых значений элементам списка ввода-вывода. Ее код имеет вид

NAw

В в о д. Все w символов с поля ввода, в том числе пробелы, если они есть, вводятся в оперативную память. Эти символы образуют текстовую константу, значение которой получает соответствующая переменная в списке ввода.

Количество символов l , используемых при образовании текстовой константы, зависит от того, какова разрядность процессора. Один символ занимает 1 байт памяти. Следовательно, для ЭВМ БЭСМ-6 максимальное значение l равно 6, для ЕС 1066 — 8. Обозначим это число буквой $г$. Если $w \geq г$, то при вводе по спецификации A в память записится $г$ символов, расположенных в правой части поля. Если $w < г$, то в памяти образуется значение переменной, состоящее из $г$ введенных символов, за которыми (справа) расположены $г - w$ пробелов. Например, с помощью операторов

READ (3, 2) X, Y, Z

2 FORMAT (A4, A8, A12)

переменные X, Y, Z , если вводятся следующие последовательности символов:

TIPE PROGRAM FORTRAN TEXT

получат соответственно значения (при $г = 8$)

TIPE PROGRAM RAN TEXT

В ы в о д. Если $w > г$, то в записи будут содержаться $w - г$ пробелов и $г$ символов. В противном случае (при $w \leq г$) выводится w символов, расположенных в левой части слова в памяти.

Предполагая, что был осуществлен приведенный выше ввод текстовых значений переменных X, Y, Z , операторы

PRINT 3, X, Y, Z

3 FORMAT (A5, A10, A12)

выведут на печать следующее:

TIPE PROGRAM RAN TEXT
X Y Z

Присвоение переменным и элементам массивов текстовых значений может быть выполнено, таким образом, операторами ввода

с использованием в соответствующем операторе FORMAT A-спецификаций.

В языке Фортран IV не предусмотрены специальные операторы для обработки текстовой информации, но переменные или элементы массива, которым присвоены текстовые значения, могут быть объединены в логическое выражение знаками операций отношения .EQ. и .NE., а также выступать в качестве правых частей операторов присваивания. Например, при соблюдении условий предыдущего примера операторы

```
IF (X.EQ.Y) GO TO 13
```

```
GO TO 27
```

```
и IF (X.NE.Y) GO TO 27
```

осуществляют одну и ту же передачу управления на оператор с меткой 27, а после выполнения оператора $Z=Y$ переменная Z получит значение текстовой константы PROGRAM_1.

В одном операторе FORMAT список спецификаций может содержать коды различных видов. Например, возможен такой оператор:

```
N2 FORMAT (A5, F4.2, E8.1, 2I7, D19.12, L2)
```

Единственное требование, которое при этом должно выполняться, заключается в том, что каждый код оператора FORMAT обязан соответствовать типу «своей» переменной в списке оператора ввода или вывода.

Примеры использования оператора FORMAT с оператором ввода.

Пр и м е р 6. Оператор

```
N2 FORMAT (5F10.2/4F9.2)
```

указывает, что вводится перфокарта с пятью числами на ней (каждое поле занимает 10 колонок) в форме F10.2, после которой вводится другая перфокарта, с которой считываются четыре числа в форме F9.2. Если есть еще третья перфокарта, то информация с нее будет считана в форме F10.2 и т. д.

Пр и м е р 7. Оператор

```
N2 FORMAT (I5/(4F5.3))
```

указывает, что с первой перфокарты вводится одно число в форме I5, а со всех остальных, сколько бы перфокарт ни было, считывается по четыре числа в форме F5.3.

Таким образом, появление символа / в списке спецификаций означает переход к другой записи. Символ / может появляться не только между элементами в списке спецификаций, но и в конце списка. Допустимо написание подряд нескольких символов /. Два символа // означают при вводе пропуск одной перфокарты, а при выводе — одну строку пробелов и т. д.

N2 FORMAT (2(F3.2, I6))

равносилен оператору

N2 FORMAT (F3.2, I6, F3.2, I6)

который вводит с одной перфокарты четыре числа.

В последнем примере реализована возможность образования повторяемой группы спецификаций. Повторяемые группы спецификаций, как уже отмечалось, могут быть образованы глубиной до двух уровней. Например:

N2 FORMAT (I5,(I4, F4.1, 4 (E8.3, F6.2)))

Скобки, помеченные нулем, соответствуют нулевому уровню, единицей — первому уровню, двойкой — второму уровню. Если в списке ввода-вывода содержится достаточно много элементов, то перебор спецификаций в списке приведенного оператора **FORMAT** будет повторен с открывающей скобки, помеченной единицей, т. е. со спецификации 14.

Использование операторов **FORMAT** из примеров 6, 7 и 8 с оператором вывода соответственно указывает:

— печать двух строк, в первой из которых располагаются пять чисел в форме F10.2, во второй — четыре числа в форме F9.2;

— печать в первой строке числа в форме I5, а в остальных — по четыре числа в форме F5.3;

— печать строки из четырех чисел.

3. Масштабный коэффициент. Спецификациям F, E, и D может предшествовать масштабный коэффициент в форме nP , где n — целая константа со знаком. Масштабный коэффициент употребляется со спецификацией F при вводе и выводе и со спецификациями E и D — только при выводе.

Использование масштабного коэффициента n^P с F-спецификацией означает:

— при вводе вводимое число умножается на 10^{-n} ; например, если число 123.456 вводится при спецификации 2PF7.3, то в память заносится значение 123.456×10^{-3} , т. е. 1.23456;

— при выводе значение из оперативной памяти умножается на 10^n и результат выдается согласно спецификации Fw.d; например, если в памяти хранилось число 1.23456, то при спецификации — 1PF9.4 выводится значение 123456, а при спецификации 3PF10.4 результатом вывода будет значение 12345600.

С Е- или D-спецификациями использование масштабного коэффициента при выводе означает перестановку точки в мантиссе и соответствующее изменение десятичного порядка числа на $-n$ единиц. Результат заполняет поле справа налево, и избыточные левые позиции поля заполняются пробелами.

Таким образом, использование масштабного коэффициента с F-спецификацией вызывает изменение вводимого или выводимого значения (после вывода в памяти сохранится неизмененное значение), в то время как при E- и D-спецификациях выводимое значение не изменяется. Например, число 0.23456789 при спецификации —1PE20.2 представляется на выходе как

□□□□□□□□□□□□□□0.02E+01

а при спецификации 5PE20.2 как

□□□□□□□□□□23456.78E—05

в то время как в случае обычной (без масштабного коэффициента) спецификации E20.2 это число представляется в виде

□□□□□□□□□□□□□□0.23E+00

Если масштабный коэффициент не указан, то он предполагается равным нулю. Однако, будучи указан один раз, он сохраняет силу для всех следующих за ним спецификаций F, E и D в данном операторе FORMAT. Чтобы ликвидировать его влияние на последующие спецификации, одной из следующих спецификаций F, E или D должен предшествовать масштабный коэффициент с $n=0$, т. е. 1P. Масштабные коэффициенты при спецификациях E и D при вводе игнорируются.

Масштабные коэффициенты используются в двух наиболее характерных случаях: для корректировки положения разрядов числа относительно точки в полях спецификации F и для печати необходимого количества цифр левее точки в полях спецификаций E и D.

4. Управляющие редакционные спецификации. '-спецификация (апостроф-спецификация или спецификация типа литерал) используется при передаче информации в виде текста, помещаемого в операторе задания формата между апострофами. В тексте могут встретиться любые символы (в том числе пробелы и апострофы). Первый апостроф обозначает начало поля для текста, последний — конец этого поля. Для того чтобы включить в текст символ апостроф, его надо написать дважды. Например, оператор FORMAT, содержащий '-спецификацию, может выглядеть так:

35 FORMAT ('□X=', F10.3, 'AB"□C') .

Оператор ввода или вывода, содержащий метку оператора FORMAT со списком спецификаций, в который входит текст, заключенный в апострофы, имеет список ввода-вывода без элементов, соответствующих '-спецификациям. Так, с приведенным оператором FORMAT может использоваться, например, такой оператор вывода:

WRITE (3, 35) X

В в о д. Из вводимой записи считываются символы до тех пор, пока в тексте не встретится апостроф или не будет заполнено все

поле в операторе FORMAT новым текстом. Первоначально, т. е. при включении в программу оператора FORMAT с '-спецификацией, достаточно поле между апострофами заполнить пробелами. Например, ввод заголовка FORTRAN PROGRAM, набитого на перфокарте, в оператор FORMAT может быть осуществлен следующим способом:

```
READ 1
1 FORMAT ('XXXXXXXXXXXXXXXXXXXX')
```

В дальнейшем оператор с меткой 1 в программе имеет вид

```
1 FORMAT ('FORTRAN PROGRAM')
```

В ы в о д. В запись переносится содержимое списка спецификаций между апострофами. Результатом действия приведенных в начале п. 4 операторов будет печать строки

```
 X=---12.345AB'C
```

если переменная X имела значение —12.345.

Размер выводимого на печать текста не должен превышать длину строки. Если отдельный апостроф случайно оказался в выводимой информации, он ограничит поле вывода.

N-спецификация имеет вид

```
wN
```

(целая константа w пишется перед индексом N) и, так же как '-спецификация, используется при вводе или выводе текстовой информации, содержащейся в операторе FORMAT, однако в коде wN указывается количество символов текста, тогда как в апостроф-спецификации это не требовалось.

В в о д. В поле оператора FORMAT из w позиций, следующих за индексом N, заносится весь текст, содержащийся в одной записи. Например, если на перфокарте набито

```
TEST AND TEXT
```

то операторы

```
READ 8
8 FORMAT (14NXXXXXXXXXXXXXXXX)
```

преобразуют оператор с меткой 8 к виду

```
8 FORMAT (14NTEST AND TEXT)
```

В ы в о д. Информация в количестве w символов, которая содержится в операторе FORMAT после индекса спецификации N, выводится на печатающее устройство. Так, использование приведенного оператора FORMAT с меткой 8, например, с оператором

```
PRINT 8
```

вызовет печать строки

```
TEST□AND□TEXT□
```

Операторы

```
PRINT 4, Q
```

```
4 FORMAT (6HINDEX =, F5.2)
```

выведут на печать строку

```
INDEX=□2.35
```

если переменная Q имела значение 2.35.

При использовании H-спецификации список ввода-вывода отсутствует. В последнем примере элемент Q списка относится к F-спецификации. Константа перед индексом H ставится и в случае $w=1$.

X-спецификация записывается в виде

```
wX
```

(константа w пишется перед индексом спецификации) и используется при вводе для пропуска w символов, а при выводе для включения в запись w пробелов. Например, оператор FORMAT в виде

```
11 FORMAT (10X, I10, 20X, 4I10)
```

совместно с оператором ввода

```
READ 11, I, J, K, L, M
```

указывает, что значение переменной I считывается с поля, занимающего позиции с 11 до 20, а значения J, K, L, M — начиная с позиции 41, т. е. три поля (по 10 колонок) на перфокарте не воспринимаются, а в результате выполнения операторов

```
PRINT 9, X, Y
```

```
9 FORMAT (F10.2, 4X, F8.3)
```

будут напечатаны значения переменных X и Y с интервалом между ними, равным четырем пробелам. Очевидно, оператор с меткой 9 эквивалентен оператору

```
9 FORMAT (F10.2, '□□□□', F8.3)
```

T-спецификация используется для задания позиции в записи, начиная с которой производится ввод или вывод, если есть необходимость считывания информации (при вводе) или занесения результата (при выводе) не с начала записи. Эта спецификация имеет вид

```
Tw
```

В одном операторе FORMAT T-спецификация может употребляться несколько раз в сочетании с любыми другими спецификациями. При этом нет необходимости в последовательном указании кодов в операторе FORMAT с возрастающими значениями констант w.

В в о д. Константа w равна номеру первой позиции в поле записи. Так, операторы

```
READ 1, S
1 FORMAT (T21, F10.2)
```

осуществляют ввод для переменной S значения, расположенного на перфокарте, начиная с колонки 21.

В ы в о д. Значение константы w совпадает с номером первой позиции поля, на которое переписывается информация, если только внешним носителем информации не является печатающее устройство. В последнем случае номер первой занятой позиции в напечатанной строке равен w-1. Например, при выводе

```
PRINT 2, X
2 FORMAT (T33, F8.2, T31, 'X=', T61, 'PRIME')
```

если полученное значение X равно, скажем, 135.41, будет напечатана строка

```
X =    135.41    PRIME
  ↑   ↑           ↑
 30  32           60  позиции
```

Не следует допускать перекрытия полей в записи несколькими спецификациями. Так, ошибочным будет оператор

```
N2 FORMAT (T12, I15, T22, I6)
```

который при выводе должен отвести под число в форме I15 позиции с 11 по 25, а под число в форме I6 — позиции с 21 по 26.

Управляющие редакционные спецификации могут быть использованы в списке кодов оператора FORMAT совместно со спецификациями преобразования в произвольной последовательности.

5. Управление печатью. Если информация выводится на печатающее устройство, то первый символ записи не печатается, а интерпретируется как *управляющий* и определяет величину интервала перед печатью строки. Как правило, этот управляющий символ задается H-спецификацией или апостроф-спецификацией, хотя, вообще говоря, допустим любой другой способ задания.

Управляющими являются символы

```
   0 + 1
```

которые соответственно вызывают: перевод одной строки перед печатью, перевод двух строк, печать без перевода строк, прогон бу маги до первой строки следующей страницы. Например, операторы

```
PRINT 5, X
5 FORMAT (3X, F8.3)
```

вызовут печатание на следующей строке значения переменной X, начиная с третьей позиции строки. Переход к следующей строке обеспечивается тем, что первым символом печатаемой строки является пробел.

Упражнения

1. Пятая запись файла с номером 3 имеет вид

0.16E-010.21E+030.61E-010.22E+030.80E+05
0.12E+030.18E-040.39E+030.71E+010.93E+03
0.57E-010.21E+02

Определить значения элементов массива E, которые они получают в результате выполнения программы

```
REAL E(10) /-19.13, 2.01, -2.86, 3.12, 3*10., 3* 4.12/  
DEFINE FILE 3 (5, 800, L, L)  
3 FORMAT (3E8.2)  
READ (3'5, 3) (E(I), I=3, 10, 3)  
STOP  
END
```

2. Указать, какой вид будет иметь восьмая запись файла 18 в результате выполнения программы

```
LOGICAL P, Q  
DEFINE FILE 18(20, 100, E, L)  
6 FORMAT (2G11.3, G9.1)  
K=18  
P=TRUE.  
Q=.NOT. P  
WRITE (18'8, 6) K, P, Q  
STOP  
END
```

3. Определить последовательность символов, которая будет помещена в двадцатую запись файла с номером 13 в результате выполнения программы

```
DEFINE FILE 13(3, 150, E, K)  
7 FORMAT (3A4)  
READ (13'5, 7) A, B, C  
WRITE (13'20, 7) A, B, C  
STOP  
END
```

если пятая запись файла с номером 13 имеет вид: OTS┐END┐
FINAL.

4. Составить программу умножения векторов A(100) и B(100) по алгоритму $C(I)=A(I)*B(I)$, $I=1, 2, \dots, 100$, где C(100) — массив результата. Результат напечатать в виде таблицы 10×10 значений элементов массива C, представленных по формату E12.5. Таблицу надписать текстом: $C=A*B$.

§ 23. Классификация операторов и наименований

Рассмотренные выше операторы делятся на пять типов (см. также § 3): описательные операторы, операторы присваивания, операторы управления, операторы ввода и вывода, подпрограммы.

Перечислим здесь операторы каждого из типов.

Операторы COMMON, COMPLEX, DATA, DEFINE FILE, DIMENSION, ENTRY, EQUIVALENCE, EXTERNAL, FORMAT, IMPLICIT, INTEGER, LOGICAL, NAMELIST, REAL и оператор определения оператор-функции относятся к первому типу; операторы ASSIGN, арифметический и логический операторы присваивания — ко второму типу; операторы BACKSPACE, CALL, CONTINUE, DO, END, END FILE, FIND, GO TO, IF, PAUSE, RETURN, REWIND, STOP являются операторами управления; операторы PRINT, PUNCH, READ, WRITE относятся к четвертому типу; операторы BLOCK DATA, FUNCTION, SUBROUTINE — к пятому.

Наименования переменных, массивов и оператор-функций, а также метки операторов и формальные параметры локализованы в той программной единице, в которой они появляются. Это значит, что всякое обращение к ним разрешается только из данного сегмента программы.

Наименования подпрограмм-функций и подпрограмм, наименования всех входов, а также наименования общих блоков являются общими (нелокализованными) для всех сегментов данной программы, и к ним можно обращаться из любой программной единицы в соответствии с установленными правилами (по идентификатору подпрограммы-функции, имени входа или идентификатору стандартной функции, с помощью операторов CALL или COMMON).

Порядок расположения операторов в программной единице следующий:

- оператор определения подпрограммы BLOCK DATA, FUNCTION или SUBROUTINE (используется только в подпрограмме);
- оператор IMPLICIT;
- описательные операторы, отличные от оператора IMPLICIT;
- определения оператор-функций;
- исполняемые операторы;
- оператор END.

Операторы DATA, FORMAT, ENTRY, RETURN могут находиться в любом месте программной единицы. Оператор DATA должен следовать после оператора IMPLICIT за описательными операторами, объявляющими величины, используемые в операторе DATA. Оператор ENTRY не должен находиться в цикле, а оператор RETURN среди описательных операторов.

Оператор DEFINE FILE должен логически предшествовать операторам ввода и вывода прямого доступа, а оператор NAMELIST — операторам ввода и вывода, которые используют величины, объявленные в операторе NAMELIST.

§ 24. Примеры программ

1. Вычисление интеграла методом Симпсона. Вычислить интеграл

$$Q(p) = \int_a^b \frac{p^v x e^x dx}{(p + e^x)^{v+1}}$$

как функцию параметра p , интервал по которому задан; задано также значение v . Формула Симпсона имеет вид

$$\int_a^b f(x) dx = \frac{h}{3} [f(a) + 4f(a+h) + 2f(a+2h) + 4f(a+3h) + \dots + 2f(b-2h) + 4f(b-h) + f(b)],$$

где $h = (b-a)/n$, n — четное количество интервалов, $f(x)$ — подынтегральное выражение.

Вычисление подынтегрального выражения может быть осуществлено с помощью оператор-функции

$$F(X) = X * \text{EXP}(X) / (P + \text{EXP}(X)) ** (NU + 1)$$

где X — формальный параметр.

Значения оператор-функции, которые умножаются на множитель 4, получаются обращением к F с фактическими параметрами, равными $A+H$, $(A+H)+2H$, ..., а те значения подынтегрального выражения, которые умножаются на множитель 2, получаются обращением к F с фактическими параметрами $(A+H)+H$, $((A+H)+H)+2H$, ... Суммирование указанных двух групп значений подынтегральной функции выполняется в цикле с помощью оператора IF.

Вычисление интеграла Q в интервале значений параметра P от P_0 до P_K осуществляется во внешнем цикле.

Программа имеет вид

```

C      SIMPSON
      REAL NU
      F(X) = X * EXP(X) / (P + EXP(X)) ** (NU + 1)
5      FORMAT (5F11.2, 2I5)
      READ 5, P0, PH, PK, A, B, NU, N
      H = (B - A) / N
      P = P0
10     S4 = 0
      S2 = 0

```

```

      I=1
      X=A+H
15    S4=S4+F(X)
      S2=S2+F(X+H)
      IF (I-(N-3)) 20, 25, 25
20    I=I+2
      X=X+2*H
      GO TO 15
25    Q=P**NU+H/3*(F(A)+F(B)+
      * 4*F(B-H)+4*S4+2*S2)
      PRINT 30, Q, P
30    FORMAT (2E15.5)
      IF (P-PK) 35, 40, 40
35    P=P+PH
      GO TO 10
40    STOP
      END

```

2. Решение системы обыкновенных дифференциальных уравнений. Требуется написать программу для решения уравнения второго порядка $y'' - 8y' + x^2y = e^x$ с начальными условиями $y(0) = y'(0) = 0$ методом Эйлера на отрезке $[0, 1]$ с шагом $h=0.1$.

Записав уравнение в виде системы

$$\begin{aligned}
 y_1' &= y_2, \\
 y_2' &= 8y_2 - x^2y_1 + e^x,
 \end{aligned}$$

воспользуемся подпрограммой для одного шага интегрирования по методу Эйлера с выбранным шагом H в виде

```

      SUBROUTINE E (F, Y, Y0, X, X0, N, H)
      DIMENSION F(N), Y(N), Y0(N)
      DO 5 I=1, N
      Y (I)=Y0 (I)+F(I)*H
5    X=X0+H
      RETURN
      END

```

В этой подпрограмме в число формальных параметров включены следующие идентификаторы: $Y0$ — массива начальных данных, Y — массива результатов интегрирования, F — массива значений правых частей, $X0$ — начального значения аргумента X , N — числа уравнений в системе, H — шага интегрирования.

Подпрограмма вычисления правых частей для уравнений системы будет выглядеть так:

```

      SUBROUTINE P
      DIMENSION F(2), Y(2), Y0(2)
      COMMON F, Y, Y0

```

```

F(1)=Y0(2)
F(2)=8*Y0(2)-X0**2*Y0(1)+EXP(X0)
RETURN
END

```

И, наконец, еще одна программная единица — основная программа — представляется в виде

```

C      EULER
      COMMON F(2), Y(2), Y0(2)
      X=0
      DO 10 I=1, 2
10     Y(I)=0
      PRINT 15, X, Y(1), Y(2)
15     FORMAT (3E15.5)
      DO 25 K=1, 10
      X0=X
      DO 20 J=1, 2
20     Y0(J)=Y(J)
      CALL P
      CALL E(F, X0, Y0, X, Y, 2, 0.1)
      PRINT 15, X, Y(1), Y(2)
25     CONTINUE
      STOP
      END

```

Окончательная программа составляется из выписанных сегментов, например, в такой последовательности:

```

EULER
SUBROUTINE P
SUBROUTINE E

```

Упражнения

1. Написать подпрограмму для нахождения корня произвольного алгебраического уравнения $f(x)=0$ методом деления отрезка пополам, если известно, что корень существует, единственный и содержится в отрезке $[x_1, x_2]$.

2. Найти на отрезке $[0.5, 2.0]$ корень уравнения $(4+x^3)(e^x - e^{-x})=18$, используя подпрограмму из предыдущего упражнения.

Глава III. БЕЙСИК

§ 1. Основные конструкции языка

1. **Режим работы.** Язык программирования Бейсик — самый распространенный язык диалога человека с машиной. Диалоговый характер Бейсика определяет его конструктивные особенности, позволяющие проводить все операции по разработке, отладке и выполнению программы за клавиатурой дисплея. При работе с языком Бейсик возможны два *режима работы*: режим *немедленной обработки* или прямого общения и *программируемый режим* или режим отсроченной обработки (режим непрямого общения). В первом случае инструкции, набираемые на клавиатуре, называются *командами* или *директивами* и каждая такая инструкция, введенная в ЭВМ, выполняется сразу. В программируемом режиме инструкции обычно называются *операторами*, хотя и большинство директив режима немедленной обработки можно использовать в режиме отсроченной обработки и наоборот. Однако есть такие команды, которые работают только в режиме немедленной обработки, и такие операторы, которые выполняются только в программируемом режиме. Характерной чертой программируемого режима является наличие у оператора *номера строки (метки)*, и все операторы выполняются в порядке возрастания их номеров.

Таким образом, основными конструктивными элементами программы являются команды (директивы) и операторы. Из операторов могут быть образованы *подпрограммы* — участки программы, содержащие операторы для выполнения некоторой вполне определенной процедуры. Обращение к таким участкам осуществляется, как правило, оператором вызова подпрограмм, хотя операторы в них могут быть выполнены и в естественной последовательности. Конструктивными элементами, используемыми в операторах, являются *выражения*, которые в зависимости от входящих операндов в виде данных могут быть числовыми или строковыми.

2. **Алфавит.** Для записи любых конструктивных элементов языка используется вполне определенный набор изобразительных средств — *алфавит* языка. В зависимости от версии языка и возможностей клавиатуры терминала (дисплея) алфавит Бейсика мо-

жет иметь различный вид, однако среди символов алфавита обязательно будут буквы, цифры и специальные знаки. Чаще всего — это буквы латинского алфавита

A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

арабские цифры

1 2 3 4 5 6 7 8 9 0

специальные знаки — символы

+	(плюс),	{	(двоеточие),
—	(минус),		(апостроф),
*	(звездочка),	"	(двойная кавычка),
{	(наклонная черта),	&	(амперсанд),
\	(обратная наклонная черта),	@	(коммерческое at),
^	(стрелочка),	?	(вопросительный знак),
=	(равно),	((открывающая круглая скобка),
<	(меньше),)	(закрывающая круглая скобка),
>	(больше),	[(открывающая квадратная скобка),
%	(процент),]	(закрывающая квадратная скобка),
!	(восклицательный знак),	—	(подчеркивание),
#	(знак номера),		(пробел).
\$	(знак доллара),		
.	(точка),		
,	(запятая),		
;	(точка с запятой),		

Символ пробел, не имеющий начертания, иногда обозначают как \square , а вместо знака доллара используется символ \square . Стрелочка, используемая для обозначения операции возведения в степень, может быть заменена на символ \uparrow , \uparrow или **, а в число букв добавлены строчные буквы латинского алфавита или прописные и строчные буквы русского алфавита. В некоторых версиях часть приведенных здесь специальных знаков может отсутствовать вообще.

3. Конструктивные элементы. С помощью символов алфавита строятся все названные выше конструктивные элементы языка и любые объекты, среди которых знаки операций отношения, знаки логических операций, данные (константы и переменные), наименования (ключевые слова) команд, операторов и встроенных числовых и строковых функций. Перечислим здесь ключевые слова, используемые для обозначения наиболее часто встречающихся команд и операторов. Список функций языка Бейсик будет приведен в приложении II.

Основные ключевые слова, используемые в качестве команд: AUTO (автоматический), DELETE (удаление), EDIT (редактирование), LIST (список), MERGE (соединить), NAME (имя), NEW (новый), OLD (старый), RUN (прогнать), SAVE (сохранить), SYSTEM (система).

Ключевые слова, чаще всего используемые в качестве операторов: CHAIN (цепь), CLOSE (закрыть), COLOR (цвет), COMMON (общий), DATA (данные), DEF (определить), DIM (размерность), END (конец), FOR (для), GOSUB (перейти к подпрограмме), GOTO (перейти к), IF (если), INPUT (ввод), LET (пусть), LINE (линия), LOCATE (размещать), MAT (матрица), NEXT (следующий), ON (на), OPEN (открыть), PRINT (печатать), READ (читать), REM (примечание), RESTORE (восстановить), RETURN (возврат), SCREEN (экран), STOP (стоп), WEND (конец цикла), WHILE (до тех пор пока), WIDTH (ширина).

§ 2. Данные

1. **Типы данных.** *Данные* подразделяются на два вида: константы и переменные. С каждым видом данных связано понятие типа. Тип данного определяет его внутреннее представление в памяти ЭВМ. Различают следующие типы данных: строковые; целые — шестнадцатеричные, восьмеричные и десятичные; десятичные с дробной частью — одинарной и двойной точности.

2. **Константы.** *Константы* — значения, неизменяемые во время выполнения программы. Константы подразделяются на строковые (символьные или текстовые) константы (строки), соответствующие строковому типу, и числовые константы.

Строка представляет собой последовательность символов, непосредственно следующих друг за другом. Обычно строка окаймляется с обеих сторон кавычками. Например:

"ВЕРСИЯ BASIC—PLUS"

Строковая константа может содержать любой из символов алфавита, однако длина строки не должна превышать 255 символов. Самая короткая строка не содержит никаких символов и называется пустой или нулевой строкой или строкой нулевой длины.

Числовые константы могут быть представлены пятью различными способами в соответствии с указанными выше типами данных, каждый из которых имеет свое определенное назначение. Десятичные числа имеют три типа представления: в виде целых констант, констант обычной и двойной точности.

Целый тип представляет собой запись числа как последовательности цифр с предшествующим знаком + или —, при этом у положительного числа знак + может быть опущен. Целое число может

лежать в диапазоне от -32768 до $+32767$ и занимает в памяти два байта. Если значение числа выходит за пределы указанного интервала, то оно перестает относиться к целому типу.

В ряде версий Бейсика (например, в Бейсик-плюс) признак целочисленности обозначается символом % после последней цифры числа. Например,

5 % 100 % 13 % 53421 %

Константы обычной точности имеют две формы представления — с фиксированной и с плавающей точкой. Первая из форм представления имеет обычный вид положительного или отрицательного числа с дробной частью с количеством цифр не более семи, в записи которого вместо десятичной запятой используется точка. Например:

123. .321 +1234567 —55.13 —1.23456

В экспоненциальной форме представления константа обычной точности имеет вид последовательности из мантиссы, являющейся целым или дробным числом, буквы E и порядка. Порядок представляется собой однозначное или двузначное целое число, указывающее на сколько разрядов вправо (при положительном знаке) или влево (при отрицательном знаке) нужно сместить точку в мантиссе, чтобы получить фактическое значение представляемой величины. Перед мантиссой в случае отрицательного числа указывается знак минус и необязательный знак плюс в случае положительного значения. Например:

—1.234567E+05 53E—1 .345E3 —.26E+10

Для хранения в памяти чисел обычной точности используется четыре байта, в записи таких чисел допускается не более семи цифр, при выводе также печатается семь цифр, из которых точными являются только первые шесть.

Числовые константы двойной точности представляются как десятичные числа, в записи которых используется до шестнадцати значащих цифр и занимающих в памяти ЭВМ восемь байтов. Форма представления с фиксированной точкой кроме количества значащих цифр ничем не отличается от соответствующей формы представления числа обычной точности, а в экспоненциальной форме используется вместо буквы E буква D. Примеры числовых констант двойной точности:

1234567890987654 —12345678.9D+13

Диапазон представимых в памяти ЭВМ констант с плавающей точкой зависит от особенностей ЭВМ. Чаще всего наибольшее допустимое число не должно превышать 10^{38} , а наименьшее отличное от нуля число имеет порядок 10^{-38} .

Константы обычной точности в форме представления с фиксированной точкой могут иметь завершающий символ !, а константы двойной точности — завершающий символ #. Например:

454.2! 1.23456# 13489# 15!

Любое число, содержащее более семи значащих цифр, автоматически рассматривается как значение двойной точности, если только в конце числа не стоит восклицательный знак. Для того чтобы некоторое число представлялось в машине с двойной точностью, надо поставить символ # за последней цифрой этого числа. Если в числе менее семи цифр, то при переводе его во внутренний код ЭВМ после последней цифры автоматически добавляются нули.

Восьмеричная константа представляет собой последовательность не более чем из шести восьмеричных цифр (1, 2, 3, 4, 5, 6, 7, 0) с префиксом &O и возможно знаком + или —. Например:

— &O777777 &O25 &O1001

Допускается (версия IBM) запись восьмеричного числа без символа O после амперсанда.

Для записи чисел в шестнадцатеричной системе наряду с десятичными цифрами от 0 до 9 используют буквы A, B, C, D, E и F. Для задания шестнадцатеричного числа перед первой его цифрой вводится префикс &H, а само число не должно содержать более четырех разрядов. Например:

&H7FFF —&H7EFE &HDA0 —&H67

В памяти ЭВМ все числовые константы представляются в системе счисления с основанием 2. Такие числа записываются в виде последовательностей нулей и единиц и называются двоичными. В некоторых версиях Бейсика двоичные целые (положительные и отрицательные) числа можно использовать в программе, указав перед последовательностью из нулей и единиц префикс &B, например:

&B101110 —&B10001

3. Переменные. Каждому типу данных — строковому, целому, обычной и двойной точности — соответствует определенный тип переменной. *Переменная* — это величина, к которой обращаются по имени переменной (идентификатору) и которая может принимать различные значения. *Идентификатор* выбирается как последовательность символов, в которой первым символом является буква, а последующие символы могут быть любыми буквами, цифрами или точками. Имя переменной может иметь любую длину при условии, что оно уместается в одной строке программы; в то же время транслятор учитывает только первые 40 символов имени и игнорирует все остальные. В качестве имен переменных запрещено ис-

пользовать ключевые слова, употребляемые для идентификации команд, операторов и функций.

Тип переменной может быть определен последним символом идентификатора. В качестве таких символов используются: % — для целого типа, ! — для обычной точности, ‡ — для двойной точности, а \$ для строкового типа. Например, идентификаторы

N% Y! D‡ A\$

являются именами переменных целого типа, обычной и двойной точности и строкового типа соответственно.

Имена, отличающиеся друг от друга лишь последним, определяющим тип, символом, воспринимаются транслятором как имена различных переменных. Так, X%, X!, X‡, X\$ — идентификаторы четырех разных переменных. Если имя переменной не оканчивается одним из указанных выше специальных символов, определяющих ее тип, то по умолчанию считается, что такой идентификатор обозначает числовую переменную обычной точности. Этот принцип нарушается в случае применения оператора DEF, пользуясь которым можно задавать тип переменной одной начальной буквой ее имени без классифицирующих суффиксов.

Каждой переменной, впервые встречающейся в программе, транслятором отводится определенная область памяти для хранения ее значений в зависимости от типа: целой переменной выделяется 2 байта, переменной обычной точности 4 байта, двойной точности 8 байт, строковой переменной 2 байта плюс количество символов значения переменной.

Переменная, в каждый заданный момент принимающая лишь одно значение, называется *простой* или *скалярной* переменной. Приведенные выше примеры идентификаторов — это имена простых переменных. В то же время одним идентификатором может быть обозначено некоторое упорядоченное множество величин, объединенных в *массив*. Каждая отдельная величина как значение элемента массива идентифицируется с помощью числовых индексов. Элемент массива — это переменная с индексами. Количество индексов у переменной с индексами определяет размерность массива. Так, один индекс задает некоторую одномерную строку (или столбец) значений — вектор, два индекса идентифицируют двумерный массив — матрицу, три индекса формируют куб, а для больших размерностей трудно найти простой геометрический образ. Максимальное число индексов для одного массива — 255.

Индексы у переменной с индексами указываются после идентификатора в круглых скобках и разделяются запятыми. Правила образования идентификаторов массивов те же, что и для простых переменных. В одной и той же программе допустимы одинаковые идентификаторы массивов и простых переменных: в этом случае

они считаются никак не связанными друг с другом, как, если бы имели различные имена, однако каждый массив должен иметь свое имя. Все элементы одного массива имеют один и тот же тип — целый, обычной или двойной точности или строковый. Примеры обозначений переменных с индексами:

A\$(1, 5) K#(7) D(3, 3, 3) N%(2, 4)

Отметим, что во многих версиях Бейсика в качестве идентификатора допускается лишь латинская буква или буква с последующей за ней цифрой и разрешены лишь одномерные и двумерные массивы.

Упражнения

1. Среди приведенных ниже записей определить те, которые являются: а) целыми числами; б) числами обычной точности; в) числами, имеющими двойную точность; г) равными по величине; д) записями, не допускаемыми правилами Бейсика:

378	50000	0.100E03	+500.0E+2.0
100	—E—6	.100D—30	777777777
+8%	387.0	#0.11241	123456.7%
15!	0.387	.25E12#	0.000008#
—2#	3.14!	11111%	1D—37!

2. Определить тип следующих констант:

237	.1E—16	"INTEGER"	&B111000
125	—2778!	&01234567	—&H01111
"4"	"BASF"	—187.331#	"&B1000"
2.6	1989%	&H1234567	&0477774
—7#	1938#	9876.1D17	"123456"

3. Среди приведенных ниже записей выбрать последовательно-сти символов, являющиеся идентификаторами:

ARRAY!	APOLLO-13	FORTRAN\$
INTEGER	NAME.BAS	LONGITUDE.110DEG
2A	.DATA	ARCTANX
ALLA	DATA.#	integer%
DOLLAR\$	RATE100%	ИДЕНТИФИКАТОР
PRINT	123CENT	"NAMEOFINDEX"

4. Указать, какие из приведенных записей можно рассматривать как обозначения переменных:

A3\$	A[3]	PI(3.14)	A(3)%	LENA
A(3)	PI3.14	PI(PI)	PI#(3)	LANA

5. Определить, допускаются ли в Бейсике следующие конструкции; в случае положительного ответа указать класс соответст-

вующих объектов:

STRING?	"A,B"	NAME: ALLA
string	"A'B"	"NAME□□□□□"
"STRING"	"A—B"	SYSTEM
.076#	NATALI	ARRAY(30, 40)
"X=25"	algol	PERCENT%
256.\$	ПРИМЕР	"%! # \$"
BASIC	A[577]	BASIC.SYSTEM

§ 3. Числовые операции и выражения

1. Понятие операции и выражения. Над данными могут совершаться различные действия, называемые *операциями*. Операции над числовыми данными (или операндами) называются *числовыми операциями*. Каждая операция обозначается знаком операции. Операнды, соединенные с помощью знаков операций в строку, которая после выполнения указанных в ней операций соответствует конкретному числовому значению, образует *числовое выражение*.

Среди операций, используемых в числовом выражении, выделяются арифметические операции, операции отношения и логические операции.

Выражения можно использовать практически во всех случаях, когда переменные и константы служат для вычисления некоторых значений. Использование выражения в конструкциях, где воспринимаются строки символов, приводит к тому, что последовательность символов, образующая выражение, интерпретируется как строка, т. е. не вычисляется. Естественно, использование того или иного выражения должно быть логически оправдано, в частности тип значения, получаемого в результате вычисления всего выражения, должен быть согласован с характером использования этого выражения.

2. Арифметические операции. К *арифметическим* операциям относятся операции, определяемые следующими знаками операций:

\wedge (возведение в степень),	\backslash (деление нацело),
$-$ (изменение знака),	MOD (арифметический модуль),
$*$ (умножение),	$+$ (сложение),
$/$ (деление),	$-$ (вычитание).

Все приведенные знаки операций используются в выражениях с двумя числовыми операндами, за исключением операции изменения знака, которая является одноместной операцией, т. е. знак — пишется перед единственным операндом. Например, выражениями будут следующие конструкции:

$X \wedge 2$ $-Y$ $X * Y$ X / Y 2.5 X

Последние два из приведенных примеров представляют собой частные случаи выражения, в котором присутствует только один операнд (константа или переменная).

Последовательность выполнения операций определяется приоритетом операции. Арифметические операции, указанные выше, перечислены в порядке убывания приоритета, при этом операции умножения и деления, так же как и сложения и вычитания, обладают одинаковыми приоритетами. В первую очередь выполняются операции с наивысшим приоритетом. В пределах одного приоритета операции выполняются последовательно слева направо. Исключение составляет лишь операция возведения в степень, которая выполняется справа налево. Стандартный порядок выполнения операций может быть изменен с помощью круглых скобок, так как первыми выполняются операции внутри скобок. Скобки могут вкладываться одни в другие практически неограниченно; первыми всегда выполняются операции, заключенные в самые внутренние скобки. Например, в выражении

$$(A-B) \wedge (X / ((K+L) * N) / M)$$

наличие скобок обеспечивает выполнение операций в последовательности: вычитание, сложение, умножение, два деления, возведение в степень.

Отметим, что возведение в целую степень выполняется как перемножение основания самого на себя заданное число раз. Если же показатель степени не является целым числом, то операция осуществляется по формуле $X \wedge Y = \exp(Y * \ln X)$, где $\ln X$ — натуральный логарифм X .

Операция перемены знака преобразует заданное значение в значение той же абсолютной величиной, но противоположного знака.

В числовые выражения могут входить операнды разных типов: при выполнении операций среди операндов выделяется тип, имеющий наибольшую точность, и все остальные операнды преобразуются к этому типу. С той же точностью вычисляется и результат всего выражения.

Деление нацело выполняется как обычное деление, только операнды перед выполнением операции деления округляются до целых чисел, которые должны быть в интервале от -32768 до 32767 , а дробная часть результата деления отбрасывается (не округляется), и остается лишь целая часть частного. Например:

$10 \setminus 4$ равно 2

$25.75 \setminus 6.98$ выполняется как деление $26 \setminus 7$ с результатом 3

Операция арифметического модуля вычисляет целое значение, которое является остатком от целочисленного деления, например:

$10 \text{ MOD } 3$ равно 1

$15.4 \text{ MOD } 3.8$ выполняется как $15/4$; остаток равен 3.

3. Операции отношения. В числовых выражениях операции отношения позволяют сравнивать значения числовых операндов возможно различной точности; при этом, как и в случае арифметических операций, среди двух операндов выбирается операнд с наибольшей точностью и другой операнд преобразуется к его типу. Допустимы следующие шесть операций отношения:

$=$ (равно), $>$ (больше),
 $< >$ (не равно), $<=$ (меньше или равно),
 $<$ (меньше), $>=$ (больше или равно).

имеющие одинаковый приоритет, т. е. при наличии в одном выражении нескольких операций отношения они выполняются слева направо, если конечно, в выражении нет скобок. Например:

$X=Y$ $X<>Y$ $X<=Y$ $X>Y$

Результат сравнения в операциях отношения есть ответ на вопрос — так ли это?, т. е. выражение $X=Y$ воспринимается как вопрос: равны ли X и Y ? Если ответом будет «да», то результатом выражения является целое значение -1 , воспринимаемое как «истина»; в случае ответа «нет» соответствующее значение равно 0 и интерпретируется как «ложь».

4. Логические операции. Эти операции имеют смысл только для значений «истина» и «ложь». Поскольку эти значения являются результатами операций отношения, операндами логических операций, как правило, являются выражения с операциями отношения. Результатом логической операции всегда является либо значение «истина», либо значение «ложь». Считается, что операнд в логической операции или результат логической операции — «истина», если он не равен нулю.

В языке Бейсик предусмотрено шесть логических операций:

NOT (логическое отрицание), XOR (исключающая дизъюнкция),
 AND (конъюнкция), EQV (эквивалентность),
 OR (дизъюнкция), IMP (импликация),

которые перечислены здесь в порядке убывания приоритета.

Логическое отрицание в отличие от остальных логических операций является односторонней операцией, которая изменяет значение операнда на противоположное. Например, если A имеет значение «истина»; то значение выражения NOT A будет «ложь» и наоборот.

Операция «конъюнкция» (логическое умножение) в выражении A AND B , где A и B принимают значения «истина» или «ложь», дает в качестве результата значение «истина» в одном случае: когда и A , и B — «истина», и значение «ложь» — в остальных трех случаях. Выражение A OR B , в котором выполняется операция «дизъюнкция» (логическое сложение), дает значение «ложь» лишь в том случае, когда и A , и B имеют значение «ложь», и значение «истина» —

в остальных случаях. Операция «импликация» в выражении $A \text{ IMP } B$ дает значение «ложь», когда A — «истина», B — «ложь», и значение «истина» — при других трех комбинациях значений операндов A и B .

Операция «исключающая дизъюнкция» обеспечит выражению $A \text{ XOR } B$ значение «ложь», а операция «эквивалентность» — выражению $A \text{ EQV } B$ значение «истина» в случае совпадающих значений операндов A и B и противоположные значения при несовпадении значений A и B .

Выражения с операциями отношения и логическими операциями наиболее часто используются при программировании в составе операторов условного перехода для задания условий, определяющих выбор очередного шага работы программы (см. § 15).

В общем случае логические операции выполняются путем преобразования их операндов в шестнадцатитривиальные целые числа в диапазоне от -32768 до 32767 . В случае отрицательного операнда соответствующее двоичное число представляется в дополнительном коде. Если операнд не лежит в этом диапазоне, то результат будет ошибочным. Логическая операция выполняется с целыми числами поразрядно (по битно), т. е. каждый бит результата зависит от значения соответствующих битов двух операндов. Например, результатом выражения $15 \text{ AND } 14$ будет 14 (15 есть двоичное число 1111 , 14 есть двоичное 1110 , поэтому $1111 \text{ AND } 1110$ есть 1110 , т. е. десятичное 14); выражению $-1 \text{ AND } 8$ соответствует результат 8 (-1 есть двоичное число 1111111111111111 , 8 есть двоичное 1000 , поэтому $1111111111111111 \text{ AND } 1000$ дает 0000000000001000 , т. е. десятичное число 8 ; результат выражения $4 \text{ OR } 2$ есть 6 (4 — это двоичное 100 , 2 — двоичное 10 , поэтому $100 \text{ OR } 10$ есть 110 , а это — десятичное число 6).

Если в одном выражении встречаются арифметические операции, операции отношения и логические операции, то в первую очередь выполняются арифметические операции, затем операции отношения и в последнюю очередь логические операции с сохранением внутри каждой группы действующего там приоритета с учетом расстановки скобок.

Упражнения

1. Определить порядок выполнения операций в следующих выражениях:

$$\begin{aligned} S*(A+B)*B(M,N+3) & \quad 2+A^2 > Y \text{ EQV } X < Y \\ A \wedge (B+C) - (A-B) & \quad X \leq Y \text{ OR } X^2 = X*Y \end{aligned}$$

2. Определить значения следующих выражений:

$$3*(I/J)*2+K \quad 3*(I \setminus J)*2+K \quad \text{NOT } A > B \text{ OR } A < C$$

если переменные I, J, K, A, B, C относятся к целому типу и их значения соответственно равны $7, 2, -3, 3, 4, 5$.

3. Составить арифметические выражения, соответствующие математическим формулам:

$$a + \frac{b}{c+a} \quad \frac{x}{a} - \frac{1}{b} (a + bc^{kx})$$

$$(x+y)^{\frac{kx}{ly}} \quad \sqrt{a^2+b^2} + 4 \sqrt[3]{\frac{a^5}{b^5}}$$

$$\left(\frac{q_1}{q_2}\right)^{\frac{y}{y-1}} \quad a^{ba+b}$$

4. Определить, соответствуют ли приведенные в правом столбце выражения на Бейсике стоящим слева математическим выражениям:

a^{b^2}	$A \wedge B \wedge 2$
ab^2	$A * B \wedge 2$
$\frac{ab}{x+y}$	$A * B / (X + Y)$
$\left(\frac{a+b+c+\pi}{2d}\right)^2$	$(A + B + C + 3.1416) / (2 * D) \wedge 2$

5. Определить, допустимы ли в Бейсике следующие конструкции и совпадают ли значения выражений:

$-2 \wedge 3 \wedge 2 + 12 * 8 / 2 * 5$	и $-2 \wedge 3 \wedge 2 + 12 * 8 \setminus 2 * 5$
$(13 - 67 / 3) \setminus 3$	и $(13 - 67 \setminus 3) / 3$
$(2 \wedge 5 + 8 * 5) \setminus 4$	и $2 \wedge 5 \setminus 4 + 8 * 5 \setminus 4$

§ 4. Строковые операции

Строковые переменные могут быть объединены в выражение, называемое *строковым* или символьным выражением с помощью операции конкатенации, обозначаемой знаком $+$. Эта операция соединяет строковые значения друг с другом с образованием одной более длинной строки, где за последним символом левого операнда сразу же следует первый символ правого операнда. Длина строки, полученной в результате, равна сумме длин операндов.

Например, если переменная A\$ имеет значение "BASIC", а переменная B\$ — значение "PROGRAMMING", то значением выражения A\$+B\$ будет строка символов

"BASIC PROGRAMMING"

В строковые выражения в качестве операторов могут входить строковые константы. Например, в результате выполнения операции конкатенации в выражении "ABC"+"DEFH" получаем новую строку "ABCDEFH".

В строковых выражениях можно сравнивать друг с другом строковые значения, используя операции отношения. Две строки сравниваются посимвольно до выявления двух первых несовпадающих символов. Затем сравниваются (как обычные числовые значения)

ния) машинные коды этих несовпадающих символов и определяется наибольший код; соответствующая строка считается большей. Если первая строка короче второй и все ее символы совпадают с соответствующими символами второй строки, то большей считается вторая, более длинная строка. Две строки одинаковой длины с совпадающими в одинаковых позициях символами считаются равными; в частности, равными считаются две нулевые строки.

Например, результат выполнения операций отношения в следующих выражениях есть «истина»:

```
"A156"="A156"
```

```
"PROGR"<"PROGRAM"
```

```
"AB">"AA" (код В имеет числовое значение, большее чем код А)
```

```
"12345"<>"ABCDE"
```

Все строковые константы, используемые в выражениях, должны быть заключены в кавычки. Использование в одном строковом выражении нескольких операций отношения подряд недопустимо. Так, выражение типа $X\$=Y\$=Z\$$ вызовет сообщение об ошибке («несоответствие типов»), поскольку при выполнении первого сравнения $X\$=Y\$$ получится числовое значение (—1 или 0), которое нельзя сравнивать со строковой переменной $Z\$$.

Упражнения

1. Определить значения следующих выражений:

```
"123"+"123" "A$"+"88" "ТАК">"ТАКТ"
```

```
"123"<"123" "A$"="B$" "STRING"="СТРОКА"
```

```
"123"="123" "A$"<>"B$" "25*5"="125"
```

2. Определить, допустимы ли следующие выражения; в случае положительного ответа вычислить их значения, если это возможно:

```
"MONEY+MONEY" "X+13"="13+X"
```

```
"MONEY"+"MONEY" "INDEX">NUMBER
```

```
"1234"+5678 "NUM"<>NUM$
```

```
"DOLLARS"+DOLLAR$ A$+"B"+"CD"
```

§ 5. Система операторов

Любая программа на языке Бейсик состоит из операторов. Оператор начинается с номера (метки), после которого следует ключевое слово и список оператора, состоящий из различных элементов в зависимости от назначения оператора. Формат оператора в общем случае имеет вид

НОМЕР КЛЮЧЕВОЕ СЛОВО список «возврат каретки»

Максимально допустимое число символов в строке программы, занимаемой одним оператором (и командой тоже), равно 255. Таким

образом, оператор может занимать до семи 40-символьных или до четырех 80-символьных дисплейных строк. В конце оператора нажимается клавиша «возврат каретки» как признак конца оператора и указание для ввода данной строки в машину.

Отметим здесь, что при включении вычислительной системы на экране появляется светящийся (иногда мерцающий) подчеркивающий символ в виде черточки, прямоугольника, квадрата или иной формы. Этот символ называется *курсором*, он указывает позицию для размещения на экране очередного символа. При наборе какого-либо символа нажатием на соответствующую клавишу этот символ высвечивается на экране, а курсор смещается на одну позицию вправо. После заполнения строки курсор автоматически переходит в начало следующей строки, т. е. в крайнюю левую позицию экрана и на одну строку вниз. Существуют специальные клавиши для управления движением курсора вправо, влево, вверх и вниз.

При нажатии клавиши «возврат каретки» курсор также перемещается в крайнюю левую позицию экрана и на одну строку вниз, т. е. выполняется действие, аналогичное работе механизма пишущих машинок и телетайпов. Отсюда и происходит название «возврат каретки».

Несколько следующих друг за другом операторов (или команд) можно помещать в одной программной строке. При этом операторы одной строки отделяются друг от друга двоеточием. Например:

```
НОМЕР КЛЮЧЕВОЕ СЛОВО 1 список 1: КЛЮЧЕВОЕ  
СЛОВО 2 список 2
```

Клавиша «возврат каретки» нажимается лишь один раз в конце программной строки, т. е. после последнего оператора в этой строке. Номер в таком случае имеет лишь первый оператор строки.

Программа выполняется всегда в порядке возрастания номеров операторов (первым исполняется оператор с наименьшим номером), поэтому в исходной программе операторы могут быть неупорядоченными. В практике программирования принято заканчивать программу оператором окончания END, хотя это условие и не является обязательным. Однако если в программе используются подпрограммы, то в основном тексте программы (основной программе) оператор END должен присутствовать, чтобы не допустить выполнения всех процедур после завершения работы основной программы.

Операторы условно делятся на две группы: исполняемые и неисполняемые. *Исполняемый* оператор — это инструкция программы, которая сообщает транслятору, что делать при выполнении программы (например, передать управление, вывести результаты на печать и т. д.). *Неисполняемый* оператор — оператор, который не вызывает никаких программных действий, а является указанием транслятору (например, указание о комментарии в тексте программы, о формировании блока данных и т. д.).

По своему функциональному назначению операторы Бейсика могут быть разделены на следующие классы: операторы описания и определения, операторы присваивания и ввода, операторы структуры программы и подпрограммы, операторы управления, операторы вывода, операторы обработки файлов, операторы машинной графики. Среди операторов указанных классов могут быть как исполняемые, так и неисполняемые, и в дальнейшем при изучении операторов определенного класса будут рассмотрены все операторы, имеющие отношение к программным действиям соответствующего класса.

Среди неисполняемых операторов языка Бейсик имеется оператор, который служит для включения в программу пояснительного текста или *комментария*, описывающего действия операторов. Этот оператор состоит из ключевого слова REM, после которого следует любой текст. Например:

```
1010 REM НАЧАЛО ПОДПРОГРАММЫ АТОС
```

Весь текст программы, следующий после ключевого слова REM, воспринимается как пояснительное замечание. Поэтому, если комментарий является частью строки, содержащей несколько операторов, соответствующий оператор REM должен быть последним оператором данной строки. Например:

```
730 END : REM КОНЕЦ ПРОГРАММЫ ВЫСОТА
```

Ключевое слово REM может быть заменено простой кавычкой. Если комментарий стоит в конце строки, состоящей из нескольких операторов, и вместо слова REM используется кавычка, то двоеточие перед кавычкой не ставится. Так, приведенные выше операторы можно переписать иначе:

```
1010 ' НАЧАЛО ПОДПРОГРАММЫ АТОС
```

```
730 END ' КОНЕЦ ПРОГРАММЫ ВЫСОТА
```

Для указания конца предшествующего оператора в строке и начала пояснительного текста в версии Бейсик-плюс вместо кавычки используется восклицательный знак.

Упражнение

Определить, нет ли ошибок в написании следующих строк:

```
70 END : 80 END : 90 REM END
```

```
100 END : REM : КОНЕЦ ПРОГРАММЫ
```

```
180 REM :
```

```
200 END :
```

```
300 'REM : END
```

```
340 END : END : REM
```

```
380 END ' REM : END ' REM
```

§ 6. Описание переменных

Основной способ указания типа переменной — с помощью последнего символа идентификатора. В то же время тип переменной может быть задан по первой букве идентификатора, если эту букву указать в списке оператора *описания типа*, начинающегося с ключевого слова DEF. В этом случае в идентификаторах специальные суффиксы (% , ! , ## , \$) не используются.

Различают четыре разновидности оператора описания типа:

DEFINT диапазон букв

DEFSNG диапазон букв

DEFDBL диапазон букв

DEFSTR диапазон букв

Диапазон букв задается либо одной буквой, либо граничными символами алфавита. Например, если в начале программы стоят операторы

```
10 DEFINT A, F—K
```

```
20 DEFSNG B, C, E
```

```
30 DEFDBL L—N
```

```
40 DEFSTR D, P—X, Z
```

то переменные, имена которых начинаются с букв A, F, G, H, I, J, K, будут переменными целого типа; переменные, идентификаторы которых начинаются с букв B, C, E, будут переменными обычной точности; переменные, имена которых начинаются с букв L, M и N, будут переменными, имеющими двойную точность, а переменные, идентификаторы которых имеют в качестве начальной буквы D, P, Q, R, S, T, U, V, W, X и Z, будут строковыми переменными.

Если в программе в идентификаторе какой-либо переменной тип указывается одним из специальных суффиксов, то он будет приоритетным по отношению к неявному заданию типа переменной операторами DEFINT, DEFSNG, DEFDBL, DEFSTR. Так, используемые в программе переменные A##, AK\$ будут соответственно двойной точности и строковая, несмотря на то, что оператор

```
10 DEFINT A, F—K
```

описывает все переменные, начинающиеся с буквы A, как целые.

В некоторых Бейсик-системах оператор с ключевым словом DEF должен находиться в программе до первого использования определенных им переменных. Начальные значения всех определяемых числовых переменных являются нулями, а строковые переменные имеют нулевую длину строки.

Упражнение

В программе содержатся операторы

```
10 DEFSTR S—Z
```

```
20 DEFINT A. I—N
```

Определить тип переменных, обозначаемых следующими идентификаторами:

A\$, AS, SA, X#, TRAF, LI, KOEFFICIENT, Y%, DBL#, SIGMAI, DBL, MAS, WIND, ZERO, STRING#, B%, PROF, QZ.

§ 7. Определение функций

1. **Оператор-функция.** С помощью оператора DEF могут быть определены функции, задаваемые как значения некоторых выражений. Такие функции иногда называют *оператор-функциями*. Общая форма задания оператор-функции следующая:

DEF FN α (X)=E

где α — идентификатор, буквы FN — обязательная составная часть имени функции FN α ; X — список *формальных параметров*, т. е. X — имена фиктивных переменных, резервирующих место в памяти для фактических значений, которые должны быть определены к моменту выполнения функции; E — выражение, определяющее, какие действия производит данная функция.

Операндами E обычно являются формальные параметры из списка X, отдельные элементы которого разделяются запятыми. В качестве операндов могут использоваться и другие переменные, элементы массивов, функции. Разрешается использовать любые допустимые виды выражений при условии, что тип результата совместим с типом, определяемым именем функции (и результат выражения, и значение функции должны быть либо оба числовыми, либо оба строковыми).

Действительное вычисление значения выражения происходит лишь в момент обращения к функции. Обращение осуществляется указанием имени функции FN α со списком *фактических параметров* A, т. е. FN α (A), в какой-либо конструкции программы, где допускается использование переменной. Число фактических и формальных параметров должно совпадать: каждое значение фактического параметра будет подставлено вместо соответствующего формального параметра, при этом требуется совместимость по типу между формальными и фактическими параметрами (числовыми или строковыми). Фактические параметры могут быть заданы выражениями.

При выполнении функции указанные значения подставляются вместо формальных параметров только в этой функции и нигде больше, даже если в программе встречаются переменные с теми же именами, что и у формальных параметров. Если в выражение, определяющее оператор-функцию, входят какие-либо нефиктивные переменные, то при вычислении функции используются их текущие

значения. Тип и точность результирующего значения определяются именем функции.

Например, оператор-функция

50 DEF FNSC (X, Y) = A*(X^2+Y^2)

задает алгоритм вычисления суммы квадратов двух чисел, умноженной на некоторое текущее значение A. Обращение к этой функции может выглядеть, например, так:

A+FNSC (B, C)

где предполагается, что к моменту обращения значения переменных A, B, C определены (здесь B, C — фактические параметры). Если $A=0.5$, $B=-1$, $C=5$, то значение указанного выражения будет равно $0.5+0.5*(-1^2+5^2)=13.5$, и оно будет использовано в конструкции, содержащей это выражение.

2. Процедуры-функции. В ряде версий языка Бейсик могут быть описаны функции, определяемые с помощью нескольких операторов. Такие функции принято называть *процедурами-функциями*. Описание процедуры-функции имеет вид

DEF FN α (X)

операторы

FNEND

Здесь FN α — имя процедуры-функции, где α — идентификатор, составленный по всем правилам образования имен; X — список формальных параметров; под словом «операторы» подразумеваются операторы языка Бейсик, необходимые для формирования значения идентификатора FN α (тело определения функции), т. е. среди этих операторов должен присутствовать хотя бы один, в котором переменная FN α получает значение (например, оператор присваивания вида FN α =E); FNEND — ключевое слово, обозначающее оператор конца процедуры-функции.

Обращение к процедуре-функции осуществляется указанием ее имени со списком фактических параметров, т. е. FN α (A), в каких-либо конструкциях программы (например, в выражениях). Значением, выдаваемым функцией, является значение FN α после того, как был выполнен оператор FNEND.

Процедуры-функции могут обращаться друг к другу или к самим себе. Это означает, что в операторах процедуры-функции может употребляться имя самой процедуры-функции или любой другой функции со списком фактических параметров.

Формальным параметрам процедуры-функции нельзя присваивать значения в теле определения функции; попытки такого присвоения игнорируются. Если какая-либо переменная не является формальным параметром, но используется в теле определения функ-

ции, то после выхода из функции эта переменная будет сохранять то значение, которое ей было присвоено при выполнении функции.

3. Модификации. В различных версиях языка могут иметь место и другие модификации оператора DEF, в частности для описания фрагментов, заканчивающихся оператором ENDDEF, для подпрограмм, содержащих оператор возврата RETURN, обращение к которым выполняется оператором вызова подпрограмм GOSUB; для указания адреса входа в подпрограмму пользователя с именем USRn (n — целое) число из диапазона от 0 до 9) или для задания адреса начала текущей программной секции. В последних двух случаях соответствующий оператор DEF имеет вид

DEF USRn=адрес

DEF SEG=адрес

Оператор DEF относится к неисполняемым операторам, и поэтому рекомендуется все операторы DEF размещать вместе с другими неисполняемыми операторами. В ряде Бейсик-систем выполнение оператора DEF должно предшествовать первому использованию определяемых им переменных или функций,

Упражнение

В программе содержатся операторы:

10 DEFINT B, L

20 DEF FNLIN (X)=K*X+LIM\M

30 DEF FNIN (X)=K*X+LIM/M

40 DEF FNINT (X)=K*X+LIM/M

Вычислить значения функций FNLIN, FNIN, FNINT при $X=0.5$, если $K=1.5$, $LIM=5$, $M=1.25$.

§ 8. Размерности массивов

Если в программе используются только одномерные и двумерные массивы, то в случае, когда в каждом измерении значение индекса не превосходит 10, по умолчанию максимальное значение индекса принимается равным десяти. Многомерные и одномерные массивы с числом элементов больше указанного должны быть описаны в программе, т. е. должны быть заданы *размерности* массивов. Это означает, что для каждого индекса указывается максимальное значение, которое он может принять.

Оператор *описания массивов* имеет вид

DIM M1, M2, ...

где M1, M2, ... — идентификаторы массивов с указанием максимальных значений индексов в виде целых констант. Так, в операторе

10 DIM A(15), M(10, 20), B(50), N(5, 5, 25)

описываются четыре массива А, М, В, N, причем максимальное значение индекса одномерного массива А равно 15, максимальные значения индексов двумерного массива М равны соответственно 10 и 20, одномерного массива В — 50, трехмерного массива N — соответственно 5, 5 и 25. Отметим, что по умолчанию минимальное значение индекса принимается равным нулю и с шагом 1 значение индекса изменяется до указанного в описании максимального значения. В общем случае значение индекса сверху не ограничивается, однако в конкретных системах могут быть введены ограничения. Например, в версии Бейсик-плюс, допускающей только одно- и двумерные массивы, максимальное значение индекса должно быть не больше, чем 32771.

Наименьшее значение индекса может быть изменено на 1 оператором установления начала отсчета индексов (базы)

OPTION BASE 1

Этот оператор используется как в режиме немедленной обработки, так и в программируемом режиме, однако перед выполнением программы (по команде RUN) наименьшее значение индекса устанавливается равным нулю. Таким образом, команда OPTION BASE 1, выполненная в режиме немедленной обработки, не окажет никакого влияния на массивы, используемые в программе. Это значит, что если индексы в программе должны меняться, начиная с единицы, программа должна содержать свой собственный оператор, например:

```
10 OPTION BASE 1
```

Отметим, что допускается оператор

OPTION BASE 0

который устанавливает с нуля начало отсчета индексов в массивах, описанных после этого оператора. Фиксация базы массивов должна быть произведена до объявления массивов или употребления элементов неописанных массивов.

В большинстве Бейсик-систем требуется, чтобы описание массива предшествовало использованию элементов массива в каких-либо конструкциях программы. При выполнении оператора DIM для каждого указанного в нем массива будет отведен достаточный объем динамической памяти, всем элементам числовых массивов будут присвоены начальные значения, равные нулю, а всем элементам строковых массивов — нулевые строки.

Оператор DIM относится к неисполняемым операторам. Если в программе имеется оператор DEF, действие которого распространяется на идентификаторы массивов, перечисленных в операторе DIM, то он должен обязательно предшествовать оператору DIM.

Оператор объявления массивов DIM можно использовать и в режиме немедленной обработки, однако в этом случае с определяе-

мыми в этом операторе массивами нельзя работать в программируемом режиме, так как после выполнения команды RUN уничтожаются все описания массивов.

В памяти элементы многомерных массивов располагаются таким образом, что первым изменяется последний индекс, что означает в случае двумерных массивов расположение по строкам.

Размерность массива, определенная оператором DIM, в процессе выполнения программы не может быть изменена никаким другим оператором DIM. Однако массив может быть уничтожен путем использования оператора *удаления*

ERASE M1, M2, ...

где M1, M2, ... — имена массивов без указания максимальных значений индексов, после чего при выполнении нового оператора DIM, переопределяющего структуру некоторого массива (из числа M1, M2, ...), всем элементам этого массива присваиваются нулевые значения. Например:

```
10 DIM M$(50), C(50), PR$(30)
...
110 ERASE C, PR$
120 DIM C(4, 20)
```

Упражнения

1. Указать, какие из приведенных строк можно рассматривать как описания массивов:

```
10 DIM Y[10], Z(10)
20 DIM S(1:5), SUM(1,5)
30 DIM M%(10, 20, 10), N(N)
40 DIM ARRAY$(50), MAS(N, M)
```

2. Описать входящие в некоторую программу идентификаторы: A, B, C, D, F, K, X, Y, Z, L, M, N, если известно, что A, B, F, X относятся к переменным двойной точности; K, Z обозначают строковые переменные; C, D — векторы с компонентами c_1, c_2, c_3 и d_1, d_2, d_3 соответственно, действительного типа; M — массив из 100 символьных строк; X — квадратная матрица из 36 целых значений; L, N — переменные целого типа.

§ 9. Присваивание значений

Начальные нулевые значения, присваиваемые переменным при их объявлении или появлении в программе, как правило, изменяются в процессе выполнения программы. Такое изменение, т. е. присваивание нового значения, чаще всего выполняется оператором *присваивания*

LET V=E

где ключевое слово LET может быть опущено, V — переменная, E — выражение. Например:

```
70 LET X=3.14
80 S$="DIAMOND"
90 N%=51
```

Оператор LET присваивает переменной с именем, указанным слева от знака равенства (в данном случае являющимся знаком присваивания), значение выражения, записанного справа от этого знака. Переменная и присваиваемое ей значение должны быть совместимы по типу данных. Если это условие не выполняется, например, переменной строкового типа присваивается числовое значение, то будет выдано сообщение об ошибке («несоответствие типов»).

Числовое значение двойной точности, присваиваемое переменной, которая имеет обычную точность или является целой, округляется в соответствии с типом переменной. Аналогично, если значение обычной точности присваивается целой переменной, это значение округляется до ближайшего целого числа. Например, в результате выполнения операторов

```
130 A!=1.23456789
140 A%=1.23456789
```

переменная A! получит значение 1.234568, а переменная A% — значение 1.

В тех случаях, когда целое значение присваивается переменной обычной или двойной точности, это значение дополняется точкой и следующими за ней нулями. Аналогично преобразуются значения обычной точности в случае присваивания их переменным двойной точности, но при этом преобразованное значение содержит не более шести точных цифр.

В некоторых системах допускается присвоение одного и того же значения нескольким переменным в пределах одного оператора. В этом случае переменные отделяются друг от друга либо знаками =, либо запятыми

```
V1=V2=...=VN=E
V1, V2, ..., VN=E
```

Во избежание возникновения сложных ситуаций, обычно приводящих к ошибкам, при использовании такой конструкции рекомендуется воздерживаться от употребления в одном списке индекса и элемента массива с этим индексом. Так, фрагмент программы

```
30 I=2 : A(2)=1 : K=1
40 I, A(I)=I+K
```

при выполнении его в разных Бейсик-системах может привести к различным результатам. Например, в Бейсик-плюс сначала вычисляются значения индексов переменных, затем значение выражения

в правой части оператора, и только после этого вычисленное значение выражения присваивается всем переменным, указанным в левой части оператора. Следовательно, в приведенном примере получаем после выполнения строк с номерами 30 и 40, что $A(2)$ и I равны 3, а $A(3)$ равно нулю.

Тот же результат получим, если в системе принят порядок выполнения операций присваивания справа налево, а индексы предварительно не вычисляются. Если же значения присваиваются в обратном порядке, то получаем, что I и $A(3)$ равны 3, а $A(2)$ равно 1, т. е. результаты не совпадают.

Отметим, что в любом случае сначала вычисляется значение выражения в правой части оператора присваивания, а затем это значение присваивается всем переменным, записанным слева. Таким образом, в общем случае оператор

$A, B = E$

не эквивалентен последовательности операторов

$A = E : B = E$

Например, в результате выполнения операторов

50 $X = 1$

60 $X, Y = X + 5$

переменным X и Y присваивается значение 6, тогда как при выполнении операторов

50 $X = 1$

70 $X = X + 5 : Y = X + 5$

переменная X получает значение 6, а Y — значение 11.

Упражнения

1. Описать в виде операторов присваивания вычислительные операции при решении квадратного уравнения $ax^2 + bx + c = 0$.

2. Найти ошибки в записи следующих операторов присваивания:

LET $X + Y = Z$ $F(X) = X \wedge 3$

3.25 = $A + B$ $A : B = A + 1$

$-M = A \wedge 2 - B \wedge 2$ $X = Y = Z, P = Q$

LET $Z = \text{"LET"}$ $Z\$ = \text{TEXT} + \text{LET}$

3. Определить значения переменных после выполнения следующих операторов присваивания:

$N = 25$ $L\% = 3.14$

$M\# = 1.347$ $J = I\# = 9876799.44$

$K = 476338976.25$ LET $N = L\% = 267.89$

§ 10. Ввод данных в программу

Оператор присваивания является наиболее распространенным средством однократного присваивания значения переменной. Однако он оказывается неудобным в случае присваивания значений большому числу переменных и ввода присваиваемых значений с клавиатуры в процессе выполнения программы. Для этой цели служат более эффективные операторы.

1. Ввод с использованием блока данных. Оператор

DATA список констант

где в списке констант могут содержаться любые допустимые числовые или строковые константы, отделяемые друг от друга запятыми, формирует в памяти *блок данных*. Используемая строковая константа должна быть заключена в кавычки, если она начинается со значащих пробелов или кончается ими, либо если один из входящих в строку символов — запятая или двоеточие. При этом в заключенной в кавычки строке запрещено использовать кавычки. Например:

```
180 DATA 5, 7.35, BASIC, "PROGRAM", 1E-2
```

```
340 DATA 701.5, 46, 3.14159
```

Оператор DATA относится к числу неисполняемых операторов, который может находиться в любом месте программы.

В одной программе может быть несколько операторов DATA, при этом списки всех таких операторов формируют один блок данных. Этот блок создается последовательно, начиная с констант, указанных в первом операторе DATA (расположенном первым в тексте программы), и кончая значениями, относящимися к последнему из них. Так, приведенные выше операторы DATA формируют блок данных из восьми элементов, где первым элементом является значение 5, последним — 3.14159.

Значения констант из блока данных присваиваются скалярным переменным и элементам массивов с помощью оператора *ввода*

READ список переменных

где в списке переменных перечислены те переменные, которым должны быть присвоены значения. Оператор READ — исполняемый.

Оператор READ присваивает первое значение из блока данных, формируемого операторами DATA, первой переменной списка переменных данного оператора READ. Второй переменной оператора READ присваивается второе значение, третьей — третье и т. д. до тех пор, пока не будут присвоены значения всем переменным этого оператора. Если в дальнейшем в программе выполняется еще один оператор READ, то переменной присваивается следующее имеющееся значение из блока данных. Например,

операторы

```
70 READ N%, A, X$, Y$  
190 READ B, C, M%, D#
```

осуществят присваивание значений переменным, указанным в списках этих операторов из блока данных, сформированного приведенными выше операторами DATA с номерами 180 и 340. Это эквивалентно следующим присваиваниям:

```
N%=5: A=7.35: X$="BASIC": Y$="PROGRAM":  
B=1E-2: C=701.5: M%=46: D#=3.14159
```

Неиспользованные значения из блока данных игнорируются, а если при выполнении некоторого оператора READ оказалось, что блок данных уже исчерпан, выдается сообщение об ошибке («нет данных»).

Типы переменных и элементов массивов, указанных в списке переменных оператора READ, должны быть совместимы с типами присваиваемых им значений. Так, недопустимо, чтобы переменная была числовой, а значение — строковым или наоборот. В случае подобного несоответствия выдается сообщение об ошибке («синтаксическая ошибка») и текст оператора READ, в котором обнаружилось несоответствие типов данных. Если и переменная, и константа являются числовыми, но их типы различаются по точности, оператор READ выполняет преобразования, аналогичные действию оператора LET.

Одновременно с присваиванием очередного значения из блока данных переменной в операторе READ происходит перемещение специального указателя, отслеживающего текущую позицию в блоке данных. Управление перемещением этого указателя может быть осуществлено оператором *восстановления*. В простейшем случае этот оператор имеет вид

```
RESTORE
```

и возвращает указатель к началу списка констант в блоке данных; при этом очередной оператор READ, следующий после оператора RESTORE, будет выбирать значения из списка повторно. Например, используя вышеприведенный блок данных, оператор

```
200 RESTORE
```

переместит указатель в начало блока, так что оператор

```
210 READ L%
```

вызовет присваивание переменной L% значения 5.

Оператор восстановления позволяет также устанавливать указатель на элемент блока данных, соответствующий любому промежуточному оператору DATA. В этом случае оператор имеет

вид

RESTORE номер строки

где номер строки — это номер соответствующего требуемого оператора DATA. Если в строке с указанным номером не окажется оператора DATA, будет выбран ближайший оператор DATA, расположенный в программе после указанной строки. Так, обращаясь к приводимому в этом пункте примеру, оператор

220 RESTORE 340

устанавливает указатель в начало списка констант оператора с номером 340, т. е. на значение 701.5. Вместо метки 340 в этом операторе может стоять любой номер от 181 до 339 при условии, что на указанном участке программы отсутствуют операторы DATA.

2. Ввод значений с клавиатуры. Значения переменных могут быть введены в программу с клавиатуры по мере их запроса самой программой. Для этого предусмотрен оператор ввода, который считывает вводимые значения с клавиатуры и присваивает их переменным и элементам массива. Оператор ввода с клавиатуры

INPUT список переменных

в списке переменных содержит имена числовых или символьных переменных. Например:

350 INPUT A, N%, X\$, D#, Y\$

Выполняя оператор INPUT, ЭВМ выводит на экран дисплея знак вопроса. Появление этого знака означает, что пользователь программы должен ввести с клавиатуры значение каждой переменной оператора INPUT, отделив их друг от друга запятыми. Количество и тип значений должны соответствовать количеству и типу переменных в списке оператора INPUT. После набора последнего значения следует нажать клавишу «возврат каретки». При ответе на запрос оператора INPUT строковое значение необходимо заключить в кавычки при условии, что оно содержит запятые, либо начинается (заканчивается) значащими пробелами; заключенная в кавычки строка не должна содержать кавычек.

Так при выполнении оператора INPUT с номером 350, приведенного выше, на экране высвечивается символ ?. На клавиатуре следует набрать значения для переменных списка этого оператора, например:

7.35, 5, BASIC, 3.14159, "PROGRAM"

и нажать клавишу «возврат каретки». Если ввод был осуществлен правильно, программа будет выполняться дальше.

После набора значений переменных производится проверка вводимого сообщения. При необходимости выполняются преобразования числовых значений для согласования их точности с точ-

носию соответствующих им переменных, подобно тому как это делается операторами LET и READ. Если количество набранных пользователем значений больше или меньше числа переменных и элементов массива оператора INPUT либо если пользователь вводит строковое значение в качестве значения числовой переменной, то выдается сообщение об ошибке («повторите ввод»). После этого сообщения осуществляется повторный ввод всех значений с самого начала строки.

Высвечивание на экране дисплея вопросительного знака при наличии в программе нескольких операторов INPUT не позволяет определить, каким именно переменным требуется ввод. Оператор INPUT позволяет кроме символа ? (или вместо него) выводить некоторое наводящее сообщение. Текст этого сообщения должен указываться в операторе INPUT в виде строковой константы, стоящей непосредственно за ключевым словом, при этом строковая константа заключается в кавычки и отделяется от имени переменной точкой с запятой. Наводящее сообщение может быть указано перед каждой переменной, соблюдая перечисленные синтаксические правила.

Например, следующий оператор:

```
360 INPUT "СКОРОСТЬ КОРАБЛЯ"; V
```

обеспечит появление на экране сообщения

СКОРОСТЬ КОРАБЛЯ?

После наводящего сообщения вместо точки с запятой можно указать запятую. В этом случае знак вопроса на экране не появится. Так, если предыдущий оператор представить в виде

```
360 INPUT "СКОРОСТЬ КОРАБЛЯ=", V
```

то информация на экране будет следующая:

СКОРОСТЬ КОРАБЛЯ=

Разновидностью оператора ввода с клавиатуры является оператор

LINE INPUT строковая переменная

который обеспечивает присваивание всех символов, вводимых с клавиатуры до первого возврата каретки, одной и той же строковой переменной. Поскольку вводится только одно значение, запятые в качестве разделителей не используются, т. е. при наборе вводимых данных на клавиатуре запятые можно использовать как прочие символы без заключения всей записи в кавычки. При выполнении оператора LINE INPUT вопросительный знак, как требование к пользователю ввести значение, на экран не выводится, но при необходимости его всегда можно включить в явном виде в текст наводящего сообщения, которое формируется по тем же

правилам, что и в операторе INPUT. Например, оператор
370 LINE INPUT "НАЗВАНИЕ ПРОГРАММЫ:"; N\$
вызовет появление на экране текста

НАЗВАНИЕ ПРОГРАММЫ:

и после набора любой строки символов длиной до 254 символов выполнение программы продолжится.

Преимущество оператора LINE INPUT перед оператором INPUT с точки зрения пользователя состоит в возможности при вводе значения свободно использовать запятые.

В расширенном Бейсике после ключевых слов INPUT или LINE INPUT можно вставить дополнительную точку с запятой

380 INPUT; "НОМЕР"; N

В таком варианте оператора после ввода с клавиатуры ответа на запрос и нажатия на клавишу «возврат каретки» автоматического возврата каретки не происходит, что удобно при заполнении последней строки экрана, так как не происходит сдвига изображения вверх и не теряется верхняя строка.

3. Обмен значениями. Значения двух переменных или элементов массива одного типа можно взаимно поменять оператором

SWAP V1, V2

где V1, V2 — идентификаторы переменных.

Так, если переменные A\$ и B\$ имели соответственно значения "BASIC" и "FORTRAN", то после выполнения оператора

140 SWAP A\$, B\$

Переменная A\$ будет иметь значение "FORTRAN", а B\$ — "BASIC".

В случае несоответствия типов переменных будет выдано сообщение об ошибке («ошибочный тип»).

Упражнения

1. Сформировать в памяти массив числовых данных из 30 элементов и обеспечить доступ к нему операторами READ таким образом, чтобы данные считывались по следующему правилу: сначала первые десять, затем первые двадцать, последние двадцать и последние десять чисел.

2. Обеспечить ввод с клавиатуры 5 строковых значений и значений элементов числового массива M (2, 3).

§ 11. Вывод данных

1. Вывод на экран. Ширина изображений на экране дисплея может быть запрограммирована либо на 40, либо на 80 символов в строке с помощью оператора *длины строки*

WIDTH N, W

где N — номер или имя устройства, W — ширина (длина строки в символах). Параметр N может быть опущен, тогда используется текущее устройство. Очевидно W может принимать два значения: 40 или 80.

Перед первым выполняемым в программе оператором WIDTH указывается оператор *режима* SCREEN 0, отменяющий максимальную ширину, установленную для данной Бейсик-системы.

Размеры символов при ширине 40 вдвое больше по сравнению с 80-символьной строкой и воспринимаются легче. Поэтому, если в качестве дисплея применяется обычный телевизор или цветной монитор, то использовать оператор WIDTH 80, как правило, не рекомендуется: изображение символов может оказаться очень нечетким и потому неудобным для чтения.

На экране можно разметить до 25 строк текста. Возможна очистка экрана с возвратом курсора в левый верхний угол. Это действие выполняется оператором CLS.

Вывод значений переменных и элементов массивов, т. е. числовых и строковых констант *на экран дисплея* осуществляется оператором

PRINT список

где в списке перечисляются идентификаторы тех переменных, значения которых должны быть распечатаны на экране, или же непосредственно указываются строковые константы, подлежащие выдаче (при этом кавычки обязательны); элементами списка могут быть и выражения.

Оператор PRINT является исполняемым и работает как в программируемом режиме, так и в режиме немедленной обработки (т. е. допускается команда PRINT).

В операторе PRINT может отсутствовать список. В этом случае на экран выводится пустая строка.

В качестве разделителей в списке оператора PRINT используются точка с запятой и запятая. В зависимости от используемого разделителя и типов переменных в списке определяется число пробелов между соседними выведенными на экран словами (значениями).

Если два значения разделены в операторе PRINT точкой с запятой, то они выводятся непосредственно друг за другом. Это, в частности, позволяет при выводе строковых значений получить конкатенацию нескольких строк. Однако слитное написание числовых значений приводит к неудобствам при чтении, поэтому после каждого числового значения обязательно дается один пробел. Положительным числам предшествует пробел, отрицательным — знак минус. При заполнении строки экрана вывод будет продолжен на следующей строке, при этом неумещающееся значение

целиком размещается на новой строке, начиная с крайней левой позиции.

Пр и м е р:

```
50 INPUT "НАЗВАНИЕ ПРОГРАММЫ"; NP$
60 INPUT "КОЛИЧЕСТВО СТРОК"; V
70 PRINT
80 PRINT NP$; "КОЛИЧЕСТВО СТРОК НА БЕЙСИКЕ"; V
```

Здесь оператор с номером 50 осуществляет запрос

НАЗВАНИЕ ПРОГРАММЫ?

Предположим, ответили: ВЫСОТА (переменная NP\$ имеет теперь это значение). На запрос

КОЛИЧЕСТВО СТРОК?

ответили: 1200 (теперь V=1200). Оператор с номером 70 пропускает одну строку, а с номером 80 печатает

ВЫСОТА КОЛИЧЕСТВО СТРОК НА БЕЙСИКЕ 1200

Символ; может стоять и в конце списка. В этом случае курсор остается в конце последнего выведенного значения и следующий оператор PRINT начинает вывод в той же строке.

При использовании в операторе PRINT в качестве разделителя запятой значения выводятся в стандартной табличной форме по столбцам. Экран делится на зоны: при ширине 40 символов — две зоны по 14 и одна зона 12 символов; при ширине 80 символов — 5 зон по 14 и одна зона 10 символов. Если в операторе PRINT перед элементом списка стоит запятая, то перед выводом соответствующего значения курсор переместится к началу очередной зоны.

Две запятые означают пропуск одной зоны, а запятая в конце списка указывает на то, что следующий оператор PRINT начинает печать в той же строке со следующей зоны. Если при выводе некоторое значение выходит за пределы зоны, следующее значение будет выведено с начала первой свободной зоны.

Пр и м е р:

```
70 PRINT "ФАМИЛИЯ, ИМЯ, ОТЧЕСТВО"," — ГОД —"
120 READ NAME$, AGE
140 PRINT NAME$,, AGE
150 READ NAME$, AGE
160 PRINT NAME$,, AGE
170 READ NAME$, AGE
180 PRINT NAME$,, AGE
800 DATA ТАММ А. А., 1940, ТАММ А. К., 1950, ТАММ С. А.,
1970
```

На экран будет выведено следующее:

ФАМИЛИЯ, ИМЯ, ОТЧЕСТВО	_____	— ГОД —
ТАММ А. А.		1940
ТАММ А. К.		1950
ТАММ С. А.		1970

В этом примере первая выводимая строковая константа в операторе с меткой 70 выходит за пределы первой зоны, поэтому вторая константа выводится в третьей зоне для того, чтобы последующие выводимые значения располагались в нужных столбцах, в соответствующих операторах PRINT с номерами 140, 160 и 180 между идентификаторами переменных ставятся две запятые и при выводе вторая зона пропускается.

Числовые значения в зависимости от количества значащих цифр выводятся в обычной форме представления (с фиксированной точкой) или в экспоненциальной форме. Значения обычной точности, содержащие более семи значащих цифр, и значения двойной точности, имеющие более 16 значащих цифр, всегда выводятся в экспоненциальном представлении. Так, значения, введенные в виде

12345678900 и 0.001234567890123456789

будут напечатаны как

1.2345678E+10 и 1.234567890123457E-03

а числа, введенные в форме

1234.56789 и 123456789.123456789

будут выведены в следующем виде:

1234.568 и 123456789.1234568

2. Форматный вывод. Оператор

PRINT USING "шаблон"; список

предназначен для вывода на экран дисплея информации, форма представления которой (*формат*) задается специальным строковым значением — *шаблоном*. Каждый символ шаблона имеет определенную интерпретацию, так что шаблон устанавливает, на какие числовые и строковые зоны разбиваются выводимые значения, какова длина и другие характеристики каждой зоны и т. д. При необходимости каждое выводимое значение списка преобразуется в соответствии с заданным шаблоном. Список оператора PRINT USING формируется точно так же как и в операторе PRINT, хотя в некоторых версиях Бейсика могут быть и отличия. Например, в IBM-системе в качестве разделителей разрешается использовать только точку с запятой.

При выводе строковых значений используются следующие символы шаблона (строковые форматы):

! — означает вывод одного символа;

\\ — вывод фиксированного количества символов: двух или более в зависимости от числа пробелов, стоящих между косыми чертами с левым наклоном;

& — вывод строки переменной длины (вывод целиком всего строкового значения).

Например, если строковая переменная A\$ имеет значение "PROGRAM", то с помощью следующих операторов:

```
40 PRINT USING "!"; A$
50 PRINT USING "\\"; A$
60 PRINT USING "\\_\\_\\_\\_"; A$
70 PRINT USING "&"; A$
```

будет выведено

```
P
PR
PROG
PROGRAM
```

Таким образом, если длина строкового значения не помещается в задаваемое шаблоном поле, то «избыточные» правые символы этого значения отбрасываются. Если же длина строкового значения меньше длины поля, то оставшиеся свободные места в поле заполняются пробелами.

При выводе числовых значений используются следующие символы шаблонов (числовые форматы):

— вывод одного разряда выводимого значения;

+ — вывод в явном виде знака + или —;

— — вывод пробела, если значение положительно, и знака —, если оно отрицательно;

. — вывод в заданной позиции поля десятичной точки;

^^^ — вывод значения в экспоненциальном представлении;

, — вывод запятых через каждые три разряда;

** — заполнение оставшихся слева незанятых позиций поля звездочками;

\$\$ — вывод перед самым значением одного символа \$;

**\$ — заполнение левых позиций звездочками и вывод непосредственно перед самым значением знака доллара.

Ниже приводятся примеры операторов PRINT USING, после которых в следующей строке указывается, в каком виде по данному шаблону выводится числовое значение:

```
10 A=123;PRINT USING "# # #"; A
123
```

```
20 A=—123:PRINT USING "##.###"; A
   └─123
30 A=1234:PRINT USING "##.##"; A
   % 1234
```

Если число содержит меньше позиций, чем предусмотрено шаблоном, то оно располагается в поле справа, а слева добавляются пробелы. Если же число содержит больше разрядов, чем задано шаблоном, то оно все равно будет выведено полностью, только перед первой его цифрой выводится символ %. При количестве заданных цифр в числе более 24 появляется сообщение об ошибке («неверен вызов функции»).

Знак плюс может содержаться в шаблоне как перед самым первым символом ##, так и после последнего символа ##, а знак минус только после последнего символа ##. Точка в шаблоне позволяет заранее установить, сколько разрядов будет в выводимом числе в целой и дробной частях. При этом возможно округление в последнем разряде дробной части. Например:

```
30 A=—68.95:B=22.449:PRINT USING "+##.##"; A; B
   —68.95      +22.45
40 PRINT USING "##.###+"; A; B
   68.95—      22.45+
50 PRINT USING "##.###+"; A; B
   68.95—      22.45
```

Для вывода числовых значений в экспоненциальном представлении четыре символа ^ указываются в конце числового шаблона:

```
60 A=—76954:PRINT USING "###.##^ ^ ^ ^"; A
   —76.95E—02
```

В мантиссе размещается максимальное число цифр, а порядок настраивается соответственно мантиссе. Если шаблон задает более одной цифры в целой части мантиссе, то крайний левый разряд всегда будет либо пробелом, либо знаком минус в зависимости от знака выводимого значения.

Для удобства чтения большие числа часто записывают с запятыми через каждые три цифры, например 123,456,789,987. Запятая, расположенная в шаблоне после любого символа ##, но до десятичной точки, обеспечивает расстановку запятых в выводимом значении, при этом каждая запятая занимает одну позицию в выводимом поле. Примеры:

```
60 A=1234E7
70 PRINT USING "##.#####.##"; A
   12,340,000,000.00
80 A=135 95:PRINT USING "###.###.##"; A
   ***135.95
```

```

90 PRINT USING "$$###.##"; A
   L$135.95
100 PRINT USING "**$###.##"; A
    *$135.95

```

В шаблонах с символом \$ в случае отрицательных чисел знак минус выводится перед знаком доллара. Использование знака плюс допускается как в начале, так и в конце шаблона. Экспоненциальный формат не применяется.

Шаблон, задаваемый в операторе PRINT USING, может содержать любые символы. Эти символы не принимают участия в определении числовых и строковых полей, а всякий раз при использовании шаблона будут выводиться в том виде, как они записаны, без каких-либо изменений. Такими символами могут быть пробелы, буквы алфавита, цифры и большинство специальных знаков. Среди них есть и такие, которые используются в символьном и числовом форматах в смысле, определенном выше. Для того чтобы символ, встречающийся в шаблоне, воспринимался как символьная константа (литерал) и выводился без изменений, надо поставить перед ним в шаблоне символ подчеркивания —.

Например:

```

110 S=567754.28
120 PRINT USING "СУММАРНЫЙ ДОХОД ##,
   ###.##"; S
   СУММАРНЫЙ ДОХОД 567,574.28
130 H=185:PRINT USING "ПОСТ":###-##"; H
   ПОСТ:185#

```

3. Вывод на печать. Данные будут выведены на печатающее устройство, если включить в программу операторы с ключевыми словами LPRINT и LPRINT USING, соответствующие операторам PRINT и PRINT USING и имеющие точно такую же структуру списков. Оператор LPRINT допускает ширину строки печати до 80 символов. Печать свыше 80 символов происходит с переводом строки.

Печатающее устройство всегда воспроизводит строку целиком. Оно имеет память емкостью в одну строку для запоминания символов, записанных в операторах LPRINT или LPRINT USING, которые оканчиваются точкой с запятой. Вывод на печать осуществляется лишь при выполнении операторов LPRINT и LPRINT USING, не имеющих в конце списка символа ; или же после накопления достаточного количества символов для заполнения всей строки целиком (в этом случае на печатающее устройство автоматически посылается сигнал «возврат каретки»). Все последующие символы, поступающие в устройство печати, накапливаются для формирования следующей строки.

Например, следующие операторы вывода на печать:

```
100 LPRINT "ОПИСАНИЕ ЯЗЫКА BASIC";
```

```
170 LPRINT "└(РУКОВОДСТВО ДЛЯ ПОЛЬЗОВАТЕЛЯ)";
```

```
200 LPRINT
```

```
210 LPRINT "ЧАСТЬ 1"
```

работают так. Поскольку операторы с номерами 100 и 170 заканчиваются точкой с запятой, и в списках их содержатся строковые константы, суммарное количество символов в которых не достигает 60, информация, подлежащая выводу, будет накоплена в памяти печатающего устройства. Оператор с номером 200 хотя сам не содержит информации для печати, заканчивается возвратом каретки (нет точки с запятой) и задействует печатающее устройство. Будет напечатано (в одной строке)

ОПИСАНИЕ ЯЗЫКА BASIC (РУКОВОДСТВО ДЛЯ ПОЛЬЗОВАТЕЛЯ)

Оператор с номером 210 выведет на печать строку

ЧАСТЬ 1

не дожидаясь заполнения на печатающем устройстве всей строки (80 символов), так как этот оператор не имеет в конце точки с запятой.

Упражнения

1. Вывести на экран шириной 80 символов значения элементов массива $A\$(80)$ каждое длиной 10 символов, расположив их сначала в последовательные строки, затем — в виде таблицы из четырех колонок.

2. Написать операторы вывода на экран дисплея 20 значений обычной точности в числовом формате с обязательным выводом знака перед числом, предусмотрев в целой части числа четыре, а в дробной части три разряда.

§ 12. Стандартные функции

Некоторые из часто встречающихся математических процедур над числами или процедур преобразования строк символов представлены в языке Бейсик в виде *стандартных* функций. Стандартной функции соответствует определенное заданное ключевое слово, являющееся ее идентификатором, после которого обычно указываются один или несколько аргументов функции в круглых скобках. В некоторых функциях аргумент отсутствует. В отличие от рассмотренных выше функций, определяемых пользователем, описаний стандартных функций в программе нет, пользователь лишь обращается к ним, указывая в конструкциях, где это допустимо, имя соответствующей стандартной функции и список ее аргументов (фактических параметров), которые могут быть и выражениями.

Количество различных стандартных функций в различных версиях языка измеряется несколькими десятками. Суммарное число различных функций около 130 (например, в версии Бейсик-плюс стандартных функций меньше 20, а в Бейсик-системе IBM около 60).

Стандартные функции реализуют различные вычисления, анализ строковых данных, преобразование типов данных, а также позволяют осуществлять непосредственное управление вычислительной машиной и ее внешними устройствами и анализировать состояние и содержание файлов данных. Большинство функций выдают в качестве результата числовое значение, но имеются функции, результатами выполнения которых являются строковые значения, а также функции, которые хотя и вычисляют числовые значения, но связаны со строковыми данными.

Список большинства стандартных функций, используемых в различных Бейсик-системах, приведен в Приложении II; остановимся здесь лишь на некоторых из них.

1. Математические функции. К функциям этого типа относятся функции вычисления значений элементарных функций и некоторые другие математические процедуры. Например, $\text{ABS}(X)$, $\text{SIN}(X)$, $\text{SQR}(X)$, RND , вычисляющие соответственно $|X|$, $\sin X$, \sqrt{X} , случайное число.

2. Функции преобразования числовых данных. Функции этой категории преобразуют данные одного типа в другой. К их числу относятся, в частности, функции $\text{FIX}(X)$, $\text{INT}(X)$, $\text{CINT}(X)$, $\text{CDBL}(X)$, которые соответственно выполняют действия: отбрасывает все цифры после десятичной точки, определяет наибольшее целое, не превосходящее значение аргумента, округляет заданное значение до ближайшего целого, преобразует значение аргумента в значение двойной точности (фактическая точность самого числа при этом не повышается). Например, если X равен 97.54, то $\text{FIX}(X)$ есть 97, $\text{INT}(X)$ равен 97, $\text{CINT}(X)$ есть 98, а $\text{CDBL}(X)$ выдает значение, равное 97.54000000000000.

3. Функции преобразования цифровых строк. Из цифр могут быть образованы цифровые строки, например, "123.456". Цифровую строку можно преобразовать в числовое значение и наоборот. Существует несколько стандартных функций для выполнения названных действий, в частности, функции $\text{VAL}(X)$ и $\text{STR}\$(X)$. Первая из этих функций преобразует цифровую строку в числовое значение, например значением функции $\text{VAL}("123")$ будет 123. Если строка начинается с нецифрового знака, то результатом выполнения функции VAL будет нуль.

Функция $\text{STR}\$(X)$ преобразует числовое значение в цифровую строку, например, результатом выполнения функции $\text{STR}\$(123)$ будет строка "123".

4. **Функции обработки строк.** К этому типу относятся функции формирования строк, функции выделения подстрок и некоторые другие функции. Например функция `STRING$(5, "*")` формирует строку из пяти повторяющихся символов*, т. е. строку `"*****"`. Здесь первый аргумент — число повторений, второй — заданный символ. Стандартная функция `MID$(C$, 7, 7)` выделяет среднюю часть строки, задаваемой первым аргументом. Вторым аргументом указывает номер позиции в исходной строке, начиная с которой будет выделяться слева направо подстрока, третий аргумент — длину подстроки. Так, если `C$` есть `"BASIC PROGRAMMING SYSTEM"` то значением функции `MID$(C$, 7, 7)` будет строка `"PROGRAM"`.

Кроме названных категорий стандартных функций существуют еще функции преобразования символов во внутренние коды машины, функции, предназначенные для работы с файлами данных, функции, обеспечивающие графический режим, функции организации управления вводом и выводом, календарные функции и др.

Упражнение

Определить, эквивалентны ли значения выражений:

`INT(ABS(X))` и `ABS(INT(X))`

`INT(Y)\Z` и `INT(Y/Z)`

`INT(Y)\FIX(Z)` и `FIX(Y)\INT(Z)`

§ 13. Система команд

Программа пользователя, составленная из операторов, как правило, выполняется в программируемом режиме. Управляется это выполнение, как известно, директивами или командами, с помощью которых осуществляются следующие действия: запуск системы, ввод программы, проверка и корректировка программы, исправление ошибок, отладка программы, запоминание текста программы, вызов текста программы из файла, выполнение расчетов.

В зависимости от конкретной операционной системы состав команд и форматы сходных по своему назначению директив могут различаться. Тем не менее существует принципиально единая система команд, обеспечивающая выполнение перечисленных действий с программой, написанной на языке Бейсик, независимо от того, какая операционная система используется.

1. **Запуск системы.** Приступая к работе, пользователь должен включить вычислительную машину или подключить свой терминал к машине путем нажатия определенной клавиши (например, `ON LINE` на терминале `EC-7168`). При работе на персональной ЭВМ

следует сначала загрузить операционную систему путем ввода через дисковод с дискета системных и прикладных программ. При этом на экране дисплея будет высвечиваться определенная информация, на которую пользователь должен реагировать нажатием соответствующих клавиш. Возможна и полностью автоматизированная загрузка системы и пользователю остается лишь ввод с клавиатуры своих программ.

Для запуска прикладной программы сначала следует набрать на клавиатуре имя файла, содержащего нужный транслятор (интерпретатор). Например, при работе на ПЭВМ фирмы IBM может быть использован дисковый интерпретатор с именем BASIC или интерпретатор расширенной версии, допускающий более широкий набор команд — интерпретатор BASICA.

При работе на удаленном терминале после установления связи с ЭВМ необходимо выполнить некоторое число операций, чтобы вычислительную машину подготовить к работе с пользователем. В системе ДОС RBP, использующей язык Бейсик-плюс, это — набор команды HELLO, шифра пользователя и пароли. На все команды пользователя ЭВМ реагирует соответствующим образом, включая информацию о последних изменениях, произведенных в системе, и в конце выводит на экран сообщение о готовности системы к работе (например, строку "OK" или READY).

Отметим, что перед тем как приступить к работе, необходимо ознакомиться с документацией по эксплуатации данной вычислительной машины и системами программирования, используемыми ею.

2. Ввод программы. Набору операторов программы на клавиатуре дисплея предшествует команда NEW, которая стирает программу, находящуюся в текущий момент в памяти, и очищает все переменные и массивы, т. е. освобождает память для новой программы. Команда закрывает все файлы. Может быть использована для задания наименования создаваемой программы, например NEW TEST.

Команда AUTO обеспечивает автоматическое генерирование номера оператора или строки программы после каждого нажатия клавиши «возврат каретки». В команде AUTO могут стоять два параметра, образуя список вида N, SN где N — начальный номер, SN — шаг изменения номера. Так, команда AUTO 10, 5 присваивает первому оператору программы номер 10, второму 15 и т. д. Если список опущен, то такая команда эквивалентна команде AUTO 10, 10. Если опущен параметр SN, но за параметром N следует запятая, то принимается шаг, определенный в предыдущей команде AUTO. Если опущен начальный номер, но определен шаг, то нумерация начинается с нуля.

Например, команда AUTO 20 присваивает операторам номера 0, 20, 40, 60 и т. д., а следующая команда AUTO 600 генерирует

номера 600, 620, 640 и т. д. (шаг, равный 20, берется из предыдущей команды AUTO).

При автоматической нумерации строк может оказаться, что номер очередной строки уже имеется в программе. В этом случае вместе с очередным номером строки на экране появляется кавычка или звездочка, означающая, что следующая строка будет введена на место уже имеющейся строки. Для сохранения имеющейся строки следует после появления символа ' или * нажать клавишу «возврат каретки», после чего команда AUTO будет генерировать следующий номер строки. С помощью этой же клавиши можно пропустить любой автоматически сгенерированный номер, нажав на нее сразу после появления этого номера на экране.

Для прекращения автоматической нумерации строк следует нажать соответствующую клавишу (или одновременно две клавиши) на клавиатуре. Последняя набранная строка при этом не сохраняется.

3. Команды корректировки. Команда RENUM NN, NO, SN перенумеровывает строки программы. Новые номера, первый из которых будет NN, присваиваются операторам текущей программы, начиная со строки с номером NO (старый номер) с шагом SN. Например, команда RENUM 100, 10, 20 осуществляет перенумерацию, начиная со строки 10, которой присваивается новый номер 100, затем номер следующей строки программы заменится на 120 и т. д.

Любой из параметров, задаваемых в команде RENUM, можно опустить. По умолчанию NN принимается равным 10, NO принимается равным номеру первого оператора программы, а SN равным 10.

Действие оператора RENUM распространяется также на номера операторов, встречающихся в операторах управления (GO TO, GOSUB, IF — THEN, ELSE, ON — GO TO, ON — GOSUB). Если в ходе замены старых номеров на новые встретится ссылка на несуществующий номер оператора (несуществующую строку), перенумерация проводиться не будет, а появится сообщение об ошибке («недопустимый вызов функции» или «неопределенный номер строки»). При этом может быть выдано сообщение, в котором указывается номер отсутствующей строки и номер строки, в которой обнаружена ссылка на нее.

Заменить можно также имя программы. Это выполняется командой

RENAME новое имя

Новое имя может быть присвоено, например, измененной версии старой программы, когда ликвидация старой программы нецелесообразна (за нею сохраняется старое имя),

4. **Вывод программы на экран.** После заполнения последней строки экрана изображение автоматически сдвигается вверх, и таким образом верхние строки исчезают. В режиме немедленной обработки исчезнувшие с экрана строки корректировать невозможно. В программируемом режиме исчезнувшие строки можно повторно вывести на экран с помощью команды LIST. Эта команда в случае, когда она состоит только из ключевого слова, выведет на экран всю текущую программу. Если программа содержит большее число строк, чем помещается на экране, то некоторые системы обеспечивают непрерывное движение строк на экране (с регулируемой или нерегулируемой скоростью) снизу вверх (это движение может быть в любой момент прекращено вмешательством с клавиатуры), в других вывод осуществляется с определенным интервалом по кадрам.

В команде LIST можно указать диапазон номеров строк, которые необходимо вывести, набрав на клавиатуре команду LIST N1 — N2. В этом случае выводится фрагмент программы, где номер первой строки есть N1, а последней — N2. Команда в виде LIST N1 — обеспечит вывод строки, начиная с номера N1 до конца программы; команда LIST — N2 выведет строки, начиная со строки с минимальным номером и заканчивая строкой с номером N2; команда LIST N осуществит вывод строки с номером N.

В любом варианте оператора LIST вместо явного номера строки можно поставить точку. При выполнении такой команды LIST под точкой подразумевается текущая строка. Например:

LIST (выводится на экран вся программа),

LIST.— (выводится фрагмент от текущей строки до конца программы),

LIST. (выводится одна текущая строка),

LIST 10 — . (выводится фрагмент, начиная со строки с номером 10 и заканчивая текущей строкой).

Любое количество строк аналогичным образом может быть выведено на печатающее устройство. Для этого предназначена команда LLIST. Способы задания номеров выводимых строк в ней остаются теми же.

В различных версиях Бейсик-систем могут встречаться и другие модификации операторов LIST и LLIST.

5. **Редактирование программы.** В процессе составления программы и ввода ее с клавиатуры может появиться необходимость добавления в программу новых строк или удаления некоторых имеющихся строк. Для замены существующей строки на новую достаточно набрать на клавиатуре новую строку с тем же самым номером, что и у заменяемой и нажать клавишу «возврат каретки». Если между двумя имеющимися строками программы требуется

вставить новую строку, надо присвоить этой строке номер в промежутке между номерами смежных с ней строк. После нажатия клавиши «возврат каретки» строка автоматически размещается в нужном месте. Чтобы такая операция могла быть выполнена без затруднений и не возникала необходимость заново перенумеровать строки, следует первоначально снабжать строки номерами с определенным интервалом (например, через 5 или 10).

Удаление одной строки выполняется также просто — достаточно набрать ее номер и нажать на клавишу «возврат каретки». Удаление нескольких строк удобнее осуществить командой DELETE, формат которой полностью совпадает с форматом команды LIST, только вместо вывода на экран из программы удаляются строки с номерами, задаваемыми списком команды DELETE. Однако могут быть и отличия от формата команды LIST. Так, в Бейсик-системе IBM список команды DELETE допускается только в следующих модификациях: N1 — N2, N и точка. Это ограничение связано с тем, что в этой системе номера строк, сообщаемые в команде DELETE, должны быть явными номерами существующих строк программы. В противном случае будет выдано сообщение об ошибке («недопустимый вызов функции»).

Напомним, что для удаления из памяти сразу всех строк программы может использоваться команда NEW. Эта команда, как отмечалось выше, должна выполняться всякий раз перед вводом очередной программы, иначе ее строки перемешаются со строками старой программы.

Редактирование отдельной строки производится командой EDIT, в которой задается номер строки. Эта команда выводит на экран дисплея строку для редактирования, а курсор устанавливается в позицию первого символа после номера строки. Затем строка может редактироваться (можно заменять символы, удалять или добавлять символы). Если строки с заданным в команде EDIT номером нет в программе, на экране появится сообщение об ошибке («неопределенный номер строки»). Для редактирования только что введенной строки можно использовать точку вместо номера. Например:

```
10 X=3.14
40 Y=X+5.6
   EDIT 40
90 Z=X+Y
   EDIT.
```

В последней строке символ \cdot заменяет номер 90.

6. Выполнение программы. Запуск текущей программы осуществляется по команде RUN. Порядок, в котором выполняются операторы программы, определяется номерами строк. Эта ко-

манда инициирует выполнение программы, начиная со строки с наименьшим номером. Для того чтобы запустить программу с любой другой строки N, после ключевого слова указывается этот номер строки: RUN N.

После выполнения всех операторов программы вычислительная машина вновь переходит в режим немедленной обработки, однако программу можно остановить и раньше с помощью оператора *останова* STOP. По оператору STOP выполнение программы прекращается, выводится сообщение, указывающее строку останова, и система переходит в режим немедленной обработки.

Прервать выполнение программы можно также нажатием соответствующей клавиши (или клавиш) на клавиатуре. При этом, как и при использовании оператора STOP, на экране появится сообщение о номере последнего выполненного оператора.

В обоих случаях прерывания выполнения программы работа ее может быть продолжена с помощью команды CONT.

Нормальное завершение программы происходит при выходе на оператор *окончания* END, который является последним оператором программы или основной программы, если в программе содержатся подпрограммы. После прерывания оператором END не выдается сообщение о номере последнего выполненного оператора.

Прерывание программы происходит также при обнаружении программной ошибки, о чем, как правило, выдается сообщение.

Например, если фрагмент программы имеет следующий вид:

```
40 X = 25.4
50 STOP
60 Y = 1.33
400 END
RUN
```

то после выполнения оператора с номером 50 произойдет останов и на экран будет выведено сообщение "останов в 50". После набора команды CONT выполнение программы продолжится с оператора номер 60 и произойдет останов по оператору END.

После окончания работы управление передается операционной системе командой SYSTEM. При этом закрываются все файлы.

В различных Бейсик-системах этой цели служат и другие команды (BUE и т. д.), а во время выполнения программы реализуются дополнительные возможности, например, сегментация программы (команда CHAIN), указание состояний системы (команда LENGTH), распечатка имен файлов данного пользователя (CATA-LOG) и др.

7. **Запись программы в файл.** Обычно пользователь не работает одновременно с несколькими программами, так что в оперативной памяти в данный момент находится лишь одна программа, следо-

вательно надо иметь возможность хранить другие программы на внешних запоминающих устройствах и при необходимости их оттуда вызывать. В языке Бейсик существует такая возможность, и каждый пользователь может организовать на внешних носителях архив, состоящий из набора файлов с записанными в них программами и каталога архива.

Пересылка программы из оперативной памяти (сформированной там, например, набором с клавиатуры) в заданный файл (чаще всего в дисковый файл) осуществляется с помощью команд **SAVE** и **LIST**, в которых после ключевого слова можно ничего не указывать — тогда на диске сохранится текущая программа со своим именем, или можно указать спецификацию файла (в кавычках), в который записывается программа (если имя файла, в который записывается программа, совпадает с именем файла, уже существующего на указанном дисковом, то старый файл стирается, а на его месте создается новый). Например,

```
LIST
SAVE
LIST "B : TEST. BAS"
SAVE "B : TEST. BAS"
```

Здесь спецификация **B : TEST. BAS** расшифровывается как текст программы на Бейсике (добавка **BAS**), который хранится под именем **TEST** на диске, установленном на дисковом **B**.

Команда **LIST** позволяет пересылать все символы программы на любое внешнее устройство точно так же, как при выводе их на экран дисплея. В частности, с помощью команды **LIST** можно записывать в файл отдельные части программы. Для этого в команде следует задать соответствующий диапазон изменения номеров операторов записываемого фрагмента, например:

```
LIST 10—80, "TEST"
```

При выполнении команды **SAVE** программа всегда записывается целиком. Обычно производится сжатие текста программы за счет сокращения длины ключевых слов до одного символа, и файл будет храниться в двоичном формате. Для исключения операции сжатия следует добавить в конце команды **SAVE** запятую и символ **A** :

```
SAVE "TEST", A
```

Команда **SAVE** позволяет записывать программу в файл в зашифрованном виде, защищая ее тем самым от постороннего вмешательства. Если зашифрованная программа снова пересылается в оперативную память, она становится недоступной для команд **LIST** и **EDIT**. Необходимо помнить, что нет способа снять защиту с такой программы. Чтобы записать программу по команде **SAVE** в зашифрованном виде, следует поставить в конце этой

команды запятой и символ R:

SAVE "TEST", R

9. Вызов из файла. Программа, размещенная на любом внешнем устройстве, например помещенная в дисковый файл с использованием команд LIST или SAVE, извлекается из файла и загружается в оперативную память командой загрузки LOAD.

При выполнении этой команды производится очистка памяти, и новая программа занимает место программы, находившейся в памяти непосредственно перед выполнением команды LOAD. После ключевого слова указывается имя файла (в кавычках). Если в имени файла устройство опущено, используется текущий диск-вод. Например: LOAD "TEST. BAS".

Рассмотренная команда LOAD только загружает программу, но не выполняет ее. Для того, чтобы извлеченная из файла программа была выполнена, следует в команде LOAD после имени файла через запятую указать параметр R. Например: LOAD "TEST. BAS", R. Программа начинает выполняться с первого оператора.

Для загрузки программы и последующего ее выполнения используется также команда RUN, в которой необходимо указать имя файла. Например, RUN "TEST. BAS".

При загрузке программы из файла находящаяся в памяти предыдущая программа может быть сохранена. При выполнении команды MERGE строки программы из заданного файла присоединяются (добавляются) к тексту программы, находящейся в памяти в текущий момент. Например, если в памяти содержалась программа PROGRAM, то команда MERGE "NUMBER" присоединяет строки программы NUMBER к строкам программы PROGRAM.

Если какие-либо строки в присоединяемой программе имеют такие же номера, как и строки программы в памяти, то строки из файла будут замещать соответствующие строки в программе, находящейся в памяти.

Команду MERGE можно использовать только для программ, записанных в файл с помощью команды SAVE с использованием символа A. Так, в предыдущем примере программа NUMBER должна быть записана в файл командой SAVE "NUMBER", A, в противном случае появится сообщение об ошибке («неверен режим файла») и программа в памяти останется неизменной.

Если при выполнении команд LOAD, RUN или MERGE вычислительной машине не удастся найти указанный файл, то выдается сообщение об ошибке («неверное имя файла»).

9. Операции над файлами. В ряде случаев бывает необходимо получить информацию об имеющихся на данном устройстве (диске) именах файлов, С помощью команды, состоящей из одного ключа

чего слова **FILES** распечатываются имена всех файлов на текущем выбранном устройстве.

Можно выводить имена файлов любого заданного дисковогода и даже получать справочную информацию только об интересующих файлах. В этих случаях необходимо задавать в команде **FILES** идентификатор дисковогода, имя файла, а возможно, то и другое; эти данные указываются в кавычках после ключевого слова **FILES**, при этом символ ***** соответствует любому имени или расширению имени. Например, по команде **FILES "*.BAS"** распечатываются все файлы с расширением **BAS** в их именах на текущем устройстве; команда **FILES "B:"** или **FILES "B : *"** выводит имена всех файлов без расширений на устройстве (дисковомде) **B**; команда **FILES "B : *.*"** выдает информацию о всех файлах (с любым именем) на дисковомде **B**; команда **FILES "*.*"** обеспечит получение информации вообще о всех файлах.

Команда **KILL "имя файла"** позволяет удалить файл с диска. Эта команда может быть использована для всех типов файлов. В имени файла может быть указан идентификатор дисковогода. Если в этой команде задается имя файла, открытого в текущий момент, то выдается сообщение об ошибке («файл уже открыт»). Например, **KILL "A : DATA1. BAS"**.

Имя любого файла может быть изменено на новое. Для этого используется команда

NAME "старое имя" AS "новое имя"

где "старое имя" и "новое имя" — строковые выражения для обозначения имени файла. Файл со старым именем должен существовать на диске, а нового имени не должно быть на диске, иначе появится сообщение об ошибке («неверное имя файла»). Если файл находится не на принимаемом по умолчанию текущем дисковомде, то перед старым именем файла следует поставить идентификатор нужного дисковогода. Любой идентификатор, стоящий впереди нового имени файла, игнорируется. Например,

NAME "A : TEST. BAS" AS "ACTS. BAS"

После выполнения этой команды файл существует на том же диске **A** в той же области дискового пространства с новым именем **ACTS. BAS**

10. Отладка программы. После того как составлена программа, как правило, возникает необходимость в проверке правильности ее функционирования. Допущенные ошибки, особенно если они синтаксические, да и многие семантические, выясняются при трансляции. Сложнее с арифметическими ошибками и ошибками в логике программы. Обычно выдача результатов или выход программы на запланированный останов операторами **STOP** или **END** еще не означают, что программа работает правильно. Необходимо ее

тщательное тестирование при различных наборах начальных данных и в ситуациях, когда результат известен.

Правильность функционирования программы можно установить оценкой промежуточных результатов и проверкой логики программной реализации алгоритма путем включения в программу операторов STOP в различных «горячих» точках. После этого в режиме немедленной обработки, пользуясь операторами вывода или другими операторами, можно получить ответы на интересующие программиста вопросы. Промежуточные «отладочные» операторы могут исполняться и в программируемом режиме, при этом отпадает необходимость в операторах останова перед ними.

В то же время такой путь проверки логики выполнения программы, особенно установления порядка обращений к подпрограммам и к вложенным циклам, выявления количества повторений в различных циклах затруднен. Для получения максимальной информации о названных здесь операциях в ходе выполнения программы и для решения других вопросов, связанных с правильной работой программы и локализацией ошибок, предусмотрен специальный отладочный режим, называемый *трассировкой*, распространяемый либо на всю программу, либо на ее часть.

Режим трассировки в процессе выполнения программы означает фиксацию номеров строк программы в той последовательности, в которой они реально исполнялись. Цепочка этих номеров оставляет след (или трассу) работы программы. Трасса может быть сразу выведена на экран или в специальный отладочный файл, содержимое которого в любой момент доступно для просмотра. При трассировке для операторов условного перехода фиксируются номера текущей и следующей строки, если переход состоялся, а для операторов присваивания и ввода фиксируются идентификаторы соответствующих переменных и их значения. Обычно этой информации достаточно для оценки правильности вычислительных схем и выявления всех переходов в программе.

Режим трассировки обеспечивается командой TRON, которая может быть включена и в программируемом режиме. По команде TRON будет печататься каждый номер строки программы, которая выполняется. Номера при выводе заключаются в квадратные скобки. Например, при выполнении программы, в которой реализуются циклические вычисления, подробно рассматриваемые ниже:

```
10 K=10
20 FOR J=1 TO 2
30 L=K+10
40 PRINT J; K; L
50 K=K+10
60 NEXT
70 END
```

TRON RUN

команда TRON обеспечивает вывод следующей информации:

[10]	[20]	[30]	[40]	1	10	20
[50]	[60]	[30]	[40]	2	20	30
[50]	[60]	[70]				

где номера выполняемых операторов заключены в квадратные скобки, а после номера оператора вывода [40] указываются значения переменных J, K, L как результат исполнения этого оператора.

Для отмены режима трассировки служит команда TROFF, которая также может использоваться в программируемом режиме. Любая команда, удаляющая текущую программу из памяти (NEW, LOAD или другие в различных системах), отменяет также режим трассировки. Однако команда RUN в ее простейшем виде не отменяет этого режима, так что использование команды TRON в режиме немедленной обработки приведет к выполнению всех последующих программ до тех пор, пока не встретится оператор TROFF.

Упражнения

1. Выполнить автоматическую нумерацию строк программы, состоящей из основной программы и двух подпрограмм. Начальный номер строки в основной программе 10, шаг изменения номеров — 20; подпрограммы начинаются соответственно со строк с номерами 500 и 800 с шагом 10.

2. Строки программы имеют номера, начиная с номера 15 с шагом 10 до номера 155. Можно ли при редактировании этой программы пользоваться командами

```
LIST — 75  
DELETE 110—135  
LIST 105  
DELETE 100
```

3. Допустимы ли команды

```
NAME "B : TRAC" AS "CART. BAS"  
NAME "B : TRAC" AS "A : CART"
```

§ 14. Безусловная передача управления

Практически во всех программах естественный порядок выполнения операторов в соответствии с возрастанием номеров строк так или иначе изменяется. Это делается в связи с тем, что некоторые операторы должны быть повторно выполнены или же некоторые из них в ряде случаев пропущены.

1. Оператор GO TO. Изменение порядка выполнения операторов и передача управления любой заранее заданной строке программы

осуществляется оператором безусловного перехода

GO TO N

где N — номер строки. Если N — номер исполняемого оператора, то выполняется этот оператор и следующие за ним, а если это неисполняемый оператор, то выполнение продолжается с первого исполняемого оператора, расположенного после заданного номера строки N.

Номер строки N в операторе GO TO задается в виде числовой константы (а не переменной или выражением). Если в константе встречается точка, то она игнорируется вместе со всеми следующими за ней десятичными цифрами. Не разрешается использовать в качестве номера строки число в экспоненциальном представлении.

Например, в следующем фрагменте программы оператор GO TO обеспечивает трехкратное исполнение операторов с выводом на печать, после чего будет выдано сообщение об ошибке («нет данных»):

```
10 DATA 5, 7, 12
20 READ R
30 PRINT "R="; R;
40 S=3.1416*R^2
50 PRINT "S="; S
60 GO TO 10
RUN
R=5          S=78.5
R=7          S=153.86
R=12         S=452.16
НЕТ ДАННЫХ
```

Если в программе отсутствует номер строки, на который передается управление, то выдается сообщение об ошибке («неопределенный номер строки»). В связи с этим следует заметить, что надо быть внимательным при передаче управления на операторы, используемые при отладке, или на комментарии, которые после окончания отладки из программы удаляются.

Использование ключевого слова GO TO в режиме немедленной обработки аналогично выполнению команды RUN, однако выполнение программы начинается с первого оператора указанной в команде GO TO строки, не присваивая никаких начальных значений переменным и не меняя никаких данных. Это может быть использовано при отладке и тестировании. В остальных случаях использовать в режиме немедленной обработки команду GO TO не рекомендуется, поскольку существует большая вероятность вводить неправильный номер строки или непреднамеренно изменить значение переменной так, что фактически изменится вся программа, и это приведет к появлению многочисленных ошибок,

2. Оператор ON — GO TO. Если заранее известно, что в зависимости от сложившейся в ходе выполнения программы ситуации управление должно быть передано в различные точки программы, то используется оператор

ON E GO TO список номеров

где E — числовое выражение, а список номеров содержит произвольное число номеров строк программы, разделенных запятыми. Действие оператора ON — GO TO заключается в передаче управления на оператор с номером из списка номеров, порядковый номер которого равен текущему значению выражения E.

Таким образом, значения выражения E должны быть заключены между 1 и значением, равным числу элементов в списке номеров. Дробные значения выражения округляются до ближайшего целого числа. Если значение этого выражения равно 0 или больше числа элементов в списке номеров, то оператор не выполняется, и управление передается следующему за ним оператору. При отрицательном значении выражения выдается сообщение об ошибке («недопустимый вызов функции»). Например, при выполнении фрагмента программы

```
70 INPUT N
80 ON N GO TO 100, 110
90 PRINT "НОМЕР НЕ ОПРЕДЕЛЕН": GO TO 70
100 PRINT "ЗАДАНИЕ 1": GO TO 700
110 PRINT "ЗАДАНИЕ 2": GO TO 780
```

если значение N не равно 1 или 2, выполняются операторы в строке номер 90 и требуется ввод нового значения N; если же N равно 1 или 2, выполняются операторы в строках с номерами 100 и 110 соответственно.

Упражнения

1. Написать фрагмент программы, который производил бы последовательно вычисления значения y по следующим формулам:

$$\frac{x^3 + x^2 - 1}{\sin x + x + 2}, \quad \frac{x^2 + 1}{\sin^2 x + 1}, \quad \frac{x - 1}{\sin x + \cos x}$$

при заданном значении x , а для вывода результата использовался бы только один оператор.

2. Не меняя последовательности расположения пяти операторов присваивания, содержащихся в программе, обеспечить их выполнение в следующем порядке: пятый, второй, первый, третий, четвертый.

3. Составить операторы, которые обеспечивают переход к строкам программы с номерами 70, 90, 110 в следующей периодической последовательности: 70, 90, 70, 110, 90, 110, 70, 90.

§ 15. Условная передача управления

В отличие от оператора ON — GO TO, в котором явно не указывается условие формирования значения выражения (по этой причине этот оператор относится к операторам безусловной передачи управления), в операторах условной передачи управления условие, определяющее, когда выполняются те или иные действия, содержится непосредственно в конструкции оператора. В языке Бейсик имеется два оператора условной передачи управления; *условный* оператор (оператор IF — THEN) и *полный условный* оператор (IF — THEN — ELSE).

1. Условный оператор. Предписание выполнить какое-либо действие только в том случае, когда выполняется заданное условие, указывается оператором

IF L THEN S

где L — выражение, принимающее логическое значение «истина» (не нуль) или «ложь» (нуль); S — один оператор или несколько операторов, разделенных двоеточием.

Действие оператора заключается в следующем: если L — «истина», выполняется S, в противном случае, т. е. когда L — «ложь», S пропускается. Например:

```
130 INPUT "ДВА ЧИСЛА", A, B
140 IF A>B THEN PRINT "A>B"
150 IF B>A THEN PRINT "A<B"
160 IF A=B THEN PRINT "ЧИСЛА РАВНЫ": GO TO 130
```

Если S представляет собой оператор GO TO, то можно опустить либо ключевое слово THEN, либо ключевое слово GO TO, но не оба сразу. Например, вместо

```
50 IF X<Y AND B<0 THEN GO TO 500
```

можно написать один из следующих вариантов:

```
50 IF X<Y AND B<0 THEN 500
50 IF X<Y AND B<0 GO TO 500
```

В последовательность операторов S, определяющих конструкцию IF — THEN, как было отмечено выше, может входить любой оператор; если при этом используется оператор GO TO, то он, очевидно, будет последним реально выполняемым. Например, перестановка последнего оператора после THEN в строке

```
90 IF A%=14 THEN, E$="АЗОТ": M=28: GO TO 40
```

недопустима.

Пр и м е р. Программа вычисления значения N! может быть записана в виде

```
10 READ N
20 F=1 : K=1
30 F=F*K
40 IF K<=N THEN 30
50 PRINT N, F
60 IF N<7 GO TO 10
70 DATA 3, 5, 7
80 END
```

2. Полный условный оператор. Условный оператор может быть дополнен ключевым словом ELSE, после которого указываются действия, подлежащие выполнению, если выражение E этого оператора имеет значение «ложь». В результате в одном операторе, называемом полным условным оператором, указаны все действия, выполняемые при любом значении выражения. Формат этого оператора следующий:

```
IF L THEN S ELSE S1
```

где L и S имеют тот же смысл, что и выше, а структура и особенности S1 те же, что и у S; S1 выполняется, если L имеет значение «ложь». Например:

```
40 IF M<>N THEN 10 ELSE PRINT "РАВЕНСТВО",
M : GO TO 20
```

Операторы IF — THEN — ELSE могут быть вложены. Вложение ограничивается только длиной строки дисплея. Допускается несовпадение количества ключевых слов THEN и ELSE; в этом случае каждому ELSE соответствует THEN, закрывая ближайший оператор IF. Например:

```
60 IF A=B THEN IF B=C THEN PRINT "A=C"
ELSE PRINT "A< >C"
```

В этом примере один оператор IF вложен в другой; на два ключевых слова присутствует лишь одно ELSE, что приводит к тому, что не будет печататься строка "A< >C", когда A{_i)B, т. е. первому в этой строке слову THEN не соответствует ELSE.

Упражнения

1. Вычислить последовательные значения F как функции переменной x, меняющейся с шагом 0,01 от значения x=0 до x=5. Напечатать значения F вместе с соответствующим значением аргумента и порядковым номером вычисленного значения. Функция F задается формулой

$$F = F_0 + F_1 \frac{x}{1!} + F_2 \frac{x^2}{2!} + F_3 \frac{x^3}{3!} + \dots$$

$$F_0 = 1, \quad F_1 = 0,5, \quad F_2 = 0,3, \quad F_3 = 0,1,$$

2. Вычислить F_x и F_y по следующим формулам: $F_x = x + \sin^2 y$, $F_y = y - \cos^2 x$, но при выполнении условия $x > y$ учесть, что F_x вычисляется по второй формуле, а F_y — по первой.

3. Вычислить сумму квадратов сумм отрицательных и положительных элементов последовательности a_1, a_2, \dots, a_k . Результат вывести на печать.

4. Вычислить

$$y = \begin{cases} a_{11}x^2 + a_{12}x + a_{13}, & \text{если } k=1, \\ a_{21}x^2 + a_{22}x + a_{23}, & \text{если } k=2, \\ a_{31}x^2 + a_{32}x + a_{33}, & \text{если } k=3, \end{cases}$$

и вывести на печать значение y с соответствующими коэффициентами a_{ij} ($i, j=1, 2, 3$) и k .

5. Составить программу, которая проверяет, можно ли построить параллелограмм из отрезков с длинами a, b, c, d . Предполагается, что все четыре величины заданы и положительны.

§ 16. Организация циклов

Циклические операции, т. е. выполнение одних и тех же действий многократно, возможно, при измененных каждый раз значениях входящих в операторы переменных, возникают в различных программах довольно часто. Как правило, многократно выполняются отдельные группы операторов, причем вполне определенное число раз. Таким образом, при организации циклических операций необходимы средства, позволяющие выделять повторяемые группы операторов, считать число повторений и в нужный момент прекратить выполняемые операции, а также обеспечить изменение значений определенных переменных при повторении операций.

Как было показано на примере вычисления значения $N!$ в предыдущем параграфе, *цикл* в программе можно организовать с помощью оператора условного перехода, содержащего условие окончания цикла, однако приходится использовать также оператор, присваивающий счетчику начальное значение и оператор, увеличивающий значение счетчика. В алгоритмическом языке Бейсик предусмотрены специальные операторы, включающие все необходимые действия по организации циклов.

1. Цикл **FOR — NEXT**. Эта разновидность цикла представляется двумя операторами: **FOR** и **NEXT**. Оператор **FOR** записывается в виде

FOR I=E1 TO E2 STEP E3

где I — простая числовая переменная, называемая параметром цикла; $E1, E2, E3$ — числовые выражения, определяющие соответственно начальное значение, конечное значение и шаг изменения параметра цикла, ключевые слова **TO** и **STEP** — разделители.

За оператором FOR располагаются операторы, образующие тело цикла, т. е. последовательность операторов, которая выполняется для каждого значения параметра цикла.

Тело цикла завершается оператором NEXT I.

При выполнении цикла FOR — NEXT в первую очередь вычисляются (если это необходимо) и запоминаются начальное значение, конечное значение и шаг изменения параметра цикла. Параметру цикла присваивается значение E1. Выполняются программные строки, образующие тело цикла. Параметр цикла изменяется на шаг (при $E3 > 0$ увеличивается, при $E3 < 0$ — уменьшается) и проверяется условие, не превосходит ли текущее значение параметра I конечное значение E2 (в случае положительного шага) или не стало ли значение параметра меньше E2 (при отрицательном шаге E3). Если значение параметра цикла содержится в интервале между начальным и конечным значениями, повторно выполняются операторы в теле цикла, если не содержится — управление передается на оператор, стоящий в программе после оператора NEXT, т. е. происходит выход из цикла.

Если шаг равен 1, то конструкция STEP 1 может быть опущена. После ключевого слова NEXT указание параметра не обязательно. Например, с использованием цикла FOR — NEXT программа вычисления N! примет вид

```
10 READ N : F=1
20 FOR K=1 TO N
30 F=F*K
40 NEXT
50 PRINT N, F : IF N<7 GO TO 10
60 DATA 3, 5, 7
70 END
```

Переменная, являющаяся параметром цикла, может использоваться внутри цикла точно так же, как любая другая числовая переменная, в частности, можно изменять ее значение, что, однако, может повлиять на число проходов цикла, так как оператор NEXT всегда дает приращение самому последнему значению параметра.

При выходе из цикла по какому-либо оператору условного или безусловного перехода значение параметра равно последнему значению внутри цикла. В случае нормального завершения цикла, т. е. при выходе из цикла через оператор NEXT, значение параметра равно последнему его значению плюс величина шага изменения цикла.

В отличие от параметра цикла начальное и конечное значения параметра цикла и шаг его изменения являются неизменными в течение работы цикла. Изменение значений переменных внутри

цикла не влияет на значения E1, E2, E3, которые были вычислены в начале цикла.

Разрешается использование циклов в цикле (вложенных циклов). В этом случае внутренний цикл должен полностью находиться в теле внешнего цикла. Глубина вложений зависит от типа ЭВМ и объема памяти, который находится в распоряжении пользователя. Например, допустимы циклы

```
100 FOR I=1 TO 5
150 FOR J=0 TO -10 STEP -2
200 FOR K=5.5 TO 10.5 STEP .5
250 NEXT K
300 NEXT J
350 FOR L=2 TO 20 STEP 2
400 NEXT L
410 NEXT I
```

Здесь последние два оператора могут быть объединены в один: 400 NEXT L, I или проще: 400 NEXT, т. е. у двух циклов, заканчивающихся в одной точке программы, конец может быть задан одним оператором.

Следует отметить, что такое совмещение нескольких операторов NEXT может ограничить возможности циклов, а опускание параметра после ключевого слова NEXT привести к ошибке. Дело в том, что оператор NEXT без указания параметра будет относиться к последнему выполненному оператору FOR, а поскольку во вложенных циклах возможны передачи управления из внутреннего цикла во внешний (но не наоборот, за исключением передачи непосредственно на внутренний оператор FOR), то может оказаться нарушенным соответствие пар операторов FOR и NEXT. Например, при выполнении программы

```
10 DIM M$(2,10)
20 FOR J=1 TO 2
30 FOR K=1 TO 10
40 INPUT "NAME:", M$(J, K)
50 IF M$(J, K)=" " THEN 70
60 NEXT
70 NEXT
```

будет выдано сообщение об ошибке ("NEXT без FOR").

В приведенной программе оператор NEXT в строке номер 60 является концом внутреннего цикла, начинающегося в строке 30. Однако, если в строке 50 произойдет передача управления на строку 70, оператор NEXT строки 70 будет воспринят как последний оператор внутреннего цикла. Но строка 70 формально является последней строкой внешнего цикла, начинающегося в строке 20. Поскольку при указанной передаче управления строка

70 интерпретируется как завершающая внутренний цикл, внешний цикл окажется без завершающей строки с оператором NEXT.

2. Цикл WHILE — WEND. В ряде случаев использование только что рассмотренного оператора цикла может быть затруднительным. Это связано с тем, что иногда возникает необходимость изменять число проходов цикла в процессе выполнения операторов тела цикла, либо прерывать процесс выполнения цикла в зависимости от сложившейся ситуации. В подобных случаях обычно в цикл вводится условный оператор, который при выполнении определенных условий передает управление за пределы цикла. Такая ситуация была реализована во внешнем цикле в программе вычисления значения $N!$, рассмотренной выше.

Если заранее неизвестно число проходов цикла, не задан однозначно шаг изменения параметра цикла или выполняется цикл с нестандартными, нерегулярными, непредсказуемыми условиями завершения, то удобнее пользоваться не условными операторами внутри цикла, а специальными операторами WHILE и WEND, которые обеспечивают работу цикла с окончанием по условию.

Оператор WHILE, с которого начинается в программе цикл WHILE — WEND, состоит из ключевого слова WHILE и выражения, заданного с помощью отношений и логических операций, т. е. имеет вид WHILE L. Когда выражение L принимает значение «истина», выполняются операторы, следующие за оператором WHILE и составляющие тело цикла, вплоть до оператора WEND. Как только встречается оператор WEND, управление передается на оператор WHILE. При повторном выполнении оператора WHILE заново вычисляется содержащееся в нем выражение и, если оно по-прежнему имеет значение «истина», снова выполняются операторы тела цикла. Как только выражение в операторе WHILE будет иметь значение «ложь», управление передается оператору, следующему за WEND.

Например, внутренний цикл в программе вычисления $N!$ может быть переписан так:

```
20 F=1 : K=1
30 WHILE K<=N
40 F=F*K
50 K=K+1
60 WEND
```

В операторе WHILE могут также использоваться числовые выражения. В этом случае выполнение цикла повторяется до тех пор, пока значение такого выражения отлично от нуля. Например, в предыдущем примере строку с номером 30 можно переписать так:

```
30 WHILE N+1-K
```

Циклы WHILE — WEND могут быть вложены друг в друга, при этом каждому оператору WHILE должен соответствовать свой оператор WEND, иначе будет выдано сообщение об ошибке ("WHILE без WEND").

В различных версиях Бейсика при организации циклов с окончанием по условию может быть использован другой формат. Например, в системе Бейсик-плюс допускаются следующие два оператора цикла:

```
FOR I=E1 STEP E3 WHILE L
FOR I=E1 STEP E3 UNTIL L
```

В этих операторах использованы принятые в данном параграфе обозначения. Если шаг E3 равен 1, то конструкция STEP 1 опускается. Логическое выражение вычисляется перед выполнением цикла и при каждом повторении цикла. Цикл FOR — WHILE обеспечивает повторное выполнение операторов тела цикла (т. е. до первого оператора NEXT) до тех пор, пока значения L суть «истина», а цикл FOR — UNTIL — до тех пор, пока L имеет значение «ложь».

Упражнения

1. Составить программу для вычисления вещественного корня уравнения $x - \sin x = 0.35$ с заданной абсолютной погрешностью, равной 0.001.

2. Найти значения положительного кубического корня нечетных чисел от 1 до 99, т. е. составить программу для решения уравнения $x^3 = a$ при $a = 1, 3, \dots, 99$, используя итерационную формулу $x_n = \frac{1}{3} \frac{a}{x_{n-1}^2} + \frac{2}{3} x_{n-1}$. В качестве первого приближения для $\sqrt[3]{a_i}$ использовать значение $\sqrt[3]{a_{i-1}}$. Точность: 3 десятичных знака.

3. Составить программу вычисления e^x для заданного x , просуммировав 20 членов ряда $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$

4. Составить программу вычисления e^x путем разложения в ряд, приведенный в упражнении 3. Прекратить вычисления после того, как очередной член окажется меньше чем число 10^{-6} , умноженное на частичную сумму, полученную до этого.

5. Составить программу для упражнения 3 из § 3 гл. I.

6. Найти наименьший и наибольший (по абсолютной величине) элемент в массивах: а) X(25, 75) б) Y(25, 75) в) Z(10, 20, 30).

Пр и м е ч а н и е. В этом, как и в последующих упражнениях, предполагается составление программы.

7. Напечатать одномерный массив Y значений функции $y(x) = x^3 / (5x^2 + 3)^2$ для аргумента x , изменяющегося следующим образом: от 0 до 1 — с шагом 0.1, от 1 до 10 — с шагом 1, от 10 до 50 — с шагом 5, от 50 до 100 — с шагом 10. Вместе со значением Y(I) напечатать значение номера I.

8. В двумерном массиве поменять местами строки с номерами I и J.

§ 17. Подпрограммы

1. Структурирование программ. Вероятность появления ошибок в программе, особенно сложной и длинной, очень велика, и организация ее функционирования представляет собой трудную задачу. Бессистемное написание операторов, наличие множества переходов, вложенных и рекурсивных процедур иногда делают программе практически необозримой и недоступной для структурного анализа. Поскольку наглядность программы, как правило, пропорциональна ее надежности, возникает необходимость разбиения программы на части ради упрощения ее структуры. Это может быть обеспечено простейшим путем — выделением в программе обособленных разделов по их функциональному назначению. Например в начале программы могут быть размещены различные описания и определения, затем, во второй ее части, — процедуры инициализации, в третьей — ввод данных, после чего следует обработка и, наконец, вывод данных.

Для того чтобы в целостной программе отделить один раздел от другого, можно использовать в каждом из них свой диапазон номеров строк, достаточно удаленный от диапазонов других разделов. Если при написании программы выбрать некоторый стандартный способ нумерации строк для различных ее частей, то впоследствии легко и удобно находить строки с операторами определенного класса (например, DIM, DEF, DATA и т. д.).

Приведенное выше разбиение программы на части не является единственным, возможны и другие способы выделения различных функциональных разделов программы. Далее полученные программные разделы можно в свою очередь подразделять на модули, а модули разбивать на отдельные сегменты и т. д. Конечная цель состоит в том, чтобы полученные в результате окончательного разбиения элементарные сегменты были достаточно малы и их можно было бы легко разрабатывать, программировать и отлаживать. Задача состоит также в том, чтобы из этих последовательно выполняемых сегментов можно было составить целостную программу, в которой отсутствовали бы сложные передачи управления на предыдущие участки программы. Не исключено, что при таком подходе некоторые сегменты будут повторяться, однако преимущества стройной, наглядной структуры программы компенсируют с лихвой отдельные повторы незначительного числа модулей или сегментов. Дополнительное преимущество такого *структурированного* подхода состоит в том, что многие из полученных модулей или сегментов можно будет использовать как некие «суперкоманды» в других программах.

2. Подпрограммы. В различных местах одной и той же программы многократно может выполняться одна и та же процедура, записанная определенным числом строк программы (например, про-

цедура вычисления N1). Переписывать каждый раз такую группу операторов заново неэкономно. Гораздо выгоднее выделить операторы данной процедуры из основной программы и при необходимости передавать ей управление из основной программы. Очевидно, после выполнения выделенных операторов управление должно быть передано в то место основной программы, где содержится оператор перехода к данной процедуре.

Такие процедуры называются *подпрограммами*, а передача к ним управления — *вызовом* подпрограммы. Оператор, который в процедуре выполняется последним и обеспечивает передачу управления обратно в основную программу, называют оператором *возврата*.

Вызов подпрограммы осуществляется оператором

GOSUB N

где N — номер первой строки подпрограммы. Действие этого оператора заключается в передаче управления на первый оператор в строке с номером N, т. е. как и в случае оператора GO TO, однако запоминается еще точка вызова, в которую необходимо вернуться после окончания работы вызванной подпрограммы. Завершает работу подпрограммы оператор RETURN. При его выполнении управление передается в основную программу на оператор, стоящий за оператором GOSUB, инициировавшим вызов подпрограммы.

Например, для вычисления значений вещественных корней двух квадратных уравнений вида $ax^2+bx+c=0$ может быть записана следующая программа:

```
10 N=1
20 READ A, B, C
30 GOSUB 90
40 PRINT "X1="; X1, "X2=" ; X2
50 N=N+1
60 IF N>2 THEN 80 ELSE 20
70 DATA 1.4, 5.2, .7, 2.6, 4.8, 1.1
80 END
90 R=SQR (B^2-4*A*C)
100 X1=(-B+R)/(2*A)
110 X2=(-B-R)/(2*A)
120 RETURN
```

В одной подпрограмме операторов возврата может быть несколько, однако срабатывает только один из них. Выход из подпрограммы разрешен также операторами передачи управления (GO TO, IF—THEN). Кроме того, для нестандартного выхода из подпрограммы на конкретную строку программы в расширенном Бейсике предусмотрена возможность задавать в операторе RETURN

номер строки. Например, оператор

440 RETURN 130

передает управление на строку 130 программы.

Поскольку оператор GOSUB обеспечивает предварительное запоминание точки возврата, выполнение оператора RETURN без предварительной передачи управления на подпрограмму оператором GOSUB невозможно, и фиксируется ошибка, хотя операторы подпрограммы могут выполняться и без передачи на них управления оператором GOSUB (в естественной последовательности при отсутствии в конце основной программы оператора END, а также после срабатывания операторов GO TO или IF — THEN и др.).

Среди операторов выполняемой подпрограммы в свою очередь могут содержаться операторы GOSUB, т. е. одна подпрограмма будет вызывать другую, а та, возможно, третью и т. д. Таким образом, вызовы подпрограмм оказываются вложенными друг в друга, хотя тексты подпрограмм представляют собой последовательные модули в программе.

Допускаются рекурсивные обращения, реализуемые в том случае, когда подпрограмма обращается к самой себе. Понятие рекурсии охватывает также и ситуацию, при которой первая подпрограмма вызывает вторую, а та в свою очередь вызывает первую. Число обращений подпрограммы к себе самой определяет глубину рекурсий, которая ограничена и зависит от используемой версии Бейсика, количества других вложенных одна в другую подпрограмм, выполняемых в этот же момент времени, и от размера рабочего стека — участка оперативной памяти для хранения списка адресов возврата из подпрограмм. Как правило, размеры стандартного стека достаточны для решения большинства задач, встречающихся на практике.

Вызов подпрограммы осуществляется также оператором

ON E GOSUB список номеров

где по аналогии с оператором ON — GO TO управление передается на оператор, метка которого в списке номеров имеет порядковый номер, равный текущему значению выражения E. Возврат из подпрограммы осуществляется на оператор, занимающий следующую после оператора ON — GOSUB строку программы.

3. Сцепление программ. Если программа настолько велика, что целиком не помещается в оперативной памяти, ее следует разделить на сегменты. Каждый сегмент в этом случае хранится на диске как самостоятельная программа, но выполняет лишь часть общей задачи. Когда текущий сегмент завершает работу, происходит загрузка очередного сегмента, предназначенного для выполнения следующей части задачи, и управление передается этому новому программному сегменту. Отдельный такой программный сегмент называется *оверлеем* (от английского слова *overlay* — наложение, перекрытие, каж-

дый вновь поступающий сегмент как бы накладывается на сегмент, находящийся в данный момент в памяти), а описанная структура программы — оверлейной.

Соединение (*сцепление*) текущего сегмента программы со следующим осуществляется оператором

CHAIN "имя файла"

где строковое значение "имя файла" в общем случае содержит имя некоторой программы на Бейсике с указанием имени файла, в котором она находится и имени дисковогода. Например, оператор

100 CHAIN "A : PROG1.BAS"

загружает в оперативную память новый сегмент (программу) с заданным именем A: PROG1.BAS, в котором A — метка дисковогода, BAS — расширение имени. Выполнение этого сегмента начинается с первого оператора.

В рассматриваемом операторе после имени файла можно указать второй параметр — номер строки или выражение, вычисляющее номер строки в вызванной программе. Это начальная точка, с которой происходит выполнение новой программы. На этот параметр не действует команда RENUM. Например:

100 CHAIN "A : PROG1.BAS", 1000

Оверлейная структура программы позволяет очень эффективно использовать имеющийся объем оперативной памяти.

4. Общие переменные. При выполнении оператора CHAIN каждый раз перед загрузкой следующего сегмента теряется из памяти предыдущий сегмент, т. е. удаляются строки программы и значения переменных. В то же время при необходимости могут быть сохранены переменные предыдущего сегмента, если их идентификаторы перечислить в списке оператора *общих переменных* с ключевым словом COMMON. В этом операторе перечисляются имена простых переменных или массивов (после идентификатора массива указывается пара круглых скобок), но не отдельные переменные с индексами. Например:

150 COMMON A, X(), B, C, Y()

Оператор COMMON является неисполняемым и может располагаться в любом месте программы (предыдущего сегмента), хотя рекомендуется использовать в ее начале вместе с операторами DEF и DIM. Одна и та же переменная не может появляться в пределах одного программного сегмента более чем в одном операторе COMMON.

После загрузки нового сегмента описания типов переменных, заданные в предыдущем сегменте операторами DEF, теряют силу. Чтобы значения общих переменных были сохранены, при необходимости следует повторить описания типов в новом сегменте.

Например, если в некотором сегменте содержатся операторы

```
10 DEFINT U
20 COMMON $$, U
```

то значения обеих переменных \$\$ и U сохранятся в новом сегменте лишь в том случае, когда там будет стоять оператор DEFINT U, в противном случае идентификатор U воспримется как имя переменной обычной точности и значение ее из предыдущего сегмента, несмотря на оператор COMMON, будет потеряно.

Для сохранения всех переменных при переходе от одного сегмента к другому можно использовать оператор CHAIN, в котором на третьем месте в списке этого оператора указывается параметр ALL. Оператор COMMON в таком случае не используется. Например:

```
450 CHAIN "PROG", 10, ALL
```

Сохранение переменных с помощью оператора CHAIN с параметром ALL связано с теми же проблемами переопределения типов переменных, что и в случае оператора COMMON. Ни один из описанных методов сохранения переменных не распространяется на функции, определяемые пользователем, т. е. задаваемые оператором DEF FN.

5. Совместное выполнение сегментов. Оператор CHAIN можно использовать для присоединения к находящемуся в памяти сегменту строк другого сегмента, т. е. в этом случае строки нового сегмента не замещают полностью прежний сегмент, а как бы вставляются в него. Если при этом номер некоторой загружаемой строки нового сегмента совпадает с номером какой-либо строки сегмента, уже находящегося в памяти, старая строка полностью заменяется на новую. Для того чтобы два сегмента совместно выполнялись, оператор CHAIN следует писать следующим образом

```
CHAIN MERGE "имя файла"
```

Например:

```
200 CHAIN MERGE "FULL"
```

Чтобы загружать сегменты с помощью оператора CHAIN MERGE, необходимо предварительно выполнить команду SAVE с буквой A в конце, указав в ней нужные сегменты. Например,

```
SAVE "FULL", A
```

В операторе CHAIN MERGE может быть указан параметр ALL. Например:

```
200 CHAIN MERGE "FULL", ,ALL
```

Дополнительная запятая перед ALL резервирует место для номера начальной строки. В данном примере этот параметр отсутствует, следовательно новый сегмент с именем "FULL" выполняется, начи-

ная с первой строки. С предыдущего сегмента будут сохранены все переменные.

Включение в оператор CHAIN MERGE параметра DELETE позволяет исключить строки, номера которых лежат в заданном после слова DELETE диапазоне. Например, оператор

200 CHAIN MERGE "FULL", ,DELETE 750—900
при загрузке программы "FULL" исключает все строки, начиная с номера 750 и заканчивая номером 900. Отметим, что операторы с номерами, являющимися граничными в задаваемом диапазоне, должны в программе присутствовать, в противном случае будет выдано сообщение об ошибке («неопределенный номер строки»). Если после слова DELETE не задан последний номер диапазона, то удаляется лишь одна строка с номером, указанным после слова DELETE.

Если в операторе CHAIN MERGE одновременно с параметром DELETE используется параметр ALL, то параметр ALL указывается первым. Например:

210 CHAIN MERGE "FANNY", ,ALL, DELETE 400

Упражнения

1. Составить подпрограмму нахождения четных значений элементов, расположенных на главной диагонали и выше для квадратной матрицы n -го порядка с целыми элементами. Предусмотреть ввод всех элементов и вывод найденных значений с указанием номеров соответствующих строки и столбца и общего их количества.

2. Поменять столбцы матрицы размером $m \times n$, состоящей из элементов x_{ij} так, чтобы элементы заданной строки убывали по модулю. В упорядоченной матрице запомнить исходные номера столбцов. Составить подпрограмму для произвольного i и выдать результаты на печать для i , равного 1.

3. Написать подпрограмму определения наименьшего по абсолютной величине элемента одномерного массива произвольного размера и использовать ее для вычисления минимального элемента среди найденных в случае k различных одномерных массивов.

4. Написать подпрограмму определения наибольшего из элементов прямоугольной матрицы.

5. Составить подпрограмму вычисления среднего квадратического для k значений элементов массива A по формуле
$$m = \left(\sum_{i=1}^k a_i^2 / k \right)^{1/2}$$
, и с помощью этой подпрограммы вычислить отношение двух средних квадратических для элементов массивов B(15), C(30).

6. Написать рекурсивную подпрограмму для нахождения наибольшего общего делителя двух положительных чисел. Составить не рекурсивный вариант этой же подпрограммы.

7. Описать в виде рекурсивной подпрограммы алгоритм m — кратного дифференцирования полинома степени n :
$$L_n(x) = \sum_{i=0}^n a_i x^{n-i}.$$

§ 18. Операции над матрицами

1. Матричные операции. Для удобства работы с матрицами, а также ради сокращения объема программы в состав языка Бейсик входит оператор с ключевым словом MAT, позволяющий выполнять ввод и вывод значений элементов матрицы, арифметические операции над матрицами, а также реализовать ряд матричных функций.

В списке оператора MAT для задания указанных операций указываются ключевые слова операторов READ, INPUT, PRINT или оператор присваивания. Все массивы, участвующие в операторе MAT, должны быть определены в операторе DIM. Все матричные операции не выполняются над элементами с нулевыми индексами, так что значения элементов с нулевыми индексами после выполнения операций в операторе MAT будут неопределенными. Поэтому при использовании матричных операций нулевыми индексами лучше не пользоваться.

2. Ввод значений. Оператор *обработки матриц* вида

MAT READ список массивов

обеспечивает присваивание значений элементам массивов, указанных в списке массивов, предварительно сформированного операторами DATA блока данных. Элементы списка массивов могут приводиться с индексами или без них. В первом случае индексы определяют количество элементов массива, которым присваиваются значения. Значения индексов не могут превышать размеров этого массива, объявленных в операторе DIM. Во втором случае из блока данных считываются значения и присваиваются элементам массива в соответствии с объявленной размерностью. Имена массивов в списке массивов разделяются запятыми.

Отсутствие значений или задание меньшего количества значений в операторе DATA приведет к выдаче сообщения об ошибке и прекращению выполнения программы. Присваивание значений элементам двумерного массива (матрицы) производится по строкам. Например, в следующем фрагменте программы:

```
10 OPTION BASE 1
20 DATA 2, 3, 4, 5, -7, 0, -6, -9, 1, 8, 14, 12, 6,
   7, 10, 16, -1
30 DIM X(25, 15), A(7), B(5)
40 READ I, J
50 MAT READ B, X(I, J), A(J)
```

индексация массивов начинается с 1 и происходит присвоение значений в операторе с номером 50 всем пяти элементам массива B, шести элементам массива X и трем элементам массива A; один элемент блока данных остался неиспользованным.

Для присваивания элементам массивов значений, вводимых с клавиатуры, используется оператор

MAT INPUT список массивов

где список массивов формируется точно так же, как и в операторе **MAT READ**, а действие этого оператора аналогично рассмотренному выше оператору **INPUT**.

Значения элементов матрицы вводятся по строкам, при этом вводимые величины разделяются запятыми. Необходимо соблюдение соответствия по типу элементов массива и вводимых значений. Строковые константы при вводе заключаются в кавычки лишь тогда, когда содержат запятые, кавычки, пробелы в начале или конце. Например, при выполнении программы

```
10 OPTION BASE 1
20 DIM C(2, 3), D$(3)
30 MAT INPUT D$, C
. . . . .
RUN
```

после выполнения оператора с номером 30 вычислительная машина требует ввода, высвечивая на экране знак ? После набора строки значений

ТЕКСТ, "␣BASIC", —PLUS, 1, 6, 2, 5, 3, 4

и нажатия на клавишу «возврат каретки» элементам массива **D\$** будут присвоены строковые значения, а массив **C** будет заполнен числовыми значениями: 1, 6, 2 — первая строка, 5, 3, 4 — вторая строка.

3. Вывод данных. Значения элементов массивов могут быть выведены на экран дисплея оператором

MAT PRINT список массивов

где список массивов содержит идентификаторы массивов с индексами или без них. Индексы указывают размеры массива, который должен быть выведен. Если индексы отсутствуют, то на экран будут выведены все элементы массива.

Элементы вектора выводятся, начиная с первого элемента, а элементы матрицы — по строкам: сначала элементы первой строки, затем — второй и т. д. Подчеркнем еще раз, что элементы с нулевыми индексами не выводятся. В качестве разделителя между элементами списка можно использовать либо запятую, либо точку с запятой. Употребление в списке массивов запятой после имени массива или отсутствие разделителя в конце списка приводит к выводу в табличной форме по зонам. Использование в списке массивов после имени массива точки с запятой приводит к размещению в строке дисплея значений элементов заданного массива непосредственно

друг за другом (с одним пробелом после каждого числового значения), т. е. аналогично обычному оператору PRINT.

Например, если в программе содержатся операторы

```
10 DIM Y (50), X (4,5)
70 MAT PRINT X (2,5); Y
```

то второй из них разместит десять значений элементов двумерного числового массива X друг за другом с одним пробелом в конце каждого значения, а 50 значений элементов массива Y будут выведены в виде пяти столбцов по десять значений в каждом из них (предполагается, что ширина экрана равна 80 символам).

Подобным же образом организуется вывод значений элементов массива на печатающее устройство. Для различных Бейсик-систем запись оператора печати может отличаться деталями.

Форматный матричный вывод так же зависит от конкретной системы. Например, в некоторых версиях, в частности в версии Дисп для ЕС ЭВМ имеется оператор

```
MAT PRINT USING N, список массивов
```

где первый элемент в списке этого оператора — параметр N является номером строки с описанием форматов печати. Например:

```
210 MAT PRINT USING 240, P, Q
240 :+ ## ## ## ##
```

Здесь все элементы массивов P и Q выводятся как четырехзначные целые числа со знаком (формат задан в операторе с номером 240 после двоеточия), исполняющего в данном случае роль ключевого слова).

4. Арифметические операции. При помощи оператора MAT по правилам матричной алгебры выполняются операции сложения и вычитания матриц, умножение массива на скаляр и перемножение матриц. Соответствующий оператор имеет вид

```
MAT A=E
```

где A — матрица (результат), E — матричное выражение вида: B, B + C, B - C, B * (K), F * D. Здесь B, C — матрицы-операнды, размерности которых совпадают с размерностью матрицы A; K — скалярное выражение; F, D — также матрицы-операнды, причем число столбцов матрицы F должно равняться числу строк матрицы D.

При выполнении операции умножения не допускаются выражения вида F=F*D или D=F*D. В некоторых версиях Бейсика разрешается опускать окаймляющие скалярное выражение круглые скобки, если множителем является не выражение, а число. Например, если в программе имеем оператор

```
10 DIM A(2,3), B(2,3), C(2,3), D(3,3), F(3,2)
```

и определены значения элементов матриц В, С и F, то допустимы следующие операторы:

```
50 MAT A=B
60 MAT B=B+C
70 MAT A=A-C
80 MAT B=2.5*B
90 MAT D=C*F
```

5. Операции инициирования. В языке Бейсик существует три оператора, которые формируют стандартные значения элементов массива. Эти операторы имеют вид

```
MAT A=ZER
MAT A=CON
MAT A=IDN
```

Первый из этих операторов присваивает значения, равные нулю, каждому элементу массива А; второй присваивает каждому элементу массива А значения, равные единице; последний образует массив, в котором элементам с одинаковыми индексами присваивается значение 1, а всем остальным 0 (в случае квадратного массива получаем единичную матрицу).

После ключевых слов ZER, CON, IDN в круглых скобках могут быть указаны размеры массива, элементы которого должны быть инициированы. Если они отсутствуют, то принимаются максимальные размеры массива. Например:

```
10 DIM A(10, 5), B(8), C(10, 10)
20 MAT A=ZER(5, 5)
30 MAT B=CON
40 MAT C=IDN(10, 10)
```

Размерность, указанная в операторах инициирования, не должна превышать ранее объявленную размерность массива. Превышение ее приведет к ошибке. Проверка размерности производится путем подсчета количества элементов с учетом элементов с нулевыми индексами несмотря на то, что они игнорируются матричными операторами. Так, если в программе содержится описание массивов, приведенное в строке номер 10 предыдущего примера, то допустимы операторы

```
50 MAT A=ZER(4, 12)
60 MAT C=CON(19, 5)
```

так как здесь не превышены объявленные размерности массивов А и С, т. е. $(10+1) \times (5+1) > (4+1) \times (12+1)$ и $(10+1) \times (10+1) > (19+1) \times (5+1)$. Оператор

```
70 MAT C=CON (20,5)
```

очевидно, будет ошибочным.

6. Матричные функции. Для нахождения транспонированной и обратной матриц существуют матричные функции с идентификаторами TRN и INV, используемые в операторе MAT:

MAT A=TRN (B)

MAT A=INV (B)

где A, B — двумерные массивы.

В функции транспонирования строки и столбцы исходной матрицы B меняются местами, так что элементы A(I, J) получают значения элементов B(J, I) и, если матрица B имела размерность M×N, то размерность результирующей матрицы будет N×M.

Функция обращения имеет смысл только для квадратной матрицы. Эта функция сводится к ссылке в результирующую матрицу A элементов обратной матрицы B⁻¹.

При нахождении обратной матрицы вычисляется определитель, и если он отличен от нуля, то операция обращения совершается; в противном случае фиксируется ошибка. В некоторых версиях, например, в системе Бейсик-плюс значение определителя запоминается как значение переменной с идентификатором DET. Например, после выполнения оператора

80 MAT A=INV (B)

допустим оператор

90 D=DET

так как значение переменной DET будет определено и равно значению определителя матрицы B. Возможны и другие модификации в использовании функции INV, например указание в конце оператора после запятой переменной, которой присваивается значение определителя. Например:

100 MAT A=INV (B), D

В Бейсик-системе для ЕС ЭВМ и ряде других принято размещать обратную матрицу на месте исходной, так что в соответствующем операторе MAT функция INV пишется без аргумента, т. е. оператор MAT имеет вид MAT A=INV. Отметим, что существуют версии Бейсика, в которых оператор MAT A=INV (A) запрещен.

Упражнения

1. Составить программу ввода матриц третьего порядка и вычисления детерминанта матриц.

2. Написать программу ввода квадратных матриц порядка M и печатания соответствующих транспонированных матриц.

§ 19. Файлы данных

1. Структура файла. Обычно в одной программе обрабатывается большой объем данных, которые удобно объединять в наборы данных и хранить вне оперативной памяти вычислительной машины. Такие

наборы данных, как известно, называются *файлами*. Наиболее эффективным способом хранения данных является организация файлов на дисках. В большинстве вычислительных систем имеется по меньшей мере один дисковод.

Файл состоит из записей. *Запись* содержит одно или несколько значений данных. Значение каждого элемента информационной записи называют *полем*. Обычно файл организуется так, что все его записи имеют одинаковую конфигурацию, т. е. количество полей, порядок их расположения и длина не изменяются при переходе от одной записи к другой. Поэтому единственное, чем отличаются записи, — это значениями полей, и компоновка файла определяется двумя факторами: конфигурацией и количеством содержащихся в нем записей, так что перечисление полей одной записи дает адекватное описание структуры файла.

2. Доступ к файлу. Отыскание нужных записей в файле реализуется двумя способами. Поиск с начала файла путем проверки по очереди каждой записи до тех пор, пока не будет найдена требуемая, называется *последовательным* доступом. Альтернативный способ позволяет обращаться к записям по номеру в любом порядке. Такой способ поиска называется произвольным или *прямым* доступом.

Любой файл может быть организован либо как файл с последовательным доступом либо как файл с прямым доступом. Одновременно оба способа доступа к одному файлу недопустимы.

Последовательный доступ прост для программирования, и при его реализации требуется меньше памяти на внешних носителях, но может потребоваться значительное время для поиска записи, находящейся в конце длинного файла. Обновление существующих записей трудноосуществимо, а иногда и просто невозможно. В последовательно организованном файле длина записи непосредственно зависит от величины каждого поля, а поскольку длины полей в каждой записи различны, то и длины записей также различны.

Файлы с прямым доступом требуют более сложного программирования и занимают обычно больше места на диске, чем последовательные файлы, но очень легко поддаются обновлению, и поиск любых записей может осуществляться с одинаковой скоростью. Длина записи в файле с прямым доступом определяется при его создании, она постоянна, и каждому полю выделяется внутри записи строго определенное место.

Чтобы произвести считывание (ввод) конкретного поля записи, программа должна, в первую очередь, открыть файл, найти соответствующую запись, затем переслать эту запись с диска в оперативную память и, наконец, выделить требуемое место в виде значения переменной. Переписывание информации в файл (вывод) производится аналогичным путем с той лишь разницей, что в этом случае значения пересылаются программой из динамической памяти в файл.

При пересылках, производимых целыми блоками значений данных, часть оперативной памяти отводится под *буфер*, в котором временно хранятся данные. Управление буфером происходит автоматически. Когда программа заканчивает работу с файлом, она должна его закрыть во избежание потери части содержимого файла.

3. Имена и номера файлов. Каждый файл обозначается *именем*, которое должно содержать не менее одного и не более восьми символов. После восьмого символа следует суффикс, называемый *расширением* имени файла и содержащий не более трех символов. В качестве символов в имени используются заглавные буквы, цифры и некоторые из специальных знаков.

Расширение имени файла, являющееся необязательным элементом имени, тем не менее обозначает его тип. В частности, расширение имени .BAS указывает, что данный файл есть программа на Бейсике, расширение имени .DAT задает файл данных и т. д.

Каждое уникальное имя обозначает свой конкретный файл. Родственные по своему функциональному назначению файлы могут быть обозначены *родовыми* (глобальными, неоднозначными) именами, в которых используются символы ? и *.

Символ ? является единственным неоднозначно интерпретируемым символом в имени файла, т. е. ему соответствует в имени любой одиночный символ. Например, имена TEST1.BAS, TEST2.BAS соответствуют родовому имени TEST?.BAS, но ему не соответствует имя TEST13.BAS

Символ * обозначает любое количество неоднозначно интерпретируемых символов, которые являются последними символами либо в имени, либо в суффиксе. Например, под именем PRG*.B* подразумевается любое имя, начинающееся с PRG и имеющее расширение, начинающееся с B. Поскольку звездочка имеет смысл только в конце атрибутов имени файла, родовые имена *PRG.BAS и *.*A, например, эквивалентны именам *.BAS и *.* соответственно.

Файлы с одним и тем же именем могут быть использованы на разных дисках. Чтобы различить такие файлы, перед именем файла указывается двухсимвольный префикс, обозначающий дисковод. Первый символ — это *метка* дисковода (буквы A, B, C и т. д.), а второй символ — двоеточие. Например, имя файла B:PROGR.BAS указывает на то, что файл с именем PROGR с расширением BAS находится на дисководе B.

Если в системе один дисковод, то метка его A и указывать префикс A: нет необходимости. Операционная система всегда регистрирует один дисковод, используемый по умолчанию. Этот дисковод называется системным (зарегистрированным или подразумеваемым). Даже если в системе несколько накопителей, файлы без указания метки дисковода по умолчанию предполагаются находящимися в системном дисководе.

Хотя файлы данных на диске и распознаются по именам, программы обращаются к файлам данных главным образом по номерам. Номер файла соотносится с его именем при *открытии* файла оператором OPEN, в простейшем случае имеющим вид

OPEN "имя файла" AS ##N

где в имени файла префикс, задающий диск, указывается лишь тогда, когда файл не находится на диске, принимаемом по умолчанию; AS — разделитель; N — номер файла, указываемый числовой константой, переменной или выражением; обычно допускаются только номера 1, 2, 3. Например,

870 OPEN "B:COEF.DAT" AS # 2

Оператор *закрывтия* файла CLOSE освобождает номер файла для повторного использования, если следующим по порядку идет оператор OPEN. Например, оператор

1080 CLOSE # 2

закрывает файл с именем "B:COEF.DAT", которому в предыдущем примере был соотнесен номер 2.

4. Открытие файлов. В предыдущем пункте был приведен простейший оператор открытия файла. В различных Бейсик-системах этот оператор имеет различные форматы, которые зависят также от того, каков режим доступа открываемого файла и каково его назначение.

Например, в Бейсик-системе IBM имеются следующие модификации оператора OPEN:

OPEN "имя файла" FOR OUTPUT AS ##N

OPEN "имя файла" FOR APPEND AS ##N

OPEN "имя файла" FOR INPUT AS ##N

OPEN "имя файла" AS ##N LEN=L

где N — номер файла, L — длина записи в байтах.

Первые три модификации относятся к файлам с последовательным доступом. Оператор OPEN с конструкцией FOR OUTPUT создает новый дисковый файл с заданным именем для размещения записей в его начало. Если файл с таким именем уже есть, то он уничтожается, и на его месте создается новый файл с тем же именем. Оператор OPEN—FOR APPEND обеспечивает поиск названного файла среди уже существующих, с тем, чтобы добавить новые записи в его конец, но если требуемого файла найти не удастся, то автоматически создается новый файл с заданным именем. Модификация оператора с конструкцией FOR INPUT реализует поиск файла с конкретным именем, отсутствие которого вызовет ошибку. Последний из приведенных выше операторов OPEN открывает файл с прямым доступом как для чтения, так и записи данных, устанавли-

вая длину каждой записи с помощью конструкции LEN. Все записи в таком файле имеют одинаковую длину, которая обычно составляет 128 байт.

Операторы с ключевыми словами PRINT# и PRINT# USING реализуют *записывание* (вывод) значений в дисковый файл, а операторы INPUT# или LINE INPUT# — *считывание* этих значений и присваивание их переменным (ввод). Действие этих операторов аналогично работе операторов PRINT, PRINT USING, INPUT, LINE INPUT, рассмотренных ранее. Например, оператор

```
170 PRINT#1, 301
```

записывает значение константы 301 в дисковый файл номер 1, а оператор

```
210 INPUT#1, R
```

считывает значение из дискового файла номер 1 и присваивает переменной R.

После открытия файла с прямым доступом в программе должна быть объявлена *структура* записей файла. Это реализуется с помощью оператора с ключевым словом FIELD, имеющим вид

```
FIELD #N, список полей
```

где N — номер файла, а список полей содержит перечисление полей в виде конструкций с форматом L AS V в том порядке, в котором они встречаются в записях файла; для каждого поля оператор устанавливает длину L и имя переменной, которая будет использована в программе для идентификации данного поля. Все поля файла должны быть перечислены в одном операторе FIELD и разделены запятыми. Например,

```
190 FIELD #3, 10 AS T$, 4 AS S$, 10 AS R$ (1)
```

Общая длина всех объявляемых в операторе FIELD полей не должна превосходить длины записи, установленной оператором OPEN. Все переменные в операторе FIELD должны быть строковыми; при помощи их в файл или из файла с прямым доступом пересылаются все значения.

Заполнение заданного поля в файле с прямым доступом строковым значением осуществляется операторами LSET и RSET вида

```
LSET V=E
```

```
RSET V=E
```

где V — строковая переменная поля, E — строковое значение. При заполнении строковое значение выравнивается по крайней левой позиции поля в случае оператора LSET и по крайней правой позиции при использовании оператора RSET; оставшиеся неиспользованными позиции поля (соответственно правые и левые) заполняются

пробелами. Например:

```
270 LSET F$="FUNCTION"  
310 RSET P$=REM$
```

Перед использованием операторов LSET и RSET для хранения в файле с прямым доступом числовых величин последние необходимо преобразовать в цифровые строки при помощи функций MKI\$, MKS\$, MKD\$ соответственно для целых чисел, чисел обычной и двойной точности. Обратное преобразование выполняется с помощью функций CVI, CVS, CVD соответственно.

Занесение готовой записи в файл с прямым доступом производится оператором

```
PUT #N, M
```

где N — номер файла, M — номер записи; последний указывать не обязательно, и если номер записи опущен, то записи присваивается очередной номер, т. е. номер на единицу больше последнего, использованного в данном файле. Например:

```
320 PUT #1, 4  
360 PUT #1
```

Считывание записи из файла с прямым доступом реализуется оператором

```
GET #N, M
```

где, как и выше, N — номер файла, M — номер записи. Если параметр M задан, то считывается запись с заданным номером, в противном случае — запись с очередным номером.

Оператор открытия файла может иметь и универсальный формат (как например, в системе CP/M—80):

```
OPEN "режим", #N, "имя файла", L
```

где "режим" — строковое выражение, первым символом которого может быть одна из букв O, I, R, означающая соответственно последовательный вывод в файл, последовательный ввод из файла и прямой ввод — вывод; N — номер файла; L — длина записи (целочисленное выражение, устанавливающее длину записи для файла с прямым доступом). Атрибуты # и L могут быть опущены (по умолчанию L=128 байт).

Этот оператор открывает буфер для ввода-вывода в файл или устройство и определяет режим доступа, который будет использоваться буфером. Файлы с произвольным или последовательным доступом могут размещаться на диске, а все другие устройства могут открываться только для последовательных файлов. При открытии файла для вывода будут уничтожены все данные, существующие в файле. Например, оператор

```
100 OPEN "O", #1, "DATA"
```

открывает файл, названный "DATA" под номером 1 для последовательного вывода.

5. Закрытие файлов. Взаимодействие с файлом, т. е. завершение ввода в файл или любое устройство, или вывода из файла (устройства) выполняется оператором закрытия файла, уже упомянутым в п. 3 этого параграфа. В общем случае этот оператор имеет вид

CLOSE #N1, #N2, ...

где N1, N2, ... — номера файлов (устройств). Номера файлов могут быть опущены, тогда оператор CLOSE закрывает все файлы и устройства. Можно не указывать символ # перед номером.

После выполнения оператора CLOSE аннулируется ассоциация между определенным файлом или устройством и его номером. Этот же файл или устройство могут быть открыты опять с тем же или другим номером файла. Этот же номер может быть использован заново, чтобы открыть любое устройство или файл. Операторы END и CHAIN, команды LOAD, NEW, RUN и SYSTEM автоматически закрывают все открытые файлы или устройства. Операторы STOP и CHAIN MERGE не закрывают файлы или устройства. Например:

110 CLOSE #1, 2, #3

370 CLOSE

Упражнение

В программе открывается уже существующий файл. С помощью каких операторов в данной программе можно определить: а) имеется ли доступ к этому файлу для считывания; б) имеется ли доступ к этому файлу для записи; в) возможен ли прямой доступ к этому файлу для записи.

§ 20. Графическое представление информации

1. Текстовый и графический режимы. Вывод на экран дисплея информации в виде строк символов алфавита осуществляется с помощью операторов PRINT и PRINT USING. Соответствующий режим работы дисплея называется *текстовым* (алфавитно-цифровым или символьным). В то же время различные вычислительные системы могут быть оснащены дополнительным оборудованием, среди которого важными являются цветной графический адаптер и цветной экран дисплея. Наличие такого оборудования позволяет реализовать графический вывод — построение отдельных точек, линий, различных геометрических фигур, заполнение любой заданной области экрана однотонным цветом. Такой режим работы дисплея называется *графическим*.

Различают две разновидности графического режима, отличающиеся числом и размером выводимых точек и количеством допустимых цветов — режим высокого разрешения и режим среднего раз-

решения. В первом случае экран разбивается на большее по сравнению с режимом среднего разрешения точек, их размеры меньше и изображение может содержать большее число деталей. В режиме высокого разрешения изображение всегда черно-белое. В режиме среднего разрешения на экране одновременно может появляться до четырех различных цветов.

Операторы, обеспечивающие графический режим работы дисплея, составляют наиболее изменчивую часть языка Бейсик. Эти операторы в различных реализациях могут иметь различное представление, что связано в основном с тем, что изменяются графические средства отображения информации для различных вычислительных систем.

2. Области экрана. На экране дисплея выделяют передний план, фон и окантовку. Передний план — это область, где располагаются текстовые данные и графические изображения. Фон — область экрана, в которой производится все, что выводится на экран (выводимая информация накладывается на фон); сразу после включения вычислительной машины можно видеть фон. Окантовка — область, окружающая фон; как правило, окантовка бывает того же цвета, что и фон, и поэтому неразличима. Благодаря наличию окантовки сглаживаются различия между экранами разнотипных телевизионных установок и мониторов. В некоторых телевизионных установках (бытовых телевизорах) почти вся окантовка остается невидимой, т. е. остается за пределами рабочей области экрана, в других — только некоторая ее часть, в мониторах же видимой является почти вся окантовка.

3. Оператор режимов. Переключение режимов работы экрана осуществляется оператором

SCREEN N

где N — значение, идентифицирующее конкретный режим работы: 0 — определяет режим текстового вывода, 1 — задается режим графического вывода со средней разрешающей способностью, 2 — графический режим с высоким разрешением. Например, оператор

120 SCREEN 1

переключает экран в режим графического вывода со средним разрешением; обратный переход к текстовому режиму может быть задан оператором

760 SCREEN 0

Если указанный в операторе SCREEN режим совпадает с текущим, никаких действий не производится. В противном случае при выполнении этого оператора происходит очистка экрана дисплея и устанавливается белый цвет для переднего плана и черный для фона.

4. Оператор цвета. Оператор с ключевым словом COLOR, называемый оператором *цвета*, в общем случае имеющий формат

COLOR N1, N2, N3

где N1, N2, N3 — числовые значения, идентифицирующие цвет, позволяет задавать различные цвета для выводимой на переднем плане информации, фона и окантовки.

Действие этого оператора зависит от того, в каком режиме работает дисплей. В графическом режиме с высоким разрешением, когда цвет переднего плана всегда белый, а фона и окантовки черный, оператор COLOR недопустим. Поэтому в дальнейшем, если особо не оговорено, будет подразумеваться графический режим со средней разрешающей способностью.

Рассмотрим сначала оператор цвета в текстовом режиме. Первое значение в списке этого оператора, т. е. N1 определяет цвет для переднего плана, N2 — для фона, а N3 — для окантовки. В текстовом режиме существует следующее соответствие между номерами и различными цветами: 0 — черный, 1 — синий, 2 — зеленый, 3 — голубой, 4 — красный, 5 — пурпурный, 6 — золотистый (коричневый), 7 — белый (серый), 8 — темно-серый, 9 — светло-синий, 10 — светло-зеленый, 11 — светло-голубой, 12 — розовый, 13 — светло-пурпурный, 14 — желтый, 15 — ярко-белый.

Например, задание зеленого цвета для переднего плана, красного для фона и белого для окантовки реализуется оператором

190 COLOR 2, 4, 7

Номера цветов от 0 до 7 допустимы как для переднего плана и окантовки, так и для фона, тогда как номера с 8 по 15 разрешается использовать только для переднего плана и окантовки. Некоторые мониторы при задании цвета на номера с 8 по 15 реагируют точно так же, как и на номера с 0 по 7 соответственно. Если к номеру цвета переднего плана прибавить 16, то выводимые символы будут мерцать.

Оператор COLOR в текстовом режиме оказывает влияние на цвет переднего плана и фона только для тех символов, которые будут выводиться после его выполнения. Цвет окантовки изменяется сразу. Информация, уже высвеченная на экране, не изменит своего цвета, а для всех последующих появляющихся на экране символов цвет переднего плана и фона будет новым. Например:

COLOR 3, 4, 14

PRINT "ЦВЕТНАЯ ГРАФИКА"

В этом примере оператор COLOR использован в режиме немедленной обработки (как и оператор PRINT), что допускается. Команда COLOR изменяет цвет переднего плана на голубой, фона — на красный и окантовки — на желтый, однако сам высвечивается на

экране с прежним цветом переднего плана и фона, так как эта команда появилась на экране до ее исполнения. Команда PRINT и выведенная в результате ее выполнения строка ЦВЕТНАЯ ГРАФИКА появятся на экране в новом, голубом цвете на красном фоне.

Любое из значений N1, N2, N3 можно опустить, тогда цвет соответствующей области экрана не изменится, т. е. возможны варианты оператора цвета:

COLOR N1 COLOR, N2 COLOR, N3

В графическом режиме в списке оператора цвета допускается лишь два параметра

COLOR N1, N2

где N1 определяет цвет фона и окантовки, а N2 — палитру цветов переднего плана (отметим, что в текстовом режиме задаваемые в операторе COLOR значения имеют прямо противоположную интерпретацию).

Номера, соответствующие различным цветам, приведены выше. Однако в графическом режиме номер N1 цвета фона и окантовки определяет, кроме того, оттенок цвета переднего плана — светлый или темный. В результате номера цветов от 0 до 7 соответствуют темной окраске фона и переднего плана, от 8 до 15 — светлым тонам этих областей, а номера цветов от 16 до 23 определяют темный оттенок фона и светлый передний план для цветов, имеющих номера от 0 до 7 соответственно.

Конкретный цвет переднего плана (всего из четырех допустимых) задается в самих операторах, выполняющих графические построения. В операторе COLOR параметром N2 задается лишь набор цветов для переднего плана — палитра. Каждая палитра состоит из четырех цветов, палитр всего две. Номер палитры 0 означает, что активной будет палитра, в которой допускаются цвета: фона, зеленый, красный и золотистый; если же номер палитры есть 1, то соответствующими цветами являются: цвет фона, голубой, пурпурный и белый. Таким образом, второй параметр N2 в операторе COLOR принимает только два значения: 0 или 1.

Каждому из четырех цветов в одной из палитр соответствуют номера 0, 1, 2, 3, задаваемые, как отмечалось выше, в операторах графического построения. Номеру 0 соответствует всегда цвет фона, 1 — зеленый или голубой, 2 — красный или пурпурный, 3 — золотистый или белый (в зависимости от палитры). Например, оператор

550 COLOR 18, 1

в графическом режиме обеспечивает зеленый цвет фона и окантовки, причем фон будет темным, а оттенок переднего плана светлым. Значение 1 палитры указывает, что цвет переднего плана может быть либо зеленым (цвет фона), либо голубым, пурпурным или белым.

Заданием одного из номеров от 0 до 3 в операторе графического построения выбирается один из названных здесь цветов.

При выполнении оператора COLOR в графическом режиме изменяется цвет всего фона вместе с окантовкой и всего переднего плана, включая изображения, выведенные на экран ранее. Любой из параметров N1 или N2 может быть опущен, т. е. допускается запись оператора COLOR в виде

COLOR N1 или COLOR, N2

и тогда цвет соответствующей области (областей) не изменяется.

5. Система координат. Любую точку экрана дисплея необходимо уметь однозначно идентифицировать. Это достигается введением в плоскости экрана системы координат с началом (точка с координатами (0, 0)) в левом верхнем углу экрана. Ось абсцисс направлена вправо, ось ординат — вниз. Таким образом, в правом нижнем углу будет расположена точка с максимальными значениями координат. Поскольку каждая светящаяся точка на экране снабжается целочисленными координатами, максимальные значения координат зависят от разрешающей способности экрана.

Существуют цветные дисплеи с разрешающей способностью 256×192 точек (Ямаха, MS X и др.) и 320×200 точек (PC IBM, графический режим со средним разрешением). В режиме высокого разрешения на экране высвечивается 640×200 точек (PC IBM). Таким образом, правая нижняя точка имеет координаты (225, 191), (319, 199) или (639, 199) в зависимости от марки дисплея и режима работы.

6. Построение точек и линий. После того как экран дисплея подготовлен к работе операторами CLS, SCREEN и COLOR на него могут быть отображены различные геометрические фигуры.

Для изображения на экране, работающем в одном из графических режимов, отдельной точки с заданными координатами и цветом используется оператор с ключевым словом PSET

PSET (X, Y), N

где X, Y — соответственно абсцисса и ордината точки, N — номер одного из четырех цветов активной палитры в режиме со средней разрешающей способностью; в режиме с высокой разрешающей способностью нечетные значения N определяют единственный допустимый в этом режиме цвет переднего плана (белый), а четные N — единственный цвет фона (черный). Например, при выполнении оператора

320 PSET (130, 80), 2

в режиме среднего разрешения на экране появится красная или пурпурная точка — в зависимости от того, какая палитра является активной; в режиме с высокой разрешающей способностью на экран будет выведена белая (цвета переднего плана) точка,

Номер цвета в операторе PSET задавать не обязательно. Если этого номера нет, то цвет точки в графическом режиме со средним разрешением в зависимости от активной палитры будет золотистый или белый (цвет, соответствующий номеру 3), а в режиме с высокой разрешающей способностью — белый. Например:

```
370 PSET (10, 190)
```

Для построения отрезка прямой используется оператор

```
LINE (X1, Y1) — (X2, Y2)
```

где X1, Y1 и X2, Y2 — координаты начала и конца отрезка соответственно. Например, при выполнении оператора

```
390 LINE (60, 120) — (10, 5)
```

на экране будет построен отрезок прямой, начало которого находится в точке с координатами (60, 120), а конец — в точке (10, 5).

В операторе LINE можно задавать цвет выводимой линии точно так же, как и в операторе PSET, т. е. добавлением в конец оператора запятой и номера цвета.

Используя несколько операторов LINE, можно построить ломаную линию. Например, с помощью двух операторов

```
400 LINE (60, 120) — (10, 5), 1
```

```
410 LINE (10, 5) — (5, 70), 2
```

на экране выстраивается ломаная линия с вершиной в точке, имеющей координаты (10, 5), причем отрезки ее будут разного цвета.

Пользуясь тем, что всякий раз при выводе на экран какой-либо линии в памяти запоминаются координаты последней выведенной точки, при построении ломаной в операторах LINE можно опустить координаты начальной точки очередного отрезка. Например, предыдущий пример можно записать в виде

```
400 LINE (60, 120) — (10, 5), 1
```

```
410 LINE — (5, 70), 2
```

Если на экране предварительно не производилось никаких графических построений, то по умолчанию предполагается, что начальная точка совпадает с центром экрана.

В оператор LINE после номера цвета через запятую может быть добавлена буква B. Это означает, что на экране при выполнении оператора LINE будет построен прямоугольник, стороны которого параллельны координатным осям (сторонам экрана дисплея), а одна из диагоналей имеет концами точки, указанные в операторе LINE. Например, оператор

```
420 LINE (70, 80) — (80, 90), 2, B
```

выстраивает на экране квадрат цвета 2 со стороной 10.

Если номер цвета не задан, по умолчанию будет использоваться цвет номер 3. Отметим, что при этом позиция для неявного номера

цвета должна быть сохранена, т. е. перед символом В должны стоять две запятые. Например:

```
430 LINE (170, 180) — (180, 190), ,В
```

Построенный на экране указанным способом прямоугольник может быть закрашен цветом, которым изображается контур прямоугольника. Для этого достаточно добавить в оператор LINE непосредственно после буквы В букву F. Например, при выполнении оператора

```
430 LINE (170, 180) — (180, 190),,ВF
```

на экране появится квадрат со стороной 10, закрашенный в цвет номер 3 (золотистый или белый в зависимости от того, какая палитра является активной).

Во всех приведенных в этом пункте примерах точки задавались абсолютными координатами. Допускается также относительная координатная идентификация, при которой координаты задаются относительно последней выведенной на экран точки. Если в операторах PSET или LINE координатам точки предшествует слово STEP, то эти координаты интерпретируются как смещения относительно координат последней выведенной точки. Например:

```
440 PSET (50, 75)
```

```
450 PSET STEP (10, —5)
```

Первый из приведенных операторов осуществит вывод на экран точки с абсолютными координатами (50, 75), а второй содержит относительные координаты и точка, которая будет изображена на экране, имеет абсолютные координаты (50+10, 75—5), т. е. (60, 70).

Если относительными координатами задается самая первая выводимая точка, то в качестве предыдущей точки подразумевается центр экрана.

7. Совместный вывод текста и графики. Во всех режимах работы экрана могут быть выведены любые символы алфавита языка Бейсик или любые имеющиеся на клавиатуре знаки. Цвет выводимых символов по умолчанию предполагается соответствующим номеру 3. При средней разрешающей способности выводимые символы имеют такую же величину, как и в текстовом режиме при 40-символьной строке экрана; в режиме высокого разрешения размер символов соответствует 80-символьной строке.

Таким образом, в соответствии с размерами символов, выводимых в текстовой строке, каждый символ занимает площадь экрана в 8 графических строк по высоте и столько же графических столбцов в ширину. Нумерация текстовых строк начинается со строки номер 1 и столбца номер 1. Эти номера могут использоваться в операторе LOCATE, применяемом наряду с рассмотренными выше операторами PRINT и PRINT USING для размещения текстовых строк на экране.

Оператор LOCATE имеет формат
LOCATE S, C, N, F, L

Этот оператор обеспечивает перемещение курсора в позицию, определяемую заданными номерами текстовой строки (S) и текстового столбца — позиции в строке (C). Параметр N определяет видимость курсора: если значение N будет 1, курсор видим, если 0 — курсор невидим. Значения параметров F и L определяют размер и форму курсора.

На уже имеющееся на экране графическое изображение может быть наложен любой текст, при этом графические элементы изображения могут частично стираться, так как вокруг каждого символа освобождается поле по 8 координатных единиц по высоте и по ширине. Если же происходит наложение графического изображения на уже имеющийся на экране текст, то контуры символов, с которыми пересекаются графические элементы, могут лишь незначительно измениться.

8. Построение окружности и ее элементов. Для построения окружности на экране используется оператор

CIRCLE (X, Y), R, N

где X, Y — координаты центра окружности, R — радиус окружности, причем единицей измерения является ширина координатного столбца экрана, N — необязательный параметр, определяющий цвет окружности; в качестве значений этого параметра могут использоваться номера 0 или 1 в режиме высокого разрешения или от 0 до 3 в режиме среднего разрешения (выбор одного из цветов активной палитры); если номер цвета не задан, то используется стандартный цвет переднего плана. Например, используя операторы

```
10 SCREEN 1
20 CLS
30 CIRCLE (190, 120), 25, 1
40 CIRCLE (100, 150), 20
```

будет выбран графический режим работы, очищен экран и построены две окружности: одна голубого цвета, другая стандартного (белого) цвета (параметр N в операторе 40 не указан). Если в дальнейшем исполнится оператор

```
300 CIRCLE (100, 150), 20, 0
```

то он сотрет белую окружность путем вывода на экран точно такой же окружности цвета фона (N=0).

После параметра N в операторе CIRCLE могут быть указаны еще два параметра B и E — так называемые начальная и конечная точки дуги. В тех случаях, когда заданы параметры B и E (используемые только в паре), на экране вместо полной окружности строит-

ся дуга с указанными концевыми точками. Параметры В и Е должны быть заданы в радианах — от 0 до 6.2831. Например, верхняя полуокружность с центром, совпадающим с центром экрана и радиусом 50 стандартного белого цвета в режиме среднего разрешения может быть задана оператором

```
50 CIRCLE (160, 100), 50,,0,3.141593
```

Отметим, что хотя в этом операторе опущен номер цвета — параметр N, позиция этого параметра задана наличием следующей за ним запятой.

Перед любым из параметров В и Е, или перед обоими может стоять знак минус. Это означает, что в соответствующую концевую точку дуги из центра окружности будет проведен радиус. Так, для выделения голубым цветом некоторого сектора, например, второй четверти построенной в предыдущем примере полуокружности, используется оператор

```
60 CIRCLE (160,100),50,1, —1.570796, —3.141593
```

9. Построение эллипсов. Для изображения на экране эллипса также используется рассмотренный только что оператор CIRCLE, только в нем необходимо в конце добавить еще один параметр CR — характеристическое отношение выстраиваемого эллипса. Параметр CR является отношением высоты эллипса к его ширине, поэтому удобнее и проще всего задавать этот параметр обыкновенной дробью, где числитель указывает, какое количество строк при выполнении оператора CIRCLE следует считать эквивалентным числу столбцов, приведенному в знаменателе. Например, оператор

```
70 CIRCLE (170,70),30,1,0,6.2831,1/3
```

вызовет на экране изображение эллипса, вытянутого по горизонтали.

Поскольку координатные единицы по горизонтали и по вертикали различаются, в режиме со средней разрешающей способностью характеристическое отношение 5/6 определяет окружность, а $CR = 5/12$ задает окружность в режиме высокого разрешения.

10. Оператор закрашивания. С помощью оператора с ключевым словом PAINТ может быть заполнена любая заданная область экрана любым допустимым в режимах графического вывода цветом. Оператор имеет вид

```
PAINT (X, Y), N, N1
```

где X, Y — координаты точки, начиная с которой экран начнет закрашиваться цветом номер N равномерно по всем направлениям; по каждому направлению это будет продолжаться до тех пор, пока не встретится точка, окрашенная в цвет с номером N1. Например:

```
290 PAINT (110,90),1,3
```

Очевидно для того, чтобы могла быть закрашена какая-то часть экрана, на нем предварительно должна быть проведена замкнутая кривая цвета N1, охватывающая заданную точку.

11. Оператор DRAW. Наряду с рассмотренным оператором PAINT в расширенный Бейсик входит оператор графического вывода

DRAW "список подкоманд"

который с помощью специального языка графического вывода позволяет изображать на экране рисунки, составленные из различных комбинаций точек и прямых линий. Список подкоманд представляет собой последовательность кодов действий по перемещению на экране и вычерчиванию линий или управляющих подкоманд. Подкоманды обеспечивают проведение линий любой длины, начиная с последней высвеченной на экране точки, в любом из восьми заданных направлений и любого из четырех цветов активной палитры или делать их невидимыми. Например, операторы

50 SCREEN 1

60 DRAW "R50, D40"

обеспечат проведение линии из центра экрана (если на экране в начале отсутствовало какое-либо графическое изображение, построение начинается из центра экрана) вправо на 50 координатных единиц (столбцов) и затем вниз на 40 единиц (строк).

Список подкоманд оператора DRAW приведен в приложении II.

12. Построение движущихся изображений. При создании движущихся изображений или «живых» картинок в кино производится съемка обычных статических рисунков, незначительно отличающихся друг от друга. Если последовательно просматривать такие снимки, быстро сменяя один другим, то отдельные изображения будут сливаться друг с другом, создавая впечатление непрерывного плавного движения.

Как было рассмотрено выше, с помощью операторов PSET, LINE, CIRCLE, PAINT, DRAW можно на экране дисплея строить и стирать неподвижные картинки. Если же после стирания это же изображение вывести на экран в новом месте, незначительно отличающемся от предыдущего местоположения, то в принципе можно реализовать идею кино, получая на экране движущееся изображение. Программная реализация такого вывода будет, очевидно, содержать операторы цикла и операторы графического вывода, однако достижение быстрой сменяемости отдельных кадров может оказаться чрезвычайно трудным, если вообще возможным. Решение этой проблемы обеспечивают операторы расширенного Бейсика — операторы с ключевыми словами GET и PUT.

Оператор GET позволяет запоминать цвета всех точек заданной прямоугольной области экрана дисплея и хранить их номера в виде

числового массива; оператор PUT повторно воспроизводит все эти цвета на экране. С помощью оператора PUT изображение воспроизводится достаточно быстро для того, чтобы можно было синтезировать движение по экрану.

Формат оператора GET:

GET (X1, Y1)—(X2, Y2), M

Здесь X1, Y1 и X2, Y2 — соответственно координаты двух противоположных углов прямоугольной области; M — идентификатор числового массива, описанного в программе и вмещающего данные о расцветке всех точек рассматриваемой области экрана; этот массив должен быть достаточно большим, чтобы указанная информация в нем могла быть размещена (количество элементов в массиве M может быть подсчитано по достаточно простой программе). Например:

150 DIM MAS(600)

370 GET (90,60)—(195,140), MAS

Формат оператора PUT:

PUT (X, Y), M

В списке этого оператора X, Y — координаты точки, в которой должен располагаться верхний левый угол воспроизводимого прямоугольника, запоминание информации о котором по раскраске области было обеспечено ранее оператором GET; M — идентификатор числового массива, содержащего данные о раскраске этого прямоугольника. Например, оператор

400 PUT (90, 60), MAS

воспроизводит исходное изображение, описанное в операторе GET из предыдущего примера, а два следующих оператора:

430 PUT (10,20), MAS

450 PUT (190,40), MAS

выводят это же изображение дважды, каждый раз на новом месте.

Полученное с помощью оператора PUT изображение можно стереть с экрана повторным исполнением того же самого оператора PUT. Например, оператор

590 PUT (190,40), MAS

стирает последнее из построенных выше изображений.

Таким образом, для синтеза движения на экране изображения, данные о раскраске которого хранятся в числовом массиве, необходимо: воспроизвести изображение с помощью оператора PUT; вычислить координаты нового местоположения изображения; стереть с помощью оператора PUT текущее изображение; повторно с помощью оператора PUT воспроизвести изображение в новом месте экрана; повторить все шаги, начиная со второго.

В операторе PUT после имени массива через запятую можно указать один из параметров смеси цветов: PSET, PRESET, AND,

OR, XOR, определяющий, по каким правилам при наложении изображений должно происходить слияние заданных цветов с уже имеющимися на экране. Эти параметры в данной книге не обсуждаются.

Упражнения

1. Составить подпрограмму для вычисления периметра произвольного многоугольника с заданными координатами вершин x_i, y_i . Изображение многоугольника вывести на экран.

2. Составить программу для следующей игры. Человек бросает камень под углом к поверхности Земли с некоторой начальной скоростью. Надо подобрать скорость так, чтобы попасть в лунку. Программа должна нарисовать на экране поверхность Земли, лунку, камень. После этого надо запросить с экрана скорость и угол. Программа должна изображать траекторию полета и в случае попадания вывести на экран поздравление с успехом, а в случае промаха предоставить очередную попытку. Побеждает тот, кто затратит меньше попыток. Расстояние от человека до лунки и размеры лунки можно менять с помощью функции RND.

§ 21. Воспроизведение звука

В персональных ЭВМ можно генерировать звуки и музыку. Источником звука является звуковой генератор с регулируемой частотой и длительностью сигнала. В ПЭВМ встроены также динамики.

Простейший оператор для генерации звука в языке Бейсик — оператор BEEP. Он выдает на динамик сигнал частотой 800 Гц и длительностью около 0,25 с. Другой оператор, позволяющий управлять частотой и продолжительностью звука, имеет вид

SOUND W, T

где первый параметр задает частоту в диапазоне от 37 до 32767 Гц; параметр T определяет длительность звука, которая измеряется числом импульсов сигнала времени или машинными единицами времени в диапазоне от 0 до 65535 (цена единицы около 55 мс). Например, оператор

20 SOUND 532.25, 16.55

генерирует звук частотой 532,25 Гц, которому в нотном стане соответствует нота «до» первой октавы с длительностью 16,55 импульсов в секунду.

Во время звучания сигнала не предполагается ожидание окончания работы оператора SOUND, а продолжается выполнение программы. Если в программе встретится второй оператор SOUND, а действие первого еще не завершено, то выполнение программы приостанавливается до окончания освобождения звукового канала. В любой момент звук можно убрать, выполнив оператор SOUND с нулевой длительностью ($T = 0$). Поскольку звуки частотой более 15000 Гц практически человеческим ухом не воспринимаются, опе-

ратор SOUND, в котором указан параметр $W > 15000$, будет генерировать паузы.

Оператор SOUND обладает достаточно широкими возможностями для создания всевозможных звуковых эффектов, сопровождающих выполнение программы, и исполнения музыкальных произведений, однако необходимый при этом перевод нот в форму много-разрядных чисел является занятием весьма затруднительным. Программирование мелодий гораздо удобнее осуществить с помощью специального музыкального языка, реализуемого оператором с ключевым словом PLAY, имеющим вид

PLAY "музыкальная строка"

где "музыкальная строка" в общем случае содержит номер октавы (от 0 до 6; первая октава имеет номер 3) и последовательность нот.

Элементы музыкальной строки называются подкомандами оператора PLAY. Они имеют самое различное назначение в соответствии с правилами музыкальной грамоты. В частности, в подкоманде, обеспечивающей звучание, чистые ноты в нужной октаве обозначаются буквами C, D, E, F, G, A, B. Если нота с диэзом, то после буквы указывается знак плюс или \sharp , с бемолем — знак минус. Октава обозначается буквой O с последующим номером; так первая октава обозначается как O3.

Всего в семи используемых октавах содержится 84 звука, для обозначения которых в операторе PLAY вместо номера октавы и буквенных наименований нот можно указать в виде подкоманды N порядковый номер ноты от 0 до 84 (0 означает паузу).

Таким образом, например, для воспроизведения всего звуко-ряда, начинающегося с «до» первой октавы с включением полутонов, может быть использован любой из следующих операторов

10 PLAY "O3 C \sharp D D \sharp E F F \sharp G G \sharp A A \sharp B"

10 PLAY "N37 N38 N39 N40 N41 N42 N43 N44
N45 N46 N47 N48"

Для явного указания длительности нот используется подкоманда L, пауз — подкоманда P; для этих же целей может быть использована точка. Существует подкоманда T для задания темпа и ряд подкоманд, управляющих порядком исполнения описанных в программе мелодий. В результате с помощью оператора PLAY можно добиться вполне приемлемого звучания простых мелодий. Составление программы сводится к компоновке блока данных оператора DATA, куда заносятся подкоманды оператора PLAY. Далее строки подкоманд считываются такт за тактом в последовательный текстовый массив и исполнение мелодии, хранящейся в массиве, уже становится просто вопросом использования соответствующей подкоманды оператора PLAY, устанавливающей очередность исполнения каждого элемента массива.

Подробно эти вопросы как и приемы программирования музыки для компьютеров с несколькими звуковыми генераторами, работающими независимо и воспроизводящими звуки заданных длительности, тембра и громкости, в этой книге не рассматриваются.

§ 22. Примеры программ

1. **Нахождение простых чисел.** Одним из методов нахождения простых чисел является метод Эйлера, заключающийся в делении любого нечетного числа N на все найденные простые числа, меньшие \sqrt{N} .

Первые три простых числа 1, 2, 3 можно считать известными, тогда проверять следует лишь все последующие нечетные числа. Если нечетное число N не делится нацело ни на одно из уже найденных простых чисел, меньших \sqrt{N} , то оно простое.

Программа для нахождения первых 100 простых чисел будет иметь вид

```
10 DEFINT F, I, N, J, K
20 DIM F(100)
30 F(1)=1:F(2)=2:F(3)=3
40 I=3:N=3
45 REM I—КОЛИЧЕСТВО НАЙДЕННЫХ ПРОСТЫХ
    ЧИСЕЛ
50 N=N+2
60 IF N-3*INT(N/3)=0 THEN 50
70 S=SQR(N)
80 FOR J=2 TO I
90 IF F(J)>S THEN 120
100 IF N-F(J)*INT(N/F(J))=0 THEN 50
110 NEXT J
120 I=I+1
130 F(I)=N
140 IF I<100 THEN 50
150 REM ЗАПИСЬ В ФАЙЛ
160 OPEN "CAS:SIMPL" FOR OUTPUT AS # 1
170 FOR J=1 TO 100
180 PRINT # 1, F(J)
190 NEXT J
200 CLOSE # 1
210 REM ВЫВОД НА ЭКРАН
220 FOR J=1 TO 95 STEP 6
230 FOR K=0 TO 5
240 PRINT USING "# # # # #"; F(K+J)
250 NEXT K
```

```

260 PRINT
270 NEXT J
280 END

```

В программе для сокращения вычислений исключаются все числа N , которые делятся на 3 (строка 60). Аналогичным образом в строке 90 проверяется, делится ли число N на $F(J)$. Если N делится на $F(J)$, то $\text{INT}(N/F(J))=N/E(J)$.

Полученный массив из 100 чисел сначала записывается в файл на магнитной ленте с именем CAS : SIMPL, затем выводится на экран по 6 чисел в строке.

2. Построение графика функции. Пусть линейная функция $y=kx$ определена для значений независимой переменной x от 0 до 200, и требуется построить семейство прямых линий при заданных значениях коэффициента k , например, в диапазоне от 0.1 до 1.0. Программа построения указанных зависимостей имеет вид

```

10 DATA 0.1, 0.2, 0.3, 0.4, 0.5
20 DATA 0.6, 0.7, 0.8, 0.9, 1.0
30 DEF FNLIN (K, X)=K*X
40 SCREEN 2
50 FOR I=1 TO 10
60 READ A(I)
70 NEXT I
80 LINE (0, 0) — (255, 0), 1
90 LINE (0, 0) — (0, 191), 1
100 XO=0
110 XK=200
120 FOR I=1 TO 10
130 LINE (XO, FNLIN (A(I), XO)) — (XK, FNLIN(A(I), XK)), 1
140 NEXT I
150 END

```

В этой программе предварительно формируется блок данных, из которого значения k считываются как элементы одномерного массива $A(10)$. В операторе LINE при построении прямолинейных отрезков достаточно указать начальное XO и конечное XK значения переменной X . Операторы в строках 80 и 90 выполняют построение осей координат.

3. Решение квадратного уравнения. Программа для вычисления корней квадратного трехчлена ax^2+bx+c в соответствии с блок-схемой, приведенной в § 3 гл. I, может быть записана в виде

```

10 INPUT A, B, C
20 IF A < > 0 THEN 100
30 IF B < > 0 THEN 70
40 K=0
50 PRINT A; B; C; K; "НЕТ РЕШЕНИЯ"

```

```

60 GO TO 210
70 K=1
80 X1R=-C/B : X2R=0
90 GO TO 190
100 K=2
110 D=B^2-4*A*C : A2=2*A
120 IF D>=0 THEN 170
130 X1R=X2R=-B/A2
140 X1I=SQR(-D)/A2
150 X2I=-X1I
160 GO TO 200
170 S=SQR(D)
180 X1R=(-B+S)/A2 : X2R=(-B-S)/A2
190 X1I=X2I=0
200 PRINT A, B, C, K, X1R, X2R, X1I, X2I
210 END

```

В этой программе ввод значений коэффициентов A, B, C предполагается с клавиатуры. В вырожденном случае ($A=B=0$) осуществляется плотная печать значений коэффициентов и признака K, а также текста НЕТ РЕШЕНИЯ; в остальных случаях ($K=1$ и $K=2$) выводимые значения размещаются каждое в своей зоне.

§ 1. Данные

Все величины, из которых могут быть образованы выражения и над которыми могут быть совершены определенные действия, задаваемые операторами, называются *данными*. В языке ПЛ/1 используются следующие виды данных: константы, переменные, массивы, структуры. Массивы бывают внутренние, подразделяемые на *n*-мерные массивы и массивы структур, и внешние массивы, или файлы.

Для записи элементов языка — данных, выражений и операторов — используется 60 символов, составляющих *алфавит* языка:

— 29 букв: прописные буквы от А до Z, \$ — знак денежной единицы, @ — коммерческий знак «at», # — знак номера;

— 10 цифр от 0 до 9;

— 21 специальный символ:

+	(плюс),	:	(двоеточие),
—	(минус),	<	(меньше),
*	(звездочка),	>	(больше),
/	(наклонная черта),		(ИЛИ),
=	(равно),	&	(И),
((открывающая скобка),	⌋	(НЕ),
)	(закрывающая скобка),	%	(процент),
,	(запятая),	—	(символ разбивки (черта под строкой)),
.	(точка),		
'	(апостроф),	?	(вопрос),
;	(точка с запятой),		(пробел, пусто).

Символ пробел не имеет начертания, но в тексте иногда пишут символ \square на месте пробела.

Кроме указанного 60-символьного алфавита, разрешается пользоваться его 48-символьным подмножеством, в котором отсутствуют символы @ и # первой группы, в третьей группе присутствует только 11 символов: плюс, минус, звездочка, наклонная черта, равно, открывающая скобка, закрывающая скобка, запятая, точка, апостроф, пробел, а вместо отсутствующих символов ; , : , < , > , | , & , ⌋ , % приняты следующие замены: (соответственно) ,. , .., LT, GT, OR, AND, NOT, //.

При этом:

— перед двоеточием должен стоять пробел, если предыдущий символ есть точка, т. е. `┐┐..`;

— если впереди или после символов `//`, заменяющих знак `%`, стоит символ `*`, то он отделяется от `//` пробелом, т. е. `*┐//┐*`;

— символ `.` обозначает точку с запятой лишь в тех случаях, когда за ним не стоит цифра и если он не стоит внутри примечания или строки символов.

Для конструкций языка, обозначаемых двумя символами, написанными подряд, в случае 48-символьного алфавита вместо символов `<=`, `>=`, `┐=`, `┐<`, `┐>`, `||`, `—>` введены обозначения (соответственно) `LE`, `GE`, `NE`, `NL`, `NG`, `CAT`, `PT`. Если такому символу предшествуют или за ним непосредственно следуют буква или цифра, то между ними должен стоять пробел, например `A┐NE┐1.2`.

1. Константы. Константы различают *арифметические* и *строковые*. Числа действительные и комплексные, двоичные и десятичные, с плавающей и фиксированной точкой, с различной точностью составляют арифметические константы. Точность числа с фиксированной точкой определяется заданием количества цифр в числе и указанием количества дробных разрядов, а точность числа с плавающей точкой — заданием количества значащих цифр в числе. К строковым константам относятся символьные строки (константы типа строки символов) и битовые строки (константы типа строки битов).

Последовательность десятичных цифр, среди которых содержится точка, с предшествующими знаками `+` или `-` или без знака, образует *действительную десятичную константу* (десятичное число) с фиксированной точкой. Например:

`+123.45 -67.890 0.9`

Действительная десятичная константа с фиксированной точкой, при записи которой отсутствует точка, называется *действительной целой десятичной константой*. Например:

`35 -67 +289`

Действительная десятичная константа с плавающей точкой образуется как последовательность, состоящая из действительной десятичной константы с фиксированной точкой (мантиссы), символа `E` (основания порядка) и действительной целой десятичной константы, содержащей не более двух цифр (порядка). Так, записи

`21.5E+3 35E-02 .035E5`

изображают числа 21.5×10^3 , 35×10^{-2} , 0.035×10^5 соответственно.

Если в изображении десятичного числа с плавающей точкой содержится шесть или меньше значащих цифр, то в памяти машины число хранится с обычной точностью и занимает полное слово (4 байта). Если количество значащих цифр больше шести, то под

данное число отводится в памяти 8 байт (двойное слово) и число представляется с повышенной (двойной) точностью.

Действительная двоичная константа (двоичное число) с фиксированной точкой записывается при помощи последовательности цифр 0 и 1, в которой может содержаться точка, а предшествовать один из знаков + или —, и обязательного элемента — символа В — в конце последовательности. Например:

1101.011В —.0010111В +100011В

Последнее число записано без точки, образуя тем самым действительную целую двоичную константу.

Действительная двоичная константа с плавающей точкой записывается, например, так:

1.11011ЕЗВ —.00101Е—4В +1101Е—02В

т. е. обязательным элементом является символ В после порядка. Порядок — это действительная целая десятичная константа, являющаяся показателем степени при основании 2. Следовательно, 1.11011ЕЗВ — это 1.11011×2^3 , т. е. 1110.11.

Под двоичное число с плавающей точкой в памяти отводится полное слово, если число содержит 21 или меньшее количество значащих цифр, в противном случае оно занимает двойное слово.

Мнимая константа — это любая действительная константа, за которой следует символ I. Например:

2.16I — мнимая десятичная константа с фиксированной точкой;

101BI — мнимая двоичная константа с фиксированной точкой;

21.6Е—1I — мнимая десятичная константа с плавающей точкой.

Комплексная константа записывается как сумма двух величин — действительной и мнимой констант. Например:

21.6+3.16I 2.35Е+0—1.75Е—0II

Комплексная константа хранится в памяти как пара действительных чисел.

Символьная строка записывается в программе в виде последовательности символов алфавита, заключенной в апострофы (кавычки). В апострофы могут быть заключены любые символы, в том числе символ ', который следует записывать дважды. Так, символьные строки

'ТЕКСТ' '//5+\$500.' 'IT'S'

представляют собой такие последовательности символов:

ТЕКСТ //5+\$500. IT'S

Битовая строка образуется из цифр 0 и 1, заключаемых в апострофы; после правого символа ' пишется буква В. Например,

битовым строкам

'1101'B '0010'B '11111'B

соответствуют последовательности

1101 0010 11111

Строка может быть повторена несколько раз, если перед открывающей кавычкой написать в скобках коэффициент повторения в виде целого десятичного числа без знака. Так, строкам

(2)'ABC' (6)'1'B (0)'TEXT'

соответствуют последовательности

ABCABC 111111

В последнем примере приведена пустая строка (коэффициент повторения равен нулю), которая представляет собой строку без символов.

2. Переменные, массивы, структуры. *Переменные* по своему назначению делятся на арифметические и управляющие. Различают два вида переменных: простые (*скалярные*) переменные и переменные с индексами. Для обозначения простой переменной служит идентификатор. Переменная с индексами записывается как идентификатор с последующим списком индексов. Переменные с индексами объединяются в *массивы*. *Структура* образуется как упорядоченная совокупность данных. Элементами структуры могут быть простые переменные, массивы и другие структуры.

Арифметические переменные делятся на переменные действительного и комплексного типов и строковые (типа строки символов), а управляющие переменные — на переменные типа метки, указателя, ветви, события, области, ячейки. Ниже из управляющих переменных будут рассмотрены только первые два типа.

Упражнения

1. Преобразовать записи, составленные из символов 60-символьного алфавита, в записи, составленные из символов 48-символьного алфавита:

A B C	M=K =L;	X& Y
P—>Q	30*%(S>=SI)	25% =S
X&Y	X >Y <Z	(1.:10.:9:10)
2.:3.14.:.5	27<72	X Y Z

2. Среди приведенных ниже записей констант определить те, которые являются: а) десятичными числами; б) двоичными числами; в) действительными числами; г) комплексными числами; д) константами с плавающей точкой; е) символьными строками; ж) битовыми строками; з) записями, не допускаемыми правилами языка:

123.45B	40E0B	'0100'B1
37.73	101E1B	'SIN'

—347.	'#5138B'	(2) 'TAN'
.12E—16	1001IB	11—11I
11E—01	101E1IB	11—11E—11I
55D—02	.01B	10B+101E1BI
'11E+01'	2733B	(11)'101'B
(7)'10'B	1E1I+1	55 ₁₀ —02

3. Представить в виде десятичных чисел с плавающей точкой так, чтобы мантисса содержала три цифры, первая из которых не нуль:

100	12×10^{-3}	2×2^3
—12,3	0,0003	1/3
2,75	—666	0,0088
—1/8	370	-10^{-10}

§ 2. Идентификаторы

Для обозначения переменной служит *идентификатор* — последовательность не более чем из 31 символа. Первым символом идентификатора должна быть латинская буква или символы @, # или \$. Кроме указанных символов, в записи идентификаторов разрешается использовать цифры, а также символ разбивки, который может быть использован в любом месте идентификатора, за исключением его начала и конца. Например,

A X5 NAME—OF—INDEX \$1000 #77

Идентификаторами обозначаются также метки операторов, формальные параметры подпрограмм, указатели входов в подпрограммы, метки подпрограмм, имена файлов.

Ряд идентификаторов в виде последовательностей букв, являющихся словами английского происхождения, используется для обозначения объектов языка, называемых операторами, описателями, дополнениями, ситуациями прерывания, встроенными функциями. Это служебные или ключевые слова, которые, однако, могут быть употреблены в любом другом смысле, например в качестве имен переменных. В таком случае истинный смысл данного идентификатора определяется транслятором контекстуально. При использовании 48-символьного алфавита сочетания букв, которыми заменяются специальные символы, т. е. составные символы LT, GT и др., а также ключевые слова могут быть использованы только в этом специальном смысле.

В качестве ключевых слов используются следующие идентификаторы (в скобках указываются допускаемые сокращения, дается также перевод на русский язык):

при записи операторов:

ALLOCATE — разместить, BEGIN — начало, CALL — вызвать, GLOSE — закрыть, DECLARE (DCL) — объявить, DELETE —

удалить, DISPLAY — показать, DO — выполнить, END — конец, ENTRY — вход, FORMAT — формат, FREE — освободить, GET — выбрать, GO TO — перейти к, IF -- если, LOCATE — разместить, ON — на, OPEN — открыть, PROCEDURE (PROC) — процедура, PUT — послать, READ — читать, RETURN — вернуть, REVERT — отменить, REWRITE — переписать, SIGNAL — сигнал, STOP — стоп, WRITE — записать;

для обозначения описателей:

ALIGNED — выровненный, AUTOMATIC — автоматический, BACKWARDS — обратный, BASED — базированный, BINARY (BIN) — двоичный, BIT — битовый, BUFFERED — с буфером, CHARACTER (CHAR) — символьный, COMPLEX (CPLX) — комплексный, CONSECUTIVE — последовательный, CONTROLLED (CTL) — управляемый, DECIMAL (DEC) — десятичный, DEFINED (DEF) — определенный по, DIRECT — прямой, ENTRY — входной, ENVIRONMENT (ENV) — оборудованный, EXCLUSIVE — с защитой, EXTERNAL (EXT) — внешний, FILE — файл, FIXED — фиксированный, FLOAT — плавающий, INDEXED — индексный, INITIAL (INIT) — начальный, INPUT — для ввода, INTERNAL (INT) — внутренний, KEYED — с признаком, LABEL — метка, LIKE — подобный, LINESIZE — размер строки, OUTPUT — для вывода, PACKED — упакованный, PAGESIZE — размер страницы, PICTURE (PIC) — шаблон, POINTER (PTR) — указатель, POSITION — позиция, PRINT — для печати, REAL — действительный, RECORD — запись, REGIONAL — региональный, RECURSIVE — рекурсивный, RETURNS — возвраты, SEQUENTIAL — последовательный, STATIC — статический, STREAM — поток, UNBUFFERED — без буфера, UPDATE — обновить, VARYING (VAR) — переменный;

при указании дополнений:

BY — шаг, BY NAME — по имени, COLUMN — столбец, COPY — копия, DATA — данные, EDIT — редактирование, ELSE — иначе, FROM — из, IGNORE — игнорировать, INTO — в, KEY — ключ, KEYFROM — ключ из, KEYTO — ключ в, LINE — строка, LIST — список, MAIN — главный, OPTIONS — выбор, PAGE — страница, REPLY — ответ, SET — установить, SKIP — пропустить, SNAP — состояние, STRING — строка, SYSTEM — системный, THEN — to, TO — до, WHILE — пока;

для задания ситуаций прерывания:

CHECK — проверка, CONDITION — состояние, CONVERSION (CONV) — преобразование, ENDFILE — конец файла, ENDPAGE — конец страницы, ERROR — ошибка, FINISH — окончание, FIXEDOVERFLOW (FOFL) — переполнение, NAME — наименование, NO — префикс ситуации прерывания, OVERFLOW (OFL) — переполнение, RECORD — запись, SIZE — размер,

STRINGRANGE (STRG) — диапазон строки, SUBSCRIPT-RANGE (SUBRG) — диапазон индекса, TRANSMIT — передача, UNDEFINEDFILE (UNDF) — неопределенный файл, UNDERFLOW (UFL) — исчезновение порядка, ZERODIVIDE (ZDÍV) — деление на ноль.

При написании в программе два идентификатора не должны непосредственно примыкать друг к другу. Они должны отделяться каким-нибудь знаком операции, примечанием, одним из символов-разделителей из набора

() , ' . ; = :

или пробелом. Сказанное относится также к константам и шаблонам (см. § 4, п. 2).

Что касается символа пробел, то он широко используется при построении конструкций языка. Пробелы в произвольном количестве могут стоять по обе стороны от знаков операций и символов-разделителей и допускаются внутри символьных строк. Однако идентификаторы, составные знаки операций, шаблоны, константы (кроме символьных строк) не должны содержать пробелов.

Идентификаторы используются также для обозначения так называемых встроенных функций — автоматически включаемых в программу подпрограмм, реализующих часто встречающиеся математические процедуры. Идентификаторы встроенных функций могут в программе использоваться для обозначения других объектов.

Упражнения

1. Среди приведенных ниже последовательностей символов выбрать идентификаторы. У остальных указать ошибки в записи.

DELTA	A50R17S	X1100LOGICAL
2PI	2.A	\$10E5
AREA	NAME_LJOE	IT'S
REAL	#1982	A_LB_LC_LD
@123	EC=1060	COEFFICIENT_L
ЗАДАНИЕ	TY_L156	DET OF INIT
FLOAT	AND	3NNHH
55ALPHA	'VARYING'	V(P)

2. Составить из символов I и 1 все возможные идентификаторы, состоящие из трех символов.

3. Сколько идентификаторов, состоящих из четырех символов, можно составить из: а) символов A, B, C, D; б) символов A, B, C, 1; в) символов A, B, 2, 1; г) символов A, 3, 2, 1; д) символов 4, 3, 2, 1?

§ 3. Операторы

Под *оператором* понимается предписание, приводящее к вполне определенному функционированию ЭВМ (к выполнению определенных действий). Все операторы могут быть разделены на два класса: *неисполняемые* и *исполняемые*.

К классу неисполняемых относятся операторы, используемые в программе для указания свойств величин, для описания формы представления данных на внешних носителях информации, для указания о распределении памяти и т. д. Это — оператор объявления данных, оператор размещения данных, оператор освобождения памяти, оператор форматов, оператор конца. При выполнении программы, если управление передается одному из этих операторов, он пропускается.

Класс исполняемых операторов составляют операторы, используемые в программе для указания конкретных действий и порядка выполнения этих действий. Это оператор присваивания, оператор перехода, условный оператор, оператор цикла, оператор блока, оператор процедуры, оператор останова, оператор вызова процедуры, оператор входа в процедуру, оператор возврата, оператор ожидания, оператор задания режима обработки прерываний, оператор отмены реакции на прерывание, операторы ввода и вывода, пустой оператор.

Различают следующие основные логические группы операторов: объявления данных, присваивания, управления, структуры программы, отладки, распределения памяти, ввода и вывода.

В общем случае оператор записывается в форме

(C): M: KC_XSL;

где C — список имен ситуаций прерывания, M — список меток, KC — ключевое слово оператора; XSL — тело оператора, представляющее собой список, элементами которого могут быть идентификаторы, обозначающие переменные, массивы, структуры, описатели, дополнения, элементы формата, другие операторы; символ-разделитель точка с запятой указывает конец оператора.

Основным (обязательным) элементом оператора (за исключением оператора присваивания и пустого оператора) является ключевое слово, обязательным элементом является также символ; (точка с запятой) в конце оператора, остальные элементы могут частично или полностью отсутствовать. В дальнейшем в каждом конкретном случае структура тела оператора будет подробно рассмотрена. Там же будет пояснен смысл элементов с различными наименованиями, которые входят в список тела оператора. В частности, если оператор содержит в списке тела оператора другой оператор, он называется составным. Соответственно простой оператор — это оператор, не содержащий в себе других операторов.

Простым оператором является пустой оператор, который никаких действий не выполняет и тело которого никак не обозначается. Следовательно, пустой оператор имеет вид

M: ;

где М — список меток — может отсутствовать. Однако если метка имеется, можно, передавая управление на пустой оператор, переходить на конец некоторого составного оператора или блока, пропуская какие-то другие операторы.

При записи операторов на бланке допускается так называемый свободный формат. Из 80 позиций строки для записи операторов разрешается использовать позиции 2—72, соответствующие колонкам 2—72 на перфокарте. Первая позиция не должна заполняться, а позиции 73 — 80, хотя и могут быть заполнены информацией, транслятором игнорируются. Для удобства чтения между элементами одного оператора и между соседними операторами разрешается оставлять пробелы. В одной строке может быть записано несколько операторов; один оператор может занимать несколько строк, при этом перенос разрешается в любом удобном месте.

Упражнение

Определить по формальным признакам, являются ли операторами следующие конструкции:

```
DECLARE A COMPLEX FIXED;  
SIZE: PROCEDURE;  
(NOSIZE): E=F*N;  
K: L: M: N: STOP;  
57: END;  
SYSTEM: ON SYSTEM;
```

§ 4. Описание данных

Для указания свойств идентификаторов, обозначающих переменные, массивы и структуры, применяются три способа: *явное*, *контекстуальное* и *неявное* (по умолчанию) *описания*.

Явное описание осуществляется оператором объявления данных (с ключевым словом DECLARE), а для данных, организованных в файлы, кроме того, еще оператором открытия файлов (с ключевым словом OPEN).

Контекстуальное описание осуществляется без явного указания каких-то свойств идентификаторов, а по имеющимся другим его свойствам или же на основе характера использования идентификатора в программе.

Неявное описание представляет собой присвоение определенных свойств идентификатору по предварительному соглашению (по умолчанию).

Оператор объявления данных имеет вид

```
DECLARE XL;
```

где элементы списка XL суть идентификаторы и описатели.

1. Описание простых арифметических переменных. Для описания арифметических данных используются описатели:

— для задания основания системы счисления DECIMAL и BINARY, указывающие соответственно, что значение величины представляется либо в десятичной, либо в двоичной системе счисления;

— для задания формы представления FLOAT и FIXED, указывающие соответственно, что значение величины представляется либо в форме с плавающей точкой, либо с фиксированной точкой;

— для задания типа REAL и COMPLEX, указывающие, что величина является либо действительной, либо комплексной;

— для задания точности (разрядности) (w, d) и (w) соответственно для данных с фиксированной точкой и данных с плавающей точкой. Здесь w — целое десятичное число без знака, представляющее собой общее (максимальное) количество цифр, допустимое для значения величины (значащие цифры в мантиссе в случае числа с плавающей точкой); d — десятичное целое число со знаком или без знака, при $d > 0$ — это количество дробных разрядов в десятичном или двоичном значении переменной (если значение d не задано, предполагается, что d равно нулю), при $d = 0$ точка располагается в конце числа; если $d < 0$, то соответствующее значение состоит из $w + |d|$ разрядов, причем младшие $|d|$ разрядов будут нулями.

При описании комплексных данных предполагается, что вещественная и мнимая части имеют одинаковые описатели (как пара действительных величин).

Некоторые из перечисленных описателей (или все) могут отсутствовать. В этом случае по умолчанию предполагается, что описываемым данным приписывается: из описателей основания системы счисления — DECIMAL, из описателей формы представления — FLOAT, из описателей типа — REAL, а точность, которая, вообще говоря, зависит от типа ЭВМ, обычно задается как (5, 0) или (6) для десятичных значений и как (15, 0) или (21) в случае двоичных значений.

Описатели в операторе объявления данных указываются после идентификатора, и могут следовать в произвольном порядке, однако непосредственно за идентификатором не может быть указан описатель разрядности.

Примеры описания простых переменных для арифметических данных:

```
DECLARE X REAL DECIMAL FIXED (5,3);  
DECLARE C REAL BINARY FLOAT;  
DECLARE F DECIMAL FIXED (3,—2);
```

Здесь с помощью трех операторов описаны переменные, принимающие значения: X — действительное, десятичное с фиксированной точкой, общее количество разрядов — 5, дробных — 3; C — действительное, двоичное с плавающей точкой и 15 (по умолчанию)

разрядами в мантиссе; F — действительное (по умолчанию), десятичное с фиксированной точкой, содержащее пять разрядов, из них младшие два состоят только из нулей (например, значением F может быть число 12300, но не 12340 или 12345).

В одном операторе может быть описано любое количество переменных. Так, приведенные операторы можно объединить в один; в этом случае между описаниями пишется разделитель запятая:

```
DECLARE X REAL DECIMAL FIXED (5,3),  
        C REAL BINARY FLOAT,  
        F DECIMAL FIXED (3,—2);
```

Разрешается использовать в программе переменные, идентификаторы которых не указаны в операторе объявления данных. В этом случае, если первый символ идентификатора есть одна из букв I, J, K, L, M или N, то соответствующей переменной приписываются по умолчанию описатели BINARY FIXED REAL (15, 0), во всех остальных случаях — описатели DECIMAL FLOAT REAL (6).

Для описания строковых переменных используются описатели CHARACTER и BIT. Первый из них является описателем символьной строки, второй — битовой строки. Длина строки указывается целым числом в скобках (p) в байтах для описателя CHARACTER и в битах — для описателя BIT. Например, оператором

```
DECLARE L CHARACTER (7), R BIT (8);
```

описываются символьная строка L, размещаемая на поле длиной 7 байт, и битовая строка R, занимающая поле длиной 8 бит (т. е. 1 байт), например:

```
'ABCDEFH' '01010101'
```

Для строковой переменной может быть задан описатель VARYING, который указывает, что строка имеет переменную длину, если он опущен, то считается, что строка имеет постоянную длину. Например, значениями переменной Y, объявленной оператором

```
DECLARE Y CHARACTER(5), VARYING;
```

могут быть строки

```
'ABCDE' 'ABCD' 'ABC' 'AB' 'A' ''
```

Описатель длины (p) должен следовать за описателем BIT или CHARACTER, опущен быть не может. Значение p — это длина в случае строк с постоянной длиной и максимальная длина для строк с переменной длиной. В качестве p допускается скалярное выражение, принимающее целое значение или символ *, если длина может быть взята из предшествующего размещения данных.

Максимально допустимая разрядность в случае описателей: DECIMAL FIXED равна (15, d), DECIMAL FLOAT — (16), BINARY FIXED — (31, d), а BINARY FLOAT — (53); для описателя длины p максимальное значение равно 32767.

Несколько переменных, которые описываются одинаково, могут быть объединены в один список, заключаемый в скобки, за которым описатели пишутся только один раз. Например, вместо
DECLARE A BIT (10), B BIT (10), C BIT (10);

можно написать

DECLARE (A,B,C) BIT (10);

2. Шаблоны. Строки символов и десятичные данные в форме с фиксированной или плавающей точкой могут быть описаны *шаблоном*.

Записывается шаблон с помощью описателя PICTURE, после которого следует шаблон, заключаемый в кавычки. Таким образом, общая форма оператора будет следующая:

DECLARE X PICTURE 'шаблон';

где X — идентификатор, 'шаблон' — строка определенных символов.

В случае комплексных данных указывается описатель COMPLEX и один шаблон, общий для вещественной и мнимой частей описываемой величины. Применение шаблонов для двоичных данных не предусмотрено.

Описание с помощью шаблона используется в случае необходимости внутреннего представления данных в памяти в виде строк, в ряде случаев, удобном для организации ввода или вывода, в частности для редактирования подлежащих печати выходных данных, когда необходимо вставить знак, подавить незначащие нули и т. д.

В шаблоне используются символы, на месте которых в описываемом шаблоне значении подразумеваются вполне определенные символы алфавита. Список символов шаблона для десятичных данных с указанием, что эти символы означают, следующий:

9 — любая десятичная цифра;

V — подразумеваемая точка;

S — место знака числа; если число больше нуля, обозначает +, если оно меньше нуля — знак минус;

± — место знака числа; если число больше нуля, обозначает +, если меньше нуля — пробел;

— — место знака числа; если число меньше нуля, обозначает минус, если число больше нуля или равно нулю — пробел;

E — место буквы E, указывающей начало показателя степени числа с плавающей точкой;

K — подразумеваемое начало показателя степени числа с плавающей точкой;

T — цифра с дополнительной пробивкой для знака числа на перфокарте над этой цифрой (в 12-й строке для знака ± и в 11-й строке для знака минус);

. — цифра с дополнительной пробивкой для знака числа над этой цифрой, если число на поле больше нуля или равно нулю;

R — цифра с дополнительной пробивкой для знака числа над этой цифрой, если число на поле меньше нуля;

CR — два символа, которые будут содержаться в соответствующих позициях, если число больше нуля или равно нулю;

DB — два символа, которые будут содержаться в соответствующих позициях, если число меньше нуля;

F — наличие в шаблоне масштабного множителя, который указывается в скобках как положительное или отрицательное целое число; не разрешается в случае полей с плавающей точкой;

Z — цифровая позиция, которая заменяется пробелом, если в ней содержится незначащий ноль (имеются в виду старшие разряды числа);

* — аналогичен символу Z, только незначащие нули заменяются звездочками;

Y — аналогичен символу Z, только все незначащие нули заменяются пробелами;

\$ — позиция, которая содержит символ \$; она может быть либо крайней левой, либо крайней правой;

, — позиция, в которую вносится символ запятой; если наличие символов Z или * вызвало подавление нулей и цифровые символы слева от запятой отсутствуют, то запятая заменяется пробелом в случае символа Z и звездочкой в случае символа *;

. — аналогична запятой, только вместо запятой используется точка;

/ — аналогична запятой, только используется наклонная черта;

B — пробел, который вводится в указанную позицию.

В шаблоне можно использовать только один символ знака из числа: S, +, —, T, I, R, CR, DB, за исключением числа с плавающей точкой, состоящего из двух частей — мантиссы и порядка, которые могут иметь собственные знаки. Символ знака должен занимать крайнее левое или крайнее правое положение в соответствующем поле. Для повторения символа в шаблоне перед ним в скобках записывается коэффициент повторения.

Примеры шаблонов с содержимым поля памяти и соответствующим числом, представляемым шаблоном:

'999V99'	12345	123.45
'(3)9V99'	12345	123.45
'S99V9'	—123	—12.3
'+99V9'	+123	12.3
'—99V9'	—123	—12.3
'V99ES99'	12E+03	.12×10 ³
'V99KS99'	12+03	.12×10 ³
'99F(—2)'	12	.12

Символы Z, *, Y, \$, ., /, B и , шаблона используются в основном при редактировании.

Символы *, Z и \$, S, +, —, если они появляются в шаблоне один раз, означают действия, описанные выше, и являются статическими символами. Появление их более одного раза в шаблоне превращает их в плавающие символы; это означает, что денежный знак, плюс, минус или пробел должны быть введены перед старшей значащей цифрой в поле или в крайнюю правую позицию плавающего символа в строке, т. е. слева будут пробелы, а знаки \$, +, — займут один символ в позиции, соответствующей самому правому пробелу.

Примеры результатов вывода числа с различными шаблонами:

12.34	'\$999V.99'	\$012.34
12.34	'\$ZZ9V.99'	\$└12.34
12.34	'\$**9V.99'	\$*12.34
12.34	'\$\$\$9V.99'	\$12.34

Для описания строк символов в шаблоне используются те же символы, что и для числовых шаблонов, но в шаблон для строки должен входить по крайней мере один из символов — A или X, где A указывает любую букву или пробел, а X — любой символ алфавита языка. Кроме того, в шаблоне символьных строк цифра 9 означает любую цифру от 0 до 9, а также пробел. Так, в операторе

```
DECLARE ТЕКСТ PICTURE 'XXXXXX',  
TEXT PICTURE '(4)A',  
TEST PICTURE '99AX';
```

описываются строковые переменные: ТЕКСТ — принимающая значение строковой константы, состоящей из пяти любых символов, TEXT — из четырех букв или пробелов, TEST — из двух цифр или пробелов, одной буквы или пробела и одного произвольного символа.

3. Описание массивов. Переменная с индексами записывается с помощью идентификатора, за которым в скобках указываются индексы в виде выражений. Количество индексов определяет размерность массива. Максимальное допустимое число индексов равно 32, однако в ряде трансляторов, где используется подмножество ПЛ/1, оно сокращено до трех. Примеры записи переменных с индексами:

```
M(7,1) A(1) XY(1,2,5,7,1,1,8)
```

При описании массива в операторе DECLARE сразу после идентификатора указываются границы изменения индексов (описатель измерения). В остальном описание массива совпадает с описанием простой переменной, следовательно, все элементы массива имеют одинаковый тип и одни и те же описатели.

Границы изменения индексов задаются в виде граничной пары, состоящей из нижней границы, двоеточия и верхней границы.

Предполагается, что индекс принимает значения от нижней границы до верхней с шагом $+1$. Границы изменения индексов — целые десятичные константы (возможно, отрицательные и нуль) или скалярные арифметические выражения. Например, оператор

```
DECLARE F(0:2) DECIMAL FIXED (6,0),  
        E(-3:5,1:8), A(1:2,1:2);
```

описывает три массива: одномерный F с элементами (переменными с индексами) $F(0)$, $F(1)$, $F(2)$; двумерный $E(-3,1)$, $E(-3,2)$, ..., $E(-2,1)$, ...; двумерный $A(1,1)$, $A(1,2)$; $A(2,1)$, $A(2,2)$.

В памяти элементы массива записываются с последовательным упорядочением индексов, начиная с первого (иными словами, первым изменяется последний индекс).

Если нижняя граница изменения индекса равна $+1$, то в описателе измерения можно указывать лишь верхнюю границу. Следовательно, последний массив из предыдущего примера можно описать так:

```
DECLARE A(2,2);
```

Допускается объявление массивов с переменными (пока неизвестными) верхними границами изменения индексов, если массив является параметром процедуры. В этом случае в операторе `DECLARE` вместо всех границ следует писать символы $*$. Например:

```
DECLARE M(*,*) BIT (4);
```

Максимальное значение, которое может быть использовано в качестве индекса, равно 32767.

Указание в выражениях или операторах идентификатора массива означает, что обращение делается ко всем элементам массива. Указание идентификатора с индексами означает обращение к конкретному элементу массива. Обращение к части массива осуществляется как обращение к *сечению* массива. Обращение к сечению массива производится при помощи имени массива, за которым следует список индексов, по крайней мере один из которых есть $*$. Так, если в i -й позиции в списке индексов стоит $*$, то сечение включает все элементы массива, получаемые при изменении i -го индекса между его границами. Размерность сечения равна количеству звездочек в списке индексов. Например:

$A(3, *)$ означает обращение к третьей строке массива A ,

$B(*, *, 2)$ — к двумерному сечению (второй плоскости) массива B ,

$A(*, *)$ — обращение ко всему массиву, т. е. эквивалентно указанию идентификатора массива A .

4. Описание структур. Описание структуры производится оператором `DECLARE`, в котором указывается идентификатор (имя) структуры, идентификаторы групп данных, входящих в структуру

(имена подструктур), и идентификаторы первичных составляющих (имена элементов структуры). Для указания иерархии соподчинения идентификаторов перед каждым из них записывается номер уровня — десятичное целое число без знака, меньшее 256. Самая внешняя структура называется старшей и номер ее равен 1; содержащиеся в ней структуры являются младшими. Младшие структуры должны всегда иметь номер уровня, численно больший номера уровня той структуры, в которой они содержатся. Младшим структурам присваиваются не обязательно последовательные номера.

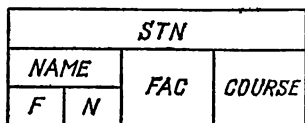


Рис. 3

П р и м е р. Требуется дать сведения о каждом из студентов-спортсменов, представляющих различные факультеты и курсы учебного заведения, т. е. необходимо указать номер факультета, на котором учится студент, номер курса, имя и фамилию студента. Организация такой структуры с идентификатором (именем) STN схематически изображена на рис. 3, где NAME — идентификатор студента, состоящий из фамилии F и имени N; FAC и COURSE — соответственно имена подструктур (факультета и курса).

Соответствующий оператор DECLARE примет вид

```
DECLARE 1 STN,
        2 NAME,
          3F CHARACTER (16),
          3N CHARACTER (16),
        2 FAC DECIMAL FIXED (3,0),
        2 COURSE DECIMAL FIXED (2,0);
```

В этом примере старшая структура с именем STN состоит из подструктур NAME, FAC и COURSE, а подструктура с именем NAME в свою очередь содержит подструктуры F и N. Запись оператора DECLARE «в столбик» наглядна, но не обязательна; повторяющиеся номера уровней и описатели можно вынести за скобки. Например, приведенный оператор можно записать так:

```
DECLARE 1 STN,
        2 (NAME, 3(F,N) CHARACTER (16),
          FAC DECIMAL FIXED (3,0), COURSE DECIMAL
          FIXED (2,0));
```

Однотипные структуры из предыдущего примера, содержащие сведения о каждом из студентов рассматриваемой группы, могут быть объединены в более сложные конструкции — массивы структур. Например, если студентов-спортсменов всего 40, то сведения о всей группе могут быть описаны оператором, совпадающим с только что приведенным, за исключением первой строки, принимающей

вид

DECLARE 1 STUDENT(40),

где идентификатор STUDENT — имя массива структур из 40 элементов.

Для обращения к элементам структуры используется составное или уточненное имя, которое представляет собой последовательность идентификаторов, записываемых один за другим в порядке увеличения номеров уровней структуры. Идентификаторы последовательности отделяются точкой, вокруг точки разрешается оставлять пробелы. Составное имя не обязательно содержит имена всех структур, но должно включать достаточно имен для того, чтобы избежать неоднозначности. Например, если имеем описание

DECLARE 1 A, 2 B, 3(D,E), 2 C, 3 D CHARACTER (4);

то составное имя A.B.D означает обращение к элементу D подструктуры B, а составное имя A.C.D — обращение к элементу D подструктуры C. Очевидно, однозначными будут также обращения B.D и C.D.

Для обращения к структуре в целом указывается ее идентификатор. Обращение к подструктуре означает, что выполняется обращение ко всем ее составляющим.

Если структура является компонентой массива структур, то обращение записывается в форме переменной с индексами, образуя составное имя с индексами. Составное имя с индексами должно иметь по крайней мере один индекс. Так, для структуры S, определяемой оператором

DECLARE 1 S(5,10), 2 T(7), 3 (R(6), Q);

составные имена S(1,3).T(5).R(2), S(1).T(3,5).R(2), S(1).T(3).R(5,2) и т. д. относятся все к одному и тому же элементу. Использование составного имени с индексами в ряде случаев допускает перемещение индексов от левых идентификаторов к правым и сокращения числа идентификаторов структур в составном имени. Так, возможны составные имена, относящиеся к одному и тому же только что приведенному элементу S.T(1,3).R(5,2), T(1).R(3,5,2), R(1,3,5,2).

Для описания двух одинаковых структур или подструктур может быть использован описатель подобия LIKE. Например, оператор

DECLARE 1 ST, 2(PS1, 3(A,B), PS2, 3(A,B), PS3, 3(A,B));

может быть переписан в следующем виде:

DECLARE 1 ST, 2 PS1, 3(A,B), 2(PS2, PS3) LIKE PS1;

Описатель подобия воспроизводит имена, описатели и размерности, появляющиеся только после указанного идентификатора, т. е. после разделителя запятой. Никакие описатели самой структуры,

указываемой вместе с описателем LIKE, не переносятся на структуру, описанную с помощью этого описателя. Так, если в предыдущем примере идентификатор PS1 имеет описатель размерности, скажем, (5), то для обеспечения этой размерности у идентификаторов PS2, PS3 она должна быть явно указана: PS2 (5) и PS3 (5). Например, оператор

```
DECLARE 1 ST, 2 PS1(5), 3 (A,B), 2 (PS2,PS3) LIKE PS1;
```

эквивалентен оператору

```
DECLARE 1 ST, 2(PS1(5), 3(A,B), PS2, 3(A,B), PS3, 3(A,B));
```

а оператор

```
DECLARE 1 ST, 2 PS1(5), 3(A,B), 2(PS2(5),PS3(5)) LIKE PS1;
```

равносилен следующему оператору:

```
DECLARE 1 ST, 2(PS1(5), 3(A,B), PS2(5), 3(A,B), PS3(5), 3(A,B));
```

5. Файлы. Под *файлом* понимается совокупность логически связанных данных, размещенных на внешних носителях информации. Описание файла выполняется оператором объявления данных в виде

```
DECLARE F FILE FL;
```

где FILE — описатель, указывающий, что описываемый идентификатор есть имя файла, F — идентификатор (имя) файла, FL — описатели файла, характеризующие способ обработки файла, его функцию, метод доступа к данным, а также признаки файла, связанные с особенностями ЭВМ. Описатели файла будут рассмотрены в § 14.

6. Повторные определения. Значения некоторой переменной (массива, структуры), которая называется определяемой, могут быть отождествлены со значениями другой переменной (массива, структуры), называемой *базовой*, с помощью описателя DEFINED, осуществляющего *повторное* определение. Повторные определения могут быть двух типов: определение по соответствию и определение по наложению (или совмещению).

При определении по соответствию некоторой ранее описанной (базовой) величине дается новое имя, либо (в случае массивов) присваивается другое имя части массива. Например, оператор

```
DECLARE A(3,3) FIXED (4,0),  
B(3,3) DEFINED A FIXED (4,0);
```

присваивает два имени A и B одному массиву. Описатели A и B должны быть идентичными.

При определении по соответствию допускается использование специальных индексов (фиктивных переменных) вида iSUB, где $i=1, 2, \dots, n$. Эти индексы указывают первый, второй и т. д. индексы базовой переменной и используются в качестве индексов повторно

определяемой переменной. Например, оператор

```
DECLARE X(15), Y(6) DEFINED X(1SUB);
```

переопределяет переменные X(1), X(2), ..., X(6) как Y(1), Y(2), ..., Y(6). Индекс 1SUB задает первое значение индекса определяемого массива Y.

Если задан, например, такой оператор:

```
DECLARE A (9,9) FIXED (5,0), C(9) FIXED (5,0)  
DEFINED A (1SUB, 10—1SUB);
```

то он определяет часть массива A, которой присваивается другое имя C. Элементы массива C определяются по правилу

$$C(j)=A(j, 10-j), j=1, 2, \dots, 9.$$

Определения по соответствию разрешаются для всех величин, за исключением массивов структур, при этом обращение к элементам определяемой величины интерпретируется как обращение к соответствующему базовому элементу.

Определение по совмещению означает, что отождествление определяемой величины с базовой осуществляется путем совмещения памяти, т. е. определяемая величина занимает либо всю память, отведенную под базовую величину, либо ее часть. Например, если задан оператор

```
DECLARE D CHARACTER (10),  
E CHARACTER (8) DEFINED D;
```

то переменная E займет те же самые 8 байт, которые выделяются для записи первых 8 символов переменной D.

Заметим, что определение по соответствию требует идентичных описаний данных, в то время как определение по совмещению не требует этого.

Для определяемой по совмещению величины может быть указан описатель POSITION(d), указывающий, что совмещение производится, начиная с позиции номер d базовой величины (d — целая десятичная константа без знака). Например, оператор

```
DECLARE D CHARACTER (10),  
E CHARACTER (8) DEFINED D POSITION (3);
```

указывает, что 8-символьное поле определяемой переменной E начнется с третьей позиции базовой переменной D.

7. Данные типа метки. Данные типа *метки* подразделяются на константы и переменные. Константы типа метки — это идентификаторы-метки, которые записываются с последующим двоеточием перед операторами. Например:

```
PROGRAM_1: S;
```

где S — оператор. Перед оператором может стоять несколько меток.

Для организации передач управления одним оператором перехода различным участкам программы служит переменная типа метки, значением которой является константа типа метки. Переменная типа метки описывается оператором объявления данных с описателем LABEL. Например, оператор

```
DECLARE (M, LB(6)) LABEL;
```

описывает переменную типа метки M и массив переменных типа метки, состоящий из 6 элементов.

Переменная типа метки не может быть использована в качестве метки оператора. Переменной типа метки может быть присвоено значение только другой переменной типа метки или константы типа метки. Константы и переменные типа метки не могут быть операндами выражений.

8. Данные типа указателя. Управление размещением данных в оперативной памяти осуществляется с помощью переменных типа *указателя*, которые описываются оператором объявления данных с указателем POINTER. Например;

```
DECLARE P POINTER;
```

Значением переменной P является адрес поля (ячейки) памяти ЭВМ, который в тексте программы не указывается. Способы оперирования переменными типа указателя будут рассмотрены ниже.

Данные типа метки и типа указателя, а также данные типа ветви, события, области, ячейки, которые здесь не рассматриваются, относятся к управляющим данным.

9. Начальные значения. При описании переменных в оператор DECLARE может быть добавлен описатель INITIAL (IL), где IL — список значений, указывающий *начальные* значения, которые будут присвоены описываемым переменным. Например:

```
DECLARE X FIXED DECIMAL (6,0) INITIAL (16), Y (0:3,2)  
        FIXED DECIMAL (2,1) INITIAL (1,0,0,0,1,0,0);
```

Если нескольким переменным присваивается одно и то же значение, то используется коэффициент повторения (повторитель), который записывается в скобках перед значением. Например, в предыдущем примере последний описатель можно написать как

```
INITIAL (1,(4)0,1,(2)0)
```

или

```
INITIAL (1,(2)(0,0),1,(2)0)
```

Допустимо указание начальных значений части массива. Например, возможен описатель

```
INITIAL (2,3,(2)*,4,*)
```

указывающий, что начальные значения получают первый, второй и пятый элементы массива, т. е. звездочка пишется на месте тех эле-

ментов, для которых начальное значение не задается. Если в массиве элементов больше, чем начальных значений, то предполагается, что оставшиеся элементы значений не получают. Коэффициент повторения может быть и произвольным выражением. Отрицательный или нулевой коэффициент повторения не вызывает присвоения начальных значений.

В случае присвоения начальных значений строковым переменным, при записи которых используется повторитель строки, повторитель начальных значений необходимо записывать первым. Так, оператор

```
DECLARE C(3) CHARACTER (5) INITIAL ((3)'H'),  
D(4) CHARACTER (6) INITIAL ((4)(2)'ABC');
```

присваивает переменной C(1) начальное значение 'H', а переменным C(2), C(3) начальные значения не присваиваются; всем четырем элементам массива D присваиваются начальные значения 'ABCABC'.

Начальное значение, которое присваивается строковой переменной, может иметь любой тип, не разрешено лишь использовать комплексное число.

Начальное значение может быть присвоено переменной типа метки. Например, оператор

```
DECLARE M LABEL INITIAL (B);
```

присваивает переменной M начальное значение B, где B — метка из того процедурного блока, где содержится данный оператор объявления данных.

Описатель INITIAL не может задаваться для структур и для переменных типа указателя.

При повторном определении присвоение начальных значений не допускается, разрешается присвоение начальных значений лишь базовым элементам. Например, возможен оператор

```
DECLARE A(3,3) FIXED (5,0) INITIAL ((9)0),  
B(3,3) DEFINED A FIXED (5,0);
```

Упражнения

1. Определить, какие из операторов объявления данных не содержат ошибок:

```
DECLARE A REAL DECIMAL, B, C REAL (5), D COMPLEX;  
DECLARE X(5) FIXED, Z COMPLEX, BINARY;  
DECLARE Y BIT (5), E FIXED, F CHARACTER VARYING (2);  
DECLARE T PICTURE '(3)X', X PICTURE 'X';  
DECLARE (H, BIT (5), G CHARACTER (5)) VARYING;  
DECLARE I(*) BIT (*), J(0:5, *) CHARACTER (5);  
DECLARE 1 S1, 5 S2, 8 S3, 7 S4, 5 S5, 3 S6, 3 S7;  
DECLARE 1 S, 7 SK, 17 SL LIKE SK;
```

```
DECLARE M FIXED (2, 0) N (2) DEFINED M FIXED (2, 0);
DECLARE P LABEL (3), Q POINTER;
DECLARE A INITIAL (15. 0), K INITIAL (11B);
```

2. Описать в виде массива матрицу целых десятичных чисел:

$$\begin{pmatrix} a_{01} & a_{02} & a_{03} & a_{04} & a_{05} & a_{06} \\ a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} \end{pmatrix}.$$

3. Массив описан оператором

```
DECLARE M(-1: 7, 2: 5) DECIMAL FIXED;
```

Вычислить порядковые номера элементов:

```
M(1,2)      M(0,5)      M(4,5)
M(6,5)      M(7,4.3)    M(5.1,5)
M(3,3)      M(6.1,2)    M(6,4)
M(5,3.2)    M(-1,3.4)   M(1,3.4)
M(6,4.7)    M(3,4.4)    M(1,2)
```

4. Составить все возможные составные имена элемента R(0,1,2, 3) массива R, входящего в состав подструктуры S массива структур, определяемого оператором

```
DECLARE 1 P(-1: 2, 0: 3), 2 R, 2 S, 4 R(3,3), 2 T;
```

§ 5. Выражения

Выражение определяется как имеющая смысл комбинация из операндов, знаков операций и скобок такая, что после выполнения всех операций получается вполне однозначный результат. Различают скалярные выражения, выражения над массивами и выражения над структурами. Эта классификация соответствует виду образующих выражение данных и виду результата выражения. Описатели данных, входящих в выражение, могут быть различными. Выражение, результат которого есть число (массив чисел), будем называть арифметическим выражением; если результат есть строка символов или строка битов, то выражение соответственно называется выражением типа строки символов и выражением типа строки битов.

1. Скалярное выражение. Операндами *скалярного* выражения могут быть константы, идентификаторы простых переменных, переменные с индексами, составные имена, задающие обращения к значениям элементов структур, обращения к функциям, скалярные выражения, заключенные в скобки. При построении выражений допустимы следующие операции, обозначаемые соответствующими символами.

Арифметические: 1) одноместные $+$ и $-$, 2) $**$ (возведение в степень), 3) $*$ и $/$ (умножение и деление), 4) $+$ и $-$ (сложение и вычитание).

Логические: 1) \neg (отрицание), 2) $\&$ (логическое умножение), 3) $|$ (логическое сложение).

Отношения: 1) $=$ (равно), 2) \neq (не равно), 3) \geq (больше или равно), 4) \leq (меньше или равно), 5) $<$ (меньше), 6) $>$ (больше), 7) $\neg \geq$ (не больше), 8) $\neg <$ (не меньше).

Сцепления: \parallel .

Скалярное выражение вычисляется слева направо с учетом скобок и следующего приоритета операций: 1) $+$ и $-$ одноместные, $**$, \neg , 2) $*$ и $/$, 3) $+$ и $-$, 4) \parallel , 5) операции отношения, 6) $\&$, 7) $|$. Исключение составляют операции наивысшего (первого) ранга, которые выполняются справа налево.

Примеры выражений:

$+AX \quad (A \& IB) \quad K \parallel L$
 $-B/C \quad IA**B5 \quad D=C$

При выполнении арифметических операций, если операнды в выражении различны, производятся преобразования:

- операнды в форме числового поля (PICTURE) преобразуются в кодовую форму, соответствующую описателям DECIMAL FIXED;
- десятичный операнд преобразуется к двоичному основанию;
- операнд с фиксированной точкой преобразуется к плавающей точке;
- действительный операнд преобразуется в комплексный тип;
- необходимое уменьшение разрядности выполняется за счет младших разрядов.

Выполнение арифметических операций над строками нецифровых знаков не разрешается. Например, выражение '123.4'+'5.678' правильное, а выражение 'ABC'+'CDE' не допускается. Выражения, в которых используются переменные ISUB, не должны содержать знаков умножения и деления между этими переменными. Так, можно писать, например, 3SUB+4*5SUB, но неверным будет выражение 3SUB*5SUB.

Логические операции используются главным образом для выполнения операций над строками битов. Вместе с тем их можно применять для любых операндов, которые перед выполнением преобразуются в последовательность двоичных знаков. Если окажется, что операнды имеют различную длину, то операнд более короткий дополняется справа нулями до длины второго. Логические операции выполняются поразрядно. Например, если переменная A имеет значение '010111'B, переменная B — значение '111111'B, а C — значение '010111'B, то результатом выражения $\neg A$ будет '101000'B, $C \& B$ — '101000'B, $A | \neg B$ — '010111'B, $\neg(\neg C | \neg B)$ — '101111'B.

Операции отношения выполняются над значениями арифметических данных и строк символов. Результатом выполнения этих

операций является строка битов единичной длины со значением '1'B, если заданное отношение истинно, и '0'B, если оно ложно. Значения '1'B и '0'B могут быть операндами при выполнении арифметических и логических операций. Например, если переменные A, B, C описаны оператором

DECLARE (A, B, C) DECIMAL FIXED (2, 0) INITIAL (5, 7, 5);
то результат выражения $A > B$ есть '0'B, выражения $A < B$ — '1'B, а $(A < B) * (B < C)$ — '0'B. В случае строк символов производится последовательно слева направо посимвольное сравнение значений операндов, при этом более короткий дополняется справа соответствующим количеством пробелов. Например, если имеет место оператор

DECLARE X CHARACTER (6) INITIAL ('ABCDEF'),
Y CHARACTER (3) INITIAL ('ABC');

то выражение $A = B$ дает значение '0'B.

Операция сцепления (конкатенации) служит для образования последовательности символов из значений левого и правого от символа || операндов, которые могут быть только строками символов или строками битов. Например, если переменная A имеет значение 'PL/1', переменная B — значение 'L', а C — 'PROGRAM', то выражение $A || B || C$ образует в качестве результата строку 'PL/1LPROGRAM', выражение $X || Y$, если X равняется '0'B, а Y — '1'B, имеет значение '01'B.

Если один из операндов имеет описатель VARYING, то и в результате образуется строка переменной длины. Если выполнение операции сцепления приведет к результату длиннее максимально допустимого значения, равного 32767, происходит усечение результата справа.

2. Выражение над массивами. Выражение *над массивами* есть выражение, состоящее из операндов-массивов в допустимой комбинации со скалярами и (или) структурами. Если в выражении над массивами встретилась структура, то операнды-массивы должны быть массивами структур.

Выражение над массивами дает в качестве результата массив значений: все операции над массивами выполняются поэлементно. Поэтому все массивы, входящие в одно выражение над массивами, должны иметь одинаковые границы, а структуры быть одинаково организованными. Например, если описаны массивы A, B и скаляр C оператором

DECLARE (A (2, 2), B (2, 2), C) FIXED (2, 0)
INITIAL (1, 3, 5, 2, 7, 0, 2, 6, 2);

то результатом выражения $C * A$ будет массив значений (2, 6, 10, 4), выражения $A + B$ — массив (8, 3, 7, 8), выражения $A * B$ — значения

(7,0,10,12), а для структур, описанных оператором

DECLARE 1 F(10), 2 (K,D), 1 E, 2 (H, I);

выражение над массивами $F+E$ эквивалентно следующим: $F(1).K+ +E.H$, $F(1).D+E.I$, $F(2).K+E.H$, $F(2).D+E.I$, $F(3).K+E.H$, $F(3).D+E.I$ и т. д.

3. Выражение над структурами. Выражение *над структурами* включает операнды, которые являются обращениями к структурам, подструктурам, а также константами и скалярными переменными. В одном выражении не допускаются имена структур и имена массивов, однако разрешаются выражения, содержащие массивы структур. В последнем случае массивы должны быть одинаковой размерности.

Выражение над структурами является сокращенной записью выражений для каждого элемента одинаково организованных структур, у которых номера уровней и описатели данных могут быть различными. Например, для объявленных в следующем операторе структур A и I

DECLARE 1 A, 2 B, 3 C, 3 D, 2 E, 3 F, 3 G,

1 I, 2 J, 5 K, 5 L, 2 M, 3 N, 3 P;

допустимо выражение $A+I$, которое эквивалентно последовательности выражений $A.B.C+I.J.K$, $A.B.D+I.J.L$, $A.E.F+I.M.N$, $A.E.G+I.M.P$, а выражение $A.B*I.M$ есть выражение над элементами подструктур, эквивалентное выражениям $A.B.C*I.M.N$ и $A.B.D*I.M.P$.

Упражнения

1. Составить выражения, соответствующие математическим формулам:

$$x + \frac{y+z}{x},$$

$$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!},$$

$$\frac{x}{x+3a-y} - (x-3a)/y,$$

$$\frac{x+y}{2a-b} (x-a) \sqrt{b},$$

$$(a^b/b^a)^c/bc/a,$$

$$(2\pi+x) y ((zx/\pi)^x + x),$$

$$A > \frac{A}{B-C} + \frac{B(A+D)}{C} \vee (0,3A)^2 \leq B,$$

$$\frac{1}{X + \frac{1}{X+Y}} < 0,5A \wedge B \vee C.$$

2. Указать, соответствуют ли приведенные в правом столбце выражения стоящим слева математическим выражениям:

$$\frac{a}{b} - \frac{c}{d}$$

$$A/B - C/D,$$

$$\frac{a-c}{b-d}$$

$$A - C/B - D.$$

$$\sqrt{ac} \frac{b-d}{bd}$$

$$(A*C) ** .5* (B-D)/B * D,$$

3. Вычислить значения следующих выражений:

70E5/1000.	'AB' 'BA'
1.1B*2	'1'B & '1001'B
11B—1B	'11'B & ('010'B '0'B)
7E2+1E7B	'AB'='AC'
31*11—1B	111000B < 111B
'111000'B < '111'B	'0'B '1'B

4. Указать порядок выполнения операций в выражениях:

—A.N B.M + C/D.I,	A + B & —C 'D**4,
X(5)**2**3 —Y.Z(7),	'A + B & C A — B
A.N*B.M C + C/D.I,	(K + L)*M — N < P Q = 2.

§ 6. Примечания

Примечания (комментарии) используются для включения в программу пояснительного текста. Общий вид примечания:

/*ТЕКСТ*/

где ТЕКСТ — последовательность любых символов алфавита, а также символов, допустимых для конкретной ЭВМ. Примечание может быть вставлено всюду, где допустим пробел, только не в строковую константу.

Примеры примечаний в операторах:

```
DECLARE /* COEFFICIENTS FOR SYSTEM MARS*/  
      (MP, MQ, MR, MS) DECIMAL FIXED (6,4);  
CX: /* ВЫЧИСЛЕНИЕ СОПРОТИВЛЕНИЯ */  
PROCEDURE (ANT);
```

Последний из приведенных операторов будет воспринят машиной лишь в случае, если устройства ввода и вывода данной ЭВМ имеют русский алфавит.

Упражнения

1. Определить правильность написания комментария в операторе

```
DECLARE /*ИСХОДНЫЕ*/ДАННЫЕ*/ A FIXED (2,0)  
      /*ПОРЯДКОВЫЙ НОМЕР В СПИСКЕ*/,  
      B CHARACTER (4) INITIAL ('PL/1')/*NAME*/,  
      C BIT(10) INITIAL((10)0)/*ЛОГИЧЕСКИЙ*/##*,  
      D CHARACTER (17) INITIAL ('## VERSION/*PL/1  
      */'▬');
```

2. Указать, в какие позиции приведенного ниже оператора может быть вставлено примечание

```
DECLARE X CHARACTER (12) INITIAL ('NAME OF PROC');
```

§ 7. Структура программы

Первичными элементами программы являются операторы. Операторы объединяются в более крупные элементы программы — группы и блоки.

Группа есть совокупность одного или нескольких операторов, первый из которых есть оператор с ключевым словом DO.

Совокупность операторов, образующих область действия каких-либо идентификаторов, составляет *блок*. Блоки бывают двух типов: *обычные* и *процедурные*.⁴ Обычный блок начинается оператором BEGIN, процедурный — оператором PROCEDURE. Так же как и одиночный оператор, группа и блок могут быть снабжены метками. И группа и блок заканчиваются оператором END, после которого допускается указание метки, стоящей перед группой или блоком. Группа и блок служат для целей управления.

Блок может содержать в себе другие блоки. Каждый обычный блок должен содержаться в каком-нибудь другом блоке. Процедурный блок (процедура), который не содержится ни в каком другом блоке, называется внешней процедурой. Процедура, содержащаяся в каком-нибудь другом блоке, является внутренней процедурой.

Исходная программа представляет собой в общем случае совокупность внешних непересекающихся блоков PROCEDURE. Среди них имеется главная (начальная) процедура, выполнение которой вызывается операционной системой. Если программа состоит из одного процедурного блока, то он должен быть оформлен как главная процедура. Главная процедура начинается оператором

MP: PROCEDURE OPTIONS (MAIN);

где MP — метка (имя входа), представляющая собой идентификатор, в написании которого могут содержаться ограничения, обусловленные операционной системой. Ключевые слова OPTIONS (MAIN) означают «выбор главной процедуры».

Главная процедура организует вызов остальных процедурных блоков исходной программы, которые по отношению к главной процедуре являются вызываемыми. Вызываемые процедурные блоки могут сами быть вызывающими.

§ 8. Оператор присваивания

Оператор *присваивания* записывается в виде

$V = E;$

где V — переменная, E — выражение, = — знак присваивания. Этот оператор осуществляет замену текущего значения переменной V значением выражения E, которое вычисляется и преобразуется в

соответствии с описателями V. Например:

```
X = A * B + C * D;   K:M:Z = Z - 1;  
BA = '0110'B;       N = L = M1;
```

Если нескольким переменным присваивается одно и то же значение, то для всех переменных может быть записан один оператор присваивания, а переменные перечислены слева от знака присваивания через запятую. Например, вместо операторов $X=0$; $Y=0$; $Z=0$; можно написать $X,Y,Z=0$;

Слева от знака присваивания может стоять как идентификатор массива, так и сечение массива. Все массивы слева от знака присваивания и все массивы в выражении над массивами должны иметь одинаковую структуру и одинаковые границы. Если справа от знака присваивания стоит скалярное выражение, то его значение присваивается всем элементам массива. Если выражение справа содержит операнды-структуры, то все массивы, входящие в оператор, должны быть массивами структур.

Структура, указанная слева от знака присваивания, должна быть идентичной структурам, указанным в выражении справа. Например, структуры A и H, описанные оператором

```
DECLARE 1 A, 2 (B,C) CHARACTER (16),  
        3 D (2) CHARACTER (4),  
        1 H, 2 (K,B) CHARACTER (16),  
        2 E, 8 U(2) CHARACTER (4) INITIAL (0);
```

могут быть использованы в операторе присваивания $A=H$; который обрабатывается как последовательность операторов, присваивающих значения соответствующих элементов структуры H элементам структуры A. Следовательно, этот оператор эквивалентен следующим трем операторам: $B=K$; $C=B$; $F=E$; последний из которых разбивается в свою очередь на такие: $D(1)=U(1)$; $D(2)=U(2)$;

Оператор присваивания, содержащий структуры, может быть снабжен дополнением BY NAME, которое пишется после выражения через запятую. В этом случае структуры слева и справа от знака присваивания не обязательно должны иметь одинаковое строение, но по меньшей мере по одному элементу в каждой структуре должны иметь одинаковое имя, и только для этих элементов выполняется оператор присваивания. Так, в предыдущем примере оператор присваивания $A=H$, BY NAME; порождает операторы присваивания для подструктур с совпадающими именами, а не для элементов в соответствующих позициях в структурах A и H, т. е. он эквивалентен оператору $A.B=H.B$;

Если слева от знака присваивания стоит переменная типа метки, то справа может стоять либо константа типа метки, либо переменная типа метки. Например, имея описание

```
DECLARE (N(3),M(3)) LABEL INITIAL (A,B,C);
```


В программе можно использовать оператор

$M=N$;

Если в левой части оператора присваивания стоит переменная типа указателя, то справа от знака присваивания может стоять произвольное выражение над указателями, образуемое по обычным правилам. Переменные типа указателя могут быть компонентами структур и массивов структур.

Упражнения

1. Найти ошибки, если они есть, в следующих операторах присваивания:

$A=B.C=D$; $L=M=N=3.14$;

$X,Y,Z=A/D**2||'XY'$; $B(1)=B$, BY NAME

$Q=((2*A+F)**2-(A+F)/(A-F)-E)*E+A$; $P(5)=B.C$;

2. Определить значение переменной V, которое она получит при выполнении программы

P: PROCEDURE OPTIONS (MAIN);

DECLARE R DECIMAL FIXED;

$R=1.2$; $V=.6$; $R=(R+V)/2+V$;

$V=R*V**2$; END P;

3. Написать оператор присваивания для вычисления значений величин по формулам:

$$z = -3.14 + \left(\frac{a+b}{1-ab} \right),$$

$$f = \sqrt{x} x (x+y^2) / \sqrt[3]{y},$$

$$x = y - 2 \left(\frac{a}{2+a} + \frac{a^3}{3(2+a)^3} + \frac{a^5}{5(2+a)^5} \right),$$

$$p = (((a_5 x + a_4) x + a_3) x + a_2) x + a_1) x + a_0.$$

§ 9. Операторы управления

Операторы управления обеспечивают необходимую последовательность выполнения операторов, возможно, отличную от последовательности записи операторов в программе.

1. Безусловная передача управления. Оператор

GO TO M;

где M — константа типа метки или переменная типа метки, осуществляет безусловную передачу управления на оператор с меткой, являющейся значением M. Начиная с этого оператора, выполняется новая последовательность операторов. Например,

GO TO M15; M: GO TO X;

Управление разрешается передавать только на операторы внутри данного процедурного блока. Если в операторе GO TO указана

переменная типа метки, то такой оператор обычно работает как многопозиционный переключатель, каждый раз передавая управление к новому оператору, имеющему в качестве метки текущее значение заданной переменной типа метки.

2. Условный оператор. Наиболее общая форма *условного* оператора

IF E THEN S1; ELSE S2;

где E — скалярное выражение, S1, S2 — операторы, группы, или блоки BEGIN. В условном операторе может отсутствовать конструкция, начинающаяся с ключевого слова ELSE. S1 или S2 могут быть, в частности, пустыми операторами.

Скалярное выражение задает правило вычисления логического условия, для получения которого при необходимости значение E преобразуется в строку битов. Если значение выражения равно '1'В или хотя бы один бит в полученной строке имеет значение 1, то выполняется оператор, записанный после THEN, т. е. оператор S1. После выполнения S1 (если это не повлияет на порядок выполнения операторов) управление передается оператору, следующему за условным оператором. Если значение скалярного выражения E равно '0'В или все биты полученной строки суть нули, то выполняется оператор S2 (в случае отсутствия конструкции ELSE S2; выполняется оператор, стоящий непосредственно за условным оператором). После выполнения S2 управление передается следующему за ним оператору, если S2 не влияет на порядок выполнения. Например:

IF X=Y THEN GO TO M; ELSE Z=0;

Перед операторами S1 и S2 могут стоять метки, на которые можно передавать управление. Независимо от способа передачи управления на операторы S1 или S2 порядок выполнения условного оператора сохраняется.

Условные операторы могут быть вложены один в другой, т. е. либо S1, либо S2, либо оба могут начинаться с IF, например:

```
A: IF X > Y THEN IF Z=W THEN
    IF W < P THEN Y=1; ELSE P=Q;
    ELSE; ELSE X=4;
J: Z=5;
```

Приведем еще два условных оператора с поясняющими блок-схемами, изображенными соответственно на рис. 4 и 5,

Пр и м е р 1.

```
IF A=B THEN IF C < D THEN
    IF E > F THEN X=Y;
    ELSE X=Z; ELSE X=W;
GO TO M;
```

Пример 2.

```
IF A=B THEN IF C < D THEN
  IF E > F THEN X=Y; ELSE;
  ELSE X=Z; ELSE X=W;
GO TO M;
```

В случае вложенных условных операторов действует правило: дополнение ELSE относится к ближайшему слева дополнению

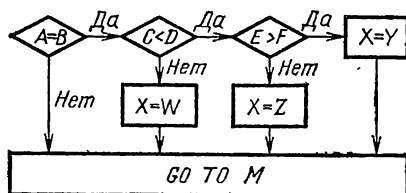


Рис. 4

THEN, не имеющему ELSE. Оно позволяет избежать недоразумений в конструкциях условных операторов, особенно если опущена часть ключевых слов ELSE.

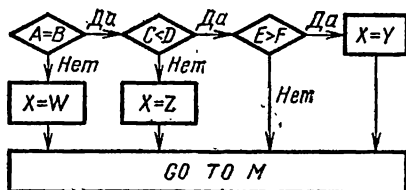


Рис. 5

3. Группа. Группа представляет собой конструкцию вида
M: DO; SS; END M;

где M — метка (наличие ее после END, как и перед ключевым словом DO, не обязательно); SS — один или несколько операторов. Группа используется для объединения нескольких операторов в любом месте программы, где может быть написан одиночный оператор (кроме оператора ON). Одна группа может быть вложена в другую. Например:

- 1) IF A>B THEN DO; C=A-B; GO TO L1; END;
ELSE IF A<B THEN DO; C=B-A;
GO TO L2; END; C=A;
- 2) B: DO; C: DO; I=K+L; A: K=L+I; L=I+K;
END C; END B;

Второй пример представляет собой случай вложенных групп. Здесь последняя строка может быть записана в виде END; END;

или же END B; т. е. в тех случаях, когда появляются несколько следующих друг за другом операторов END, допускается написание только одного из них при условии, что он включает метку внешней группы.

Извне группы разрешена передача управления на любой внутренний оператор группы. Так, в случае примера 2) допустим оператор GO TO A;

Пример 3. Рассмотренных операторов, за исключением операторов ввода и вывода, которые будем пока условно обозначать в виде ⟨ввод⟩, ⟨вывод⟩, достаточно, чтобы составить программу вычисления корней квадратного уравнения $ax^2+bx+c=0$ в соответствии с блок-схемой, приведенной на рис. 1 § 3 гл. I:

```
SQR: PROCEDURE OPTIONS (MAIN);  
  <ввод (A,B,C)>;  
  IF A=0 THEN DO; IF B=0 THEN  
    DO; K=0; <вывод (A,B,C,K)>; GO TO L; END;  
    K=1; X1R=-C/B; X2R=0; GO TO M; END;  
    K=2; D=B**2-4*A*C;  
    IF D < 0 THEN DO;  
      X1R, X2R=-B/(2*A); X1I=SQRT (-D)/(2*A);  
      X2I=-X1I; GO TO N; END;  
      S=SQRT (D); X1R=(-B+S)/(2*A);  
      X2R=(-B-S)/(2*A);  
M: X1I, X2I=0;  
N:<вывод (A,B,C,K,X1R,X2R,X1I,X2I)>;  
L: STOP; END SQR;
```

Здесь SQRT(D) — встроенная функция, вычисляющая квадратный корень аргумента D, STOP — оператор останова.

4. Оператор цикла. Группа может рассматриваться как частный случай оператора *цикла*, который определяется как последовательность, состоящая из ключевого слова DO, конструкций, обеспечивающих управление циклом (спецификаций цикла), операторов, выполняемых в цикле (тела цикла), и оператора END.

В наиболее общем виде оператор цикла может быть записан так:

```
DO CC; SS; END;
```

где CC — спецификации цикла; SS — операторы или группы. Если CC пусто, цикл превращается в группу. Существует несколько разновидностей спецификаций цикла, каждая из которых вместе с ключевым словом DO составляет заголовок цикла.

На месте CC в заголовке цикла может стоять конструкция
WHILE (E)

где E — скалярное выражение. В этом случае выполнение операторов, образующих тело цикла, производится до тех пор, пока зна-

чение скалярного выражения, преобразованного в строку битов, равно '1'В или содержит хотя бы один бит, равный 1. Как только значение Е становится равным '0'В или строка битов содержит только нули, операторы тела цикла пропускаются и управление передается на оператор, следующий после цикла (т. е. после оператора END, завершающего тело цикла). Если значение выражения с самого начала оказывается равным '0'В, операторы в теле цикла не выполняются совсем.

Пример 4. Значение кубического корня из числа N методом Ньютона по формуле $X = (2X_0 + N/X_0^2)/3$, где X_0 , X — последовательные итерации, можно вычислять в цикле

```
DO WHILE (ABS(X-X0)>D); X0=X;  
X=(2*X0+N/X0**2)/3; END;
```

если предварительно задать начальные значения X_0 , X и требуемую точность, например, операторами $X_0=0$; $X=1$; $D=.0001$; Здесь ABS(X) — встроенная функция, вычисляющая абсолютное значение аргумента X.

Другая разновидность спецификации цикла включает параметр цикла (управляющую переменную) с указанием выражений, определяющих диапазон и шаг изменения значений параметра. Эта спецификация имеет вид

```
P=E1 TO E2 BY E3 WHILE (E)
```

где P — скалярная переменная любого типа — параметр цикла; E1, E2, E3 — скалярные выражения, определяющие соответственно начальное и конечное значения и шаг изменения параметра цикла, E — скалярное выражение, рассмотренное выше. Операторы тела цикла выполняются при значениях P, удовлетворяющих условию $E1 \leq P \leq E2$, если выполнено условие E. Проверка на окончание цикла делается в начале каждого прохождения цикла.

Конструкция TO E2 BY E3 может быть написана в другой последовательности BY E3 TO E2 или же совсем отсутствовать. В последнем случае цикл выполняется только один раз при значении P, равном E1, даже если имеется конструкция, начинающаяся с WHILE. Если конструкция BY E3 отсутствует, но есть TO E2, то по умолчанию E3 равно 1.

Пример 5. Оператор цикла, в котором вычисляется сумма квадратов нечетных чисел от 1 до 9, имеет вид

```
DO J=1 TO 10 BY 2; X=X+J**2; END;
```

Здесь предполагается, что перед циклом выполнен оператор $X=0$;

Пример 6. Заголовок цикла

```
DO K=1 TO 25 WHILE (A>K)
```

обеспечивает выполнение цикла 25 раз при условии, что A остается больше K. Если, например, A равно 17, то выполнение цикла за-

вершится при значении параметра K, равном 16. Шаг изменения параметра в этом примере равен 1.

В заголовке цикла одновременно может содержаться любое число различных спецификаций, отделяемых друг от друга запятыми. Например, заголовок цикла

```
DO I=1 TO 8,13,17 TO 29 BY 2, WHILE (D>.001);
```

обеспечивает выполнение цикла для $I=1,2,3,4,5,6,7,8,13,17,19,21,23,25,27,29$ и еще столько раз, сколько необходимо, чтобы переменная D получила значение, меньшее или равное 0.001.

Значение параметра цикла после выхода из цикла определено, и допускается изменение значения параметра в теле цикла. Операторы в теле цикла могут быть в свою очередь операторами цикла. В этом случае реализуются вложенные циклы. Например, фрагмент программы вычисления суммы элементов матрицы A с тремя строками и пятью столбцами содержит вложенные циклы:

```
DECLARE A(3,5) FIXED (5,0) DECIMAL INITIAL  
      (1,2,(3)3, (2)5, (8)7), (I, J, S) FIXED  
      (10,0) DECIMAL; S=0; DO I=1 TO 3;  
      DO J=1 TO 5; S=S+A (I, J); END; END;
```

Отметим, что, в отличие от группы, вход в цикл разрешен только через заголовок цикла, т. е. через оператор DO.

5. Оператор останова. Оператор *останова* имеет вид
STOP;

Результатом выполнения этого оператора является немедленное прекращение выполнения программы.

Упражнения

1. Найти минимальное число операторов безусловной передачи управления, обеспечивающих выполнение N операторов присваивания в обратном порядке, не меняя последовательности их расположения.

2. Написать условие для проверки принадлежности значения x заданному интервалу $[a, b]$ и присвоить значение x, которое заключено в $[a, b]$, переменной y.

3. Вычислить сумму квадратов сумм отрицательных и положительных элементов последовательности A_1, A_2, \dots, A_k , исключив до вычисления соответствующих сумм элементы, равные нулю.

4. Составить программу с использованием переменных типа метки, выполняющую вычисления:

$$y = a + bx + cx^2 \quad \text{при} \quad 0,5 \leq x < 1,5,$$

$$y = (abx)^c \quad \text{при} \quad 1,5 \leq x < 2,5,$$

$$y = \sqrt{a + bx^3} - c \quad \text{при} \quad 2,5 \leq x < 3,5.$$

5. Написать последовательность операторов, в которой комплексным переменным C1 и C2 присваиваются значения корней квадратного уравнения $x^2 + px + q = 0$, а переменной T присваивается

значение 0, если корни комплексные, значение 1, если корни действительные и совпадают, значение 2, если корни действительные и различные.

6. Одномерный массив содержит 100 элементов. Просуммировать элементы: а) с четными номерами; б) с нечетными номерами; в) с номерами, кратными 5; г) с отрицательными значениями и номерами, кратными 10.

§ 10. Блоки

Программа, как уже отмечалось выше, в общем случае состоит из нескольких сегментов, каждый из которых является процедурным блоком (блоком PROCEDURE или процедурой). Управление последовательностью выполнения процедурных блоков осуществляется главной процедурой. Обычный блок (блок BEGIN) и процедурный блок объединяют части алгоритма в одно целое. Эти блоки определяют область действия идентификаторов, описанных в них, и управляют резервированием основной памяти.

1. Обычный блок. Общая форма обычного блока следующая:

M: BEGIN; SS; END M;

где M — метка, SS — операторы или блоки. Метка после оператора END должна совпадать с меткой, непосредственно предшествующей ключевому слову BEGIN (если меток несколько), метка после END может и отсутствовать.

Обычный блок может использоваться в любом контексте программы, где допускается написание оператора. В состав обычного блока можно включать любые операторы языка, в том числе и другие блоки. Блок начинает выполняться после передачи управления оператору BEGIN — первому оператору блока. Управление может быть передано либо оператором перехода, либо после выполнения предшествующего оператора, за которым записан блок. Следовательно, в записи обычного блока метка не является обязательным элементом.

Областью определения явно описанных идентификаторов (т. е. идентификаторов, перечисленных в операторе DECLARE или использованных в качестве метки оператора или в качестве имени входа в процедурный блок, а также идентификаторов, включенных в список формальных параметров процедурного блока) является блок (процедурный или обычный), в котором идентификаторы описаны, за исключением тех внутренних блоков, содержащихся в данном, где эти же идентификаторы описаны вновь. Например, в следующем фрагменте программы:

```
P: PROCEDURE;  
  DECLARE X CHARACTER (15), Y FLOAT (8),  
  Z FIXED (5,2); ...; IF Y>0 THEN GO TO B; ...;  
B: BEGIN;
```

```

DECLARE U CHARACTER (6), V BINARY
FIXED (23,0), W BIT (5), X FIXED (6,0); ...;
END B; ...; END P;

```

идентификаторы U, V, W, X определены в блоке B; X, Y, Z — вне блока. В процедуре P можно пользоваться переменными X (строковая), Y, Z; переменными U, V, W — нельзя. Внутри блока B можно ссылаться на переменные U, V, W, X (с фиксированной точкой). Кроме того, внутри блока можно пользоваться переменными Y, Z. Строковая переменная X, описанная вне блока, в нем не определена, и на нее ссылаться в блоке B нельзя.

Переменная, описанная в блоке, может стать доступной в каждом блоке, в котором содержится данный блок, если переменную объявить *внешним* именем. Для этого в операторе DECLARE, в котором объявляется данная переменная, следует указать описатель EXTERNAL, называемый описателем области действия. В идентификаторах внешних имен содержится не более семи символов. Например, если в предыдущем примере изменить начало оператора объявления данных в блоке B на

```

DECLARE U CHARACTER (6) EXTERNAL, V BINARY...;

```

то переменная U становится внешним именем. Описатель EXTERNAL присваивается по умолчанию идентификаторам, которые обозначают точки входа во внешние процедурные блоки.

2. Процедурный блок. Процедурный блок имеет следующую форму записи:

```

M: PROCEDURE (FP); SS; END M;

```

где M — метка, SS — операторы или блоки, FP — список формальных параметров. Метка является именем входа в процедурный блок и используется для вызова блока, после оператора END метка может отсутствовать. Формальными параметрами в блоке являются идентификаторы, которым во время вызова процедуры ставятся в соответствие фактические параметры — объекты программы, над которыми производятся действия в процедурном блоке. Формальные параметры могут отсутствовать.

Для процедурного блока область определения идентификаторов устанавливается точно так же, как для обычного блока. Блок PROCEDURE выполняется только после того, как был произведен его вызов. Блок может быть вызван либо как функция, либо как подпрограмма. В первом случае вызов осуществляется посредством оператора в выражении, во втором — оператором вызова процедуры.

Операнд в выражении, вызывающем процедурный блок как функцию, имеет вид M(FP), где M — имя входа в процедурный блок — метка, стоящая перед оператором PROCEDURE (имя основного входа), или метка, стоящая перед одним из операторов

ENTRY (имя дополнительного входа, см. п. 5 этого параграфа), возможно, содержащихся в вызываемом процедурном блоке; FF — список фактических параметров, количество которых совпадает с числом формальных параметров. Формальные параметры могут быть включены в операторы DECLARE процедурного блока, фактические параметры при необходимости описываются в вызывающем блоке.

Вызов процедурного блока означает, что производится передача управления к процедуре с данной меткой (именем входа) М с одновременной заменой формальных параметров фактическими. После выполнения процедуры управление передается на продолжение выполнения операторов вызывающего блока.

Оператор *вызова* процедуры имеет вид

CALL M(FF);

где CALL — ключевое слово, а М и FF имеют тот же смысл, что и в случае процедуры-функции. Выполнение процедурного блока заканчивается либо оператором END, либо оператором RETURN.

3. Процедура-функция. Процедурный блок принято называть *процедурой-функцией* или функцией, если он вызывается как функция, а при обращении к процедурному блоку исполняется хотя бы один оператор *возврата* вида

RETURN (FE);

где FE — выражение, являющееся результатом выполнения процедуры-функции. Идентификатором функции является имя входа в данный процедурный блок.

Для функции в операторе PROCEDURE могут быть указаны описатели, характеризующие вычисляемое в функции значение FE. Одновременно аналогичные описатели для имени входа должны быть указаны в операторе объявления данных с описателями имени входа ENTRY и RETURNS в вызывающем блоке

DECLARE M ENTRY RETURNS (LF);

где М — имя входа, LF — описатели результата.

Если описатели не заданы (как в операторе PROCEDURE, так и после описателя RETURNS в приведенном операторе), то они устанавливаются по умолчанию в соответствии с первой буквой идентификатора, обозначающего точку входа в процедуру-функцию. В этом случае данный оператор DECLARE выполняет лишь одну функцию — объявляет в вызывающем блоке М как имя процедуры. Такой оператор

DECLARE M ENTRY;

обязателен, если процедура-функция М не имеет параметров,

Пр и м е р. Следующий фрагмент программы представляет собой процедуру-функцию для вычисления $N! = 1 \cdot 2 \cdot \dots \cdot N$:

```
NF: PROCEDURE (N) DECIMAL FIXED;  
    DECLARE N FIXED (10);  
    IF N <= 1 THEN RETURN (1);  
    F = 1; DO I = 2 TO N; F = F * I; END;  
    RETURN (F); END NF;
```

Здесь формальный параметр N описан явно оператором объявления данных. Описатели DECIMAL FIXED в операторе PROCEDURE характеризуют вычисляемое значение, которое задается в двух операторах возврата RETURN. Эти же описатели подразумеваются у идентификатора NF, обозначающего точку входа (метку процедуры-функции).

Отметим, что первая строка в приведенном примере допускает другую форму записи:

```
NF: PROCEDURE (N) RETURNS (DECIMAL FIXED);
```

где описатели, указанные в скобках, характеризуют вычисляемое значение. В вызывающем блоке следует написать оператор

```
DECLARE NF ENTRY RETURNS (DECIMAL FIXED);
```

а само обращение может выглядеть, например, так:

```
MN = NF(10) + 10;
```

где константа 10, указанная в скобках, — фактический параметр. В этом операторе вычисляется значение переменной MN как $10! + 10$.

Если в списке фактических параметров указано имя входа в процедуру со списком своих фактических параметров или же указано в скобках имя функции без параметров, то считается, что такое имя входа представляет функцию и в список фактических параметров соответствующей вызываемой процедуры передается численное значение. Например, операторы

```
CALL P(F(A,B),N);  
CALL P((E),N);
```

где F и E-функции, описание которых имеет вид

```
F: PROCEDURE (A,B); ...;  
E: PROCEDURE; ... ;
```

передадут в вызываемую процедуру значения функций F и E соответственно,

Если в списке фактических параметров имеется имя входа, которое не заключено в скобки, то в вызываемую процедуру передается это имя входа. Например, в фрагменте программы

```
DECLARE E ENTRY; ...; CALL P(E,N);
```

в процедуру P будет передано имя входа E.

4. Процедура-подпрограмма. При обращении к функции в вызывающий блок передается (возвращается), как правило, одно численное значение. Если же процедурный блок используется для вычисления набора значений, его следует оформить как *процедуру-подпрограмму*.

В операторе PROCEDURE для подпрограммы, как и для функции, указаны формальные параметры, но в нем нет описателей, поскольку в операторе возврата RETURN не указывается выражение, значение которого возвращается в вызывающую программу. Для возврата в вызывающую программу можно также использовать оператор конца END, замыкающий данную процедуру. Действие этого оператора полностью эквивалентно выполнению оператора возврата RETURN.

Значения, которые вычисляются в подпрограмме и возвращаются в вызывающий блок, соответствуют формальным параметрам. Части формальных параметров, как и для функции, соответствуют фактические входные данные.

При обращении к подпрограмме оператором CALL указываются на месте входных данных идентификаторы, определенные в вызывающем блоке. Идентификатор точки входа в операторе CALL считается описанным в вызывающей программе контекстуально, т. е. появление идентификатора в операторе CALL есть его описание.

Описатели формальных параметров и описатели фактических параметров процедуры могут быть согласованы описанием фактических параметров в вызывающей процедуре оператором объявления данных с теми же описателями, что и у формальных параметров вызываемой подпрограммы. В то же время такое согласование описателей может быть выполнено указанием в вызывающей процедуре оператора

DECLARE M ENTRY (LFP);

где M — имя вызываемой процедуры, LFP — список требуемых описателей соответствующих формальных параметров, ENTRY — описатель имени входа. Описатели каждого формального параметра отделяются в списке LFP от описателей другого формального параметра запятой, при этом в случае уже имеющей место согласованности описателей в соответствующей позиции списка LFP описатели опускаются, однако все разделяющие запятые должны быть сохранены.

Пр и м е р. Процедура-подпрограмма, в которой вычисляются значения $F=N!$ и $L=(N!)^2+N!+N$, имеет вид

```
NFL: PROCEDURE (N, F, L);  
  DECLARE (N, F, L, I) DECIMAL FIXED;  
  IF N=0 THEN DO; F=1; L=2; RETURN; END;  
  IF N=1 THEN DO; F=1; L=3; RETURN; END;
```

```
F=1; DO I=2 TO N; F=F*I, END;
L=F**2+F+N; RETURN; END NFL;
```

Из вызывающего блока, начинающегося оператором

```
P: PROCEDURE;
```

в котором содержится, например, описание

```
DECLARE (FF,LF) DECIMAL FIXED;
```

или же оператор

```
DECLARE NFL ENTRY (,DECIMAL FIXED,DECIMAL
FIXED);
```

можно обратиться к данной процедуре оператором

```
CALL NFL (10,FF,LF);
```

в результате которого идентификатор FF получит значение 10!, а LF — значение $(10!)^2 + 10! + 10$.

В этом примере в списке описателей после слова ENTRY первый элемент может быть опущен, так как первый фактический параметр задается десятичным числом в форме с фиксированной точкой, т. е. имеет место согласованность описателей формального и фактического параметров.

Если процедуру, к которой производится обращение, разместить непосредственно в вызывающем блоке, то к ней можно обратиться, не задавая фактических параметров. Так, в приведенном примере, если процедура вычисления F и L содержится в том блоке, из которого она вызывается, можно оператор

```
DECLARE (N,F,L,I) DECIMAL FIXED;
```

перенести в вызывающий блок, одновременно разместив там операторы

```
N=10; FF=F; LF=L;
```

т. е. процедурный блок P примет вид

```
P:PROCEDURE; ...;
  DECLARE (FF, LF, N, F, L, I) DECIMAL FIXED; ...;
NFL: PROCEDURE;
  IF N=0 THEN DO; F=1; L=2; RETURN; END;
  IF N=1 THEN DO; F=1; L=3; RETURN; END;
  F=1, DO I=2 TO N; F=F*I; END;
  L=F**2+F+N; RETURN; END NFL;
N=10; CALL NFL;
FF=F; LF=L; ...; END P;
```

Здесь процедура вычисления значений $F=N!$ и $L=(N!)^2 + N! + N$ не содержит параметров и обозначена меткой NFL.

Любая подпрограмма может вызывать другие подпрограммы, в том числе и саму себя, т. е. может быть рекурсивной. Последнее означает, что при выполнении процедурного блока может встретиться обращение к имени входа этого же процедурного блока. В этом случае у оператора PROCEDURE указывается описатель RECURSIVE, который относится ко всем входам этой процедуры. Например, рекурсивный вариант процедуры вычисления $N!$ может быть составлен так:

```
NFR: PROCEDURE (N) DECIMAL FIXED RECURSIVE;
      DECLARE N FIXED(10);
      DECLARE NFR RETURNS (DECIMAL FIXED);
      IF  $N \leq 1$  THEN RETURN (1);
      F = NFR (N-1)*N;
      RETURN (F); END NFR;
```

5. Оператор входа в процедуру. Оператор *входа* в процедуру указывает дополнительный вход в процедурный блок и имеет вид

```
M: ENTRY (FP);
```

где M и FP имеют тот же смысл, что и в операторе PROCEDURE.

Оператор входа в процедуру задает точку входа, в которую передается управление при вызове процедуры. Поэтому оператор ENTRY, как и оператор PROCEDURE, должен иметь по крайней мере одну метку. Эти метки называются именами точек входа. При любом обращении к процедуре задается какое-либо имя точки входа и тем самым указывается, с какого места следует выполнять данную процедуру. Когда процедура вызывается по имени, заданному в операторе ENTRY, обработка начинается с первого исполняемого оператора, следующего после оператора ENTRY. Оператор входа при обычном последовательном выполнении операторов процедуры во внимание не принимается.

Оператор входа в процедуру не может быть внутренним по отношению ни к какому блоку, содержащемуся в этой процедуре; он не может также содержаться внутри группы, для которой в операторе задан цикл. При вызове процедуры список фактических параметров должен быть согласован со списком формальных параметров для соответствующей точки входа. Списки формальных параметров, заданные в различных точках входа процедуры, могут не совпадать друг с другом.

Пример. Рассмотрим процедурный блок

```
P1: PROCEDURE (A,B,C); ...; B1: BEGIN; ...; END B1;
E1: ENTRY (D,E); ...; P2: PROCEDURE; ...; END P2;
E2: ENTRY; ...; END P1;
```

Обращение к этой процедуре с помощью оператора вызова процедуры возможно в трех вариантах:

```
CALL P1(X,Y,Z); CALL E1(T,P); CALL E2;
```

где X, Y, Z, T, P — фактические параметры.

Оператор вызова процедуры может быть использован в описателе INITIAL при присваивании начальных значений идентификаторам. Например, в операторе

```
DECLARE D(16) INITIAL CALL DATA (A,B,C);
```

присваивание начальных значений элементам массива D производится путем обращения к подпрограмме DATA. Здесь A, B, C — фактические параметры. Массив D определяется в DATA заранее либо он передается как параметр. После обработки вызванной процедуры управление передается на описатель INITIAL.

Операторы BEGIN, PROCEDURE, ENTRY, END являются операторами структуры программы. К ним можно отнести также оператор DO, одновременно выполняющий функции управления.

Упражнения

1. Указать, каковы будут значения элементов массива R после выполнения следующей программы. HBOUND (A, 1) — встроенная функция, результатом которой является целое число, равное текущему максимальному значению индекса одномерного массива A.

```
A: PROCEDURE OPTIONS (MAIN);
```

```
  DECLARE R(4) FIXED(2);
```

```
  DO J=1 TO 4; R(J)=B(J)+C(J); END;
```

```
  CALL D (R,07);
```

```
B: PROCEDURE (K); K=K*2;
```

```
C: ENTRY (K); RETURN (K-2); END B;
```

```
D: PROCEDURE (A,L); DECLARE (L,A(*)) FIXED (2);
```

```
  DO J=1 TO HBOUND (A,1); IF A(J)>L THEN
```

```
    RETURN; A(J)=A(J)+1; END; END D;
```

```
END A;
```

2. Составить описание процедуры-функции для упорядочивания последовательности N чисел в порядке их возрастания.

3. Оформить в виде процедуры-функции алгоритм умножения полинома степени n на полином степени m.

4. Составить процедуру-функцию для вычисления значения $(k+l)/(kl+l)$ (k, l — натуральные числа).

5. Составить подпрограмму, которая вычисляет координаты точки пересечения прямых, заданных уравнениями

$$a_1x + b_1y = c_1;$$

$$a_2x + b_2y = c_2.$$

Если прямые параллельны или совпадают, то подпрограмма должна передать управление оператору с меткой NO.

6. Составить процедуру-функцию, которая вычисляет сумму квадратов n ее аргументов, где $1 \leq n \leq 5$. Для каждого значения n предусмотреть отдельную точку входа в процедуру.

§ 11. Встроенные функции

Наиболее часто встречающиеся математические процедуры представлены в виде составной части языка и получили название *встроенных функций*. К числу таких функций относятся математические функции, арифметические функции, функции для обработки строк, функции для обработки массивов, функции специального назначения.

Записывается встроенная функция с помощью идентификатора и стоящего за ним аргумента (аргументов) в скобках. Идентификаторы встроенных функций заранее предопределены, как и форма представления и свойства аргументов и получаемых значений. Используются встроенные функции (в зависимости от назначения функции) в выражениях в качестве операндов как обычные переменные.

К математическим функциям относятся тригонометрические и гиперболические функции, функции извлечения квадратного корня, возведения в степень и логарифмические функции. Аргументы математических функций могут быть скалярными выражениями и массивами и должны быть представлены в форме с плавающей точкой. В противном случае перед вызовом функции они преобразуются в эту форму. Если аргументом является массив, то результатом обращения к функции будет также массив с размерностью и границами, как у аргумента; функция выполняется над каждым элементом массива. Значение, вычисляемое математической функцией, есть число с плавающей точкой, с основанием и разрядностью аргумента.

Арифметические функции имеют аргументы с описателями `FLOAT` или `FIXED` и `DECIMAL` или `BINARY`. В случае отсутствия этих описателей преобразование аргумента производится автоматически. Аргумент может быть скалярным выражением или массивом. Если аргументом является массив, результат выполнения функции есть также массив с описателями аргумента. Если не указаны описатели результата, они будут такими же, как и у аргумента.

Аргументы функций для обработки строк представляют собой строки символов, или массивы, состоящие из строк символов. Те аргументы, которые не являются строками символов, перед вызовом функции преобразуются в строку битов, или в символьную строку.

Функции для обработки массивов в качестве аргументов содержат массивы. Результат выполнения функции является скалярным значением, поэтому обращение к любой функции для обработки массивов рассматривается как скалярное выражение.

К функциям специального назначения относятся функции, которые не объединены общим назначением и не связаны с другими классами встроенных функций. В их число входят функции-ситуации, не содержащие аргументов и принимающие значения, харак-

теризующие прерывания; эти функции могут употребляться только в операторах, входящих в оператор ON, или блоках, вызываемых оператором ON.

Некоторые из встроенных функций могут быть использованы слева от знака присваивания в операторе присваивания, в качестве параметра цикла в операторе цикла или же как элемент списка данных в операторе ввода, вследствие чего их называют псевдопеременными. Например, псевдопеременной является встроенная функция $\text{COMPLEX}(X,Y)$, где X , Y — идентификаторы, возможно, с различными описателями. Так, в результате выполнения оператора $\text{COMPLEX}(X,Y)=E$; вещественная часть выражения E присваивается переменной X , мнимая — Y . Если E есть $5.1+7.5I$, X будет равно 5.1 , Y — 7.5 . Теперь выполнение оператора присваивания $C=\text{COMPLEX}(X,Y)$; где C — комплексная переменная, приведет к тому, что C получит значение $5.1+7.5I$; здесь конструкция $\text{COMPLEX}(X,Y)$ выступает как арифметическая функция, формирующая комплексное число.

Список встроенных функций зависит от версии языка, транслятора и может пополняться новыми функциями. Встроенные функции перечисляются в приложении III; их список с различной степенью полноты содержится, например, в [11, 12].

§ 12. Обработка прерываний

1. Оператор задания режима обработки прерываний. В различных исключительных случаях выполнение программы может прерываться. Это происходит, если оператор, записанный в программе, невыполним. Для управления выполнением программы в таких случаях, иначе — при возникновении *ситуации прерывания*, особенно в процессе отладки программы, используется оператор *задания режима обработки прерываний*, записываемый в двух разновидностях:

```
ON C SNAP S;  
ON C SYSTEM;
```

где ON — ключевое слово оператора, C — ключевое слово, обозначающее ситуацию, при возникновении которой управление передается оператору S, указанному после дополнения SNAP и определяющему режим обработки ситуации прерывания. На месте оператора S может стоять непомеченный обычный блок (но не процедурный блок) или одиночный оператор (но не группа), за исключением операторов DECLARE, END, DO, FORMAT, RETURN.

В записи оператора ON дополнение SNAP может быть опущено, однако если оно указано, печатается определенная информация о состоянии программы, позволяющая оценить ситуацию прерывания и полезная при отладке. Выдача этой информации происходит до обработки ситуации прерывания, т. е. до выполнения оператора S.

Оператор ON с дополнением SYSTEM указывает, что имеет место стандартная реакция операционной системы на данную ситуацию, или стандартный режим обработки прерывания. В такой форме оператор ON используется обычно тогда, когда есть необходимость отменить действие предыдущего оператора задания режима обработки прерываний и по умолчанию передать управление операционной системе.

Реакция на данную ситуацию прерывания, заданная оператором ON, сохраняет силу в пределах того блока, в который входит этот оператор ON. Режим обработки ситуации прерывания распространяется также на все внутренние блоки и все вызванные из данного блока процедурные блоки.

Выполнение оператора ON заключается в том, что для соответствующей ситуации прерывания устанавливается указанный режим обработки. Выполнение следующего оператора ON для такой же ситуации прерывания задает для нее другой режим обработки. При этом, если новый оператор ON выполняется в блоке, вызванном данным блоком, режим обработки, заданный предшествующим оператором ON, временно отменяется и восстанавливается лишь при завершении работы блока, содержащего новый оператор ON, или же в результате выполнения оператора REVERT (см. примеры в пп. 3 и 8 этого параграфа).

Предусмотрены следующие ситуации прерывания: реакции системы, вычислительные, ввода и вывода, контроля программы, обработки списков, ситуация, определяемая программистом. Для ситуаций контроля программы и вычислительных ситуаций имя ситуации как средство ее включения задается в программе в виде приставки к операторам (см. п. 6 этого параграфа); остальные ситуации включаются по умолчанию.

2. Ситуации реакции системы. Ситуациями *реакции системы* являются ERROR и FINISH; последняя возникает, когда программа заканчивается и управление будет передано операционной системе, ситуация ERROR — когда программа заканчивается необычным образом, например из-за ошибки в программе. Если для данной ситуации предусмотрена реакция на прерывание, то после выполнения этой реакции возникает ситуация FINISH.

3. Вычислительные ситуации. К *вычислительным* ситуациям относятся ситуации со следующими именами: CONVERSION, FIXEDOVERFLOW, OVERFLOW, UNDERFLOW, ZERODIVIDE.

Ситуация CONVERSION возникает при преобразовании строки символов, содержащей символы, отличные от 0 и 1, в строку битов при наличии недопустимых символов. Результат не определен. Стандартная реакция системы — печать сообщения об ошибке (сигнализация) и возникновение (вызов) ситуации ERROR.

Ситуация **FIXEDOVERFLOW** возникает при выполнении арифметических операций над числами с фиксированной точкой, если результат операции выходит за пределы числового поля (15 разрядов для описателя **DECIMAL** и 31 бита в случае описателя **BINARY**). Результат усекается слева. Стандартная реакция системы — сигнализация и продолжение счета.

Ситуация **OVERFLOW** возникает при выполнении арифметических операций над числами с плавающей точкой, если порядок результата превышает максимально допустимый (75 при основании 10 или 252 при основании 2). Результат не определен. Стандартная реакция системы — сигнализация, ситуация **ERROR**.

Ситуация **UNDERFLOW** возникает при выполнении арифметических операций над числами с плавающей точкой, если порядок результата меньше допустимого минимума (—79 при основании 10 или —260 при основании 2), и не возникает при вычитании равных чисел. Результат равен нулю. Стандартная реакция системы — сигнализация, продолжение счета.

Ситуация **ZERODIVIDE** возникает при попытке деления на нуль. Результат не определен. Стандартная реакция системы — сигнализация, ситуация **ERROR**.

Пр и м е р. При выполнении программы

```
P:PROCEDURE OPTIONS (MAIN); ...;  
  ON OVERFLOW GO TO M; ...;  
  CALL P1;  
P1:PROCEDURE;...;  
  ON OVERFLOW GO TO M1;...;  
END P1; ...; END P;
```

все переполнения, происходящие в процедуре **P** до появления оператора **ON**, вызывают стандартную реакцию системы. Переполнение, происшедшее после выполнения оператора **ON**, приведет к передаче управления оператору процедуры **P** с меткой **M**. Это положение сохраняется до выполнения оператора **ON**, вызванного оператором **CALL P1**; процедурного блока **P1**. С этого момента и до тех пор, пока не закончится выполнение процедуры **P1**, переполнение будет приводить к передаче управления оператору с меткой **M1**, который может находиться как в процедуре **P1**, так и в процедуре **P**. После выполнения процедуры **P1** действие оператора

ON OVERFLOW GO TO M;

восстанавливается до окончания выполнения программы.

4. Ситуации ввода и вывода. К ситуациям *ввода* и *вывода* относятся ситуации со следующими именами: **ENDFILE**, **ENDPAGE**, **KEY**, **NAME**, **RECORD**, **TRANSMIT**, **UNDEFINEDFILE**. Ситуации ввода и вывода всегда относятся к конкретному файлу, имя ко-

торого F указывается в скобках после соответствующей ситуации прерывания, например NAME (F).

Ситуация ENDFILE возникает при попытке считать из указанного файла что-либо, находящееся за ограничителем этого файла, т. е. после того, как прочитана последняя запись. Стандартная реакция системы — сигнализация, ситуация ERROR.

Ситуация ENDPAGE возникает при попытке печати за последней строкой на странице. Стандартная реакция системы — начинается новая страница.

Ситуация KEY возникает, если заданный признак (ключ) не обнаружен (при вводе) или задан уже существующий признак (при выводе). Стандартная реакция системы — сигнализация, ситуация ERROR.

Ситуация NAME возникает, если при вводе появляется нераспознаваемый идентификатор или имя, отсутствующее в списке данных. Стандартная реакция системы — сигнализация, игнорируется присваивание для отсутствующего имени.

Ситуация RECORD возникает, если фактическая длина записи не соответствует длине, указанной в объявлении файла. Стандартная реакция системы — сигнализация, ситуация ERROR.

Ситуация TRANSMIT возникает при наличии ошибки в операторах ввода или вывода, т. е. при передаче данных. Стандартная реакция системы — сигнализация, ситуация ERROR.

Ситуация UNDEFINEDFILE возникает при попытке открыть файл, который не определен, или когда совокупность описателей файла противоречива. Стандартная реакция системы — сигнализация, ситуация ERROR.

5. Ситуации контроля программы и обработки списков. К ситуациям *контроля программы* относятся ситуации с именами CHECK, SIZE, STRINGRANGE, SUBSCRIPTRANGE.

Ситуация CHECK (L), где L — список элементов (каждый элемент списка может быть идентификатором скалярной величины, массива, или структуры, константой типа метки, или меткой входа), возникает каждый раз, когда выполняется оператор, метка которого содержится в списке L, или изменяется переменная из списка элементов, т. е. ситуация заключается как бы в проверке элементов списка L. Стандартная реакция системы — печатаются элемент и (в случае переменной) его новое значение.

Ситуация SIZE возникает, если значение, полученное при вводе или в результате выполнения оператора присваивания, не помещается в отведенном поле. Результат не определен. Стандартная реакция системы — сигнализация, ситуация ERROR.

Ситуация STRINGRANGE возникает, если при обращении к встроенной функции с идентификатором SUBSTR допущена ошибка в длине списка аргументов. Стандартная реакция системы — сиг-

нализация, длина списка преобразуется в допустимые границы, программа продолжает выполняться.

Ситуация SUBSCRIPTRANGE возникает, когда вычисленное значение индекса находится вне указанных для него границ. Результат не определен. Стандартная реакция системы — сигнализация, ситуация ERROR.

6. Использование ситуаций прерывания в программе. Автоматически устанавливается режим игнорирования ситуаций SIZE, SUBSCRIPTRANGE, CHECK, однако при необходимости реакция системы на каждую из этих ситуаций может быть включена. Реакция на ситуации UNDERFLOW, OVERFLOW, ZERODIVIDE, CONVERSION, FIXEDOVERFLOW обычно включена, но может выключаться программистом. Реакция на все остальные ситуации включена всегда и не подвластна программисту.

Включение реакции на ситуацию осуществляется указанием имени ситуации в списке имен ситуаций (см. § 3) перед меткой оператора. Указанное включение относится только к данному оператору (или процедуре). Например:

(SIZE): M: $A = (B - 3) * 4$;

Выключение реакции на ситуацию осуществляется добавлением к имени ситуации приставки NO и записью полученного ключевого слова в списке имен ситуаций перед оператором. Например, запись вида

(NOUNDERFLOW, SIZE): P: $Y = 3.14 - X$;

обеспечивает реакцию на ситуацию SIZE и отсутствие реакции на ситуацию UNDERFLOW.

Действие приставки может быть отменено, если написать имя ситуации непосредственно перед данным оператором. Например, в следующем фрагменте программы:

(NOOVERFLOW): P: PROCEDURE; ...;

(OVERFLOW): S: $Y = 2.3 * (X + 1)$; ...; END;

для всей процедуры реакция на ситуацию OVERFLOW отменяется, за исключением одного оператора с меткой S.

Область действия ситуации прерывания, указанной перед оператором присваивания, распространяется только на данный оператор, но ни на одну из процедур, которые этот оператор вызывает, а в случае условного оператора — на выражение, следующее за ключевым словом IF. Отметим, что перед конструкциями, начинающимися с дополнений THEN и ELSE, могут быть свои собственные имена ситуаций. Если ситуация прерывания указана перед оператором DO, то область ее действия распространяется на выражения в заголовке цикла, но не на операторы группы DO. В случае блоков PROCEDURE и BEGIN ситуация прерывания оказывает действие

на все операторы вплоть до оператора END, включая все вложенные блоки. На вызываемые процедуры, лежащие вне блока, влияние ситуации прерывания, указанное перед данным блоком, не распространяется.

7. Оператор имитации ситуации прерывания. Оператор, записываемый в виде

SIGNAL C;

где C — имя ситуации прерывания, вызывает немедленное возбуждение указанной в нем ситуации и управление передается оператору ON, включающему в свой состав данную ситуацию C и выполненному к моменту появления оператора SIGNAL. Этот оператор позволяет упростить процесс отладки сложной программы, так как создается эффект действительного возникновения указанной ситуации и порядок выполнения операторов изменится. После выполнения действия по прерыванию управление возвращается к оператору, следующему сразу за оператором SIGNAL.

Если к моменту выполнения оператора SIGNAL указанная в нем ситуация не включена (не был выполнен оператор ON, или такого оператора вообще нет в программе, или реакция на данную ситуацию выключена), прерывания не происходит. Например, если некоторый фрагмент программы содержит операторы

M: PROCEDURE; P:ON ENDFILE (F) Y, Z=0; ...;

S: SIGNAL ENDFILE (F); Q:X=1;...;

T: SIGNAL SIZE; ... ; END M;

то оператор SIGNAL с меткой S вызовет прерывание так же, как если бы действительно произошла попытка считывания чего-либо, находящегося за ограничителем файла. Управление будет передано оператору, указанному в ON с меткой P, после чего выполнится оператор с меткой Q. Оператор SIGNAL с меткой T прерывания не вызывает.

8. Оператор отмены реакции на прерывание. Оператор, имеющий вид

REVERT C;

где C — имя ситуации, отменяет любой режим обработки прерывания для ситуации C, заданный в текущем блоке, т. е. отменяется действие оператора ON с той же ситуацией C в пределах блока, в котором содержатся как оператор ON, так и оператор REVERT с одной и той же ситуацией C.

После выполнения оператора отмены реакции на прерывание устанавливается тот режим обработки, который действовал в момент входа в данный блок. Если же в данный момент для какой-либо ситуации не задан никакой режим обработки прерывания, то при возникновении этой ситуации будет применяться стандартный режим

обработки прерывания. Например, в процедуре

```
PR:PROCEDURE;...;  
  REVERT ZERODIVIDE;...;  
R:ON ZERODIVIDE GO TO P;...; END PR;
```

оператор с меткой R воспринимается как пустой оператор, и в случае деления на нуль возникнет стандартная реакция системы ERROR.

9. Оператор вывода на дисплей. Для выдачи на пульт машины определенной информации, например директив для программиста, сообщений о выполнении отдельных блоков программы и др., а также для прерывания выполнения программы используется оператор *вывода на дисплей*, имеющий вид

```
DISPLAY (E) REPLY (V);
```

где E — скалярное выражение, V — строковая переменная, REPLY — дополнение. Конструкция REPLY (V) может отсутствовать. В этом случае оператор DISPLAY выдает на пульт сообщение, представляющее собой строку символов, полученную в результате вычисления значения выражения E и преобразования его, если это необходимо, в строку символов. Максимальная длина этой строки зависит от применяемой аппаратуры. После выполнения оператора DISPLAY управление передается следующему оператору. Например, в результате выполнения оператора

```
DISPLAY ('END_BLOCK' || W);
```

где переменная W получила значение 13, на пульт будет выдано сообщение END BLOCK 13.

Если в операторе содержится конструкция REPLY (V), то переменной V присваивается значение E и выдается в качестве сообщения, после чего выполнение программы прерывается и состояние ожидания будет продолжаться до тех пор, пока оператор-программист у пульта не ответит. Ответ оператора-программиста в виде строки символов становится значением переменной V и может использоваться в программе. Переменная V не должна иметь длину больше допустимой для данного устройства (в случае пишущей машинки 72 байт).

Оператор DISPLAY используется обычно в режиме диалога человек — машина.

10. Ситуация, определяемая программистом. Ситуация, определяемая программистом, имеет вид CONDITION (X), где X — идентификатор, задаваемый программистом; CONDITION — имя ситуации. По умолчанию предполагается, что X — внешнее имя, следовательно, во всей программе ситуации с именем CONDITION с одинаковыми идентификаторами представляют одну и ту же ситуацию. Стандартная реакция системы при обработке прерываний из-за

ситуаций **CONDITION** — сигнализация (печать соответствующего сообщения) и продолжение выполнения программы.

Ситуации **CONDITION (X)** возбуждаются только оператором имитации прерываний в виде

SIGNAL CONDITION (X);

Например, оператор

ON CONDITION (CONTROL) GO TO ROL;

обеспечивает в программе после выполнения оператора

SIGNAL CONDITION (CONTROL);

включение реакции на ситуацию **CONTROL** и выполнение оператора **GO TO ROL**;

Операторы, рассмотренные в этом параграфе, относятся к операторам отладки.

Упражнения

1. Указать, какие из приведенных операторов содержат ошибки:

ON ZERODIVIDE SYSTEM;

ON ERROR RETURN;

ON SIZE;

ON SIZE BEGIN; I=1; J=10; END;

(CHECK (X)): X=Y;

M: ON SIZE GO TO M;

ON ERROR SYSTEM SNAP;

ON SIZE ON FIXEDOVERFLOW SYSTEM;

(NOCHECK (K, L)): DO K=1 TO 7; L=K; END;

ON ERROR SNAP;

2. Определить, какое значение получит переменная **Z** в конце выполнения программы:

P: PROCEDURE OPTIONS (MAIN);

ON SIZE Z=-1; ON CONDITION (Y1) Z=Z+1;

ON CONDITION (Y2) Z=Z+1; ON ERROR Z=Z+1;

ON FINISH Z=Z+1; Z=0;

SIGNAL CONDITION (Y1); SIGNAL SIZE;

IF Z<0 THEN M: SIGNAL FINISH;

SIGNAL CONDITION (Y2); SIGNAL ERROR;

END;

§ 13. Распределение памяти

Под *распределением памяти* подразумевается выделение участков памяти для хранения программы и обрабатываемых данных. Для переменных, которые описаны в блоке, транслятором автоматически выполняется распределение памяти. В то же время програм-

мист может активно вмешиваться в процесс распределения памяти, указывая при объявлении данных в операторе DECLARE соответствующие описатели.

1. Описатели области действия. Описателями *области действия* являются описатели EXTERNAL и INTERNAL, служащие для указания областей алгоритма, в которых объявляемые имена доступны.

По умолчанию все имена переменных имеют описатель INTERNAL и считаются описанными как *внутренние*, а имена процедур и входов — описатель EXTERNAL, и считаются описанными как *внешние* имена. Последний из этих описателей уже рассматривался в § 10.

Имя, объявленное как внутреннее, считается описанным и может быть использовано только в блоке, в котором оно определено, а также в обычных блоках и процедурах, являющихся внутренними для блока с описанием имени. Внешнее имя может быть использовано в любом месте программы.

2. Описатели выравнивания и упаковки. Данные в памяти могут размещаться либо последовательно, без каких-либо промежутков, либо таким образом, чтобы начало данных приходилось на границу, соответствующую этому типу данных. Первый случай размещения данных обеспечивается описателем UNALIGNED, и такие данные называются невыровненными или упакованными. Во втором случае размещение обеспечивается описателем ALIGNED, и такие данные называются выровненными или неупакованными.

Использование описателя UNALIGNED позволяет экономить память, но замедляется доступ к данным. Доступ к данным с описателем ALIGNED ускоряется, но данные в памяти могут быть размещены с разрывом (образуются неиспользуемые участки памяти). По умолчанию для строковых переменных принимается описатель UNALIGNED, а для остальных типов данных — описатель ALIGNED.

3. Описатели классов памяти. Распределение памяти может быть статическим (до выполнения программы) или динамическим (во время выполнения программы). Для управления размещением данных в памяти предусмотрены четыре различных класса памяти: статическая память, автоматическая, управляемая и базированная. Объявление класса памяти осуществляется соответственно описателями STATIC, AUTOMATIC, CONTROLLED, BASED (P), где P — указатель, которыми программист может воспользоваться в операторе DECLARE. Три последних описателя характеризуют динамическое распределение памяти. В случае структур описатели классов памяти могут быть заданы лишь для старшей структуры.

Память для переменной с описателем STATIC распределяется перед выполнением программы и не освобождается до конца работы программы. Значения таких переменных можно использовать и при

повторном входе в блок, в котором имеется их описание. Описатель **STATIC** употребляется совместно с описателями **EXTERNAL** или **INTERNAL**. Для переменных с описателем **EXTERNAL**, если класс памяти не указан, по умолчанию подразумевается описатель **STATIC**.

Описатель **AUTOMATIC** означает, что данные размещаются в динамической памяти, распределение которой производится при каждом входе в блок. После выхода из блока память освобождается и значения переменных, описанных в этом блоке, теряются. Исключение составляет только рекурсивное обращение. Для переменных с описателем **INTERNAL**, если класс памяти не указан, по умолчанию подразумевается описатель **AUTOMATIC**.

Размещение в памяти переменных, имеющих описатели **CONTROLLED** или **BASED**, т. е. управляемых и базированных переменных, выполняется специальными операторами. Память, отведенная под управляемые или базированные переменные, освобождается при выполнении соответствующих операторов.

4. Размещение управляемых данных. Для отведения памяти под управляемые переменные используется оператор *размещения* данных вида

ALLOCATE X L;

где **X** — идентификатор, **L** — описатели управляемой переменной, массива или структуры. В последнем случае перед идентификатором указывается номер уровня. Список описателей **L** может включать в себя только описатели измерения и описатели **BIT**, **CHARACTER** с описателями разрядности, а также описатели начальных значений. При отсутствии описателей действует принцип умолчания. Память, отведенная оператором **ALLOCATE**, будет занята и после выхода из блока, хотя обращение к переменной не разрешается вне блока.

Освобождение памяти производится оператором

FREE X;

где **X** — один или несколько идентификаторов, память под которые была отведена оператором **ALLOCATE**. Оператор **FREE** вызывает освобождение памяти, последний раз отведенной под переменные **X**. Оператор освобождения памяти и соответствующий ему оператор размещения данных должны принадлежать одной и той же процедуре.

Описатель **CONTROLLED** используется при динамическом распределении памяти, например для размещения в памяти массивов, у которых диапазоны изменения индексов не определены в момент входа в блок, а вычисляются в процессе выполнения блока. Следующий фрагмент программы иллюстрирует сказанное:

```
BEGIN; DECLARE Z(M,N) CONTROLLED FIXED;...;
      M=10; N=20; ALLOCATE Z;...;
L1: FREE Z; ...; M, N=30; ALLOCATE Z;...;
L2: FREE Z; END;
```

Здесь при описании массива *Z* границы заданы переменными и при входе в блок распределение памяти для данного массива не производится. После того как *M* и *N* получили значения, оператором *ALLOCATE Z*; резервируется память для *Z*. После освобождения памяти, отведенной для *Z*, оператором с меткой *L1* массив *Z* может быть использован в блоке с новыми границами, если написать новый оператор *ALLOCATE Z*;. Вторично память освобождается оператором с меткой *L2*.

Определить, выделялась ли память для управляемой переменной, можно с помощью встроенной функции *ALLOCATION (X)*. Если память для *X* выделялась, то значение, вычисляемое этой функцией, равно '1'B, если не выделялась, то оно равно '0'B. Например, можно написать следующие операторы:

```
DECLARE T(M,N) CONTROLLED;...; IF ALLOCATION (T)
      THEN ALLOCATE T(I,J) INITIAL ((I*J) 0);
```

5. Размещение базированных данных. Оператор размещения для базированных данных имеет вид

```
ALLOCATE X SET (P);
```

где *X* — имя базированной переменной, массива или структуры, *P* — скалярная переменная типа указателя (возможно элемент массива или структуры); переменная *P* не обязана совпадать с переменной типа указателя, связанной с переменной *X* и описанной в операторе *DECLARE*. Этот оператор резервирует ранее свободные участки памяти, причем размер каждого участка равен размеру соответствующей переменной *X*. Положение (адрес) начала участка определяется либо значением указателя *P*, либо, в случае отсутствия конструкции *SET (P)*, — значением указателя, заданного в описателе *BASED* переменной *X*. Допускается смешанная форма оператора *ALLOCATE*, когда вперемежку в одном операторе размещаются как базированные, так и управляемые переменные.

Освобождение памяти, отведенной под базированные переменные с помощью оператора размещения, выполняется оператором освобождения памяти вида

```
FREE P → X;
```

где знак *→* обозначает освобождение памяти, отведенной для переменной *X* с данным указателем *P* (конструкция *P→* может быть опущена). Начало освобождаемого участка задается указателем переменной *X*, размер участка определяется размером переменной *X*. Освобождаемый участок должен совпадать с ранее занятым

участком. Например, в операторе

DECLARE T(100) BASED (P);

описан одномерный массив Т, который размещается в базированной памяти. Переменная Р задана описателем POINTER. Размещение массива Т в памяти осуществится после выполнения оператора

ALLOCATE T SET (P);

который присваивает указателю Р начальный адрес размещения элементов массива Т. Освобождение памяти производится оператором

FREE P → Т;

Упражнения

1. Найти ошибки в следующих операторах объявления данных:

DECLARE A(16) STATIC FIXED AUTOMATIC;

DECLARE B(10) AUTOMATIC EXTERNAL;

DECLARE C CHARACTER (N) STATIC;

DECLARE 1 D, 5 E STATIC, 5 F STATIC;

DECLARE H(L,M) EXTERNAL;

2. Определить, какую длину будут иметь переменные X,Y,Z в конце выполнения процедуры

C: PROCEDURE;

DECLARE (X,Y,Z) CONTROLLED CHARACTER (9);

ALLOCATE X, Y, Z CHARACTER (4);

ALLOCATE X, Y CHARACTER (6), Z;

FREE X, Z;

ALLOCATE X, Y CHARACTER (*), Z CHARACTER (*);

END C;

3. Какие из операторов объявления данных недопустимы и почему?

DECLARE A BASED (P), P BASED (S), S POINTER;

DECLARE 1 B BASED (P), 2 C, 2 D BIT (1), P POINTER;

DECLARE C(L) BASED (P), P POINTER;

DECLARE (D BASED (S), S POINTER) EXTERNAL;

DECLARE E BASED (P.S), 1 P, 2 (S,R) POINTER;

4. Написать программу для вычисления значения каждого очередного члена ряда $a_1 + a_2 + \dots + a_n + \dots$, если $a_1 = a_2 = 1$, а при $n \geq 3$; $a_n = a_{n-2} + a_{n-1}$. Для заданного n сформировать массив значений всех n членов.

§ 14. Описатели файла

Данные, размещенные на внешних носителях информации, составляют комплекты (или наборы) данных. Из них образуются файлы, состоящие из отдельных записей. *Запись* — это основная еди-

ница обмена информацией между внешними носителями и памятью. Запись содержит значения одной или целого набора переменных. Обмен между внешним носителем и памятью производится обычно через специальную область памяти — *буфер*. При объявлении файла указываются различные описатели файла, от которых будет зависеть обработка файла при вводе и выводе. Описатели одного и того же файла в нескольких внешних процедурах не должны противоречить друг другу.

К числу описателей файла относятся ключевые слова: FILE, RECORD, STREAM, INPUT, OUTPUT, UPDATE, SEQUENTIAL, DIRECT, BUFFERED, UNBUFFERED, PRINT, BACKWARDS, EXCLUSIVE, KEYED, ENVIRONMENT, CONSECUTIVE, INDEXED, REGIONAL, LINESIZE, PAGESIZE.

Описатель FILE определяет данный идентификатор как имя файла. Этот описатель используется со всеми другими перечисленными выше описателями и может быть опущен, если употребляется хотя бы один из этих описателей, наличие описателя FILE в таком случае определяется контекстуально.

1. Описатели способа обработки. Описатели способа обработки RECORD и STREAM определяют метод использования данных в файле.

Описатель RECORD указывает, что данный файл следует рассматривать как последовательность записей и определяет передачу комплекта данных, состоящего из отдельных записей. Передача идет без преобразования либо прямо в память или из памяти, либо на буфер.

Описатель STREAM указывает, что данный файл рассматривается как последовательность символов и определяет передачу комплекта данных как непрерывный поток символов. В этом случае записи, которые находятся в буфере, разделяются на составляющие элементы. Они редактируются и преобразуются из внешнего представления во внутреннее.

Файл с описателем RECORD может использоваться только в операторах OPEN, CLOSE, READ, WRITE, REWRITE, LOCATE, DELETE, UNBLOCK. Файл с описателем STREAM может использоваться только в операторах OPEN, CLOSE, PUT, GET и не употребляется с описателями RECORD, UPDATE, DIRECT, SEQUENTIAL, BUFFERED, UNBUFFERED, BACKWARDS, EXCLUSIVE, KEYED.

2. Описатели функции файла. Описателями функции файла являются INPUT, OUTPUT, UPDATE. Эти описатели указывают назначение файлов. Первый указывает, что данные из файла будут передаваться в память (в программу), второй — что данные будут передаваться из памяти на внешний носитель (в файл). Описатель

UPDATE указывает, что файл может использоваться как для ввода, так и для вывода данных.

Файл с описателем INPUT не может иметь описателей EXCLUSIVE и PRINT, а файл с описателем OUTPUT — описателей EXCLUSIVE и BACKWARDS. Файл с описателем UPDATE не может употребляться с описателями BACKWARDS, STREAM и PRINT.

3. Описатели доступа. Описатели DIRECT и SEQUENTIAL указывают возможность соответственно прямого и последовательного доступа к файлу с описателем RECORD.

Последовательный доступ заключается в том, что устройство ввода просматривает данные в порядке их следования. Обмен между внешним носителем и памятью, как правило, начинается с первой и заканчивается последней записью. Последовательный доступ осуществляется для файлов на перфокартах и магнитных лентах. Например, объявление файла с описателями UPDATE и SEQUENTIAL определяет способ обновления данных, т. е. следующий процесс: ввод записи в память, выполнение над элементами записи операций и вывод записи обратно на то же место, которое она занимала на внешнем носителе. Обращение к данным в таких файлах должно осуществляться последовательно операторами READ и затем REWRITE.

Прямой доступ заключается в том, что к любой части внешнего носителя, в которой размещены данные, можно обратиться непосредственно. Для этого физическим записям файла присваивается ключ, который указывает положение записи, следовательно, файлы с описателем DIRECT должны также иметь описатель KEYED.

4. Описатели буферизации. Описатели буферизации, обозначаемые ключевыми словами BUFFERED и UNBUFFERED, применимы только для файла с описателями SEQUENTIAL и RECORD и определяют, необходимо или нет пользоваться промежуточной памятью (буфером) во время обмена данными между памятью и внешними носителями.

5. Дополнительные описатели. Описатель PRINT из этой группы описателей файла указывает, что окончательно данные должны быть расположены на печатной странице. Файл с описателем PRINT подразумевает описатели OUTPUT и STREAM, описатель PRINT не может быть указан для файла с описателем RECORD.

Описатель BACKWARDS указывает, что к файлу с описателями SEQUENTIAL и INPUT обращения происходят в обратном порядке, т. е. от последнего элемента файла к первому.

Описатель EXCLUSIVE указывает, что файл с описателями DIRECT и UPDATE будет использован таким образом, чтобы запрещалось чтение, исключение или перепись записи в одной ветви, в то время как в другой ветви эта запись находится в одном из перечисленных состояний. Здесь речь идет о параллельном (асинхрон-

ном) выполнении отдельных частей программы, что в этой книге не рассматривается.

Описатель KEYED (K) указывает, что каждая запись в файле имеет связанный с ней признак, т. е. свой ключ. После описателя в скобках пишется десятичная целая константа — (K), которая определяет длину признака в символах. Файл с описателем KEYED может иметь описатели STREAM и PRINT. Описатель KEYED должен быть указан для каждого файла, содержащего ключ, даже если записи читаются последовательно.

Описатель ENVIRONMENT (D), где D — список дополнений, характеризует особенности физического размещения данных на внешнем носителе. Состав и правила задания компонент в списке дополнений существенно зависят от используемого оборудования ЭВМ, ее программного обеспечения и способа организации файла. В частности, в списке дополнений могут быть указаны форматы записей, устанавливающие размеры записей и наборов записей (блоков), подлежащих передаче.

По способу организации файлы подразделяются на стандартные последовательные, индексно-последовательные и региональные (с прямым доступом). В описателе ENVIRONMENT в списке дополнений способ организации задается одним из описателей: CONSECUTIVE, INDEXED, REGIONAL.

Последовательный (CONSECUTIVE) способ организации файла характеризуется тем, что блоки данных запоминаются на внешнем носителе в следующих друг за другом областях. Записи на перфокартах — типичный пример последовательной организации файла. Файлы на магнитных лентах и файлы, выведенные на печать, также являются последовательно организованными.

Региональная (REGIONAL) организация файла заключается в том, что память на внешнем носителе подразделяется на области. В каждой из них может размещаться одна или несколько записей (записи в блоки не объединяются). Перед записью указываются ключи, которые обеспечивают прямой доступ к записи файла. Региональная организация файлов возможна только на дисках.

Последовательная и региональная организации файлов обеспечивают соответственно последовательный и прямой доступ к записям. При индексно-последовательной (INDEXED) организации файла записи образуют логически упорядоченную последовательность в соответствии с заданным признаком (ключом), значения которого возрастают от записи к записи. Расположение записей файла описывается с помощью индексов. Доступ возможен как прямой, так и последовательный. Индексно-последовательная организация возможна для файлов, размещаемых на дисках. Если файл не является стандартным последовательным, он должен иметь ключи, т. е. либо в опе-

раторе DECLARE, либо в операторе OPEN необходимо указать описатель KEYED (K).

Если ни один из трех способов организации файла не указан, то по умолчанию предполагается способ CONSECUTIVE. Остальные дополнительные описатели никогда не выбираются по умолчанию, а должны всегда задаваться явно либо в операторе объявления файла, либо в операторе открытия файла. Единственным исключением является описатель PRINT, присваиваемый по умолчанию стандартному выводному файлу SYSPRINT. Все другие рассмотренные выше описатели, называемые взаимно исключающими, могут быть выбраны по умолчанию, а именно: из описателей способа обработки — присваивается описатель STREAM, из описателей функции — INPUT, из описателей доступа — SEQUENTIAL, из описателей буферизации — BUFFERED, а из описателей области действия — всегда описатель EXTERNAL.

6. Описатели при печати. Имеются два описателя, которые применяются только для файлов с описателем PRINT. Это описатели LINESIZE (E) и PAGESIZE (E), где E — выражение. Первый из них указывает длину печатаемой строки (включая символ управления). По умолчанию длина строки равна 120 печатным знакам. Описатель PAGESIZE указывает число строк, приходящихся на одну страницу. По умолчанию число строк на странице равно 60.

Упражнения

1. Составить оператор описания файла F, предназначенного для обработки данных как потока символов, если известно, что файл будет использоваться: а) для вывода, но не на печать; б) и для ввода и для вывода; в) для вывода на печать; г) для ввода и имя файла не должно быть внешним;

2. Указать ошибки при описании файлов в операторах:

```
DECLARE A FILE REAL;  
DECLARE B FILE STREAM INPUT RECORD;  
DECLARE C INPUT STREAM PRINT;  
DECLARE FILE OUTPUT INPUT;  
DECLARE D INTERNAL RECORD PRINT;
```

§ 15. Подготовка файлов к вводу-выводу

Прежде чем приступить к чтению или записи какого-либо файла, он должен быть открыт. Открытие файла означает, что некоторое имя ставится в соответствие конкретному файлу. Любой файл можно открыть двумя способами: явно — путем использования оператора открытия файлов — и неявно.

Оператор *открытия файлов* имеет вид

```
OPEN FILE (F) FL;
```

где F — имя файла, FL — совокупность описателей. В операторе OPEN конструкций вида FILE (F) FL может быть несколько. Разделителем в этом случае выступает символ запятая. Для файла, который уже открыт, оператор открытия файлов игнорируется.

Файл открывается неявным образом, если выполняется хотя бы один из операторов ввода или вывода, относящийся к файлу, который не был открыт. Открытие файла неявным образом не позволяет программисту вводить какие-либо описатели во время открытия.

После обработки последней записи файла он должен быть закрыт. Для этого служит оператор

CLOSE FILE (F);

где конструкций вида FILE (F), записываемых через запятую, может содержаться произвольное число. Оператор CLOSE для неоткрывшегося файла игнорируется.

После закрытия файл может быть открыт снова. Однако если файл закрывается, то теряются те описатели, которые были указаны в операторе OPEN, остаются лишь описатели, записанные в операторе DECLARE. Поэтому можно закрыть файл, использованный для вывода, и затем открыть его повторно, но уже как файл для ввода. Например, в процедуре

```
PRINT: PROCEDURE (P);  
      DECLARE P(*) FIXED,F FILE PRINT;...;  
      S: OPEN FILE(F) LINESIZE(49);...;  
          CLOSE FILE(F);...;  
      END PRINT;
```

файл F используется для вывода. Если данные печатаются до выполнения оператора с меткой S или после оператора CLOSE, то печатная строка будет иметь стандартную длину, в противном случае оператор OPEN с меткой S обеспечит длину строки в 49 символов.

Для удобства организации ввода и вывода информации, кроме файлов, которые объявляются в программе и имена которых выбираются произвольно, существуют два файла — SYSIN и SYS-PRINT,— называемые соответственно стандартным вводным и стандартным выводным файлами. Эти файлы объявлять не надо; по умолчанию считается, что описателями файла SYSIN являются STREAM и INPUT, а файл SYSPRINT имеет описатели STREAM и PRINT.

Стандартный вводной файл, как правило, используется для ввода информации с перфокарт. С каждой перфокарты может быть введено до 80 символов. Стандартный выводной файл практически всегда используется для печати. При этом как стандартная длина строки, так и стандартное число строк на странице могут быть изменены оператором OPEN,

Упражнение

Составить оператор открытия файла FL, предназначенного для передачи потока данных, если файл явно не описан и известно, что файл будет использоваться для: а) вывода на печать со стандартными размерами строк и страниц; б) вывода, но не на печать; в) ввода; г) вывода на печать со строкой из 50 символов и страницей из 50 строк; д) вывода на печать со стандартной по размеру строкой и страницей из 29 строк; е) вывода на печать со строкой длиной 60 символов и стандартной по размеру страницей.

§ 16. Ввод и вывод данных

Под *вводом* и *выводом* понимается передача данных из внешних носителей информации в память ЭВМ и соответственно из памяти на внешние носители. Ввод и вывод осуществляются операторами, которые в качестве составной части имеют список ввода-вывода.

Список ввода-вывода определяет участки памяти, используемые для ввода или вывода, и имеет вид (A), где A — список элементов, разделяемых запятыми. В качестве элемента списка допускаются переменные любого типа, имена массивов, структур и подструктур, отдельные элементы массивов и структур, псевдопеременные, индексированные элементы, а также выражения (в списке вывода). Конкретный вид списка ввода-вывода зависит от способа обработки информации.

Если в качестве элемента списка ввода-вывода указана переменная, то это означает, что необходимо выполнить ввод (вывод) значения этой переменной. Если в качестве элемента списка ввода-вывода указано имя массива или старшей структуры, то это означает, что надо выполнить ввод (вывод) всех элементов этого массива или структуры в том порядке, в котором эти элементы размещены в массиве или в соответствии со строением структуры и с порядком размещения элементов массивов, входящих в эту структуру.

Индексированный элемент имеет вид

24 ((A(I, J, K) DO I=I1 TO I2 BY I3) DO J=J1 TO J2 BY J3) DO K=K1 TO K2 BY K3)

и представляет собой сокращенную запись заголовков вложенных циклов для переменных с индексами. Если в качестве элемента списка ввода-вывода указан индексированный элемент, то это означает, что необходимо ввести (вывести) значения величин, указанных в списке индексированного элемента, организовав порядок их следования в соответствии с изменением параметров циклов, число которых не обязательно равно трем, как в данном примере. Использование индексированных элементов в списках ввода-вывода иногда позволяет избежать сложных групп и операторов цикла, в других дает возможность организовать единственный простой способ пере-

дачи данных в удобной форме. Например, индексированный элемент ((X(I,J) DO I=2 TO 4, 7) DO J=1 TO 5 BY 2)

представляет собой совокупность элементов массива X в такой последовательности: X(2,1), X(3,1), X(4,1), X(7,1), X(2,3), X(3,3), X(4,3), X(7,3), X(2,5), X(3,5), X(4,5), X(7,5).

В соответствии с тем, что файлы, подлежащие обработке, могут иметь описанием либо STREAM, либо RECORD, существует два способа передачи данных: ввод-вывод, ориентированный на поток, и ввод-вывод, ориентированный на запись.

Первый способ передачи данных предполагает, что входной или выходной поток состоит из непрерывной последовательности символов. Объем данных, передаваемых за одно выполнение оператора ввода или вывода, определяется списком ввода-вывода, элементам которого присваиваются данные из этого потока или из значений, формирующих выходной поток данных. Преобразования данных из символьной формы представления во внутреннюю, определяемую характеристиками элемента списка ввода-вывода, или, наоборот, из внутренней формы в символьную осуществляются по необходимости.

Возможны следующие способы передачи потока данных: передача, управляемая списком, передача, управляемая данными, передача, управляемая редактированием. Кроме обмена информацией между памятью и внешними носителями, возможны внутренние пересылки данных. Операторы ввода, ориентированные на поток данных, начинаются с ключевого слова GET, операторы вывода — с ключевого слова PUT.

Способ передачи данных, ориентированный на запись, предполагает, что каждый файл рассматривается как последовательность записей. При вводе запись целиком и без преобразования присваивается некоторой переменной (массиву, структуре), при выводе значение переменной (массива, структуры) целиком и без преобразования образует некоторую запись. В общем случае запись представляет собой конструкцию вида KR, где K — последовательность символов, называемая ключом (признаком), K может и отсутствовать, R — данное. В качестве R может быть действительное число с фиксированной или плавающей точкой, комплексное число (пара действительных чисел, разделенных запятой), строка символов или строка битов.

Различают операторы ввода-вывода записей последовательного доступа и операторы ввода-вывода записей прямого доступа. При использовании операторов ввода-вывода записей последовательного доступа записи могут быть переданы с использованием буфера или без него. Операторы ввода, ориентированные на записи, начинаются с ключевых слов READ, DELETE, операторы вывода — с ключевых слов WRITE, LOCATE, REWRITE,

Упражнения

1. Составить список ввода-вывода для обработки элементов двумерного массива $A = \{a_{ij}\}$, ($1 \leq i \leq m$, $1 \leq j \leq n$, где m , n — заданы) в следующем порядке:

а) $a_{11}, a_{21}, \dots, a_{m1}, a_{12}, a_{22}, \dots, a_{m2}, \dots, a_{1n}, a_{2n}, \dots, a_{mn}$;

б) в порядке, обратном по отношению к случаю а);

в) $a_{11}, a_{31}, a_{51}, \dots, a_{13}, a_{33}, a_{53}, \dots, a_{22}, a_{42}, \dots, a_{24}, a_{44}, \dots$

2. Составить список ввода-вывода для обработки элементов обеих главных диагоналей матрицы A (см. упражнение 1) в прямом и обратном порядке ($m=n$).

3. Составить список ввода-вывода для обработки элементов одномерных массивов $B = \{b_i\}$ и $C = \{c_i\}$ ($1 \leq i \leq n$, где n задано) в следующем порядке:

а) $b_1, c_1, b_2, c_2, \dots, b_n, c_n$;

б) $b_1, c_n, b_2, c_{n-1}, \dots, b_n, c_1$;

в) $b_n, b_{n-1}, \dots, b_2, b_1, c_1, c_2, \dots, c_{n-1}, c_n$.

§ 17. Операторы ввода-вывода потока

1. Ввод-вывод, управляемый списком. Ввод данных обеспечивается оператором

GET FILE (F) LIST (A);

где F — имя файла, A — список ввода, LIST — дополнение, специфицирующее список. Данные читаются символ за символом с вводного устройства, рассматриваются как строки символов и присваиваются переменным из списка ввода с соответствующими преобразованиями. Данные на вводном устройстве разделяются запятой или пробелом. Границы карт игнорируются, и вводимые данные обрабатываются как непрерывный поток.

Вывод данных осуществляется оператором

PUT FILE (F) LIST (A);

где A — список вывода, остальные обозначения такие же, как и в операторе ввода. Границы карт или строк игнорируются; значению каждой переменной отводится столько места, сколько нужно; в качестве разделителя используется пробел.

Конструкция FILE (F) в операторах GET и PUT может быть опущена. В этом случае при вводе и выводе используются стандартные файлы (системные устройства) с именами SYSIN и SYSPRINT соответственно.

Пр и м е р. Операторы

DECLARE A FIXED (5,2), B FIXED (3,2), C FIXED (5,1);
GET LIST (A,B,C);

обеспечат ввод перфокарт с системного устройства с отперфорированными на них, например, следующими данными:

1.3 3.45 6.7,
80 — 9.87 6.54 3 — 2
1

Если оператор GET проработал один раз, переменные A, B, C получат соответственно значения 001.30, 3.45, 006.7; после второго выполнения — 080.00, —9.87, 0006.5; после третьего — 003.00, —2.00, 0001.0.

К оператору GET может быть добавлено дополнение COPY, которое обеспечивает печать вводимых данных на SYSPRINT. Например, оператор

```
GET LIST (A) COPY;
```

выполняет чтение и преобразование переменной A во внутреннюю форму, а также выдает на печать значение переменной A в том виде, в каком оно было воспринято на устройстве ввода.

2. Ввод-вывод, управляемый данными. Передача, управляемая данными, позволяет читать или записывать данные, представляемые в виде допустимых констант с именами передаваемых значений. Операторы ввода и вывода, управляемые данными, соответственно имеют вид

```
GET FILE (F) DATA (A);
```

```
PUT FILE (F) DATA (A);
```

где F — имя файла, DATA — дополнение, специфицирующее данные, A — список ввода или вывода; при вводе A — список идентификаторов простых переменных или массивов, при выводе A не может содержать выражений; как при вводе, так и при выводе список A не может содержать формальных параметров, а также имен базированных и повторно определяемых переменных. Конструкция FILE (F) может быть опущена, тогда предполагается использование SYSIN для оператора GET и SYSPRINT для оператора PUT. Список ввода-вывода в операторах GET и PUT может отсутствовать.

При вводе значения могут быть присвоены только переменным, идентификаторы которых указаны в списке ввода, однако не обязательно всем перечисленным там переменным. На вводном устройстве данные представляются в виде элементов V=C, где V — переменная, C — константа, которые друг от друга отделяются запятой или пробелом. В конце списка ставится точка с запятой. Порядок следования элементов в списке на вводном устройстве может отличаться от порядка переменных в списке идентификаторов оператора GET. Например, оператор

```
GET DATA (Q,X,Y,Z);
```

может прочитать входные данные, набитые на перфокарте в виде

```
Y=12, X=34 Z=56;
```

в результате чего значения будут присвоены переменным X, Y, Z, а идентификатор Q не получит значения.

Если в операторе ввода отсутствует список ввода, то будут введены все элементы, указанные на вводном устройстве во входной по-

следовательности. Так, если на вводном устройстве имеются указанные выше входные данные, то они могут быть введены оператором

GET DATA;

который эквивалентен оператору

GET DATA (Y,X,Z);

Во входных данных могут использоваться переменные с индексами. Например, если на перфокарте набито

M(4,3)=8, M(1,2)=3, M(9,9)=5;

где M — идентификатор массива, описанного оператором

DECLARE M(10,10) FIXED;

то после выполнения любого из операторов

GET DATA (M); или **GET DATA;**

будут введены только значения элементов M(4,3), M(1,2), M(9,9). Такой способ используется, когда необходимо изменить лишь несколько значений в большом массиве. К оператору GET может быть добавлено слово COPY, которое обеспечивает выдачу входных данных на SYSPRINT.

Оператор PUT обеспечивает вывод значения каждой переменной как элемента в виде V=C. Элементы разделяются пробелом, в конце списка ставится точка с запятой. Так, если значения переменных X, Y, Z из предыдущего примера не изменились, оператор

PUT DATA (X,Y,Z);

осуществит их вывод на устройство SYSPRINT в виде следующей печатной строки:

X=34 Y=12 Z=56;

Вид выводимых значений определяется соответствующими описателями, константы в виде строк символов заключаются в кавычки, а после строки битов ставится буква B. Если в операторе вывода отсутствует список, то подразумевается такой список вывода, который содержит имена всех переменных и известен в той точке программы, где расположен данный оператор PUT.

Отметим, что ввод-вывод, управляемый данными, требует значительного объема оперативной памяти в рабочей программе.

3. Ввод-вывод, управляемый редактированием. Передача данных, при которой можно детально задавать форму поля данных в потоке (формат), осуществляется операторами ввода-вывода, управляемыми редактированием. Операторы ввода и вывода, управляемые редактированием, соответственно имеют вид

GET FILE (F) EDIT (A) (B);

PUT FILE (F) EDIT (A) (B);

где F — имя файла, A — список ввода или вывода, B — список форматов, EDIT — дополнение, специфицирующее редактирование. Если в записи операторов опущена конструкция FILE (F), то предполагается SYSIN для ввода и SYSPRINT для вывода. В операторе GET после списка форматов можно указать дополнение COPY, что вызовет печать входных данных на SYSPRINT. Список форматов состоит из элементов, являющихся спецификациями формата. Порядок размещения переменных в списке идентификаторов соответствует тому порядку, в котором располагаются вводимые значения.

Спецификации форматов подразделяются на спецификации формата данных и спецификации управления (управляющие форматы). Спецификации формата данных задают форму полей данных в потоке. Управляющие форматы определяют при выводе операции над страницами, строками и обеспечивают разрядку текста, а при вводе используются для пропуска части символов. Обычно соблюдается поэлементное соответствие между переменными списка ввода-вывода и спецификациями форматов списка форматов. Однако такое соответствие не является обязательным, более того, оно невозможно, если число элементов списка ввода-вывода неизвестно. В таком случае передача данных осуществляется по правилам а) если список ввода-вывода исчерпан, передача данных прекращается: б) если список форматов исчерпан, то спецификации повторяются от начала списка форматов до тех пор, пока не будут переданы все элементы списка ввода-вывода.

4. Спецификации формата данных. Спецификации формата данных описывают форму представления данных в потоке данных. Имеется шесть типов спецификаций формата: спецификация формата десятичного числа с фиксированной точкой, спецификация формата десятичного числа с плавающей точкой, спецификация формата комплексного числа, спецификация формата с шаблоном, спецификация формата строк символов, спецификация формата строк битов.

В общем случае спецификация формата данных может быть представлена в виде

$$D(w, d, s)$$

где D — индекс спецификации, w, d, s — выражения, принимающие целые значения, причем обычно предполагается, что $d \leq s \leq w$. Выражение w задает длину поля данных в символах, используемых при внешнем представлении (включая знаки, точки, пробелы и буквы E, B, которые используются при записи констант). Значение d определяет количество позиций после точки, а s указывает количество значащих цифр (спецификация формата десятичного числа с плавающей точкой) или является масштабным множителем (спецификация формата десятичного числа с фиксированной точкой). В записи спецификации формата индекс спецификации D является обязатель-

ным элементом, а в ряде спецификаций некоторые из компонентов w , d , s (или все) могут отсутствовать.

В качестве индекса спецификации формата данных используются следующие символы: F — для элемента списка ввода-вывода с фиксированной точкой, E — для элемента с плавающей точкой, C — для комплексного элемента, P — для элемента с шаблоном, A — для строк символов, B — для строк битов.

Спецификации формата данных с фиксированной точкой имеют вид $F(w)$, $F(w,d)$ и $F(w,d,s)$.

В случае спецификации $F(w)$ входное поле содержит w символов, соответствующих десятичному числу с фиксированной точкой. Если точка на перфокарте не набита, то предполагается, что число целое. Перед отрицательным числом должен стоять знак минус. При выводе данные воспринимаются как целые числа, при этом точка не изображается (не перфорируется и не печатается).

Для спецификации $F(w,d)$, если при вводе данных на входном поле точка не содержится, по умолчанию считается, что она стоит перед последней (считая справа) из d десятичных цифр в w -символьном поле. Если же точка отперфорирована, то заданное значение d во внимание не принимается. При выводе данных точка печатается или перфорируется всегда. В этом случае для записи цифр после точки отводится d десятичных разрядов, а для записи цифр до точки $w-d-1$ разрядов, если число положительное, или же $w-d-2$ разрядов, если число отрицательное.

В случае спецификации $F(w, d, s)$ при вводе данных считываемая величина перед преобразованием умножается на 10^s . При выводе число, хранимое в оперативной памяти, умножается на 10^{-s} . В этой спецификации s — масштабный множитель и на него ограничение $d \leq s \leq w$ не распространяется.

П р и м е р. В результате выполнения операторов

$A=1.234$; $B=-5678$; $C=90$;

$PUT\ EDIT\ (A,B,C)\ (F(6,3),F(8,3,3),F(5));$

будет напечатана строка

$\underbrace{\square 1.234 \square \square}_{A} \underbrace{-5.678 \square \square \square}_{B} \underbrace{90}_{C}$

Спецификации формата данных с плавающей точкой имеют вид $E(w,d)$ и $E(w,d,s)$

В случае спецификации $E(w,d)$ предполагается, что при вводе внешний носитель содержит число с плавающей точкой, расположенное в любом месте поля, имеющего длину w символов. Допустимы различные формы записи чисел в поле, однако требуется, чтобы перед показателем степени присутствовал по крайней мере один из символов E , $+$, $-$. Если точка не отперфорирована, количество раз-

рядов после точки определяется форматом. При выводе показатель степени всегда записывается с помощью четырех символов в форме $E \pm aa$, где a — десятичная цифра, и число печатается с d дробными десятичными разрядами. Число располагается справа в w -символьном поле. Если число положительное, то оставлять место для знака (слева) не требуется.

В случае спецификации $E(w,d,s)$ при вводе данных значение s во внимание не принимается. При выводе для записи числового значения выделяется $s-d$ позиций перед точкой и d позиций после точки. Знак минус предшествует самой левой цифре отрицательных чисел. Так, если в предыдущем примере написать оператор

PUT EDIT (A,B,C) (E(10,4),E(10,2),E(10,0,2));

то будет напечатана следующая строка:

1.2340E+00 — 5.67E+03 90.E+00
A
B
C

Отметим, что спецификация $E(w,d,d+1)$ эквивалентна $E(w,d)$.

Спецификация формата комплексного числа имеет вид $C(D_1,D_2)$, где D_1 и D_2 — любая разновидность спецификаций F, E или R. Используется эта спецификация для комплексных элементов списка ввода-вывода. Спецификация D_1 характеризует вещественную часть, D_2 — мнимую часть комплексной величины; D_2 может быть опущена, тогда обе части комплексного числа специфицируются одинаково.

Числовые данные могут быть описаны при помощи числового шаблона с использованием спецификации формата с шаблоном в виде R 'шаблон'. При вводе спецификация шаблоном описывает форму данных на внешнем носителе, а также способ их числовой интерпретации. Из вводного потока считывается определяемое шаблоном количество символов, и полученная строка символов присваивается соответствующей переменной. Если прочитанные символы не удовлетворяют указанному шаблону, возникает ситуация CONVERSION. При выводе значение элемента списка вывода преобразуется перед передачей к виду, задаваемому шаблоном. Двоничные числовые поля будут иметь после передачи символьное представление.

Пример. Ввод из потока в качестве значений переменных A, B, C трех наборов символов 123, ABC1 и 1.2 по спецификациям R'9V99', R'AAx9' и R'999' соответственно приведет к присвоению переменным A и B значений '123' и 'ABC1', при этом полученное числовое значение в первом случае равно 1.23; переменной C не будет присвоено никакое значение, а возникает ситуация CONVERSION. При выводе числового значения 12.3 по шаблону R'9V99' получаем выводимую строку в виде 230.

Спецификации формата строк символов имеют вид A(w) и A. В случае спецификации A(w) при вводе значение строки символов

длины w присваивается символьной переменной. Если переменная содержит больше чем w символов, то она справа дополняется соответствующим числом пробелов. Если переменная содержит меньше чем w символов, справа производится усечение вводимого поля. При выводе переменная располагается слева в поле, правая часть дополняется пробелами. Выводимая переменная должна иметь тип строки символов.

Спецификация формата A применяется только при выводе. Здесь значение w берется равным длине переменной, имеющей тип строки символов; при этом в списке идентификаторов может непосредственно содержаться строка символов, а не идентификатор переменной типа строки символов.

Спецификации формата строк битов имеют вид $B(w)$ и B . В случае спецификации $B(w)$ при вводе строки символов длиной w считывается и преобразуется во внутреннюю строку символов из нулей и единиц; при выводе переменная располагается слева в выводимом поле. Правая часть поля дополняется пробелами, если длина полученной строки меньше w .

Спецификация формата B используется только при выводе. Здесь значение w предполагается равным длине выводимой строки битов. Например, после выполнения операторов

```
DECLARE X CHARACTER (6), Y BIT (4);  
X='CONST='; Y='1001'B;  
PUT EDIT (X,Y,'_BINAR') (A(6),B,A);
```

будет выведена следующая печатная строка:

```
CONST=1001 BINAR
```

Отметим, что если в приведенных спецификациях $w \leq 0$, то при вводе соответствующие элементы списка ввода и списка форматов пропускаются, если только элемент в первом списке не является строкой (в этом случае значение будет равно пустой строке). При выводе элемент списка форматов пропускается, если $w \leq 0$.

Если в списке форматов несколько спецификаций имеют одинаковую структуру, то возможно применение коэффициента повторения в виде целой константы в скобках перед спецификацией. Так, список форматов $(F(8,3), F(8,3), F(8,3))$ может быть переписан как $((3)F(8,3))$.

5. Управляющие форматы. Управляющие форматы используются в тот момент, когда они встречаются при поиске очередной спецификации формата данных. Самой спецификации управления в списке ввода-вывода не соответствует никакой элемент. После того как список ввода-вывода исчерпан, оставшиеся управляющие форматы соответствующего списка форматов игнорируются. Спецификации управления подразделяются на формат для задания интервалов и форматы для печати.

Формат для задания интервалов имеет вид $X(w)$. При вводе формат X означает, что очередные w символов потока данных должны быть пропущены. При выводе в поток данных добавляется w пробелов. Ситуация $w < 0$ эквивалентна $w = 0$.

Спецификации управления для печати используются только в случае файлов с описателями STREAM и PRINT. Каждый такой файл состоит из страниц, а страница — из строк, расположенных друг под другом с некоторым интервалом. Первая строка на любой странице имеет номер один. Индексами форматов для печати являются дополнения: PAGE, SKIP, LINE, COLUMN.

Спецификация PAGE означает, что необходимо поток продвинуть до новой страницы (происходит прогон бумаги до первой строки следующей страницы).

Спецификация SKIP(w) означает, что необходимо страницу продвинуть на w строк, т. е. пропускается $w-1$ строк до печати следующей строки. Если при этом происходит выход за пределы страницы, то действие эквивалентно формату PAGE. Если $w=1$, то в формате это число может быть опущено.

Спецификация LINE (w) означает продвижение страницы к строке с порядковым номером w на этой странице. Совместное использование двух спецификаций в виде PAGE LINE (w) означает переход к строке с порядковым номером w на следующей странице.

Спецификация COLUMN (w) указывает, что некоторое количество пробелов должно быть вставлено в поток с тем, чтобы очередной символ имел порядковый номер w в текущей строке. Если в текущей строке уже записано не менее w символов, то эта строка считается заполненной; начинается новая строка, в начало которой вставляется $w-1$ пробелов, так что эта новая строка начинается с символа, имеющего порядковый номер w .

Во всех спецификациях управления для печати, если $w < 1$, принимается $w = 1$.

Пример. При выполнении оператора

```
PUT EDIT ((Y(I,K) DO K=1 TO 7), B(I)
DO I=1 TO 5) (SKIP,(7)E(14,6),E(24,6));
```

осуществится переход к следующей строке, после чего будет напечатано пять строк, каждая из которых содержит восемь величин.

Во всех случаях выдачи файлов на печать место первого символа в каждой строке не занимается выводимыми данными, а этот символ используется для управления движением бумаги перед печатью. Управляющими символами могут быть: 1 (указывает начало новой страницы), \sqcup (предписывает перевод строки); 0 (перевод двух строк), — (перевод трех строк), + (означает блокировку перевода). Не следует предполагать, что в начале выполнения программы бумага находится в начале страницы.

6. Оператор форматов. Иногда весь формат или часть формата используется в нескольких операторах ввода или вывода, управляемых редактированием. Можно избежать повторной записи спецификаций в списке форматов, если там предусмотреть косвенный (удаленный) формат вида $R(M)$, где R — индекс спецификации косвенного формата, M — константа типа метки или переменная типа метки, принимающая значение метки оператора *форматов*, записываемого в виде

M: FORMAT (B);

Здесь список форматов B состоит из спецификаций, которые заменяются в операторах GET или PUT косвенным форматом. В этом списке могут присутствовать другие косвенные форматы, но при этом форматы не должны ни непосредственно, ни через другие косвенные форматы ссылаться на первоначальный оператор форматов (т. е. рекурсии здесь не допускаются).

Косвенный формат и соответствующий ему оператор FORMAT должны быть внутренними для одного и того же блока. Если оператор GET (или PUT) является оператором некоторой реакции на прерывание, то он не может содержать косвенного формата. Количество спецификаций косвенных форматов в одном операторе GET или PUT произвольное, как и число обычных спецификаций форматов в формате R .

Оператор FORMAT относится к числу неисполняемых операторов. Это означает, что если управление будет передано оператору форматов, то он пропускается и управление получает следующий за ним оператор.

Пример использования косвенных форматов:

```
PUT EDIT (A,B,C,D,E) (R(KOC),F(5),R(KC));
KOC: FORMAT (F(6,3),F(7,3,3));
KC: FORMAT (F(6),F(10));
GET EDIT (X,Y) (R(KC));
P: FORMAT (SKIP(2),(2)F(6,3),F(5),SKIP(3));
PUT EDIT (K,L,M) (R(P));
```

В последнем из выписанных здесь операторов используется косвенный формат с меткой P , в соответствии с которым сначала пропускается одна строка, затем печатаются значения элементов K, L, M списка вывода. Поскольку список вывода исчерпан, элемент SKIP(3) списка форматов игнорируется.

7. Операторы внутренней пересылки данных. В операторах GET и PUT вместо употреблявшейся выше конструкции FILE (F) можно использовать конструкцию вида STRING (T), где STRING — дополнение, T — переменная типа строки символов. После такой замены вместо вводимой извне или выводимой в файл информации

используется содержащаяся в памяти строка символов с заданным именем, т. е. рассматриваемые операторы осуществляют внутреннюю пересылку данных. Оператор GET распределяет данные по отдельным переменным, оператор PUT объединяет данные в одну переменную.

Конструкция STRING (T) может служить для обозначения строк символов со всеми тремя рассмотренными способами ввода и вывода, однако чаще всего она используется в операторах ввода-вывода, управляемых редактированием.

Пример. Для объединения в одну карточку (одну строку символов) выходных данных о статье в журнале может быть использован следующий оператор:

```
PUT STRING (CARD) EDIT (NAME, TITLE,  
VOLUM, PAGE) (A(50), A(20), (2)F(5));
```

где CARD — наименование переменной, которая содержит все данные о статье, NAME — название статьи, TITLE — название журнала, VOLUM — номер тома, PAGE — номер страницы. Переменная CARD должна иметь описатель CHARACTER и длину по крайней мере 80.

Если переменная CARD, заданная в операторе DECLARE с описателем CHARACTER (80), имеет значение строки символов, представляющей собой выходные характеристики некоторой статьи, как приведено выше, то для того, чтобы разместить эти характеристики в различных участках памяти и обработать их, следует выполнить оператор

```
GET STRING (CARD) EDIT (NAME, TITLE,  
VOLUM, PAGE) (A(50),A(20),(2)F(5));
```

Упражнения

1. Указать, какие из следующих операторов ввода и вывода содержат ошибки:

```
PUT FILE (F) SKIP(1); PUT FILE (F) SKIP COPY;  
PUT PAGE FILE (F) SKIP (2);  
GET COPY SKIP FILE (F) LIST (X);  
GET SKIP 5 FILE (F);
```

2. Определить, что будет напечатано после выполнения операторов:

```
a) DECLARE F OUTPUT;  
OPEN FILE(F) LINESIZE (20);  
PUT FILE (F) LIST (1,2,3,4,5,6,7,8) SKIP;  
б) DECLARE E PRINT, I FIXED (2);  
OPEN FILE (E) LINESIZE (15);  
PUT LIST ('BEGIN') FILE (E) SKIP;  
DO I=2 TO-2 BY-1; PUT LIST (I||'END');  
FILE (E) SKIP (I); END;
```

- в) PUT EDIT ('XYZ','101'B) (A(2),B(2));
- г) PUT EDIT ('ABC','111'B) (A,B);
- д) DOI=1 TO 6; PUT EDIT ('A') (X(I),A); END;
- е) PUT, EDIT (A,B,C) (P*/99',P'Z/99');

(в случае е) значения переменных А, В, С равны соответственно 123, 45 и 6).

3. Составить программу, которая вводит значения элементов определителя $D = ||d_{ij}||$ ($i, j = 1, 2, 3$), вычисляет значение определителя и печатает это значение.

4. Составить такой оператор ввода, управляемого редактированием, после выполнения которого переменным А, В и С будут присвоены значения, соответственно равные $1.23E+01$, $4.56E-01$ и $1.00E+02$, при условии, что входной поток имеет вид:

- а) $+123E+00 \square\square 45.6E-02 \square 100E+01 \square$;
- б) $12300 \square\square +4.56E-11000E+02$;
- в) $123E-0145.6E-2100 \square\square\square$;
- г) $\square 12.3E+00 \square\square 0.456 \square\square 100E+02 \square$.

§ 18. Операторы ввода-вывода записей

При вводе или выводе записей обрабатываемая информация понимается как дискретные записи, формат которых совпадает с форматом внутренних данных для языка. Назначение ввода заключается в том, чтобы скопировать запись, передав ее с устройства ввода в оперативную память. Назначение вывода — копирование записи с передачей ее из оперативной памяти на устройство вывода. Преобразование при этом не предусматривается.

В качестве переменных, участвующих в операторах ввода и вывода записей, допускаются простые переменные, массивы и старшие структуры. Не разрешены подструктуры и отдельные элементы массивов и структур, формальные параметры процедур. Переменные могут быть также базированными, иметь описатели VARYING или EXTERNAL. Ввод-вывод записей осуществляется в машинном коде, поэтому он удобен для файлов, хранимых на лентах и дисках. Это более быстрая операция, чем ввод-вывод потока.

1. Операторы ввода-вывода последовательного доступа. Для ввода записей с последовательно и индексно-последовательно организованных файлов с последовательным доступом как с описателем INPUT, так и с описателем UPDATE (в режиме обновления, или исправления) используются операторы

```

READ FILE (F) INTO X;
READ FILE (F) INTO X KEYTO (T);
READ FILE (F) IGNORE (E);
READ FILE (F) INTO X KEY (ET);

```

где F — имя файла, с которого осуществляется ввод, X — переменная, массив или структура, T — переменная типа строки символов,

Е, ЕТ — скалярные выражения, INTO, KEYTO и IGNORE — дополнения.

Первый из перечисленных операторов передает данные из файла в рабочий участок памяти, идентификатор которого указан после дополнения INTO. Записи читаются последовательно одна за другой с начала файла. Оператор READ точно в такой же форме используется при последовательном чтении без ключей при вводе из файлов с описателем INDEXED.

Для файлов с описателем KEYED в операторе READ может быть указано дополнение KEYTO (Т), где переменной Т присваивается значение ключа, хранящегося в файле перед записью. Второй из приведенных операторов отвечает этому случаю.

Третий из приведенных операторов обеспечивает в случае положительного значения Е (дробная часть значения отбрасывается) пропуск соответствующего количества (Е) записей из указанного файла. Иначе говоря, следующий оператор ввода при обращении к этому же файлу получит доступ к (Е+1)-й записи. Пустые записи в счет не идут. При отрицательных значениях Е оператор игнорируется. Операторы

```
READ FILE (F) IGNORE (I);  
READ FILE (F);
```

эквивалентны, т. е. в обоих вариантах пропускается одна запись.

Четвертый (последний) из перечисленных операторов позволяет последовательное чтение записей установить с любого места в файле с описателем KEYED (начиная с заданного ключа). При выполнении этого оператора запись из файла F с ключом, заданным выражением ЕТ в дополнении KEY (значение ЕТ преобразуется в строку символов), вводится в память, заданную в виде переменной после дополнения INTO. Начало последовательного ввода для следующих операторов READ устанавливается на запись, непосредственно следующую за записью, прочитанной данным оператором.

Для вывода записей в последовательно организованный файл с последовательным доступом с описателем OUTPUT используется оператор

```
WRITE FILE (F) FROM (X);
```

где дополнение FROM (X) указывает, что значение переменной X необходимо записать в файл с именем F. Каждое выполнение этого оператора выводит очередную подготовленную запись из переменной X.

В случае индексно-последовательных файлов вывод осуществляется оператором

```
WRITE FILE (F) FROM (X) KEYFROM (ET);
```

Каждое выполнение этого оператора выводит очередную запись из переменной X, указанной в дополнении FROM, причем ключ записи задается в дополнении KEYFROM выражением ET, преобразуемым в строку символов. Записи располагаются последовательно с начала файла F.

Исправления записей файлов с описателем UPDATE для любых файлов с последовательным доступом выполняются оператором вывода в виде

REWRITE FILE (F) FROM (X);

При выполнении этого оператора на место прочитанной перед этим записи засылается содержимое переменной X, т. е. последняя прочитанная запись исправляется (обновляется).

Такой же вид имеет оператор вывода из буфера, выполняемый в режиме обновления и используемый для замены существующей записи. Конструкция FROM (X) здесь может быть опущена; в файл заносится последняя прочитанная в буфер запись.

Для файлов, допускающих пустые записи подобно оператору REWRITE, можно использовать оператор *исключения записи*

DELETE FILE (F);

который последнюю прочитанную запись в файле превращает в пустую. Этот оператор применим только в том случае, если файл имеет индексно-последовательную организацию.

При обработке записей в буфере используется оператор

READ FILE (F) SET (P) KEYTO (T);

где P — переменная типа указателя, T — переменная типа строки символов. Действие этого оператора заключается в том, что очередная запись из файла F переносится в буфер, а указатель P, заданный дополнением SET, принимает значение, указывающее на начало данных этой записи. Это значение сохраняется до тех пор, пока в другом операторе READ следующее дополнение SET не устанавливает тот же самый указатель. Для файлов с описателем KEYED использование дополнения KEYTO (T) аналогично описанному выше.

Оператор *размещения* в буфере имеет вид

LOCATE X FILE (F) FROM (X);

где X — базированная переменная (массив, структура), обеспечивающая совмещение данных X с областью, определяемой указателем P в файле F. Выполнение оператора LOCATE не сопровождается выводом; вывод данных производится немедленно перед следующим оператором LOCATE или WRITE, в котором встречается обращение к тому же самому файлу.

Оператор размещения в буфере для файлов с описателем KEYED имеет вид

LOCATE X FILE (F) SET (P) KEYFROM (ET);

где смысл указателя P и выражения ET описан выше.

2. Операторы ввода-вывода прямого доступа. Особенностью обработки файлов с прямым доступом является то, что программист сам определяет структуру размещения записей (на дисках) и до выполнения программы с помощью специальных директив (не являющихся элементами языка ПЛ/1) обеспечивает подготовку необходимого количества участков для хранения записей.

Для ввода записей с индексно-последовательно организованных файлов с прямым доступом как с описателем INPUT, так и с описателем UPDATE используются операторы

READ FILE (F) INTO (X) KEY (ET);

READ FILE (F) SET (P) KEY (ET);

где KEY (ET) — дополнение, в котором ET — скалярное выражение. При выполнении этих операторов в конструкции KEY (ET) вычисляется значение выражения ET, которое, будучи представленным целым числом без знака и преобразованным в строку символов, определяет, какая запись вводится. Если записи с указанным ключом в файле не окажется, возникает ситуация KEY.

Второй из приведенных операторов используется в случае файлов с описателем BUFFERED для размещения данных в буфере.

Для вывода записей используется оператор

WRITE FILE (F) FROM (X) KEYFROM (ET);

где дополнение KEYFROM (ET) указывает, что целое десятичное число без знака, являющееся значением выражения ET, преобразуется в строку символов и представляется в виде ключа записи, который переносится в файл. Длина K строки символов, изображающей ключ, задается описателем файла KEYED (K) в операторе DECLARE. При выполнении этого оператора запись выводится из переменной, указанной в дополнении FROM, и помещается в файл в соответствующем (согласованном с требованием возрастания ключей) месте. Если в файле уже окажется запись с таким же ключом, то возникнет ситуация KEY. Ситуация KEY возникает также и в том случае, если для выводимой записи не хватает в файле места.

Оператор, исключаящий из файла F запись с ключом ET, прочитанную из переменной X, имеет вид

DELETE FILE (F) FROM (X) KEYFROM (ET);

где конструкция FROM (X) может отсутствовать. Оператор исключения записи применим только к файлам с описателем UPDATE.

Если запись, подлежащей исключению, в файле не окажется, возникает ситуация KEY.

Для замены (обновления) содержания существующей в файле записи используется оператор

REWRITE FILE (F) FROM (X) KEY (ET);

Перед выполнением этого оператора данная запись должна быть прочитана оператором READ. Поэтому в выполняющихся по порядку операторах READ и REWRITE в дополнении KEY должно быть указано одно и то же значение ключа. В дополнении FROM указывается переменная, содержащая возвращаемую запись.

Операторы ввода и вывода для файлов с региональной организацией имеют точно такой же вид, как и только что рассмотренные операторы для файлов с описателем INDEXED. Оператор WRITE обеспечивает создание файла в режиме прямого доступа, чем достигается рациональное использование внешней памяти. Оператор READ обеспечивает доступ к любой заданной записи в файле, не прибегая к обработке других записей.

Упражнения

1. Указать, в каких операторах содержатся ошибки:

```
READ FILE (F) INTO (X); DECLARE X (10,10) FIXED;  
READ FILE (F) INTO (X) IGNORE (5);  
READ FILE (F) INTO (X (*,3));  
WRITE FROM (X); DECLARE X(10);  
WRITE FILE (F) FROM (X); DECLARE X(5) CHARACTER  
(7);  
WRITE FILE (F) FROM (X); DECLARE (Y DEFINED X,  
X) CHARACTER (80);  
REWRITE FILE (F);  
REWRITE FILE (F) FROM (X(16));  
REWRITE FILE (F) FROM (X); DECLARE X  
CHARACTER (20) VARYING;  
LOCATE X FILE (F); DECLARE X BASED (P);  
LOCATE X SET (P); DECLARE X BASED (Q);  
DECLARE F UPDATE; LOCATE A FILE (F).
```

2. Определить, из каких записей будет состоять набор данных, сопоставленный файлу F, после выполнения программы

```
T: PROCEDURE OPTIONS (MAIN);  
  DECLARE F ENVIRONMENT (F(10)),  
  A PICTURE 'S(9) 9' BASED (P);  
  DO I=1 TO 3; LOCATE A FILE (F);  
  A = I; END; CLOSE FILE (F);  
END T;
```

3. Определить, какие значения будут иметь переменные X, Y и Z после выполнения процедуры и указать, в каких случаях будет возникать ситуация RECORD:

```
P: PROCEDURE;
  DECLARE F ENVIRONMENT (F(10)),
  X CHARACTER (5), (Y,Z VARYING) CHARACTER (15);
  ON RECORD (F); OPEN FILE (F) OUTPUT;
  X='BEGIN'; WRITE FILE (F) FROM (X);
  Y=(2) 'OUTPUT'; WRITE FILE (F) FROM (Y);
  Z=SUBSTR (Y,2,10); WRITE FILE (F) FROM (Z);
  CLOSE FILE (F); OPEN FILE (F);
  READ FILE (F) INTO (X); READ FILE (F) INTO (Y);
  READ FILE (F) INTO (Z); END P;
```

§ 19. Пример программы

Программа для вычисления определенного интеграла методом Симпсона базируется на формуле

$$\int_a^b f(x) dx = \frac{h}{3} [f(a) + 4f(a+h) + 2f(a+2h) + 4f(a+3h) + \dots + 2f(b-2h) + 4f(b-h) + f(b)],$$

где $h=(b-a)/n$, n — четное количество интервалов, $f(x)$ — подынтегральное выражение.

Пусть задан интеграл

$$Q(p) = \int_a^b \frac{p^v x e^x dx}{(p + e^x)^{v+1}}$$

как функция параметра p . Заданы интервал изменения p и константа v . Вычисление подынтегрального выражения может быть осуществлено с помощью процедуры-функции вида

```
FX: PROCEDURE (X);
  F=X*EXP(X)/(P+EXP(X))**(V+1);
  RETURN(F); END;
```

где FX — метка, X — формальный параметр, F — идентификатор результата. Значения этой функции, которые при вычислении интеграла $Q(p)$ умножаются на множитель 4, получаются обращением к ней с фактическими параметрами, равными $A+H$, $(A+H)+2H$, ..., а те значения подынтегрального выражения, которые умножаются на множитель 2, получаются обращением к процедуре-функции с фактическими параметрами $(A+H)+H$, $((A+H)+H)+H$, ...

Суммирование указанных двух групп значений подынтегральной функции может быть выполнено в цикле. Во внешнем цикле

вычисляется интеграл Q в интервале значений параметра P от P_0 до P_K .

Программа имеет вид

```
SIMPSON:PROCEDURE OPTIONS (MAIN);
  DECLARE (A,B) DECIMAL FIXED (6,3),
  (V,P) DECIMAL FLOAT (8) EXTERNAL,
  (Q,X, PO, PH, PK) DECIMAL FLOAT (8),
  N DECIMAL FIXED (5,0);
  GET EDIT (PO, PH, PK, A, B, V, N)
  ((3)E (12,5), (3) F (6,3), F(5)); H=(B-A)/N;
  DO P=PO TO PK BY PH; DO; S2; S4=0;
  DO X=A+H TO B-3*H BY 2*H; DO;
  S2=S2+FX (X+H); S4=S4+FX (X); END;
  END; END;
  Q=P**V*H /3*(FX (A)+FX (B)+
  4 * FX (B-H)+4*S4+2*S2);
  PUT EDIT (Q, P) ((2) E (13,5)); END;
STOP; END SIMPSON;
FX: PROCEDURE (X);
  F=X * EXP (X)/ (P+EXP (X)) ** (V+1);
  RETURN (F);
END FX;
```

В этой программе переменные V , P описаны как внешние имена, поэтому их значения доступны при вычислении F , когда выполняется обращение к процедуре-функции с меткой FX .

Глава V. КРАТКИЕ СВЕДЕНИЯ О НЕКОТОРЫХ ВХОДНЫХ ЯЗЫКАХ

§ 1. Структура и функции операционной системы

Математическим (программным) обеспечением или *системой математического обеспечения* ЭВМ называют комплекс программных средств, предназначенных для повышения эффективности использования машины, облегчения ее эксплуатации и снижения трудоемкости подготовительной работы при решении задач на машине. Система математического обеспечения по своей структуре обычно может быть разделена на общее и специальное. Общее математическое обеспечение представляет собой комплекс программ, обеспечивающих применение ЭВМ как некоторой универсальной системы обработки информации, в то время как специальное математическое обеспечение образует программы, которые добавляются к общему математическому обеспечению и решают задачу применения ЭВМ как некоторой специализированной системы обработки информации.

Операционная система, набор пакетов прикладных программ, комплекс программ технического обслуживания и система документации являются составными частями общего математического обеспечения. Здесь будут даны только самые общие сведения о структуре и функциях лишь одной составной части математического обеспечения — *операционной системы*, которые необходимы при рассмотрении в последующих параграфах этой главы различных входных языков и вопросов подготовки задания для ЭВМ с использованием трансляторов с этих языков. Отметим лишь, что прикладные программы, входящие в общее математическое обеспечение, позволяют расширить возможности операционной системы для специальных комплексов технических средств и способов их применения, а также составляют комплексы программ для решения типовых задач.

В простейшем варианте математическое обеспечение ЭВМ сводится к программным средствам, представляющим собой комплекс из транслятора с одного из алгоритмических языков, библиотеки стандартных программ, программ отладки и, возможно, транслятора с автокода. В этом случае обычно говорят не об операцион-

ной системе ЭВМ, а о *системе программирования* на данном языке. Например, это система МикроДОС, мониторная система «Дубна», система Бейсик-плюс, UNIX и др.

Под операционной системой или монитором понимают набор программ, которые организуют непрерывную работу машины в различных режимах без вмешательства оператора. Режим работы ЭВМ — это способ организации решения задачи (задач). Основными являются следующие режимы: однопрограммный, режим пакетной обработки, режим разделения времени, режим запрос-ответ, комбинированные режимы. Все эти режимы, кроме первого, относятся к мультипрограммным (многопрограммным) режимам. В однопрограммном режиме работы ЭВМ все устройства машины заняты выполнением только одной программы. В режиме пакетной обработки информации одновременно решается несколько задач, для которых вся необходимая информация (программы, исходные данные) вводится в память машины заранее, до начала решения задач, а в ходе решения вмешательство абонента (пользователя) с пульта не допускается. В режиме разделения времени некоторое число независимых абонентов с помощью периферийных устройств ввода и вывода имеет в процессе решения своих задач непосредственный постоянный и одновременный доступ к ЭВМ. Режим запрос-ответ — это способ организации решения задач, при котором их программы постоянно хранятся в запоминающем устройстве машины, а запросы на их выполнение и необходимые исходные данные поступают извне от абонентов, имеющих прямой доступ к машине. Комбинированный режим работы ЭВМ может быть организован, например, сочетанием режима запрос-ответ, или режима разделения времени, с режимом пакетной обработки, или однопрограммным режимом.

В дальнейшем, при рассмотрении входных языков в различных операционных системах, будут использованы понятия и определения, общие для всех операционных систем. Поэтому остановимся на них здесь подробнее.

Основная независимая единица работы ЭВМ, формируемая извне программистом (абонентом, пользователем), называется *заданием*. Каждое задание обычно описывается с помощью определенных управляющих операторов операционной системы. Описание задания определяет, например, имя задания, начало и конец задания, название программы, которая должна быть выполнена. Задания могут быть представлены либо в виде одиночного задания, либо в виде пакета, образующего входной поток заданий. Задания не зависят друг от друга и могут выполняться одновременно.

По усмотрению программиста задание может быть разбито на *шаги* задания. Шаги задания для одного задания выполняются последовательно, причем выполнение очередного шага зависит

от того, насколько успешно реализованы предыдущие шаги. Например, возможны шаги задания: трансляция, редактирование, выполнение программы. Простейшее задание состоит из одного шага.

Совокупность процедур, включающая прием заданий, их контроль, подготовку необходимых программ к выполнению, запуск этих программ и автоматический переход к новому, очередному, заданию, представляет собой *управление* заданиями. Осуществляется управление заданиями управляющей программой операционной системы.

Работа, которая должна быть выполнена программой, указанной в шаге задания, представляет собой *задачу*. Задача может решаться в однопрограммном или мультипрограммном режимах. В зависимости от этого операционная система выделяет для решения задачи необходимые *ресурсы* — время процессора, определенную область оперативной и внешней памяти, необходимые внешние устройства. В однопрограммном режиме все ресурсы находятся в распоряжении одной задачи, в мультипрограммном режиме их распределяет операционная система.

Часть области оперативной памяти, предоставляемой рабочим программам абонента, называют *разделом*. Размеры разделов и их число не являются строго фиксированными и обычно имеют определенные названия.

Совокупность единиц информации образует *запись* (логическую запись), а совокупность записей, объединенных по некоторому общему признаку, образует *файл*. Файлы данных размещаются во внешней памяти. Форматы записей, организация их в физические блоки, способы хранения и поиска отдельных записей определяются свойствами операционной системы. Обработка записей, их поиск, хранение и передача осуществляются программами операционной системы.

Емкость стандартного блока внешней памяти измеряется в *томах*. Например, том — это емкость одной магнитной ленты или одного пакета дисков. Том, содержащий несколько файлов информации, называется многофайловым томом, а файл, располагающийся в нескольких томах, является многотомным файлом. Характеристики файла (или тома), позволяющие операционной системе его опознать, составляют *метку* файла (тома).

В операционной системе обычно устанавливается стандартный набор символических имен, принятых для обозначения логических устройств, которые программист использует в своей исходной программе во всех случаях, когда необходимо обратиться к внешнему устройству. Следовательно, в программе указывается логическое устройство, а не конкретный физический адрес требуемого внешнего устройства. Такой метод, обеспечивающий неза-

висимость программ операционной системы и исходных программ абонентов от конкретных физических адресов внешних устройств, принято называть *методом логических устройств*. Логические устройства, которые используются операционной системой для собственных надобностей, называются *системными* логическими устройствами. Для них символические имена зафиксированы постоянно, и каждому из них ставится в соответствие конкретный тип физических внешних устройств.

Программа, составленная на одном из алгоритмических языков, допускаемых данной операционной системой (*исходная программа* или *исходный модуль*), транслируется в так называемый *объектный модуль*. Объектный модуль — это некоторый промежуточный вид записи программы. В объектном модуле специфика исходного языка программирования теряется, однако это еще не рабочая (машинная) программа. Для выполнения на машине объектный модуль должен пройти этап редактирования, после которого получается *абсолютный модуль* (программная фаза). Абсолютный модуль представляется на машинном языке и не подлежит дроблению при вызове его в оперативную память для выполнения. Все модули, как правило, хранятся в соответствующей библиотеке — библиотеке исходных модулей, библиотеке объектных модулей или библиотеке абсолютных модулей. В библиотеке абсолютных модулей хранятся в виде программных фаз также программы самой операционной системы (*системные программы*), поэтому библиотека абсолютных модулей в операционной системе является обязательной (две другие библиотеки могут отсутствовать). В операционной системе могут существовать как системные, так и личные (принадлежащие конкретным абонентам) библиотеки объектных и исходных модулей.

Поскольку перечень и объем операций, выполняемых операционной системой, различен для различных типов операционных систем, то по структуре они могут отличаться друг от друга, однако основные компоненты операционных систем, как правило, одни и те же. Это — управляющая программа и обрабатывающие программы.

Основные функции операционной системы реализуются с помощью *управляющей программы*. Именно она обеспечивает автоматическое выполнение как обрабатывающих программ самой операционной системы, так и программ абонентов в различных режимах работы в рамках одного задания или в пределах различных заданий. Управляющая программа операционной системы выполняет следующие три группы действий: управление данными, управление заданиями и управление задачами.

Управление данными осуществляется комплексом программ (в дисковой операционной системе ЕС ЭВМ — ДОС ЕС — про-

граммой, называемой **УПРАВЛЕНИЕ ДАННЫМИ**), предназначенных для управления процессами организации, идентификации, хранения и выборки всех данных, обрабатываемых в системе.

Комплекс программ, предназначенных для управления процессом прохождения через вычислительную машину непрерывного потока заданий, осуществляет управление заданиями. Например, в ДОС ЕС программа **УПРАВЛЕНИЕ ЗАДАНИЯМИ** обеспечивает подготовку операционной системы для выполнения пакета заданий.

Управление задачами — это комплекс программ, предназначенных для управления процессом выполнения задач при различных режимах работы, в том числе для управления процессом одновременного выполнения нескольких задач по обработке данных. Этот комплекс программ выполняет функции распределения оперативной памяти, загрузки программ в оперативную память, управления выполнением задач, переключения управления от одной задачи к другой и др. В число программ этой группы входят, например, программы **ЗАГРУЗЧИК** и **СУПЕРВИЗОР**. Первая из них (называемая в ДОС ЕС **ПЕРВОНАЧАЛЬНАЯ ЗАГРУЗКА**) обеспечивает подготовку операционной системы к работе и загрузку необходимых программ в оперативную память. Среди них будет также программа **СУПЕРВИЗОР** (или **ДИСПЕТЧЕР**), осуществляющая управление всем вычислительным процессом на машине.

В состав обрабатывающих программ входят трансляторы, системные обслуживающие (сервисные) программы, а также обрабатывающие программы, написанные абонентами. Обрабатывающие программы предназначены для сокращения объема работы и времени, затрачиваемых на написание, подготовку и выполнение программ абонентов.

Используя языки программирования данной операционной системы, программист получает доступ к ее средствам. Результатом работы трансляторов являются объектные модули. Объектные модули объединяются сервисной программой **РЕДАКТОР** для получения программ, готовых к загрузке в оперативную память. **РЕДАКТОР** позволяет вносить изменения в программу без повторной трансляции всей программы, а также разделять программу на части (сегменты), если в этом есть необходимость (например, вследствие недостаточной емкости оперативной памяти).

Среди сервисных программ могут содержаться программа **БИБЛИОТЕКАРЬ**, которая выполняет функции корректировки и обслуживания библиотек операционной системы, программа **СОРТИРОВКА**, осуществляющая сортировку или объединение файлов, а также программы **УТИЛИТЫ** — набор системных программ, обеспечивающих выполнение часто повторяющихся вспомогательных операций (передачу информации из одного за-

поминающего устройства или устройства ввода (вывода) в другое, обновление программ и данных в библиотеке системы и др.). Имеются также различные *отладочные* программы, назначение которых состоит в оказании помощи программисту при отладке программ. Эти отладочные программы задают так называемый *отладочный режим* выполнения программы.

В целом операционная система ЭВМ представляет собой программно-аппаратный комплекс средств организации и управления процессом функционирования машины. В аппаратную часть операционной системы входят переключатели режимов работы машины и операционной системы, схемы выработки сигналов прерываний работы ЭВМ, регистры прерываний и др. В общем случае чем больше удельный вес аппаратной части, тем сложнее ЭВМ по своей конструкции, но операционная система работает быстрее и наоборот. Программная часть операционной системы, представляющая собой иерархический набор управляющих программ, определяет стратегию и тактику прохождения задач через ЭВМ. Анализ характеристик задач и имеющихся ресурсов определяет стратегию управления, в то время как тактика в основном определяется текущим состоянием машины, количеством задач в машине, временем на их обслуживание и т. д.

Предметом рассмотрения в последующих параграфах этой главы являются главным образом входные языки для различных операционных систем и классов ЭВМ. При этом основное внимание уделяется отличиям (как ограничениям, так и расширениям) входного языка от стандарта соответствующего алгоритмического языка. Вопросы, связанные с оформлением задания для конкретной операционной системы, рассматриваются в минимальном объеме. Поэтому программисту, прежде чем приступить к решению своей задачи на конкретной ЭВМ, необходимо ознакомиться с такими вопросами, как:

- наличие в операционных системах трансляторов с данного языка программирования;
- особенности входного языка выбранного транслятора;
- какие управляющие операторы необходимы для формирования задания;
- правила формирования задания.

В этой главе дается краткая характеристика лишь некоторых входных языков и трансляторов для отечественных вычислительных машин, в настоящее время находящихся в эксплуатации.

§ 2. Фортран IV ОС ЕС ЭВМ

В операционную систему ОС ЕС ЭВМ входит система программирования на Фортране, которая состоит из языка программирования Фортран IV, библиотеки программ и трансляторов. Язык

Фортран IV ОС ЕС ЭВМ включает в себя в виде правильного подмножества язык ASN-Фортран, подмножеством которого является в свою очередь язык Basic FORTRAN.

В систему программирования на Фортране входят трансляторы: отладочный транслятор ФОРТРАН(G) и оптимизирующий транслятор ФОРТРАН(H). Минимальный объем оперативной памяти, необходимый для работы первого из этих трансляторов, составляет 128, второго — 256 Кбайт. Транслятор ФОРТРАН(G) имеет высокую скорость трансляции, а транслятор ФОРТРАН(H) — более качественные объектные программы.

Язык обоих трансляторов содержит, с одной стороны, общие элементы расширения языка Фортран IV, ставшие уже традиционными для современных реализаций и полностью приводимые в описании языка Фортран IV в данной книге, с другой — средства совместимости с ранними версиями языка, в частности с Фортраном II. Для совместимости с языком Фортран II в Фортране IV ОС ЕС ЭВМ используется ряд системных программ библиотеки Фортрана IV, вызов которых осуществляется оператором CALL.

Транслятор ФОРТРАН(H) допускает указание повышенной точности во всех описательных операторах, в которых разрешено использование указателя числа байтов при описании величин. Повышенная точность задается конструкциями

REAL*16 COMPLEX*32

т. е. действительные величины объявляются длиной 16 байт, а комплексные — 32 байта. Ввод и вывод данных с повышенной точностью производятся по спецификации Q.

В языке Фортран ОС ЕС ЭВМ допускается переменный формат, описанный в п. 4 § 3 этой главы.

Большой набор системных программ операционной системы позволяет эффективно контролировать выполнение программы на всех этапах. В частности, возможен опрос машинных индикаторов, отражающих состояние ЭВМ и устанавливающихся при возникновении ситуаций, которые приводят к так называемому программному сбою машины (например, деление на ноль, переполнение или исчезновение порядка и др.). Возможна также установка программных индикаторов, используемых обычно абонентами в качестве переключателей.

Каждый индикатор может быть в состоянии «включен» или «выключен» (в зависимости от содержимого этого индикатора). Если индикатор содержит ноль, то он находится в состоянии «выключен». Если содержимое индикатора отлично от нуля, то он находится в состоянии «включен».

Для установки индикаторов используется программа

SLITE (K)

где K — выражение целого типа. С помощью этой программы может быть включен один из четырех индикаторов (если K принимает значения 1, 2, 3 или 4) или же все четыре индикатора выключаются (при $K=0$).

Проверка и запоминание индикаторов выполняются программой

SLITET (K, J)

где K принимает значения 1, 2, 3, 4 и указывает, какой индикатор проверяется, J — переменная целого типа. В результате выполнения этой программы переменная J получает значение 1 или 2 в зависимости от того, включен или выключен индикатор.

Распечатка данных из оперативной памяти (так называемый сброс памяти) осуществляется программами

DUMP ($B1, E1, K1, B2, E2, K2, \dots$)

PDUMP ($B1, E1, K1, B2, E2, K2, \dots$)

из которых первая прекращает выполнение программы после распечатки, а вторая обеспечивает продолжение выполнения программы после распечатки.

Здесь BN и EN задают начало и конец N -й области оперативной памяти, подлежащей сбросу по формату, задаваемому целой константой KN ; BN, EN — идентификаторы переменных как с индексами, так и без них, $N=1, 2, \dots$

Установлено следующее соответствие между константами KN и форматами:

0 — шестнадцатеричный	6 — REAL*8
1 — LOGICAL*1	7 — COMPLEX*8
2 — LOGICAL*4	8 — COMPLEX*16
3 — INTEGER*2	9 — текстовый
4 — INTEGER*4	10 — REAL*16
5 — REAL*4	11 — COMPLEX*32

Системная программа

DVCHK (K)

выполняет включение индикатора контроля деления и фиксирует состояние деления на нуль, устанавливая K равным 2, если деление на нуль имеет место, и равным 1 в противном случае.

Программа для проверки переполнения и исчезновения порядка записывается в виде

OVERFL (K)

Если произошло переполнение порядка, т. е. превышение значения, допустимого для порядка, устанавливается K , равное 1. Если будет зафиксировано состояние исчезновения порядка, т. е. значение порядка стало меньше допустимого значения, K стано-

вится равным 3. В обоих этих случаях индикатор включается. В отсутствие перечисленных состояний К устанавливается равным 2. После обращения к программе индикатор выключается.

Выполнение программы прекращается после обращения к системной программе

EXIT

в записи которой нет параметров.

При использовании транслятора ФОРТРАН(G) в программу могут быть включены операторы отладки DEBUG, AT, TRACE ON, TRACE OFF.

Оператор DEBUG устанавливает режим отладочных операций, относящийся к одной программной единице. Этот оператор предшествует операторам, составляющим вместе с исходной программой пакет отладки, и имеет вид

DEBUG R1, R2, ...

где R1, R2, ... — параметры, определяющие режим отладки. Записываются параметры R1, R2, ... в любом порядке и представляются в одной из следующих форм:

UNIT (N1)	INIT(V1,V2,...)
TRACE	SUBCHK(A1,A2,...)
SUBTRACE	

Режим отладки, задаваемый параметром UNIT (N1), где N1 — номер файла, указывает, что все результаты отладки выводятся в файл с номером N1. Если этот режим не указан, результаты отладки выводятся на системное устройство печати. Файл, в который выводятся результаты отладки, называется файлом отладки.

Режим отладки, называемый режимом трассирования меток, задается параметром TRACE и используется для прокрутки программы, т. е. вывода в файл отладки меток выполняемых операторов участка программы, заданного в пакете отладки. Если режим TRACE не указан, прокрутка не выполняется.

Параметр, записанный в виде SUBTRACE, осуществляет заказ на трассирование имени данной программной единицы.

Режим, задаваемый параметром INIT (V1, V2, ...), где V1, V2, ... — наименования переменных или массивов, действует на всю программную единицу и осуществляет вывод в файл отладки значения любого элемента из списка V1, V2, ... каждый раз, когда этот элемент получает значение.

Режим проверки правильного использования индексов указывается параметром SUBCHK (A1, A2, ...), где A1, A2, ... — наименования массивов. Если индекс какого-либо элемента любого из массивов A1, A2, ... превышает допустимое для данного массива значение, в файл отладки выдается сообщение с указанием соответствующего элемента.

Оператор

AT M

где M — метка оператора, определяет начало того участка программной единицы, где должны выполняться операторы отладки.

Операторы

TRACE ON и TRACE OFF

осуществляют соответственно включение и отключение трассирования меток (прокрутку программы), если в операторе DEBUG был указан режим TRACE.

Оптимизирующий транслятор ФОРТРАН(H) предоставляет дополнительные гибкие средства организации ввода и вывода с помощью операторов READ и WRITE, называемых операторами асинхронного ввода и вывода, и оператора WAIT — оператора асинхронного ожидания.

Асинхронный ввод выполняется оператором

READ (N1, I=K) A

где N1 — номер файла, I — целая переменная, K — выражение целого типа, значение которого однозначно идентифицирует данный оператор, A — список наименований массивов.

Асинхронный вывод осуществляется оператором

WRITE (N1, I=K) A

где смысл обозначений N1, I, K, A тот же, что и в предыдущем операторе.

Асинхронное ожидание вызывается оператором

WAIT (N1, I=K, COND=J, NUM=N) A

где N1 — номер файла, I — целая переменная, K — выражение целого типа, идентифицирующее своим значением один из операторов асинхронного ввода или вывода (не обязательно принадлежащий той же программной единице, что и оператор WAIT). По оператору WAIT производится приостановка выполнения программы вплоть до момента завершения ввода или вывода соответствующими операторами READ или WRITE. Оператор WAIT всегда относится к тому оператору READ или WRITE, который еще не завершен другим оператором WAIT, имеет тот же идентифицирующий номер и относится к тому же самому файлу.

В операторе WAIT параметр J — идентификатор целой переменной, значение которой после выполнения оператора устанавливается равным 1 при успешном завершении оператора ввода или вывода, к которому данный оператор WAIT относится, равным 2 при возникновении состояния ошибки при передаче информации и равным 3 при обращении к записи END FILE; N — идентификатор целой переменной, значение которой после завершения дей-

ствия оператора WAIT становится равным числу байтов переданной информации.

Материал этого параграфа соответствует версии 4.1 операционной системы ОС ЕС ЭВМ.

§ 3. Фортран для ЭВМ БЭСМ-6

Версия языка Фортран, транслятор с которого установлен на машине БЭСМ-6, разработана на основе входного языка Церн-Фортран для ЭВМ CDC. Транслятор с этой версии является составной частью системы программирования «Дубна» (мониторной системы «Дубна»), поэтому входной язык называется обычно дубненской версией Фортрана (Фортран-Дубна).

1. Отличия входного языка от Фортрана IV. Отличия, которые имеет дубненская версия языка Фортран от языка Фортран IV, вызваны главным образом особенностями ЭВМ БЭСМ-6. Так, целая константа может занимать полную ячейку памяти в виде 48-рядного ненормализованного двоичного числа, откуда следует, что целая константа может состоять не более чем из 12 десятичных цифр, т. е. абсолютное значение должно находиться в диапазоне от 0 до $2^{40}-1$.

При записи действительных констант в форме E, а также чисел двойной точности наличие порядка после символов E или D обязательно (даже если порядок равен нулю). Например, число 1.2345 можно записать так:

1.2345E+00 1.2345E0 1.2345D0

Обязательна хотя бы одна цифра в целой или дробной части мантиссы, которая может иметь до 12 (форма представления E) или 24 (форма D) десятичных цифр. Значение действительной константы лежит в интервале 10^{-10} , 10^{+10} .

Не рассматриваются комплексные константы двойной точности и шестнадцатеричные константы. В то же время вводятся восьмеричные константы, которые представляются в виде последовательности восьмеричных цифр, заканчивающейся символом B. Восьмеричная константа может содержать до 16 цифр. Допускаются отрицательные константы, цифрам которых предшествует знак минус. Например:

1234567B -376B +77300B

Для восьмеричных констант арифметические действия не определены, поэтому эти константы обычно используются лишь в операторе DATA для получения нестандартного содержимого ячейки оперативной памяти.

Для ввода и вывода восьмеричных величин используется спецификация

OW

где O — индекс спецификации, w — целая константа, указывающая длину поля.

При вводе прочитывается содержимое w позиций (колонок на перфокарте). Пробелы в поле ввода игнорируются, символ B не указывается.

При выводе по O -спецификации выводятся w младших восьмеричных цифр, если $w < 16$. Если $w \geq 16$, то выводятся 16 восьмеричных цифр — содержимое полной ячейки памяти, а оставшиеся левые позиции поля занимаются пробелами. Символ B на поле вывода не указывается.

Текстовая константа может содержать не более 120 символов.

Алфавит транслятора расширен за счет заглавных букв русского алфавита и символа \diamond .

Максимальное целое число, используемое в качестве метки, равно 32767.

На одной перфокарте можно располагать несколько операторов. В этом случае они отделяются друг от друга символом \diamond . После символа \diamond нельзя записывать оператор с меткой. Нельзя символ \diamond ставить непосредственно после оператора `FORMAT`.

Символ $\$$ не является буквой и может использоваться вместо символа \diamond для указания конца оператора.

Идентификатор, кроме букв и цифр, может содержать любое число пробелов. При этом пробелы транслятором игнорируются, а остаются лишь значащие символы. Таким образом, идентификаторы, состоящие из одних и тех значащих символов (например, `SIGMA`, `S_I_G_M_A`, `S_I_I_G_L_M_A`) считаются одинаковыми. Если идентификатор состоит более чем из шести значащих символов, то такой идентификатор при трансляции преобразуется к виду, содержащему лишь первые шесть значащих символов, выдается соответствующая информация (предупредительная диагностика) и трансляция продолжается.

В операторах описания типа не должен содержаться указатель длины. В соответствии с этим из величин нестандартной длины рассматриваются только действительные величины двойной точности, описываемые оператором `DOUBLE PRECISION`. Операторы описания типа не используются для присваивания начальных значений. Отсутствует оператор `IMPLICIT`.

Максимальная размерность массивов равна трем. В качестве индекса может быть использовано лишь целое выражение вида $K, J, J \pm K, K * J, K * J \pm K_1$, где K, K_1 — целые константы, J — целая переменная. Если элемент массива указан без индексов, то считается, что указан первый элемент массива. Это правило соблюдается для всех операторов, кроме операторов ввода и вывода.

Все арифметические операции одного ранга выполняются последовательно слева направо (включая возведение в степень).

Так выражение $A**B**C$ понимается как $(A**B)**C$. Поскольку комплексные величины двойной точности не определяются, в одном арифметическом выражении не могут участвовать величины комплексные и двойной точности.

Первым в основной программе должен быть оператор

PROGRAM NAME

где NAME — идентификатор (наименование) программы.

Библиотека стандартных подпрограмм содержит 55 функций.

Наименование входа в операторе ENTRY должно быть согласовано с наименованием подпрограммы-функции (если оператор ENTRY используется в подпрограмме-функции) по типу, а также по списку формальных параметров. Максимальное число дополнительных входов в подпрограмму, т. е. включенных в подпрограмму операторов ENTRY, не должно превосходить 19.

Количество формальных параметров в описаниях оператор-функций, подпрограмм-функций и подпрограмм не должно быть больше 63.

Отсутствуют формальные параметры в виде символа *, а также формальные параметры, заключенные в символы /, т. е. вызываемые по наименованию. Не определяется оператор RETURN I.

В одной программной единице идентификатор общего блока может встречаться не более одного раза. Длина общего блока, уже загруженного в память, например в основной программе, не может быть увеличена подпрограммами.

Допускается отличная от Фортрана IV форма записи оператора DATA и переменный формат. Не рассматриваются спецификации преобразования Z и G и редакционная управляющая спецификация T.

В операторах ввода и вывода используются логические номера внешних устройств. Расширен список управляющих символов. Отсутствуют операторы

READ (N1, N2, END=N3, ERR=N4) L
NAMELIST P

В то же время дополнительно введены операторы внутренней передачи информации ENCODE и DECODE.

2. Видоизмененный оператор DATA. Оператор присваивания начальных значений записывается следующим образом:

DATA (A=R), (B=S), ...

где A, B, ... — простые переменные, переменные с индексами или идентификаторы массивов, R, S, ... — константы или последовательности констант, разделенных запятыми.

Если в списке констант имеются повторения, то запись списка можно сократить, заключив повторяющиеся константы в скобки

и поставив перед открывающей скобкой целую константу, равную числу повторений (коэффициент кратности). Например, если в программе содержатся операторы

```
DIMENSION X(2, 3), Y(5)
```

```
DATA (A=3.5), (X(1, 3)=2.7), (Y=1.4, 3(1.5), 1.6)
```

то начальные значения получат: простая переменная A, элемент массива X с индексами 1 и 3, все элементы массива Y.

Тип переменных в операторе DATA определяется типом присваиваемых им констант, а не оператором описания типа. Например, хотя оператор

```
DIMENSION Z(4)
```

описывает действительный массив, после выполнения оператора

```
DATA (Z=1.5, 2, 2 (.TRUE.))
```

только первый элемент массива будет иметь действительное значение; элемент Z(2) получит целое значение, а Z(3) и Z(4) — логические значения.

Оператором DATA можно оставлять часть массива незаполненной, однако в списке оператора DATA не может быть указано больше констант, чем требуется для заполнения массива.

В операторе DATA допускается запись переменных в виде неявного цикла. Например:

```
DIMENSION A(5)
```

```
DATA ((A(I), I=1, 4)=2 (1., -1.))
```

В этом примере элементы A(1) и A(3) получают значения, равные +1.0, а элементы A(2) и A(4) — значения, равные -1.0. Элемент A(5) не получает начального значения.

Наряду с описанной формой записи оператора DATA возможна запись этого оператора в обычной форме, принятой в языке Фортран IV.

3. Ввод и вывод. Каждому устройству, с которым производится обмен информацией, задаваемой операторами ввода и вывода, присвоен определенный номер, называемый логическим номером. На БЭСМ-6 магнитофонам (накопителям на магнитных дисках) присвоены логические номера с 1 по 15, магнитному барабану — 16, вводному перфоратору — 50, печатающему устройству — 51, выводному перфоратору — 52.

Логический номер означает, что реальное устройство, имеющее данный логический номер, может быть любым из однотипных устройств (например, магнитофонов). Соответствие логических номеров реальным устройствам устанавливается операционной системой, распределяющей эти устройства между параллельно решаемыми задачами. Одна программа может использовать не более

одного АЦПУ, не более одного выводного перфоратора, не более 15 магнитофонов и не более одного магнитного барабана. Использование накопителей на магнитных дисках вместо магнитофонов организуется путем запроса их в управляющих операторах пакета задачи. Логический номер устройства, с которым производится обмен, указывается в операторах ввода и вывода.

Управляющими символами при печати являются

0 + 1 4 5 6 S

которые соответственно вызывают:

- перевод двух строк перед печатью;
- печать без перевода строк;
- прогон до первой строки следующей страницы, представляющей один блок из 66 строк;
- прогон до первой строки следующего блока, где на странице два блока по 32 строки;
- прогон до первой строки следующего блока, где на странице три блока по 21 строке;
- прогон до первой строки следующего блока, где на странице четыре блока по 15 строк;
- отмену проверки переполнения страницы.

Любой символ в первой позиции строки, отличный от приведенных, вызывает пропуск одной строки перед печатью. Управляющие символы в первой позиции строки не печатаются.

Поиск признака конца предыдущего файла с целью формирования очередного файла осуществляется в прямом направлении управляющей программой SCHEOF, обращение к которой имеет вид

CALL SCHEOF (N1)

где N1 — целая константа или переменная, равная логическому номеру магнитной ленты (диска). По окончании поиска лента (диск) устанавливается в начале следующего файла, если таковой имеется, т. е. вызовом программы SCHEOF можно перемотать ленту (провернуть диск) вперед на один файл.

Вместо операторов ввода и вывода можно использовать комплекс управляющих программ, позволяющих работать с внешними накопителями информации на БЭСМ-6 как устройствами прямого доступа.

Единицей записи на БЭСМ-6 является: одна полная строка бумажной ленты на печатающем устройстве (128 позиций), одна перфокарта (80 колонок), одна физическая запись (144 символа), один или несколько физических записей (рекордов), составляющих одну логическую единицу записи, массив ячеек памяти, содержащий определенное число символов, указанное в операторах ENCODE и DECODE.

Одна логическая единица записи образуется при работе одного бесформатного оператора вывода. В общем случае каждый массив информации, записанный с помощью одного оператора вывода, образует одну или несколько логических единиц записи. Логическая единица записи подразделяется на массивы длиной по 24 машинных слова (физические записи) длиной 48 двоичных разрядов. При образовании физической записи к ней приформировываются два служебных слова. В каждой зоне магнитной ленты (диска) или в каждый тракт магнитного барабана помещается по 39 таких записей (состоящих из 26 машинных слов). При этом емкость одной ленты составляет 512 зон, а емкость одного магнитного барабана — 32 тракта.

4. Переменный формат. В процессе выполнения программы допускается формирование списка спецификаций оператора FORMAT, т. е. может быть использован так называемый *переменный формат*. В этом случае список спецификаций вместе с окаймляющими его скобками заносится в виде текстовой константы в заранее резервированный массив или присваивается простой переменной. В операторе ввода или вывода вместо метки оператора FORMAT указывается наименование переменной или массива переменных, в котором хранится список спецификаций.

Элементы списка спецификаций или весь список вместе с окаймляющими скобками как текстовые константы могут быть присвоены переменным или элементам массивов либо с помощью оператора DATA, либо вводом по спецификации A. Например, если в программе содержатся операторы

```
DIMENSION A(6), K(6)
25  FORMAT (F10.3 /2(F8.2, F5.1), F7.0)
READ 25,A
```

то значения элементов массива будут введены, согласно списку спецификаций оператора FORMAT, с меткой 25. Использование части списка, например, стоящей после символа /, возможно, если написать новый оператор FORMAT, списком спецификаций которого будет указанная часть предыдущего списка. Однако удобнее занести полный список рассматриваемого оператора FORMAT в определенное место памяти, например в массив K, и в каждом из двух случаев обращаться к началу соответствующего списка спецификаций. Для этого достаточно вместо оператора FORMAT в программу включить следующий оператор:

```
DATA (K(1)=(F10.3/), (K(2)=2(F8.2, F5.1), (K(3)=F7.0))
```

Теперь оператор

```
READ K, A
```

указывает, что значения элементов массива А будут введены в соответствии со списком спецификаций, который в виде символов составляет список значений элементов массива К.

Для того чтобы использовать весь список спецификаций, в операторе READ указывается идентификатор массива или первый элемент массива. Если же в операторе READ указать, например, элемент К(2), то преобразование вводимых значений будет выполняться по списку спецификаций, начиная со значения элемента К(2), т. е. используется список (2(F8.2, F5.1), F7.0).

5. Операторы внутренней передачи информации. Операторы внутренней передачи информации, или операторы ENCODE и DECODE, аналогичны операторам WRITE и READ с той разницей, что в передаче информации с помощью операторов ENCODE и DECODE внешние устройства не участвуют. Происходит передача информации с одного участка оперативной памяти на другой с преобразованием передаваемой информации согласно спецификациям оператора FORMAT.

Оператор ENCODE преобразует информацию, заданную в форме, определяемой списком спецификаций оператора FORMAT, в последовательность текстовых символов. Оператор DECODE производит обратное преобразование, т. е. перекодирует информацию, заданную в виде последовательности текстовых символов, в ту форму ее представления, которая определяется соответствующей спецификацией оператора FORMAT.

Оператор ENCODE записывается в виде

ENCODE (N, N2, M) L

где N — количество символов в записи (т. е. длина логической единицы записи), это целая константа или простая переменная целого типа, N2 — метка оператора FORMAT, в соответствии с которым производится преобразование информации, M — наименование массива или переменной, куда передается информация, получаемая в текстовом виде, L — список переменных, имеющий структуру списка ввода-вывода.

Оператор DECODE имеет вид

DECODE (N, N2, M) L

и производит перекодировку текстовой информации, заданной на участке оперативной памяти, определяемом идентификатором M, в соответствии со списком спецификаций оператора FORMAT, имеющего метку N2, с последующим запоминанием этой информации на участке оперативной памяти, указанном списком L.

Таким образом, оператор ENCODE преобразует информацию из двоичных слов, размещая ее по шесть символов в ячейку. Если значение параметра N не кратно шести, то единица записи заданной длины N заканчивается в середине ячейки (машинного слова),

и оставшееся свободное место в последней ячейке заполняется пробелами. В операторе DECODE, производящем обратное преобразование, избыточные символы, т. е. символы, заполняющие последнюю ячейку в единице записи длиной N, пропускаются. Например, если переменная L имеет значение 251, то с помощью операторов

```
ENCODE (3, 7, 1) L
7 FORMAT (13)
```

произойдет перекодировка этой величины в текстовую константу вида 3H251, значение которой получает переменная I. Оператор

```
DECODE (3, 7, 1) L
```

произведет перекодировку текстовой константы 3H251 в целую константу 251 и обратное присваивание этого значения переменной L.

6. Формирование пакета задачи. Трансляторы со всех входных языков имеют своим выходным языком так называемый язык загрузки. Управляющая программа МОНИТОР организует процесс обработки всех задач, вызывая соответствующие системные программы для выполнения необходимых действий. Определение необходимых действий и порядок их выполнения основаны на анализе информации, содержащейся в управляющих операторах. Управляющие операторы вместе с операторами самой задачи (подпрограммами и данными) составляют пакет задачи.

В управляющих операторах содержится как обязательная информация о задаче, так и необязательная информация. К обязательной информации относится заказ на необходимые ресурсы, которые, в частности, включают в себя объем оперативной памяти, отводимой для задачи, лимит машинного времени, требуемый для решения задачи, внешние устройства, которые использует задача, библиотеки программ, к которым обращается задача, а также информация о режиме работы системы, указывающая, какие трансляторы и в каком порядке должны быть вызваны для трансляции отдельных подпрограмм задачи, какие дополнительные действия требуются от системы в процессе трансляции (печать текстов подпрограмм, выдача подпрограмм на устройства и т. д.), какие данные использует задача — текстовые или двоичные, требуется ли задаче выход на счет, надо ли печатать (в случае выхода на счет) таблицу загрузки, содержащую адреса подпрограмм и общих областей памяти, какие дополнительные операции до выхода задачи на счет надо выполнить (редактирование, запись в личные библиотеки и др.).

Без заказа задаче выделяются устройства ввода и вывода, магнитные барабаны, а также нестандартные внешние устройства (например, графопостроители). Без заказа каждая задача получает

доступ к библиотеке стандартных подпрограмм. Предусмотрено в случае отсутствия заказа стандартное выделение ресурсов в объеме 24К ячеек оперативной памяти и 5 минут машинного времени. В заказе можно указать объем оперативной памяти не более 32К. В процессе счета задача не может затребовать ресурсы сверх указанных.

В случае отсутствия информации о режиме работы системы устанавливается стандартный режим, который содержит режим трансляции с языка Фортран, выдает абоненту тексты подпрограмм на их входных языках, снабженные некоторой дополнительной информацией, не предусматривает выход на счет, выдает таблицы загрузки, рассматривает все данные как тестовые.

Обязательная информация содержится в трех управляющих операторах и одном дополнительном (не управляющем) операторе.

Управляющий оператор содержит символ `*` в первой позиции, после которой указывается фиксированный текст, определяющий тип управляющего оператора; далее следует некоторая дополнительная информация, зависящая от типа управляющего оператора.

Пакет задачи начинается с управляющего оператора

`* NAME □ ТЕКСТ`,

где ТЕКСТ — это произвольный набор символов, содержащий хотя бы один символ, отличный от пробела; обычно здесь помещается имя (фамилия) абонента.

Следующим обязательным управляющим оператором в пакете является оператор

`* PASS □ ШИФР`

где ШИФР — это специальное наименование (шифр, пароль), присвоенное данному абоненту и открывающее ему доступ к машине. Шифр, состоящий не более чем из шести букв и цифр и начинающийся с буквы, заранее вносится в каталог абонентов данной машины, который хранится в мониторной системе.

После этих управляющих операторов, если в этом есть необходимость, следуют операторы заказа ресурсов, за ними расположены операторы самой программы, т. е. операторы всех программных единиц.

Последним управляющим оператором пакета должен быть оператор

`* END □ FILE`

после которого ставится еще один дополнительный оператор — признак конца ввода пакета (так называемый диспетчерский конец).

Скорость трансляции пакета — около 10—20 операторов в секунду. Качество оттранслированной программы зависит от со-

става операторов исходной программы. Трансляция арифметических выражений, не содержащих переменных с индексами, производится практически оптимально.

Трансляция программ с Фортрана осуществляется в два этапа. На первом этапе производится трансляция на Автокод, затем трансляция с Автокода, в результате которой выдается окончательный результат трансляции в виде программного модуля на языке загрузки.

Диагностика ошибок производится на всех этапах обработки пакета задачи, а именно: при декодировке, при трансляции с Фортрана на Автокод, при трансляции с Автокода на язык загрузки, при загрузке подпрограмм в память и при счете. Транслятором выдается информация, достаточно точно определяющая характер каждой ошибки и позволяющая легко их обнаруживать и исправлять.

§ 4. Фортран ДОС ЕС

В дисковой операционной системе ДОС ЕС Фортран представлен двумя версиями — базисным (основным) Фортраном ДОС и Фортраном IV ДОС. Система программирования для каждого из этих языков включает язык программирования, транслятор и библиотеку программ. Варианты языка Фортран, реализованные в ДОС ЕС, полностью согласуются с соответствующими стандартами указанных версий. По сравнению со стандартами языка ДОС ЕС содержат ряд дополнений, связанных с особенностями ЕС ЭВМ и с расширением возможностей языков.

Язык базисный Фортран ДОС отличается простотой и отсутствием сложных конструкций. Он легок при изучении и удобен для программирования. Вместе с тем этот язык предоставляет программисту большие возможности по использованию средств ввода и вывода. Фортран IV ДОС полностью включает базисный Фортран ДОС. К дополнительным возможностям Фортрана IV ДОС следует отнести наличие операторов отладки и более широкие возможности ввода и вывода данных.

Трансляторы с базисного Фортрана ДОС и Фортрана IV ДОС переводят программу на исходном языке в объектный модуль, который представляет собой программный модуль в промежуточном формате, общем для всех трансляторов операционной системы. Во время трансляции выдается диагностическая информация, облегчающая поиск ошибок в исходной программе. Объектный модуль может содержать символические адреса, не определенные во время трансляции, а также некоторую другую информацию, поэтому до непосредственного выполнения на машине объектный модуль проходит обработку программой РЕДАКТОР.

Объединение объектных модулей в программные фазы (абсолютные модули) происходит независимо от того, когда и с какого языка программирования транслировался тот или иной модуль. Фаза может собираться из независимо оттранслированных частей и подпрограмм, хранящихся в библиотеках. В соответствующих библиотеках могут храниться все программы на любой стадии подготовки (исходный модуль, объектный модуль, абсолютный модуль). Библиотека абсолютных модулей обязательна для функционирования операционной системы ДОС ЕС. Библиотеки базисного Фортрана ДОС и Фортрана IV ДОС являются частью библиотеки объектных модулей и включают программы вычисления различных математических функций, организации ввода и вывода и программы выполнения ряда служебных (системных) действий.

Трансляторы с базисного Фортрана ДОС и Фортрана IV ДОС в качестве рабочих файлов могут использовать диски и магнитные ленты. Оба транслятора могут быть отредактированы для выполнения в режиме пакетной обработки в любой части оперативной памяти (в любом разделе в мультипрограммном режиме). Для работы транслятора с базисного Фортрана ДОС требуется не менее 10 Кбайт оперативной памяти, а для выполнения трансляции с Фортрана IV ДОС — 40 Кбайт оперативной памяти.

1. Отличия Фортрана IV ДОС от стандарта языка Фортран. Фортран IV ДОС включает в себя ряд элементов, отсутствующих в более ранних версиях языка Фортран, в том числе и в стандарте языка Фортран. Изложение языка Фортран IV в гл. II велось с учетом всех дополнительных возможностей, так как эти дополнения уже стали неотъемлемой частью языка, называемого Фортраном IV. Не рассматривались лишь средства отладки и управляющие операторы, т. е. служебные программы Фортрана IV, которые могут существенно зависеть от операционной системы. Переменный формат описан в этой главе в § 3.

Перечислим элементы, которые содержатся в языке Фортран IV ДОС, но отсутствуют в стандарте (1966 г.) языка Фортран. К ним относятся:

- знаки & и ' среди символов языка;
- шестнадцатеричные константы;
- текстовые константы, заключенные в апострофы;
- размерность массива, превосходящая 3;
- выражения смешанного типа;
- многократное возведение в степень без скобок, указывающих порядок вычислений;
- произвольная форма индексного выражения;
- присвоение начальных значений в операторах явного описания типа;

- оператор IMPLICIT;
- указание длины переменных и массивов в операторах явного описания типа и длины функции в операторе FUNCTION;
- наименование массива в операторе DATA;
- оператор PAUSE 'ТЕКСТ';
- стандартные встроенные функции: DINT, DMOD, DFLOAT, IFIX, DCMPLX, DCONJG;
- стандартные внешние функции: CDEXP, CDLOG, CDSIN, CDCOS, DTANH, CDSQRT, CDABS, ARSIN, DARSIN, ARCOS, DARCOS, SINH, DSINH, COSH, DCOSH, ERF, DERF, ERFC, DERFC, GAMMA, DGAMMA, ALGAMA, DLGAMA, COTAN, DCOTAN, TAN, DTAN, CABS, CDABS;
- формальные параметры, заключенные в символы /;
- текстовая константа как фактический параметр при обращении к подпрограмме-функции;
- оператор RETURN I;
- оператор ENTRY;
- переменные размеры массивов, передаваемые через общие области COMMON;
- оператор READ N2, L;
- оператор PUNCH N2, L;
- оператор PRINT N2, L;
- оператор READ с параметрами ERR и END;
- операторы ввода и вывода прямого доступа;
- оператор NAMELIST;
- спецификации T, Z, G;
- операторы отладки;
- служебные программы библиотеки.

2. Средства отладки. Средства отладки представляют собой набор операторов языка, посредством которых можно проследить последовательность выполнения операторов программы, вывести на печать значения переменных и массивов, проверить правильность использования индексов и обращений к подпрограммам. Операторы отладки являются составной частью входного языка Фортран IV ДОС, но отсутствуют в базисном Фортране ДОС.

К операторам отладки относятся неисполняемые операторы DEBUG и AT и исполняемые операторы TRACE ON, TRACE OFF, DISPLAY. Отладочные действия, задаваемые операторами отладки, выполняются для одной программной единицы. Для этого в конце исходной программы (перед оператором END) должен быть помещен оператор DEBUG, за которым могут следовать пакеты отладки. Каждый пакет отладки состоит из оператора начала пакета AT, одного или более исполняемых отладочных операторов, а также, если необходимо, из операторов исходной программы.

Операторы DEBUG, AT, TRACE ON и TRACE OFF рассматривались в § 2 этой главы. Оператор DISPLAY имеет вид

DISPLAY L

где L — список идентификаторов простых переменных или массивов. Элементы в списке разделяются запятыми.

Оператор DISPLAY предназначен для вывода в файл отладки в формате NAMELIST значений переменных и массивов, указанных в списке. Значения элементов списка выводятся перед выполнением оператора, метка которого указана в операторе начала пакета отладки AT. Таким образом, оператор DISPLAY выполняет те же действия, что и группа операторов

```
NAMELIST /X/ L
```

```
WRITE (N1, X)
```

где X — наименование списка L, N1 — номер файла.

Если оператор DISPLAY используется в подпрограмме типа SUBROUTINE, то в списке оператора DISPLAY не должны употребляться формальные параметры этой подпрограммы. Оператор DISPLAY не должен появляться в логическом операторе IF.

3. **Служебные программы.** Служебные программы, содержащиеся в библиотеке базисного Фортрана ДОС и Фортрана IV ДОС, используются для выполнения некоторых вспомогательных действий, например таких, как завершение выполнения программы, вывод на печать содержимого участков памяти и т. д. Каждая служебная программа вызывается указанием ее наименования в операторе CALL. Наименованиями служебных программ являются следующие идентификаторы: SLITE, SLITET, DUMP, PDUMP, DVCHK, OVERFL, EXIT.

Описание служебных программ дано в § 2. Отметим лишь, что параметрами KN при обращении к программам DUMP и PDUMP в базисном Фортране ДОС могут служить константы 0, 4, 5, 6, а в Фортране IV ДОС — все перечисленные в § 2 константы, кроме констант 10 и 11.

4. **Ввод и вывод.** В Фортране ДОС ЕС обрабатываются файлы на перфокартах, магнитных лентах, файлы последовательного и прямого доступа на дисках, файлы печати и файлы пишущей машинки. Файлы на магнитных лентах могут быть с метками или без меток. Файлы на дисках всегда должны иметь метки. Не обрабатываются многотомные файлы и файлы, разделенные на участки (многоучастковые), а также многофайловые тома магнитных лент.

В исходной программе обращение к устройствам ввода и вывода производится через номера файлов. В управляющих операторах программы УПРАВЛЕНИЕ ЗАДАНИЯМИ для описания файлов на дисках и на магнитной ленте с метками используются однозначно определенные в ДОС ЕС имена файлов. Между номе-

рами файлов, логическими устройствами и именами файлов установлено определенное соответствие, которое не может изменяться программистом. Программист может не заботиться о соответствии логических и физических устройств, ему следует лишь указывать соответствующий номер файла.

Некоторые номера файлов могут быть использованы только с определенными типами устройств. Так, номер 1 относится к вводу, 2, 3, 15 — к выводу. С номерами 1, 2 связываются устройство ввода с перфокарт, накопитель на магнитной ленте и накопитель на дисках для размещения последовательного файла; с номером 3 — печатающее устройство, накопитель на магнитной ленте и накопитель на дисках для размещения файла последовательного доступа; с номером 15 связывается вывод на печатающее устройство или пишущую машинку. Номера файлов 4—14 могут быть использованы как при вводе, так и при выводе с любыми устройствами (в том числе с дисками с прямым доступом), кроме пишущей машинки.

Операторы управления файлами `BACKSPACE`, `END FILE` и `REWIND` могут использоваться только для файлов на магнитных лентах и последовательных файлов на дисках.

По аналогии с понятиями логического и физического устройства различают логическую и физическую записи. Логическая запись (запись Фортрана) — это совокупность данных, связанных между собой с точки зрения программной обработки. Физическая запись — это блок данных, размер которого определяется используемым физическим устройством. В зависимости от вида записи (форматная или бесформатная) и метода доступа к данным (прямой или последовательный) логическая запись может состоять из одной или нескольких физических записей. Размер физической записи в Фортране устанавливается в байтах в зависимости от номера файла и от типа физического устройства, назначенного файлу.

Для перфокарточных устройств ввода и вывода всегда (независимо от номера файла) устанавливается размер физической записи 80 байт. По 80 байт отводится под физическую запись для накопителей на магнитной ленте и диске, если файл имеет номер 1 и по 81 байт, если номер файла — 2. Из них первый символ является управляющим. Для файла с номером 3 размер физической записи для всех допустимых устройств равен 121 байт, из них первый символ является управляющим. Максимальный размер физической записи в базисном Фортране для файла с номером 15 равен 260 (пишущая машинка) или количеству позиций в строке печати, но не больше 260 для устройства печати.

В случае файлов с номерами 4—14 размер физической записи для печатающего устройства равен количеству позиций в строке печати, но не больше 260, для накопителя на магнитной ленте от

18 (от 15 в базисном Фортране) до 260, для накопителя на магнитных дисках с последовательным доступом — 260, для накопителя на дисках с прямым доступом в соответствии с заданием в операторе DEFINE FILE, но не больше 3625 (1726 в базисном Фортране ДОС) байт для устройств EC-5052, EC-5055, EC-5056 и 7294 (1726) байт для устройств EC-5061.

5. Отличия версий Фортрана ДОС ЕС. Как уже отмечалось выше, язык Фортран IV ДОС полностью включает в себя средства языка базисный Фортран ДОС. Следующие элементы языка Фортран IV ДОС отсутствуют в базисном Фортране ДОС:

- комплексные, логические и шестнадцатеричные константы;
- использование текстовых констант где-либо, кроме оператора FORMAT;
- типы COMPLEX и LOGICAL в операторах явного описания типа и в операторе FUNCTION;
- размерность массива, превосходящая 3;
- представление индекса элемента массива в виде произвольного арифметического выражения;
- массивы с переменными размерами;
- оператор IMPLICIT;
- указание длины переменных и массивов в операторах явного описания типа и в операторе FUNCTION;
- присвоение начальных значений данным в операторах явного описания типа;
- оператор DATA;
- оператор BLOCK DATA;
- операции отношения;
- логические операции;
- логические выражения;
- логический оператор присваивания;
- логический оператор IF;
- присваиваемый оператор GO TO;
- оператор ASSIGN;
- оператор PAUSE 'ТЕКСТ';
- текстовая константа как фактический параметр при обращении к подпрограммам типа FUNCTION и SUBROUTINE;
- формальные параметры, заключенные в символы /;
- формальный параметр в виде символа *;
- метка оператора в качестве фактического параметра;
- оператор RETURN I;
- оператор ENTRY;
- именованные общие блоки;
- оператор NAMEDLIST;
- оператор READ N2, L;
- оператор PRINT N2, L;

- оператор PUNCH N2, L;
- оператор READ с параметрами ERR и END;
- спецификации L, G, Z в операторе FORMAT;
- использование форматов в массиве (переменного формата);
- вложенность групп спецификаций в операторе FORMAT, превосходящая 1;

- операторы отладки;
- расширение форматов вывода в служебных программах DUMP и PDUMP.

В базисном Фортране ДОС, кроме перечисленных отличий от Фортрана IV ДОС, число стандартных функций (встроенных и внешних) равно 45. Оператор DEFINE FILE должен находиться среди описательных операторов.

6. Количественные ограничения. Программы, которые обрабатываются трансляторами с языка Фортран в операционной системе ДОС ЕС, должны быть составлены с учетом ограничений, налагаемых на элементы исходной программы. Для транслятора с базисного Фортрана ДОС эти ограничения сводятся к следующим.

Максимальное значение, равное 65532 байт, установлено для размеров рабочей программы, общей области, определенной оператором COMMON, и области данных вне общей области (т. е. области данных, включающих переменные, не входящие в общую область, константы и рабочие поля, построенные транслятором).

Вложенность операторов цикла ограничена 25 операторами DO, а вложенность ссылок к подпрограмме-функции не должна превосходить 15.

Не должно превышать 511 общее число каждого из следующих объектов программы:

- уникальных целых констант;
- уникальных действительных констант;
- уникальных констант двойной точности;
- переменных;
- массивов;
- наименований в операторах REAL;
- наименований в операторах INTEGER;
- наименований в операторах DOUBLE PRECISION;
- переменных и массивов в общей области;
- наименований в списках оператора эквивалентности вместе с количеством списков оператора EQUIVALENCE;
- определений оператор-функций;
- уникальных наименований подпрограмм (имен точек входа в подпрограммы);
- формальных параметров в подпрограмме или оператор-функции;
- параметров во всех подпрограммах и оператор-функциях;

— помеченных операторов (включая одну дополнительную метку на каждый оператор DO или неявный цикл в списке ввода-вывода).

Данные этого параграфа соответствуют версии 2.0 операционной системы ДОС ЕС ЭВМ.

§ 5. Фортран-Дубна и Фортран ДОС ЕС

Языки программирования Фортран для ЭВМ БЭСМ-6 и Фортран IV ДОС ЕС ЭВМ являются довольно близкими версиями языка Фортран, тем не менее при переводе программ с одной машины на другую необходимо принимать во внимание имеющиеся различия. В этом параграфе перечислены основные ограничения, налагаемые на язык Фортран правилами версии Фортран IV ДОС ЕС (которая в дальнейшем называется для краткости Фортран ЕС) по сравнению с правилами версии языка Фортран для ЭВМ БЭСМ-6 мониторной системы Дубна (называемой в дальнейшем Фортран-Дубна). Сюда включены также основные различия в интерпретации одних и тех же конструкций языка Фортран-Дубна трансляторами мониторной системы Дубна и ДОС ЕС и приведены краткие рекомендации по использованию возможностей языка Фортран-Дубна для достижения максимальной совместимости с языком Фортран ЕС.

Нарушение каждого из указываемых ниже правил при использовании языка Фортран ЕС может привести к различным последствиям. В целях удобства как использования материала этого параграфа, так и изложения правил в тексте каждого пункта правил, в квадратных скобках указывается одно (или несколько) условное обозначение в виде символов: Т, С, Р, П, И, *. Каждый из этих символов означает соответственно следующее:

Т — нарушение данного правила будет обнаружено транслятором ДОС ЕС, т. е. при трансляции будет выдана диагностика ошибки;

С — нарушение данного правила может быть обнаружено в процессе счета на ЕС ЭВМ;

Р — нарушение данного правила может привести к получению неверных результатов при счете на ЕС ЭВМ;

П — данная особенность представления числовой информации на ЕС ЭВМ может привести к значительному увеличению погрешности в результатах по сравнению с БЭСМ-6;

И — при переходе с ЭВМ БЭСМ-6 на ЕС ЭВМ (и обратно) необходимо внести изменения в программу;

* — ограничение на язык для краткости сформулировано более жестко по сравнению с правилами версии Фортран ЕС.

Если у одного пункта правил указано несколько символов, то это означает, что в данном случае возможен любой из исходов, определяемых каждым из этих условных обозначений.

1. Основные ограничения языка Фортран ЕС. Правилами языка Фортран ЕС запрещается:

- использовать символ \$ (или \diamond) в качестве разделителя между операторами в пределах одной строки; на ЕС ЭВМ символ \$ является буквой [T];

- использовать оператор PROGRAM для указания начала основной программы [T];

- использовать операторы ENCODE и DECODE [T];

- использовать восьмеричные константы; вместо них можно использовать шестнадцатеричные константы [T];

- указывать целые константы, превосходящие по модулю $2^{31}-1 \approx 2 \times 10^9$ [T]; на БЭСМ-6 аналогичный предел равен $2^{40}-1 \approx 1 \times 10^{12}$;

- указывать константы двойной точности, превосходящие по модулю $16^{63} \approx 7 \times 10^{75}$ [T]; на БЭСМ-6 такой предел равен приблизительно 1×10^{1000} ;

- использовать текстовые константы, состоящие более чем из четырех символов. [ТС*];

- использовать идентификаторы, содержащие буквы русского алфавита или состоящие более чем из шести символов [T];

- указывать текстовые константы в операторах присваивания и в логическом операторе IF [T];

- использовать элементы массивов без индексов или с меньшим числом индексов, чем указано в описании соответствующих массивов; исключения допускаются только в операторах EQUIVALENCE и DATA, а также в фактических параметрах при вызове подпрограмм [ТСР];

- использовать одинаковые метки у исполняемых операторов и у операторов FORMAT в пределах одной программной единицы [T]; на БЭСМ-6 в одной и той же подпрограмме допускаются, например, операторы

7 A=3.14

7 FORMAT (F10.3)

- применять оператор DATA для передачи данных в элементы непоименованного общего блока, а также в элементы помеченных блоков COMMON, расположенных вне подпрограммы BLOCK DATA [T];

- располагать оператор DATA ранее описательных операторов, содержащих те переменные и массивы, которым присваиваются начальные значения с помощью данного оператора DATA [T]; рекомендуется располагать оператор DATA последним среди других неисполняемых операторов данной программной единицы;

- записывать оператор DATA в форме, использующей скобки в качестве ограничителей списка констант; нельзя, например,

написать в виде DATA (X=5.), но можно написать в виде DATA X/5./ [T];

— допускать несовпадение по типу в операторе DATA соответствующих элементов в списке переменных и в списке констант; нельзя, например, написать DATA X/5/, а можно написать только DATA X/5./ [T];

— использовать переменные и массивы типов REAL или INTEGER вместе с величинами любого из других типов в одном блоке COMMON или в одной группе эквивалентности в операторе EQUIVALENCE; например, такая совокупность операторов:

```
COMPLEX C(2)
COMMON /A/ B, C
```

где B — простая переменная типа REAL*4, недопустима [T*];

— использовать помеченные блоки COMMON разной длины в различных подпрограммах, описывающих одни и те же общие блоки [T]; на БЭСМ-6 нарушение этого правила иногда остается без последствий и не приводит к неправильным результатам;

— указывать массивы в качестве фактических параметров, если соответствующие им формальные параметры являются простыми переменными [СИ];

— использовать при вводе по спецификациям I, F, E, D пробелы в поле ввода, за исключением самых левых позиций в каждом поле; на БЭСМ-6 такие пробелы при вводе игнорируются, а на ЕС ЭВМ интерпретируются как нули [CP];

— указывать в операторах FORMAT скобки с глубиной вложения более двух уровней [T]; на БЭСМ-6 допускается глубина вложения скобок до трех уровней;

— использовать спецификацию преобразования Ow для восьмеричных чисел; для преобразования шестнадцатеричных чисел используется спецификация Zw [T];

— располагать в операторе FORMAT в первой позиции каждой строки какие-либо символы, кроме управляющих, т. е. символов 0, 1 и + [C], смысл которых тот же, что и на БЭСМ-6; остальные символы, используемые на БЭСМ-6 в качестве управляющих, на ЕС ЭВМ не являются управляющими;

— использовать библиотечную подпрограмму ASIN(X) для вычисления значения арксинуса действительного аргумента [И]; взамен используется стандартная функция ARSIN(X).

2. Основные рекомендации. При переходе с ЭВМ БЭСМ-6 на ЕС ЭВМ в целях достижения максимальной совместимости с языком Фортран ЕС, а также во избежание получения неверных результатов не рекомендуется:

— использовать вычисляемый оператор перехода

```
GO TO (N1, N2, ..., NK), M
```


в случае, когда условие $1 \leq M \leq K$ не будет выполнено; на ЕС ЭВМ произойдет передача управления следующему исполняемому оператору [CP], а на БЭСМ-6 будет неправильная передача управления;

- использовать оператор ENTRY во всех случаях, кроме тех, когда соответствующая подпрограмма является подпрограммой типа SUBROUTINE без параметров; правила языка Фортран ЕС требуют, чтобы у каждого оператора ENTRY был указан список его формальных параметров, если таковые имеются [ПСИ]; на БЭСМ-6 список формальных параметров у оператора ENTRY не указывается, а предполагается, что список параметров у оператора ENTRY тот же, что и у заголовка подпрограммы, т. е. у операторов SUBROUTINE или FUNCTION;

- записывать выражение вида $A**B**C$ без скобок, указывающих точный порядок выполнения операций возведения в степень; на ЕС ЭВМ запись $A**B**C$ понимается как $A**(B**C)$, а на БЭСМ-6 — как $(A**B)**C$ [CP];

- использовать «пустые» циклы в операторах DO, т. е. циклы вида

DO M I=N1, N2, N

при $N1 > N2$ ($N > 0$); если это условие выполнено, то на ЕС ЭВМ цикл будет выполнен один раз, на БЭСМ-6 — ни разу [CP];

- использовать константы для обозначения логических номеров устройств ввода и вывода [СИ]; вместо констант рекомендуется использовать простые переменные целого типа;

- указывать в операторе DATA текстовые константы, содержащие более четырех символов [СИ].

Перечисленные здесь отличия языка Фортран ЕС от языка Фортран-Дубна необходимо учитывать не только при переходе с ЭВМ БЭСМ-6 на ЕС ЭВМ, но и при обратном переходе.

3. Особенности представления числовой информации. При переходе с БЭСМ-6 на ЕС ЭВМ рекомендуется учитывать следующие особенности представления числовой информации в ЕС ЭВМ:

- точность машинного представления величин действительного типа при использовании машинных слов стандартной длины (4 байта) составляет приблизительно 7 десятичных цифр против 12 десятичных цифр на ЭВМ БЭСМ-6; при переходе на машинные слова двойной длины (8 байт) эта точность увеличивается до 17 десятичных цифр, что, однако, меньше точности машинного представления величин типа DOUBLE PRECISION на БЭСМ-6 (24 десятичные цифры) [СПИ];

- представление действительных величин в ЕС ЭВМ в виде машинных слов двойной длины увеличивает только точность их машинного представления, но не расширяет, в отличие от БЭСМ-6, диапазон чисел, представимых в машине [СП];

— в ЕС ЭВМ арифметические операции над величинами, представляемыми машинными словами двойной длины, реализованы, в отличие от БЭСМ-6, не программно, а аппаратно, поэтому при переходе на двойную точность время счета возрастает всего в 1,3—1,5 раза, а не в 8—10 раз, как на БЭСМ-6 [С];

При переходе с ЭВМ БЭСМ-6 на ЕС ЭВМ с учетом возможного использования величин двойной точности (длиной 8 байт) взамен действительных рекомендуется:

— не описывать оператором REAL на БЭСМ-6 переменные и массивы действительного типа, которые, возможно, потребуется перевести на двойную точность [С];

— описывать оператором REAL переменные и массивы действительного типа, которые при переходе с БЭСМ-6 на ЕС ЭВМ не предполагается переводить на двойную точность.

При переходе на двойную точность с соблюдением правил, указанных в п. 3, достаточно в исходной программе, написанной на языке Фортран-Дубна, добавить оператор языка Фортран ЕС

```
IMPLICIT REAL * 8 (A—Z)
```

в каждой подпрограмме, где производится переопределение действительного типа на тип с двойной точностью.

Точное соблюдение правил, приведенных в пп. 1 и 2, обеспечивает практически полную совместимость исходных программ на уровне языка Фортран. В то же время ввиду отличий в способе машинного представления числовой информации в БЭСМ-6 и ЕС ЭВМ, это не обеспечивает, вообще говоря, достаточно точного совпадения результатов счета, полученных на этих машинах. Дело в том, что, как это следует, в частности, из сказанного в п. 3, стандартное представление действительных величин в машинах серии ЕС ЭВМ в виде машинных слов длиной 4 байта, из которых для представления мантиссы отводится 3 байта, довольно часто не обеспечивает удовлетворительной точности результатов. В таких случаях и возникает необходимость представления действительных величин в ЕС ЭВМ с двойной точностью, т. е. в виде машинных слов, состоящих из 8 байт, из которых под мантиссу отводится 7 байт. Это требует, с одной стороны, введения дополнительных ограничений при составлении программ, совместимых для БЭСМ-6 и ЕС ЭВМ, и, с другой стороны, введения дополнительных описательных операторов языка фортран ЕС при переходе с БЭСМ-6 на ЕС ЭВМ. Эти требования будут удовлетворены, если соблюдать рекомендации, перечисленные в п. 3.

§ 6. Стандарт Фортран 77

Первоначально язык Фортран разрабатывался для численных приложений, и стандарт 1966 г. отражал запросы соответствующего класса пользователей. Программа на Фортране, разработан-

ная для некоторой вычислительной системы, если она удовлетворяла стандарту, могла без всяких изменений использоваться и на другой вычислительной системе. Однако сфера применения Фортрана расширялась, охватывая не только область сугубо численных приложений, но и такие, например, области, как обработка текстов и работа с файлами. Происходила модификация языка, в него включались более мощные средства обработки данных, появлялись все новые и новые версии (диалекты) Фортрана. Отличия диалектов от стандарта касаются главным образом расширения возможностей языка и включения дополнительных элементов. В результате увеличилась область потенциальных приложений Фортрана, но за счет ухудшения переносимости написанных на Фортране программ. Возникла необходимость создания нового стандарта.

Проект нового стандарта был опубликован в 1976 г., а в 1978 г. была принята окончательная версия нового стандарта для Фортрана, получившего название Фортран 77. Новый стандарт не исключает использования старых фортранных программ, он лишь расширяет возможности языка при вводе и выводе, при описании данных, описании подпрограмм и конструкций, в которых ранее допускались только значения целого типа, а также включает ряд изменений разнообразного характера, в том числе синтаксические и семантические усовершенствования.

1. Расширения по сравнению с предыдущим стандартом. В алфавит добавлен символ: (двоеточие). Отсутствуют зарезервированные слова, поэтому идентификатор может по написанию полностью совпадать с основным символом (например, ключевым словом); смысл последовательности литер определяется по контексту.

Введены переменные текстового типа, принимающие в качестве значений текстовые константы в 'представлении. Текстовая константа в H-представлении (холлеритовская константа) может встречаться только в операторе DATA, в списке параметров оператора CALL и в операторе FORMAT. Текстовый тип описывается оператором

CHARACTER *S X *S1, Y *S2, . . .

где S, S1, . . . — описатели длины, X, Y, . . . — идентификаторы простых переменных или описатели массива, S относится ко всем элементам списка X, Y, . . ., у которых нет собственного описателя длины, а S1, S2, . . . — только к предшествующему элементу X, Y, Описатель длины может быть целой константой без знака, целым выражением из констант в скобках, имеющим положительное значение, или звездочкой в скобках. Если описатель длины отсутствует, длина принимается равной единице. Описатель длины (*) означает, что длина должна быть определена иным образом

(например, при вызове подпрограммы). Так, в операторах

```
CHARACTER *4 L, K(10) *8, C *(*), M  
CHARACTER A, B  
CHARACTER *(*), E, P *1, D
```

переменные L, M имеют длину 4 символа, A, B, P — длину 1, элементы массива K — длину 8, а для переменных C, E, D длина не определена. Длина текстовой переменной может быть вычислена с помощью стандартной функции LEN. Так, значение LEN (M) есть 4.

Может быть образована текстовая подцепочка (подстрока) — последовательность расположенных подряд литер, входящих в состав некоторого элемента данных текстового типа. Подстрока имеет также текстовый тип. Значение подстроки образуется указанием после идентификатора текстовой переменной в скобках с разделителем: (двоеточие) порядковых номеров первого и последнего символа из тех литер, которые должны быть включены в подстроку. Например, если текстовая переменная Z имеет значение 'FORTRAN', то величина Z(1 : 3) является подстрокой со значением 'FOR', а Z(4 : 6) имеет значение 'TRA'. Аналогично образуется подстрока в случае переменной с индексами. Так, значением T(I, J)(K : L) является подстрока от K-го до L-го символа элемента T(I, J) массива текстовых значений.

Для текстовых величин определена операция, называемая конкатенацией (сцеплением) и обозначаемая как //. Используется эта операция при построении текстовых выражений — выражений, значением которых являются константы текстового типа. Например, если A и B являются текстовыми переменными со значениями '1+' и '123' соответственно, то выражение A//B//'X' имеет значение '1+123X'. Операция сцепления выполняется после арифметических операций перед операциями отношения.

Операндами в операции отношения могут быть текстовые выражения. В таком случае сравниваются двоичные коды текстовых операндов, а результатом операции является «истина» или «ложь», и зависит результат от упорядоченности кодов литер по возрастанию. Если текстовые операнды имеют разную длину, более короткий операнд при сравнении дополняется справа пробелами до длины более длинного.

Определен оператор присваивания для текстовых переменных. При этом слева от знака присваивания может стоять текстовая переменная, элемент текстового массива или наименование подстроки, справа — текстовое выражение. При вычислении текстового выражения и присваивании его значения текстовой переменной никакая из изменяющихся литерных позиций в переменной не должна использоваться при вычислении выражения. Если длина выражения

меньше длины переменной, значение его дополняется справа пробелами до длины переменной. Если длина выражения больше длины переменной, значение выражения при присваивании укорачивается справа. Присваивание подстроке определяет только те литерные позиции, которым делается присваивание. Например, после выполнения операторов

```
CHARACTER P *3, R *6, S *7  
P='ABC'  
R=P// 'DEF'  
S(3 : 6)=R(2 : 5)
```

переменная R имеет значение 'ABCDEF', а подстрока S(3 : 6) — значение 'BCDE'.

Определены логические операции .EQV. — эквивалентность и .NEQV. — неэквивалентность. Результатом выражения L1.EQV.L2, где L1, L2 — логические величины, будет значение .TRUE., если L1 и L2 имеют одинаковые значения и .FALSE. — в противном случае. Результат выражения L1.NEQV.L2 имеет противоположное по сравнению с выражением L1.EQV. L2 значение.

Основная программа начинается с оператора PROGRAM A, где A — идентификатор (имя основной программы).

Определен оператор неявного описания типа по первой букве с ключевым словом IMPLICIT.

Допускается размерность массивов до семи. Нижняя граница изменения индекса у переменной с индексами не обязательно равна 1, и в качестве индекса массива допускаются отрицательные и нулевые целые константы, переменные и выражения, а диапазон изменения индекса при описании массива можно задавать граничной парой вида I : J, где I, J — соответственно нижний и верхний пределы изменения индекса. Например, оператор

```
INTEGER A(0 : 99), B(-100 : 0, -10 : 10, 101)
```

описывает одномерный массив A, индекс которого изменяется от 0 до 99 и трехмерный массив B, первый индекс которого изменяется от -100 до 0, второй — от -10 до 10, третий — от 1 до 101.

Введен оператор SAVE, действие которого заключается в том, что перечисленные в его списке переменные, массивы или общие блоки не окажутся неопределенными после выполнения в подпрограмме операторов RETURN или END. Например:

```
SAVE /Q/, X, M
```

где Q — имя общего блока, X — переменная, M — идентификатор массива. Таким образом, значения переменных и массивов от одного обращения к подпрограмме до следующего могут быть сохранены, если они заданы:

— в операторе SAVE;

— в непомеченном общем блоке, который, по существу, связан с основной программой;

— в помеченном общем блоке, который описан и в подпрограмме и в вызывающей программной единице.

В операторе DATA разрешено использование идентификаторов массивов, текстовых переменных, подстрок и неявных циклов. Введено описание символических констант оператором

```
PARAMETER (P1=K1, P2=K2, . .)
```

где P1, P2, . . . — идентификаторы (параметры), K1, K2, . . . — выражения из констант. Каждая пара величин (P1 и K1, P2 и K2 и т. д.) должна быть согласована по типу: Текстовые параметры с длиной, отличной от 1, нуждаются в описателе длины в операторе IMPLICIT или CHARACTER. Например:

```
LOGICAL L
```

```
CHARACTER N *(*), M *3
```

```
PARAMETER (L=.TRUE., K=105, M='CDC', N='PI')
```

Параметр и константа не являются взаимозаменяемыми. Параметр можно использовать только там, где это явно разрешено, например в качестве операнда в выражении, или в операторе DATA. Нельзя параметр использовать, например, для формирования комплексной константы и для обозначения текстовой константы в операторе FORMAT.

Целую переменную, которой присвоена метка оператором присваивания метки, можно использовать не только в присваиваемом операторе перехода, но и для задания формата в операторах ввода и вывода.

Допускаются пустые строки, которые обрабатываются как комментарии, т. е. игнорируются. В качестве символа для указания комментария, кроме буквы C, можно использовать символ *; при этом строка комментария может содержать произвольную последовательность литер в позициях со второй по 72.

В вычисляемом операторе перехода GO TO (N), M на месте M может стоять любое целое выражение. Если значение выражения больше количества элементов списка меток или если это значение не положительно, выполнение программы продолжается с оператора, который следует за вычисляемым оператором GO TO. В присваиваемом операторе перехода GO TO M, (N) список меток может отсутствовать.

Условные операторы управления дополнены конструкцией IF THEN — ELSE, позволяющей задавать условное выполнение групп операторов. Рассматриваемая конструкция имеет следующую структуру:

```
IF (L) THEN  
(группа IF)
```

```

ELSE IF (L1) THEN
<группа ELSE IF>
. . . . .
<другие группы ELSE IF>
. . . . .
ELSE
<группа ELSE>
END IF

```

в которой могут быть опущены <группа IF>, <группа ELSE> или <группы ELSE IF>, представляющие собой один или несколько операторов. В этой записи L, L1 — логические выражения, IF (L) THEN, ELSE IF (L1) THEN, ELSE и END IF — операторы. Для каждого оператора IF должен существовать свой оператор END IF.

Входящие в данную конструкцию группы операторов, как следует из ее структуры, могут быть вложенными, причем вложение должно быть полным, т. е. если одна группа IF содержится внутри другой группы IF, то все операторы первой из них (вложенной) должны находиться между операторами IF (L) THEN и END IF второй (внешней) группы IF. Для каждого из входящих в IF — THEN — ELSE операторов определяется уровень оператора l как разность $m - n$, где m — число операторов IF (L) THEN от начала конструкции до l включительно, а n — число операторов END IF в конструкции до l , но не включая l .

В операторе IF (L) THEN (или ELSE IF (L) THEN) вычисляется значение логического выражения L и, если оно истинно, выполняется ближайшая группа IF (или соответственно группа ELSE IF) и управление передается на оператор END IF того же уровня, что и у оператора IF (или соответственно ELSE IF), если, конечно, внутри группы IF (или группы ELSE IF) нет операторов передачи управления. Если L — ложно, выполнение операторов продолжается со следующего оператора ELSE IF, ELSE или END IF с тем же уровнем, что и у оператора IF (или ELSE IF). Управление не может быть передано извне в группу IF (или группу ELSE IF).

Оператор ELSE обеспечивает выполнение группы ELSE вплоть до оператора END IF. Передача управления внутрь группы ELSE извне запрещена. Оператор END IF отмечает место в программе и не выполняет никаких действий. На оператор END IF можно передать управление (при наличии метки), а на операторы ELSE IF и ELSE нельзя. Операторы IF, ELSE IF, ELSE, END IF не могут выступать в качестве последнего оператора в области цикла.

Пример конструкции IF — THEN — ELSE:

```

IF(L) THEN
I=I1

```

```

J=J1
ELSE IF (M) THEN
  A=A1
  B=B1
ELSE
  X=X1
  Y=Y1
END IF

```

В операторе цикла компоненты заголовка цикла: параметр цикла I, нижняя N1 и верхняя N2 границы и шаг N изменения параметра могут быть выражениями целого или действительного типа или двойной точности. Сказанное относится и к неявным циклам. Если типы компонент заголовка цикла не согласованы, выполняются преобразования к типу параметра цикла. Выполнение цикла начинается с проверки числа повторений цикла, определяемого как значение $\text{MAX}(\text{INT}((N2-N1+N)/N), 0)$, и если оно равно нулю, цикл не выполняется ни разу. Можно изменить значения компонент N1, N2, N во время выполнения цикла, однако это не повлияет на количество повторений цикла, так как оно вычисляется прежде, чем цикл выполнится в первый раз. При выходе из цикла параметр цикла сохраняет последнее присвоенное ему значение.

В записи операторов STOP и PAUSE после ключевого слова может быть указана целая константа не более чем из пяти цифр или текстовая константа.

Стандартные функции подразделяются на встроенные и внешние. При обращении к встроенной функции генерируется последовательность команд, непосредственно реализующая эту функцию, обращение к внешней функции есть обращение к библиотечной подпрограмме. Все стандартные функции, соответствующие одной и той же математической функции (процедуре), имеют общее имя, называемое универсальным именем, которым можно пользоваться независимо от типа аргумента (аргументов) функции. Обращение к соответствующей специфической функции (с конкретным типом аргументов и значения) обеспечивается автоматически. Например, ко всем специфическим функциям ABS(X), IABS(K), DABS(D), CABS(C) можно обратиться по универсальному имени ABS. Общее число стандартных функций — 81.

Если при обращении к внешней процедуре (т. е. к подпрограмме-функции или к подпрограмме) фактическим параметром является стандартная функция, то следует использовать специфическое имя функции, описав его в операторе INTRINSIC в той программной единице, где находится обращение к внешней процедуре. Например:

INTRINSIC SIN CALL FS(A, M, N, SIN)

Как и в случае оператора EXTERNAL, оператор INTRINSIC не позволяет завести переменную вместо функции, заданной в качестве фактического параметра в вызывающей программной единице. Но если же в операторе EXTERNAL встретилось имя стандартной функции, то оно в данной программной единице не будет рассматриваться как имя стандартной функции. Никакое имя в пределах одной программной единицы не может встречаться одновременно в операторах INTRINSIC и EXTERNAL. Наименования встроенных функций преобразования типов (INT, IFIX, IDINT, FLOAT, SNGL, REAL, DBLE, CMPLX, ICHAR, CHAR), а также функций для вычисления наибольшего и наименьшего значения (MAX, MAX0, AMAX 1, DMAX 1, AMAX0, MAX1, MIN, MIN0, AMIN1, DMIN1, AMIN0, MIN1) нельзя упоминать в операторе INTRINSIC.

Допускаются описания оператор-функций и подпрограмм-функций без параметров, но с сохранением скобок после идентификатора функции, например:

```
CHARACTER *4 FUNCTION C( )
```

В качестве формального параметра в подпрограмме допускается символ *, которому соответствует фактический параметр в виде *M, где M — метка оператора в вызывающей программе. Оператор возврата в этом случае имеет вид RETURN I, где I — целое выражение, и обеспечивает несколько точек выхода из подпрограммы. Для подпрограммы без параметров допустимы две формы записи: SUBROUTINE S и SUBROUTINE S(). Может быть указано несколько точек входа во внешние процедуры оператором ENTRY.

В операторе DATA не требуется совпадения типа переменной и типа соответствующего ей значения, преобразование будет происходить так, как если бы переменная и значение участвовали в операторе присваивания. Начальные значения могут быть присвоены не всем элементам массива, а лишь части его. Для этого в списке переменных указывается конструкция вида M(I1) : M(I2), где M — идентификатор массива, I1, I2 — индексы. Такая конструкция означает, что значения должны быть присвоены всем элементам массива M, начиная с элемента с индексом I1 и заканчивая элементом с индексом I2.

Допускается описание нескольких подпрограмм данных. В этом случае после оператора BLOCK DATA указывается имя, например:

```
BLOCK DATA LIST1
```

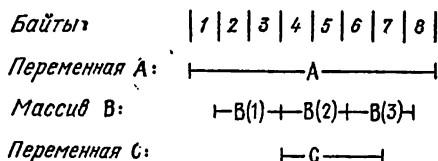
Имя подпрограммы данных может быть указано в списке оператора EXTERNAL.

В операторе эквивалентности разрешено использование идентификаторов массивов и идентификаторов текстовых переменных, текстовых массивов, элементов текстовых массивов, имен подстрок. Если в операторе EQUIVALENCE указано имя массива, соответствие устанавливается точно так же, как если бы был задан его первый элемент. Текстовые величины можно совместить при помощи эквивалентности только с другими текстовыми величинами. Совмещаются первые байты памяти, занятые объектами, входящими в список эквивалентности оператора EQUIVALENCE, что приводит также к совмещению других текстовых элементов. Например, операторы

CHARACTER A *8, B(3) *2, C *3

EQUIVALENCE (A(2 : 7), B(1)), (B(2), C)

расположат переменные в памяти следующим образом:



Не разрешается для определения эквивалентности задавать *n*-мерный массив как одномерный. В стандарте Фортрана 1966 г. это допускалось.

Ввод и вывод информации определен как для файлов последовательного, так и прямого доступа. Определена также внутренняя передача данных (из одной области оперативной памяти в другую) как передача внутренних файлов-областей памяти, описанных как текстовая переменная, текстовый массив или подстрока. Передача данных с внешних носителей — передача внешних файлов осуществляется через канал, который задается в виде целого положительного или нулевого значения.

Имеется две формы оператора ввода READ; вывод осуществляется операторами WRITE и PRINT:

READ (N) L WRITE (N) L
 READ N2, L PRINT N2, L

где N — список управляющей информации, L — список ввода-вывода, N2 — формат (элемент списка N). Элементами N являются: UNIT=N1 — канал, FMT=N2 — формат, REC=R — номер записи, IOSTAT=IS — код ответа, END=N3 — возврат по концу файла, ERR=N4 — возврат по ошибке. Первые два элемента называются позиционными параметрами, их место в списке строго определено, остальные параметры являются ключевыми и могут быть расположены в произвольном порядке, их присутствие в списке N необязательно. Не обязательно также указывать конструкции

UNIT= и FMT=. N1 может быть целым неотрицательным выражением (номер внешнего канала), * (внешний системный канал, предварительно подсоединенный для ввода-вывода), текстовой величиной (идентификатор внутреннего файла); N2 — это либо метка оператора FORMAT, либо целая переменная, которой присвоено значение метки оператора FORMAT, либо имя текстового массива, либо текстовое выражение определенного вида, либо символ *, означающая свободный формат (свободный формат подразумевается и в том случае, если отсутствует элемент FMT=N2); R — целое положительное выражение, указывающее номер записи при вводе-выводе с прямым доступом (отсутствие элемента REC=R подразумевает последовательный доступ); IS: — целая переменная или элемент целого массива, получает значение в конце операции ввода-вывода: 0 — в случае отсутствия ошибки, положительное значение, если произошла ошибка, отрицательное значение, если встретилась запись конца файла; N3, N4 — метки операторов, куда передается управление в случаях встречи конца файла или ошибки соответственно.

Операторы READ N2, L и PRINT N2, L реализуют возможность обмена данными со стандартными системными устройствами без явного указания номера канала.

В списке ввода-вывода могут содержаться текстовые величины. Свободный формат означает, что данные вводятся так, как они представлены на внешнем носителе. Значения данных на внешнем носителе отделяются друг от друга запятой или пробелами и должны иметь вид, допустимый для ввода в языке Фортран. Можно использовать коэффициент повторения в виде K*, где K — целая константа без знака (не нуль). При выводе элементов списка вывода записываются в одну или несколько последовательных записей в некотором подходящем формате, который зависит от реализации. Выводимые значения разделяются запятыми или пробелами.

Установление связи между каналом и файлом (подсоединение) выполняется оператором открытия файла OPEN. Канал может быть отсоединен оператором закрытия файла CLOSE.

Оператор открытия файла имеет вид

OPEN (ON)

где ON — список параметров, элементами которого являются: UNIT=N1 — канал, IOSTAT=IS — код ответа, ERR=N4 — возврат по ошибке, FILE=F — файл, STATUS=S — диспозиция, ACCESS=A — доступ, FORM=FM — форматность, RECL=RL — длина записи, BLANK=B — пробел. Параметр UNIT=N1 обязательный, конструкция UNIT= может быть опущена, остальные параметры могут отсутствовать. Смысл обозначений N1, IS, N4 приведен выше, F — текстовое выражение, задающее имя файла,

который следует подсоединить, S — текстовое выражение с одним из значений OLD (старый), NEW (новый), SCRATCH (временный), UNKNOWN (неизвестный), характеризующих файл, A — значения DIRECT (прямой) или SEQUENTIAL (последовательный), по умолчанию принимается SEQUENTIAL, FM — значения FORMATTED (форматный) или UNFORMATTED (бесформатный), по умолчанию принимается FORMATTED для файлов последовательного доступа и UNFORMATTED — для файлов прямого доступа, RL — целое положительное выражение, обозначающее длину записи в байтах для форматного файла и в единицах, зависящих от транслятора, для бесформатного файла, имеет смысл лишь для файлов прямого доступа, B — текстовые значения либо NULL (пусто), либо ZERO (нуль), по умолчанию выбирается NULL, значение NULL означает, что при вводе числовых значений пробелы надо игнорировать, значение ZERO — все пробелы, кроме начальных, нужно воспринимать как нули.

Оператор закрытия файла имеет вид

CLOSE (CN)

где CN — список параметров, элементами которого являются: UNIT=N1 — канал, IOSTAT=IS — код ответа, ERR=N4 — возврат по ошибке, STATUS=S — диспозиция. Параметр UNIT=N1 обязательный, конструкция UNIT= может быть опущена, остальные параметры могут отсутствовать. Смысл обозначений N1, IS, N4 приведен выше, S — текстовое значение с одним из выражений KEEP (сохранить), DELETE (уничтожить). По умолчанию для файлов, которые были открыты как SCRATCH, принимается значение DELETE, для всех остальных — KEEP.

Для получения информации о свойствах и текущем состоянии файла или канала служит оператор

INQUIRE (IN)

где IN — список параметров, в который входит либо параметр UNIT=N1 (конструкция UNIT= может быть опущена), если оператор относится к каналу с номером N1, либо параметр FILE=F, если необходимо получить информацию о файле с именем F. Остальные параметры следующие: IOSTAT=IS — код ответа, ERR=N4 — возврат по ошибке, EXIST=EX — существует?, OPENED=OP — открыт?, NUMBER=NU — номер канала, NAMED=NA — с именем?, NAME=NE — имя, ACCESS=A — доступ, SEQUENTIAL=SE — последовательный?, DIRECT=DI — прямой?, FORM=FM — форматность, FORMATTED=FO — форматный?, UNFORMATTED=UF — бесформатный?, RECL=RL — длина записи, NEXTREC=NR — следующая запись, BLANK=B — пробел. В зависимости от того, какими свойствами обладает файл или канал, входящие в параметры выражения (справа от символа =), получают

соответствующие значения. Каждый из параметров может входить в оператор INQUIRE не более одного раза.

Операторы управления файлами REWIND, BACKSPACE, END FILE могут содержать в качестве списка либо номер канала N1, либо конструкцию (UNIT=N1, IOSTAT=IS, ERR=N4), где смысл входящих параметров был пояснен выше, в частности, N1 может быть целым выражением (конструкция UNIT= может быть опущена).

В список спецификаций оператора FORMAT введены дополнительные спецификации, управляющие табуляцией (T, TL, TR), выводом знака (S, SP, SS) и вводом пробелов (BN, BZ).

T-спецификация имеет вид Tw, где w — целая ненулевая константа без знака, и указывает, что (при выводе) следующий символ будет выведен в позицию номер w в записи или (при вводе) следующий символ будет взят из позиции номер w в записи. TL= и TR-спецификации имеют вид TLw и TRw и указывают, что текущую позицию во внешней записи данных следует переместить на w позиций назад (т. е. влево) или вперед (т. е. вправо) соответственно.

Спецификации S, SP, SS управляют выводом знака плюс для чисел и влияют на действие спецификаций преобразования I, F, E, D и G. Действие спецификации S, SP или SS продолжается до тех пор, пока не встретится другая спецификация управления выводом знака. При вводе эти спецификации игнорируются. S-спецификация указывает, что следует вернуться к стандартному режиму проставления знака плюс, SP-спецификация обеспечивает вывод знака плюс везде, где он возможен, а SS-спецификация указывает, что везде, где возможно, знак плюс следует опускать.

Спецификации управления вводом пробелов указывают, как следует при вводе чисел интерпретировать пробелы, кроме старших пробелов. Режим ввода пробелов устанавливается оператором OPEN или по умолчанию. Спецификации BN и BZ, если они встретятся в операторе FORMAT, изменяют предыдущий режим. BN-спецификация указывает, что пробелы следует игнорировать, а остальные символы прижимать к правому краю поля, как если бы все опущенные пробелы располагались с левого края поля. Поле, состоящее из одних пробелов, соответствует нулевому значению. BZ-спецификация обеспечивает интерпретацию пробелов нулями.

Допускается вариант спецификации I вида Iw.d. При вводе спецификации Iw.d действует так же, как Iw. При выводе отличие от Iw в том, что выводится не менее d цифр — при этом могут появиться незначащие нули в начале числа. Если выводимое значение равно нулю и d=0, то поле вывода будет состоять целиком из пробелов.

Имеется вариант E-спецификации вида Ew.dEe. При вводе редактирование для этой спецификации выполняется так же, как и

для Ew.d. При выводе отличие в том, что порядок числа будет содержать цифру. Вариант G-спецификации вида Gw.dEe при вводе эквивалентен Gw.d. При выводе Gw.dEe эквивалентна Fw.d, если $0.1 \leq K < 10^d$, и Ew.dEe, если $K < 0.1$ или $K \geq 10^d$. Здесь K — абсолютная величина элемента данных.

В качестве самостоятельной спецификации редактирования или в качестве разделителя в списке форматов можно использовать символ : (двоеточие). Если двоеточие встретилось при выводе, когда в списке вывода не осталось ни одного элемента, то завершается выполнение операции вывода. Если же двоеточие встретилось при вводе или при выполнении операции вывода в списке вывода еще остались элементы, двоеточие игнорируется.

2. Ограничения по сравнению с Фортраном IV. Некоторые элементы и конструкции языка Фортран IV не вошли в новый стандарт Фортран 77. К ним относятся:

- символ &; символ § относится к специальным знакам;
- указатель длины *S в операторах REAL, INTEGER, LOGICAL, COMPLEX, IMPLICIT, FUNCTION; для указания двойной точности действительных величин используется оператор DOUBLE PRECISION;

- присваивание начальных значений при описании данных в операторах REAL, INTEGER, LOGICAL, COMPLEX;

- стандартные функции COTAN(X), DCOTAN(D), HFIX(X), REAL(C), CLOG10(C), ERF(X), DERF(D), ERFC(X), DERFC(D), GAMMA(X), DGAMMA(D), ALGAMA(X), DLGAMA(D), DCMPLX(D1, D2), DCONJG(CD), CDSIN(CD), CDCOS(CD), CDSQRT(CD), CDEXP(CD), CDLOG(CD), CDLG10(CD), CDABS(CD);

- операторы DEFINE FILE, NAMELIST, FIND, PUNCH; различен способ записи операторов ввода-вывода файлов с прямым доступом;

- Z-спецификация формата; в списке форматов после X- и H-спецификаций требуется разделитель.

3. Отличия Фортрана IV. Следующие элементы языка Фортран 77 отсутствуют в Фортране IV:

- символ алфавита : (двоеточие);
- оператор CHARACTER;
- понятие подстроки;
- операция конкатенации //;
- оператор PARAMETER;
- оператор присваивания для текстовых переменных и выражений;

- логические операции .EQV. и .NEQV.;
- оператор PROGRAM;
- отрицательный и нулевой индексы у элементов массивов; граничная пара при определении границ индексов;

- оператор SAVE;
- переменная из оператора ASSIGN для указания оператора FORMAT;
- пустая строка и символ * в первой позиции строки для указания комментария;
- целое выражение M в вычисляемом операторе передачи управления GO TO (N), M;
- отсутствие списка (N) в присваиваемом операторе GO TO;
- конструкция IF—THEN — ELSE;
- выражения целого и действительного типов и двойной точности в качестве компонент заголовка цикла;
- выполнение цикла 0 раз, если $N1 > N2$, $N > 0$ (или $N1 < N2$, $N < 0$);
- сохранение параметром цикла последнего присвоенного ему значения при любом выходе из цикла;
- стандартные функции ICHAR(T), CHAR(K), DINT(D), ANINT(X), DNINT(D), NINT(X), IDNINT(D), DDIM(D1, D2), DPROD(X1, X2), LEN(T), INDEX(T1, T2), LGE(T1, T2), LGT(T1, T2), LLE(T1, T2), LLT(T1, T2);
- универсальные имена;
- оператор INTRINSIC;
- оператор-функции и подпрограммы-функции без параметров; начальная строка подпрограммы-функции без параметров имеет вид **ТИП FUNCTION F()**;
- оператор возврата RETURN I, где I — целое выражение;
- отсутствие требования в операторе DATA, совпадения типа переменной и типа соответствующего ей значения; возможность присвоения начальных значений части массива;
- наличие имени у подпрограммы данных;
- имя массива и текстовая величина в группе эквивалентности оператора EQUIVALENCE;
- внутренняя передача данных;
- операторы OPEN, CLOSE, INQUIRE; различен способ записи операторов ввода-вывода файлов с прямым доступом;
- целые выражения в операторах REWIND, BACKSPACE, ENDFILE;
- спецификации формата TL, TR, S, SP, SS, BN, BZ, атрибуты d в спецификации I и Ee в E- и G-спецификациях;
- двоеточие в качестве разделителя в списке форматов.

§ 7. Бейсик-плюс для СМ ЭВМ

Язык Бейсик-плюс разработан в 1975 г. специалистами фирмы DEC (Digital Equipment Corporation, США). По сравнению с используемой в то время первоначальной версией языка Бейсик

новый вариант имел существенные расширения. В составе операционной системы разделения времени ДОС РВР, эксплуатируемой на ЭВМ типа СМ-4, используется модифицированная версия алгоритмического языка Бейсик-плюс, основными характерными особенностями которой являются:

- матричные операции, исполняемые набором матричных операторов;
- операции с символьными строками, аналогичные операциям с цифровыми данными;
- операции с файлами (управление данными и хранение данных);
- средства редактирования программы;
- режим немедленной обработки.

В алфавите языка Бейсик-плюс, реализованном на машинах серии СМ ЭВМ, используются 93 символа. Этот набор лишь незначительно отличается от символов, приведенных в § 1 гл. III. Имеются особенности в использовании тех или иных символов. Так, буквы русского алфавита (за исключением Ё и Ъ) употребляются, главным образом, как элементы символьных строк. Возведение в степень обозначается наряду с символом \wedge также \uparrow или **, символьные данные обозначаются знаком \square (или \$), комментарий может быть указан знаком!, а написание подряд двух знаков = означает приблизительно равно (так величины А и В, входящие в отношение $A \approx B$, могут отличаться в последнем десятичном разряде). В качестве разделителя двух операторов в одной строке программы наряду с двоеточием используется символ \.

Константы и переменные различают трех типов: целые, действительные и текстовые. Действительные величины могут иметь обычную или двойную точность в зависимости от параметров, установленных при генерации операционной системы. Целые константы или переменные имеют в качестве последнего символа в их записи символ %, текстовые переменные обозначаются идентификатором, заканчивающимся символом \square (или \$). Идентификатор состоит либо из одной латинской буквы, либо из латинской буквы с одной последующей цифрой. Количество индексов у элемента массива не более двух. Минимальное значение индекса равно нулю, максимальное значение индексного выражения — 32767. К действительным константам относится константа, обозначаемая как PI (значение $\pi = 3.1415927$).

Операция отношения $=$ допустима и над переменными строкового типа. Так, выражение $X\square = Y\square$ имеет значение «истина», если значения переменных $X\square$ и $Y\square$, т. е. соответствующие строки, идентичны. Отметим, что при сравнении символьных строк (за исключением случая сравнения с помощью операции $=$) пробелы в конце строк игнорируются. Например, выражение "TEMP" =

"TEMP□" имеет значение «истина», т. е. строки "TEMP" и "TEMP□" эквивалентны, но не идентичны, так как отношение "TEMP"= = "TEMP□" имеет значение «ложь».

Допускается описание процедуры-функции оператором DEF, содержащее несколько операторов и заканчивающееся оператором FNEND. Оператор DIM может быть использован для описания виртуального массива как разновидности файла прямого доступа, размещаемого в дисковой памяти, где для него выделяется необходимый участок в соответствии с объявленными в операторе DIM параметрами. Рассматриваемый оператор имеет вид

DIM#N, список массивов

где N — целая константа, являющаяся внутренним номером файла, а список массивов имеет обычный вид. Например

10 DIM#2%, A(20), B□(10, 10)

По умолчанию максимальная длина элемента виртуального массива строковых значений принимается равной 16 символам. В то же время эта длина может быть явно указана после описания массива через символ =. Например:

20 DIM#1 C□(30)=32

Использование идентификаторов виртуальных массивов и их элементов в программе ничем не отличается от обращения к обычным внутренним массивам.

В операторе присваивания в списке левой части между различными переменными используется разделитель запятая, например:

30 V1, V2, V3=E

Ключевые слова операторов LINE INPUT и LINE INPUT # имеют вид INPUT LINE и INPUT LINE # соответственно.

Из математических функций дополнительными в Бейсик-плюс являются FIX(X), осуществляющая выделение целой части своего аргумента, и LOG10(X), вычисляющая десятичный логарифм. Функция выдачи случайного числа RND может употребляться как с аргументом, так и без него. Для перенастройки генератора случайных чисел используется оператор RANDOMIZE.

Для обработки текстов используются функции, результат выполнения каждой из которых поясняется ниже после имени соответствующей функции:

INSTR (I, A□, B□) — вхождение B□ в A□, начиная с символа переменной A□ с номером I;

LEN(A□) — длина строкового значения;

LEFT(A□, N) — левая подстрока из N символов;

MID(A□, I, N) — центральная подстрока из N символов, начиная с I-го;

RIGHT (A□, N) — правая подстрока из N символов;

SPACE \square (N) — строка из N пробелов;

STRING \square (N, K) — строка из N одинаковых символов, код которых в стандарте ASCII есть K.

Для преобразования типов данных предусмотрены следующие функции:

ASCII(A \square) — код в стандарте ASCII (целое число) первого символа значения A \square ;

CHR \square (K) — символ, код которого в стандарте ASCII равен K;

CVT% \square (I) — символьная строка из двух символов, полученная в результате изменения формата представления в памяти целого числа I;

CVTF \square (R) — символьная строка из четырех (восьми) символов, полученная в результате изменения формата представления в памяти действительного числа R;

CVT \square %(A \square) — целое число, полученное в результате преобразования первых двух знаков символьной строки A \square ;

CVTF(A \square) — действительное число, полученное в результате преобразования первых четырех (восьми) знаков символьной строки A \square ;

CVT \square \square (A \square , I) — обеспечивает редактирование текстового значения A \square в соответствии со шкалой I ($1 \leq I \leq 383$);

NUM \square (E) — обеспечивает представление числового значения выражения E в виде строки символов;

VAL(A \square) — машинный формат числа, соответствующий его представлению в виде строки A \square ;

XLATE(A \square , B \square) — строка символов, полученная в результате перекодировки исходной строки A \square с помощью словаря B \square .

Для программирования временных процедур используются три системные функции с именами DATE \square , TIME \square и TIME и два оператора с ключевыми словами SLEEP и WAIT. Упомянутые функции имеют следующий формат и соответствующие значения:

DATE \square (0) — текущая дата в виде строки символов вида "ЧЧ—ММ—ГГ" (число—месяц—год);

DATE \square (N%) — календарная дата с номером N%, заданным относительно даты 01.01.1970;

TIME \square (0) — текущее время дня в виде строки символов вида "ЧЧ—ММ" (часы—минуты);

TIME \square (N%) — астрономическое время для момента за N% минут до полуночи;

TIME(0) — текущее время дня в секундах;

TIME(1) — время процессора, затраченное на решение задачи, в десятых долях секунды;

TIME(2) — время сеанса, исчисляемое в минутах от момента входа пользователя в систему;

TIME(3) — количество обращений к ячейкам оперативной памяти, состоявшееся с момента запуска задачи, в тысячах обращений.

Оператор SLEEP N приостанавливает выполнение программы на N секунд, а с помощью оператора WAIT N устанавливается предельное время в секундах, отводимое пользователю на ввод данных по запросу оператора INPUT. Если этого времени оказалось недостаточно для набора необходимых значений, происходит прерывание и выдается сообщение об ошибке с кодом ERR=15.

При выполнении оператора цикла FOR—NEXT в случае недопустимого значения параметра цикла тело цикла обходится, т. е. цикл не выполняется ни разу, а параметр цикла сохраняет свое начальное значение. При выходе из цикла параметр цикла всегда сохраняет последнее значение, при котором цикл исполнялся. Допускаются операторы цикла FOR—WHILE и FOR—UNTIL.

В программе, написанной на языке Бейсик—плюс, может быть предусмотрена подпрограмма анализа и обработки возникших при ее выполнении ошибок. Управление на подпрограмму анализа ошибок при их устранении осуществляется оператором

ON ERROR GO TO N

где N — номер начальной строки подпрограммы анализа и обработки ошибок. Этот оператор используется для назначения точки входа (N), поэтому должен быть выполнен до обнаружения ошибки. В противном случае обнаруживаемые ошибки обрабатываются системой или выполнение программы прекращается после выдачи соответствующего диагностического сообщения. Операторы

ON ERROR GO TO 0 или ON ERROR GO TO

отменяют действие оператора передачи управления на подпрограмму анализа и обработки ошибок и возлагают обработку обнаруживаемых ошибок на операционную систему.

В подпрограмме анализа и обработки ошибок используются переменные целого типа ERR и ERL. При выявлении ошибки переменная ERR принимает значение, равное коду ошибки, а переменная ERL — значение, равное номеру строки, при выполнении операторов которой произошла ошибка.

Возврат из подпрограммы анализа и обработки ошибок в основную программу в строку, выполнение оператора в которой привело к ошибке, осуществляется оператором RESUME. Если же выполнение основной программы должно быть продолжено, начиная с некоторой другой строки с номером N, то этот номер должен быть указан в операторе возврата, т. е. оператор имеет вид: RESUME N.

При обмене с файлами допускаются матричные операторы MAT INPUT#N и MAT PRINT#N, где N — номер файла. При выполнении оператора MAT—INV переменной DET присваивается значение определителя обращаемой матрицы,

С целью указать необходимость условного или многократного выполнения операторов в языке Бейсик—плюс предусмотрено пять модификаторов

```
IF L
WHILE L
UNTIL L
UNLESS L
FOR I=E1 TO E2 STEP E3
```

которые указываются после оператора. Здесь L — условие, I — параметр цикла, E1, E2, E3 — соответственно нижняя граница, верхняя граница и шаг изменения параметра цикла.

Применение модификатора обеспечивает выполнение оператора до тех пор, пока на него распространяется действие модификатора. Например, в результате выполнения оператора

```
110 A=25 IF X=0
```

переменной A присваивается значение 25 лишь тогда, когда $X=0$. Оператор

```
180 X=X**2 WHILE X<640
```

выполняется (многократно) до тех пор, пока значение X не превосходит 640.

§ 8. Интерпретатор MINIBAS ОС МикроДОС

Интерпретатор MINIBAS ориентирован на вычислительные системы, обладающие малой памятью. Он предназначен для решения математических, инженерных, обучающих и игровых задач. Объем памяти составляет 8 Кбайт. Типы данных, поддерживаемые в этой версии, соответствуют указанному кругу задач. Пользователю предоставляется значительное число команд, операторов и функций, причем имеющиеся ограничения на типы данных, числовые операции и синтаксис языка облегчают изучение языка и упрощают программирование и редактирование программ для неподготовленных пользователей.

Основное ограничение интерпретатора MINIBAS заключается в том, что разрешается использовать только десятичные данные. В то же время предоставляются дополнительные возможности работы с портами ввода-вывода, памятью, внутренним стеком и специальным стеком ловушки ошибки, а также могут быть вызваны подпрограммы на машинном языке.

Вызов интерпретатора осуществляется командой MINIBAS. Если в этой команде в виде строковой константы без кавычек указывается также имя файла, то будет загружаться наряду с интерпретатором MINIBAS и программа с данным именем.

Константы различают числовые и строковые, переменные — только числовые. Идентификатор переменной может быть длиной от одного до двух символов. Первый символ в имени должен быть латинской буквой, а второй символ — цифрой. Допускаются только одномерные массивы. Отсутствуют логические и строковые операции.

Для обозначения команд, выполняемых только в прямом режиме, используются ключевые слова: ERA, LIST, LLIST, NAME, NEW, NULL, OLD, RUN, SAVE, SYSTEM, UNSAVE.

Команды с ключевыми словами ERA и UNSAVE выполняют одинаковые действия — стирают с диска файл с заданным именем. Имя файла обязательно должно включать расширение. Например:

```
ERA PROGRAM.BSC
UNSAVE PROG.BSC
```

Команда NULL N определяет количество N нулевых символов, напечатанных в конце строки за возвратом каретки.

Операторы версии языка Бейсик в интерпретаторе MINIBAS обозначаются следующими ключевыми словами: CLEAR, DATA, DIM, END, FOR, GOSUB, GOTO, IF, INPUT, LET, LPRINT, NEXT, OUT, POKE, PRINT, PUSH, READ, REM, RESTORE, RETURN, STOP, TRAP.

Оператор CLEAR устанавливает для всех числовых переменных нулевые значения и освобождает всю память, используемую для данных, не стирая текущей программы. После выполнения оператора CLEAR массивы не определены, их следует описывать после CLEAR. Например:

```
120 CLEAR
130 DIM M2 (50)
```

Оператор

```
OUT P [N1, N2,..., NN]
```

где P — номер порта в диапазоне 0÷255; N1, N2,..., NN — байты данных в диапазоне от 0 до 255, служит для размещения байтов данных N1, N2,..., NN в специальный порт ввода-вывода. Например,

```
10 OUT 180 [2, 43]
```

Оператор

```
POKE A [N1, N2,..., NN]
```

где A — адрес памяти (целое число в диапазоне 0÷65535); N1, N2,..., NN — байты данных в диапазоне 0÷255, заносит в память байты, начиная с адреса A. Например:

```
20 POKE 106 [58, 92, 31]
```

Этот оператор используется при хранении данных, загрузке подпрограмм на машинном языке, передаче аргументов и результатов в подпрограммы на машинном языке или передаче из таких подпрограмм.

Операторы

PUSH A1, A2,..., AN

TRAP A1, A2,..., AN

где A1, A2,..., AN — адреса памяти в диапазоне от 0 до 65535, заносят эти адреса соответственно во внутренний и специальный стеки. Обеспечивается проверка переполнения стека. Последний адрес в списке будет вершиной стека, т. е. он будет считываться первым. Например:

130 PUSH 146, 253, 97, 326

В случае оператора TRAP занесенные в специальный стек ловушки значения обрабатываются как номера строк. Когда встречается ошибка, значение вершины будет выбираться из стека ловушки, управление будет передаваться оператору, соответствующему этому номеру строки. Если стек пустой, то сообщение об ошибке будет печататься как обычно.

В интерпретаторе MINIBAS используются стандартные функции с идентификаторами: ABS, ARG, CALL, COS, INP, INT, PEEK, POP, RND, SGN, SIN, SQR, TAB, TAN, UNTRAP. Из них отличными от общепринятых являются следующие функции: ARG, CALL, INP, PEEK, POP, UNTRAP.

Функция CALL(N) вызывает подпрограмму на машинном языке в адрес N и загружает пару специальных регистров последним значением, переданным функцией ARG(N), где N — целое число из диапазона 0÷65535. Например:

120 Y=ARG(100)

130 CALL(7000)

Функция INP(P), где P — целое число в диапазоне 0÷255, имеет результатом байт, считанный из порта с адресом P. Размещение байтов в порте осуществляется оператором OUT.

Результатом выполнения функции PEEK(A) является байт, считанный из адреса A памяти. Значение этого байта есть целое число в диапазоне 0÷255. Адрес A — также целое число в диапазоне 0÷65535. Размещение байтов в адресах памяти выполняет оператор POKE.

Функция POP(N) считывает N значений из внутреннего стека, куда они были занесены оператором PUSH и запоминает последнее значение, которое и будет результатом функции. Действие функции UNTRAP(N) аналогично действию функции POP, только значение запоминается из специального стека ловушки ошибки.

§ 9. Версия интерпретатора BASIC

Версия языка Бейсик, поддерживаемая интерпретатором BASIC, предназначена для решения более широкого круга задач по сравнению с рассмотренной в § 8 этой главы версией интерпретатора MINIBAS. Интерпретатор BASIC занимает 24 Кбайт памяти. Эта версия имеет большие возможности, при этом сохраняются принципы построения программы и синтаксис языка, так что она является расширением версии языка, поддерживаемой интерпретатором MINIBAS. Интерпретатор позволяет использовать восьмеричные и шестнадцатеричные числа, работать со строковыми данными и файлами данных, представляет возможности редактирования, управления форматом печати, работы с подпрограммами.

Для вызова интерпретатора необходимо ввести команду BASIC, в которой может быть указано имя файла программы, загружаемой или выполняемой. Это — строковая константа, не заключенная в кавычки. Например,

```
BASIC FILE.ASC
```

При таком вызове по умолчанию устанавливается число файлов, которые могут быть открыты в любое время в течение работы интерпретатора — 3 (каждый блок файла данных требует 166 байт памяти); максимальное число байт, которое может быть использовано как рабочее пространство интерпретатора — 64 Кбайт; максимальный размер записи для файлов с произвольным доступом — 128 байт. Принимаемые по умолчанию значения могут быть изменены путем явного задания в команде BASIC после имени файла трех параметров в виде списка /F : N1/M : N2/S : N3, где N1, N2, N3 — соответственно число файлов, рабочее пространство и размер записи. Отметим, что максимальное значение N1 может быть 6. Например, при вызове интерпретатора

```
BASIC PROG/F : 6/M : 32768
```

используется шесть файлов, 32 Кбайт памяти и выполняется программа PROG.

В имени файла первыми символами могут быть A:, B:, C: и т. д., что определяет дисковод. Если дисковод не определен, то по умолчанию выбирается текущий дисковод. Если имя файла не содержит точки, то по умолчанию используется расширение BAS. Например имена PROG и PROG.BAS эквивалентны.

Интерпретатор BASIC использует все команды, описанные в гл. III. В дополнение к этому опишем особенности команд CLEAR и RESET, также используемые в данном интерпретаторе.

Команда CLEAR имеет вид

```
CLEAR, N1, N2
```

где N1 — счетчик байтов, который устанавливает максимальное число байт, доступных интерпретатору для запоминания программы и данных, т. е. максимально доступный адрес; N2 — число байт, установленное для стекового пространства (по умолчанию N2 устанавливается меньшее из чисел — 1000 или 1/8 доступной памяти). Эта команда присваивает всем числовым переменным нулевые значения, а строковым переменным — значения нулевой длины, устанавливает конец памяти и количество стекового пространства и освобождает всю память, используемую для данных, не стирая текущей программы.

После выполнения команды CLEAR числовые переменные принимают нулевые значения, строковые переменные имеют нулевую длину и очищаются, массивы не определены и любая информация, определенная операторами DEF, теряется. Чтобы зарезервировать память для программы на машинном языке, необходимо задать параметр N1. При использовании в программе большого количества операторов GOSUB или циклов FOR—NEXT следует задать параметр N2. Примеры:

```
CLEAR
```

```
CLEAR, 32768, 2000
```

Команда RESET используется для общего сброса и перезагрузки системы. Закрываются все файлы и очищается системный буфер. Если все открытые файлы находятся на диске, то команда RESET действует так же, как команда CLOSE без указания номера файла.

Большинство операторов могут быть использованы как команды в прямом режиме. Ниже будет дана краткая характеристика некоторых операторов, не имеющих применения во многих версиях Бейсика, но реализованных интерпретатором BASIC.

Оператор

```
CALL V(X1, X2,..., XN)
```

вызывает подпрограмму на машинном языке, начальный адрес которой есть значение числовой переменной V. Переменные X1, X2, ..., XN передаются как аргументы в подпрограмму на машинном языке.

Оператор

```
DEF USRN=E
```

где N — любая цифра от 0 до 9, E — целочисленное выражение в диапазоне 0÷65535, определяет начальный адрес подпрограммы на машинном языке, которая позднее вызывается функцией USR. Цифра N соответствует номеру подпрограммы USRN, адрес которой определяется. В случае отсутствия N предполагается, что N=0. Значение адреса может задаваться в десятичном и шестнадцатеричном формате и является действительным начальным адресом под-

программы **USR**. Для переопределения начального адреса подпрограммы в программе может появляться любое число операторов **DEF USR**, так как разрешается доступ к стольким подпрограммам, сколько необходимо. Например:

```
40 DEF USR1=7000
600 X=USR1(X+100)
```

Оператор **ERROR E**, где **E** — целочисленное выражение в диапазоне 0÷255, моделирует появление ошибки интерпретатором или распознает коды ошибок, которые определяет пользователь. Если **E** — код ошибки, то оператор **ERROR** будет моделировать появление этой ошибки, и если оператором **ON ERROR** определена подпрограмма обработки ошибки, то будет осуществляться переход на эту подпрограмму, в противном случае будет выдано сообщение об ошибке, и выполнение программы будет остановлено. Для определения собственного кода ошибки используется значение, которое отличается от значений, используемых для этой цели интерпретатором, поэтому рекомендуются значения больше 200. Если собственная ошибка пользователя определяется таким способом и не обрабатывается в подпрограмме обработки ошибок, то будет выдано сообщение «неопределенная ошибка» и выполнение программы будет остановлено.

Операторы **ON ERROR** и **RANDOMIZE** используются точно так же как в языке Бейсик-плюс, а оператор **RESUME** имеет три модификации

```
RESUME 0 или RESUME
RESUME NEXT
RESUME N
```

Этот оператор обеспечивает выполнение программы после процедуры устранения ошибок. В первой модификации выполнение восстанавливается с оператора, который вызвал ошибку, во второй — с оператора, следующего непосредственно за оператором, который вызвал ошибку, и в последнем случае выполнение программы восстанавливается с оператора с заданным номером **N** строки. Оператор **RESUME**, для которого нет подпрограммы обработки ошибки, вызывает сообщение об ошибке.

Порядок выполнения операторов **NULL**, **POKE** и **OUT** описан в § 8 этой главы. Имеются лишь отличия в формате записи операторов **POKE** и **OUT**; а именно

```
POKE N, M
OUT P, M
```

где **N** — адрес памяти, число в диапазоне 0÷65535; **P** — номер порта в диапазоне 0÷255, **M** — байт данных, число в диапазоне от 0

до 255. Эти операторы используются для записи байта данных *M* соответственно по адресу памяти *N* и в выходной порт *P*.

Оператор, обозначаемый ключевым словом **RESET**, закрывает все файлы и очищает системный буфер. Если все открытые файлы находятся на диске, то оператор **RESET** действует так же, как оператор **CLOSE**, в котором не задан номер файла.

Для обмена значений двух переменных *X* и *Y* используется оператор

SWAP X, Y

Переменные *X* и *Y* могут иметь любой тип, но обе переменные должны быть одного типа. Допускается обмен значениями между элементами массива. Например, после выполнения операторов

30 **A=25 : B=10**

40 **SWAP A, B**

переменная *A* имеет значение 10, а переменная *B* — значение 25.

Имеется возможность приостанавливать выполнение программы пока обрабатывается состояние входного порта машины, т. е. пока входной порт машины не выработает определенного сигнала (образа бита). Для этой цели используется оператор

WAIT P, N, M

где *P* — номер порта в диапазоне 0÷255, *N*, *M* — целочисленные выражения. Величина *M* может быть опущена, в этом случае полагается *M* равным нулю. Данные, считанные в порт, подвергаются операции **XOR** с выражением *M*, а затем операции **AND** с выражением *N*. Если результат равен нулю, то выполнение программы продолжается со следующего оператора.

Применять оператор **WAIT** следует с осторожностью, так как с помощью этого оператора можно войти в бесконечный цикл, для останова которого необходимо перезагружать вычислительную машину.

С помощью операторов

WRITE список выражений

WRITE #N, список выражений

данные могут быть соответственно выведены на экран и записаны в файл с последовательным доступом, имеющий номер *N*, под которым файл был открыт для вывода. В операторе **WRITE** список выражений может быть опущен, тогда выводится строка пробелов. В обоих операторах выражения в списке могут быть числовыми и/или строковыми, они отделяются друг от друга запятыми или точками с запятой. Выводимые значения отделяются друг от друга запятыми, строки заключаются в кавычки, и положительным числам не предшествует пробел — в этом отличие от соответствующих

операторов PRINT и PRINT#N. После вывода последнего элемента списка выражений происходит возврат каретки (перевод строки).

Набор стандартных функций, реализуемых в интерпретаторе BASIC, практически совпадает со списком встроенных функций других наиболее полных версий языка Бейсик, приведенном в Приложении II.

Для сокращения объема памяти, занимаемой программой, и времени ее выполнения при работе с интерпретатором BASIC рекомендуется:

- стереть все ненужные операторы REM, оставить только необходимые;

- не использовать лишние пробелы в любых конструкциях (между операторами, операциями, и т. д.);

- при возможности размещать несколько операторов в каждой программной строке (каждая программная строка «стоит» дополнительно 5 байт);

- опускать в операторе присваивания ключевое слово LET;

- где возможно, пользоваться целыми переменными, так как целые числа занимают 2 байта, а действительные числа — 4 байта, кроме того, тратится время на округление чисел;

- если подпрограмма всегда вызывается из одного места в программе, то надо использовать оператор перехода GO TO, а не вызов подпрограммы, так как активный оператор GOSUB занимает 6 байт памяти, а оператор GO TO — ничего;

- в выражениях меньше использовать скобки, что позволяет экономить память за счет того, что не надо запоминать результаты, содержащиеся в скобках;

- экономить размер массива, так как незаполненные элементы все равно занимают память;

- вместо специальных символов, применяемых в идентификаторах для указания типа переменных, использовать операторы DEF;

- по возможности использовать переменные, а не константы, так как доступ к переменным занимает меньше времени, чем перевод констант в представление с плавающей точкой;

- использовать оператор POKE, что может ускорить графический вывод;

- описывать сначала наиболее используемые переменные, что позволит сократить временные затраты на отыскание переменной в памяти.

§ 10. Подмножество ПЛ/1 ОС ЕС ЭВМ

Версия языка ПЛ/1 для операционной системы ОС ЕС (ПЛ/1 ОС ЕС) представляет собой подмножество полного языка ПЛ/1. Транслятор с этой версии содержит оптимизирующий режим и соз-

дает рабочие программы, немного уступающие по своим качествам программам, составленным на языке Ассемблера. В обычно реализуемое подмножество не входит целый ряд элементов полного языка ПЛ/1, описание которого с незначительными сокращениями приведено в гл. IV. Ниже будут перечислены основные из не включенных в подмножество ПЛ/1 ОС ЕС элементов языка. К ним относятся:

- данные комплексного типа, выражения над комплексными данными и встроенные комплексные функции;
- выражения над массивами и структурами;
- возможность нахождения оператора DECLARE во внутренних процедурах и блоках;
- возможность нахождения в операторе ON не только единичного оператора, но и целого блока;
- дополнительные точки входа в процедуру, задаваемые оператором ENTRY;
- рекурсивное обращение к процедурам;
- описатель управления памятью STATIC;
- описатель управления памятью CONTROLLED;
- переменные типа области;
- ввод и вывод; управляемый данными;
- обработка записей в буфере операторами READ с дополнением INTO и оператором WRITE с дополнением FROM;
- обработка записей в буфере файла с помощью наложения на запись соответствующей базированной переменной с использованием оператора READ с дополнением SET и оператора LOCATE;
- файлы с описателями REGIONAL(2) и REGIONAL(3);
- телеобработка файлов, которые объявляются с описателем TRANSIENT и являются доступными с абонентских пунктов при использовании TCAM (общего телекоммуникационного метода доступа);
- параллельное выполнение нескольких задач, а также использование описателей EVENT, TASK, PRIORITY и оператора WAIT, позволяющих организовать параллельное выполнение процедур-подпрограмм и синхронизацию выполнения операций ввода-вывода и ряда других операций;
- использование средств препроцессора, позволяющее вносить изменения в текст программы непосредственно перед компиляцией, например возможность копировать хранящиеся в библиотеке фрагменты программы (объявления данных, тексты внутренних процедур и т. п.), изменять и добавлять части отдельных операторов.

Перечисленные элементы не являются запрещенными для транслятора с языка ПЛ/1, ограничения имеют лишь некоторые реализации подмножества ПЛ/1 ОС ЕС. Транслятор имеет определенные количественные ограничения, среди которых выделим следующие:

— размер одного оператора, кроме DECLARE, ограничен 3500 символами, что эквивалентно 50 перфокартам;

— максимальное число переменных в исходной программе зависит от полного размера словаря программы, который ограничен приблизительно 65 Кбайт; это эквивалентно ограничению приблизительно 1200 переменными для пользователя;

— количество параметров процедур должно быть не более 64;

— максимальная глубина уровня при объявлении структур — 63;

— размер блока должен быть не более 32760 байт;

— число блоков PROCEDURE, BEGIN, групп DO и операторов ON в сумме должно быть не более 255;

— в любой точке компиляции не должно быть больше 50 уровней гнездования; степень гнездования в данной точке определяется как число операторов PROCEDURE, BEGIN или DO без соответствующих операторов END, плюс число текущеактивных составных операторов IF, плюс число текущенепарных левых скобок, плюс число размерностей в каждом выражении над массивами, плюс максимальное число измерений в каждом активном выражении над структурами;

— максимальное число элементов, допускаемых в списке оператора ввода, управляемого данными, — 320;

— выдаваемое оператором DISPLAY сообщение имеет длину 72 символа, ответ — 127 символов;

— максимальный размер записи не должен превышать 32760 байт.

При трансляции исходной программы могут быть выданы четыре типа диагностических сообщений: предупреждение, ошибка, серьезная ошибка, ошибка окончания, которые записываются вслед за листингом исходной программы и всеми другими листингами.

§ 11. ПЛ/1 ДОС ЕС

Реализованный в операционной системе ДОС ЕС язык ПЛ/1 представляет собой подмножество полного языка ПЛ/1, в котором отсутствуют такие элементы полного языка как синхронное выполнение ветвей программы, предтрансляторная обработка исходного текста, передача потоком, управляемая данными, рекурсивные обращения к процедурам, возможность обработки комплексных данных, присваивания для структур с дополнением BY NAME и ряд других особенностей.

Отсутствует региональная организация файла REGIONAL(2). В то же время на магнитной ленте и магнитных дисках допускаются многофайловые тома и многотомные файлы. Описатель ENVIRONMENT(D) должен присутствовать в каждом объявлении файла.

В списке дополнений D этого описателя может быть указано большое количество режимов, описывающих те свойства и признаки файлов, которые связаны с особенностями ДОС ЕС.

На уровне входного языка в исходной программе можно выполнить ряд отладочных операций, среди которых укажем печать значений переменных, печать изменяющихся значений и печать переходов.

Печать значений переменных выполняется с помощью подпрограммы DYNDUMP, для вызова которой используется оператор

```
CALL DYNDUMP(FA);
```

где FA — список аргументов. Аргумент может быть скалярным выражением, именем массива или структуры. Значения, выдаваемые на печать, представляют собой шестнадцатеричную форму внутреннего представления указанных аргументов.

Печать изменяющихся значений выполняется после обращения к подпрограмме IJKEHHC оператором

```
CALL IJKEHHC (FA);
```

где элемент списка аргументов FA может быть скалярной переменной, именем массива или структуры, а также строковой или арифметической константой.

Имена аргументов в описателе AUTOMATIC печатаются всегда при первом выполнении оператора CALL IJKEHHC; в блоке. Идентификаторы с описателем STATIC печатаются только тогда, когда оператор печати изменяющихся значений выполняется в первый раз, если данный блок является внутренним. Если блок определения имен аргументов внешний, они печатаются каждый раз при выполнении этого оператора. При последующих выполнениях оператора CALL IJKEHHC; имена и соответствующие значения печатаются только в том случае, если со времени последнего выполнения этого оператора значения изменились.

Оператор печати переходов обеспечивает выдачу на печать информации о каждом выполнении оператора GO TO в данном блоке. Для организации (включения) печати переходов используется оператор

```
CALL IJKTRON;
```

а для выключения ее — оператор

```
CALL IJKTROF;
```

Язык ПЛ/1 и транслятор ДОС ЕС ЭВМ накладывают на объекты исходной программы количественные ограничения, которые необходимо учитывать при разработке алгоритма и написании программы. Количественные ограничения, присущие языку, были отмечены в гл. IV, перечислим здесь основные ограничения, связанные с

транслятором (там, где не оговорено, имеется в виду максимальное значение):

- длина строки символов — 255;
- длина строки битов — 64;
- минимальная длина строк — 1;
- размерность массива — 3;
- размер массива, байт — 32767;
- количество массивов во внешней процедуре — 32;
- уровень вложенности описателей при объявлении наименований — 8;
- глубина вложенности структур — 8;
- уровень вложенности условных операторов IF — 100;
- уровень вложенности групп DO — 12;
- уровень вложенности блоков — 3;
- уровень вложенности повторяющихся спецификаций в списках данных операторов PUT и GET — 12;
- уровень вложенности списков элементов формата — 5;
- уровень вложенности списка элементов формата, в котором допустимо использование косвенного формата — 2;
- количество блоков внешней процедуры — 63;
- размер блока, байт — 32К;
- количество знаков в выводимом оператором DISPLAY сообщении — 80;
- количество символов в сообщении-ответе в операторе DISPLAY — 255;
- количество фактических параметров (аргументов) в обращении к функции или процедуре — 12;
- длина ключа для файла с описателями INDEXED или REGIONAL (3), символов — 255;
- минимальная длина ключа для файла INDEXED, символов — 1;
- минимальная длина ключа для файла REGIONAL (3), символов — 9;
- единственно допустимая длина исходного ключа для файла с описателем REGIONAL (1), символов — 8;
- значение исходного ключа для файла REGIONAL (1) — 16777215.

Для работы транслятора ПЛ/1 в операционной системе ДОС ЕС требуется 12 Кбайт основной памяти, один процессор, один селекторный и один мультиплексный каналы. При этом обеспечивается возможность многопрограммной работы машины. Мультипрограммирование, организуемое ДОС ЕС, выполняется таким образом, что при генерации системы для конкретной машины основная память делится на фиксированное число частей — разделов. Таких разделов может быть один, два или три. В каждом разделе одновременно мо-

жет выполняться только одна программа. Каждому из разделов присвоено свое наименование и обозначение: фоновый раздел — BG, первый раздел переднего плана — F1, второй раздел переднего плана — F2. Каждому разделу отводится не только участок основной памяти, но и внешние устройства ввода-вывода.

Транслятор ПЛ/1 может быть выполнен только в фоновом разделе, протранслированная и отредактированная программа — в любом из разделов.

Ошибки, которые вызваны несоблюдением синтаксических правил языка, обнаруживаются транслятором. В этом случае для каждой обнаруженной ошибки печатается сообщение. Сообщение содержит степень грубости ошибки, номер оператора, десятичный код ошибки и текст ошибки.

Приложение I. БИБЛИОТЕЧНЫЕ ПОДПРОГРАММЫ ФОРТРАНА IV

- 1) SIN (X) — вычисление синуса $\sin x$;
- 2) COS (X) — вычисление косинуса $\cos x$;
- 3) TAN (X) — вычисление тангенса $\operatorname{tg} x$;
- 4) COTAN (X) — вычисление котангенса $\operatorname{ctg} x$;
- 5) SINH (X) — вычисление гиперболического синуса $\operatorname{sh} x$;
- 6) COSH (X) — вычисление гиперболического косинуса $\operatorname{ch} x$;
- 7) TANH (X) — вычисление гиперболического тангенса $\operatorname{th} x$;
- 8) ARSIN (X) — вычисление арксинуса $\operatorname{Arcsin} x$;
- 9) ARCOS (X) — вычисление арккосинуса $\operatorname{Arccos} x$;
- 10) ATAN (X) — вычисление арктангенса $\operatorname{Arctg} x$;
- 11) ATAN2 (X1, X2) — вычисление арктангенса частного $\operatorname{Arctg} (x_1/x_2)$;

- 12) SQRT (X) — вычисление квадратного корня \sqrt{x} ;
- 13) EXP (X) — вычисление показательной функции e^x ;
- 14) ALOG (X) — вычисление натурального логарифма $\ln x$;
- 15) ALOG10 (X) — вычисление десятичного логарифма $\lg x$.

Результат действия этих подпрограмм — действительного типа, аргументы — тоже действительного типа. Аргументы подпрограмм 1—7 должны быть выражены в радианах, у подпрограммы 12 аргумент неотрицательный. У каждой из подпрограмм 14 и 15 аргумент только положительный. Подпрограммы 8—11 вычисляют главное значение соответствующих функций.

- 16) DSIN (D);
- 17) DCOS (D);
- 18) DTAN (D);
- 19) DCOTAN (D);
- 20) DSINH (D);
- 21) DCOSH (D);
- 22) DTANH (D);
- 23) DARSIN (D);
- 24) DARCOS (D);
- 25) DATAN (D);
- 26) DATAN2 (D1, D2);
- 27) DSQRT (D);
- 28) DEXP (D);
- 29) DLOG (D);
- 30) DLOG10 (D).

Подпрограммы 16—30 выполняют действия, совпадающие с действиями функций 1—15 соответственно, только результат получается двойной точности. Двойную точность имеют также аргументы D, D1, D2.

- 31) ABS (X);
- 32) DABS (D);
- 33) IABS (K).

Подпрограммы 31—33 вычисляют абсолютную величину аргумента, который имеет действительный тип, двойную точность и целый тип соответственно.

34) AMIN1 (XZ);

35) MIN1 (XZ);

36) AMIN0 (IZ);

37) MIN0 (IZ).

Подпрограммы 34—37 определяют наименьшее значение в списке аргументов XZ или IZ; аргументов в списке может быть два или больше. В списке аргументов элементы разделяются запятыми. Аргументами могут быть константы, простые переменные и переменные с индексами. В одном списке все аргументы должны иметь один и тот же тип и перечисляться. Так, нахождение наименьшего среди четырех элементов массива X выполняется стандартной функцией

MIN1 (X(1), X(2), X(3), X(4))

Результат подпрограмм 34 и 36 — действительного типа, 35 и 37 — целого типа. Аргументы в 34 и 35 имеют тип действительный, а в 36 и 37 — целый. Таким образом, подпрограммы 35 и 36 осуществляют, кроме нахождения минимального значения, еще и преобразование типа.

38) DMIN1 (DZ).

Стандартная функция 38 аналогична подпрограмме 34, только в списке аргументов DZ содержатся величины двойной точности.

39) AMAX1 (XZ);

40) MAX1 (XZ);

41) AMAX0 (IZ);

42) MAX0 (IZ);

43) DMAX1 (DZ).

Подпрограммы 39—43 определяют наибольшее значение в списках аргументов XZ, IZ или DZ по правилам, полностью совпадающим с правилами действия подпрограмм 34—38 соответственно.

44) FLOAT (K);

45) DFLOAT (K);

46) IFIX (X);

47) HFIX (X);

48) DBLE (X);

49) SNGL (D).

Аргумент подпрограмм 44, 45 — целого типа, 46—48 — действительного, 49 — двойной точности. Эти подпрограммы преобразуют тип своих аргументов. Тип результата у функции 44 и 49 — действительный, 45 и 48 — двойной точности, 46 — целый, 47 — целый, нестандартной длины. Таким образом, при выполнении стандартных функций 46 и 47 игнорируется дробная часть, а при выполнении подпрограммы 49 — младшие разряды. Например, оператор

Y=FLOAT (K)

присваивает переменной Y то же числовое значение, которое имеет K, однако тип уже будет действительный (ср. с оператором Y=K).

50) AINT (X);

51) INT (X);

52) IDINT (D).

Подпрограммы 50—52 выделяют целую часть аргумента. Результат 50 имеет действительный тип, 51, 52 — целый, аргумент X —

действительный, D — двойной точности, у результата сохраняется знак аргумента.

Например, после выполнения оператора

$I = \text{INT} (-7.5)$

переменная I получает значение, равное — 7

53) $\text{AMOD} (X1, X2)$;

54) $\text{MOD} (K1, K2)$;

55) $\text{DMOD} (D1, D2)$.

Стандартные функции 53—55 содержат всегда по два аргумента и осуществляют деление первого аргумента по модулю второго аргумента, т. е. результат этих подпрограмм — остаток от деления. Он имеет действительный тип для функции 53, целый — для 54; двойной точности — для подпрограммы 55.

Например, оператор

$Y = \text{AMOD} (23., 4)$

дает в результате действительное число 3., а оператор

$\text{IF} (\text{MOD} (N, 2)) 1, 2, 1$

передает управление к оператору с меткой 1 в случае нечетных значений N и оператору с меткой 2 при четных N (результат функции MOD равен 0 при четных N и 1 при нечетных N).

56) $\text{SIGN} (X1, X2)$;

57) $\text{ISIGN} (K1, K2)$;

58) $\text{DSIGN} (D1, D2)$.

Подпрограммы 56—58 осуществляют присвоение знака второго аргумента абсолютному значению первого аргумента. Тип результата функции 56 — действительный, 57 — целый, 58 — двойной точности; таков же тип аргументов.

59) $\text{DIM} (X1, X2)$;

60) $\text{IDIM} (K1, K2)$.

Стандартные функции 59 и 60 называются подпрограммами уменьшения аргумента или вычисления положительной разности. Каждая из них определяет меньшее значение одного из двух своих аргументов, и первый аргумент уменьшается на это значение. Аргументы и результат функции 59 — действительные, 60 — целые.

Например, результат оператора

$I = \text{IDIM} (3, 8)$

есть 0, а операторов

$I = \text{IDIM} (8, 3)$ и $I = \text{IDIM} (-3, -8)$

— целое значение 5

61) $\text{ERF} (X)$;

62) $\text{DERF} (D)$;

63) $\text{ERFC} (X)$;

64) $\text{DERFC} (D)$.

В подпрограммах 61 и 63 действительный тип имеют как аргумент, так и результат, а в 62 и 64 — двойную точность. Эти подпрограммы вычисляют функции ошибок, причем 61 и 62 по формуле

$$\frac{2}{\sqrt{\pi}} \int_0^x e^{-y^2} dy, \text{ а } 63 \text{ и } 64 \text{ — по формуле } \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-y^2} dy.$$

- 65) GAMMA (X);
- 66) DGAMMA (D);
- 67) ALGAMA (X);
- 68) DLGAMA (D).

Подпрограммы 65 и 66 вычисляют гамма-функцию (тип аргумента и результата у стандартной функции 65 — действительный, у 66 — двойной точности). Значение натурального логарифма от функции 65 и 66 вычисляется соответственно подпрограммами 67 и 68. Результат 67 — действительный, 68 — двойной точности.

- 69) CSIN (C) — $\sin(a+bi)$;
- 70) CCOS (C) — $\cos(a+bi)$;
- 71) CSQRT (C) — $\sqrt{a+bi}$;
- 72) CEXP (C) — $\exp(a+bi)$;
- 73) CLOG (C) — $\ln(a+bi)$;
- 74) CLOG10 (C) — $\lg(a+bi)$;
- 75) CABS (C) — $\sqrt{a^2+b^2}$.

В подпрограммах 69—75 предполагается, что C — выражение комплексное, т. е. в обычной математической записи имеет вид $a+bi$. Результат этих функций имеет комплексный тип, за исключением функции 75, имеющей действительный тип.

- 76) CDSIN (CD);
- 77) CDCOS (CD);
- 78) CDSQRT (CD);
- 79) CDEXP (CD);
- 80) CDLOG (CD);
- 81) CDLG10 (CD);
- 82) CDABS (CD).

Подпрограммы 76—82 аналогичны соответствующим функциям 69—75, однако в них как аргумент CD, так и результат являются комплексными двойной точности, т. е. типа COMPLEX*16, за исключением стандартной функции 82, результат которой действительный нестандартной длины.

- 83) REAL (C);
- 84) AIMAG (C).

Подпрограммы 83 и 84 выделяют вещественную и мнимую части комплексного аргумента C соответственно. Результат здесь имеет действительный тип.

- 85) CMPLX (A, B);
- 86) DCMPLX (D1, D2).

Подпрограмма 85 имеет два действительных аргумента. Результат этой стандартной функции — комплексное число, которое образуется из аргументов A и B, т. е. $A+Bi$. Функция 86 образуется из двух аргументов двойной точности величину типа COMPLEX*16.

- 87) CONJG (C);
- 88) DCONJG (CD).

Эти функции имеют комплексные как аргумент, так и результат. Они дают сопряженную с аргументом величину, т. е. $A-Bi$, если C равно $A+Bi$. Функция 87 имеет комплексного типа стандартной длины аргумент и результат, 88 — типа COMPLEX*16.

Из перечисленных стандартных функций функции 31—33, 44—60 и 83—88 в Фортране IV являются встроенными, остальные — внешними.

Приложение II. СТАНДАРТНЫЕ ФУНКЦИИ ЯЗЫКА БЕЙСИК

Стандартные функции — это встроенные программы, после выполнения которых с идентификатором функции связывается вполне определенное числовое значение или же строка символов. В § 12 гл. III приведены сведения о классификации стандартных функций языка Бейсик. Здесь функции, содержащиеся в большинстве версий Бейсика, в частности в системах Бейсик-плюс и МикроДОС и в расширенном и дисковом Бейсике IBM PC, перечисляются в алфавитном порядке. В качестве аргументов, которых обычно один или два, могут использоваться константы, переменные, элементы массивов, другие функции и выражения, если специально не оговорена иная интерпретация. Для некоторых функций ограничивается диапазон изменения значений их аргументов. В приводимых ниже функциях аргументы обозначаются следующим образом: X, Y — числовые значения; A, B — строковые значения; F — номер файла; P — номер порта; M, N — числа; I, J — номера строк и столбцов соответственно; V — переменная; C — символ.

- 1) ABS (X) — вычисление абсолютной величины X;
- 2) ASC (A) — определение кода первого символа строкового значения A;
- 3) ASCII (A) — определение кода первого символа строкового значения A в стандарте ASCII;
- 4) ATN (X) — вычисление арктангенса X;
- 5) CDBL (X) — преобразование X в число двойной точности;
- 6) CHR\$ (M) — определение символа, соответствующего заданному коду M;
- 7) CINT (X) — округление X до ближайшего целого числа;
- 8) COS (X) — вычисление косинуса X;
- 9) CSNG (X) — преобразование X в число обычной точности;
- 10) CURLIN — выдача номера экранной строки, соответствующей текущему положению курсора;
- 11) CVD (A) — преобразование восьмисимвольного значения A в значение двойной точности;
- 12) CVI (A) — преобразование двухсимвольного значения A в значение целого типа;
- 13) CVS (A) — преобразование четырехсимвольного значения A в значение обычной точности;
- 14) CVTFS\$(X) — преобразование X в символьную строку из четырех (восьми) символов;
- 15) CVT\$F (A) — преобразование первых четырех (восьми) символов A в действительное число;
- 16) CVT%\$ (X) — преобразование целого значения X в символьную строку из двух символов;
- 17) CVT\$% (A) — преобразование первых двух символов A в целое число;
- 18) DATE\$ — выдача текущей системной даты;
- 19) EOF (F) — определение факта достижения конца файла (значения функции: «истина» или «ложь»);
- 20) ERL — выдача номера строки, в которой обнаружена последняя по счету ошибка;
- 21) ERR — выдача кода последней по счету обнаруженной ошибки;
- 22) EXP (X) — вычисление экспоненты e^X ;
- 23) FIX (X) — усечение X до целого числа;

24) FRE (X или A) — определение числа свободных байтов в памяти (в области памяти, отведенной для хранения строк — в случае аргумента A);

25) HEX\$ (X) — преобразование X в шестнадцатеричное представление;

26) INKEY\$ — выдача информации о том, какая клавиша нажимается в текущий момент (значение функции, равное нулю, означает, что клавиатура заблокирована);

27) INP (P) — выдача сообщения о назначенном машинном порте;

28) INPUT\$ (M, #F) — считывание строки символов с клавиатуры, либо, если задан аргумент F, с соответствующего файла (устройства);

29) INSTR (C, A, B) — поиск в исходной строке A искомой строки B, начиная с заданного символа C или, при отсутствии C, с первого символа A;

30) INT (X) — определение наибольшего целого числа, не превосходящего X;

31) LEFT\$ (A, M) — выделение из строки A подстроки длиной M, начиная с крайнего левого символа;

32) LEN (A) — определение количества символов в A;

33) LOC (F) — определение текущей позиции в F;

34) LOF (F) — определение длины F;

35) LOG (X) — вычисление натурального логарифма X;

36) LOG10 (X) — вычисление десятичного логарифма X;

37) LPOS (X) — определение номера текущей позиции печатающего устройства X;

38) MID (A, M, N) — выделение центральной подстроки строки A длины N, начиная с M-го символа;

39) MID\$ (A, C, N) — выделение части строки A, начиная с символа C, длины N, если N — задано;

40) MKD\$ (X) — преобразование X в число двойной точности, а затем представление его в виде строкового значения из восьми символов;

41) MKI\$ (X) — округление X до целого числа и представление последнего в виде строки из двух символов;

42) MKS\$ (X) — преобразование X в число обычной точности, а затем представление его в виде строки из четырех символов;

43) OCT\$ (X) — преобразование X в восьмеричное представление;

44) PEEK (M) — выдача содержимого ячейки памяти по заданному адресу M;

45) PEN (X) — выдача информации о работе светового пера, допустимые значения аргумента X лежат в диапазоне 0—9;

46) POINT (J, I) — определение цвета точки с заданными координатами (для графического режима);

47) POS (X) — выдача номера столбца, соответствующего текущему положению курсора на экране (в текстовом режиме вывода);

48) RIGHT\$ (A, M) — выделение из строки A подстроки длиной M, начиная с крайнего правого символа;

49) RND (X) — выдача случайного числа из диапазона 0 ÷ 1 с равномерным законом распределения (аргумент может отсутствовать);

50) SCREEN (I, J) — определение числового кода символа, высвеченного на экране и имеющего координаты I, J (в текстовом режиме);

- 51) SGN (X) — определение знака X;
 52) SIN (X) — вычисление синуса X;
 53) SPACE\$ (M) — генерация строки из M пробелов;
 54) SPC (M) — пропуск M позиций в выводимой на печать операторами PRINT или LPRINT строке;
 55) SQR (X) — вычисление квадратного корня из X;
 56) STRING\$ (M, A или N) — генерация строки длины M, все символы которой совпадают с первым символом строки A, либо с символом, имеющим код N);
 57) STR\$ (X) — преобразование X в строку символов;
 58) TAB (J) — подведение к позиции J в выводимой на печать операторами PRINT или LPRINT строке;
 59) TAN (X) — вычисление тангенса X;
 60) TIME — выдача системного времени;
 61) TIME\$ — выдача текущего времени;
 62) USR N (A или X) — передача управления программе (с номером N, если он указан) на машинном языке, либо по числовому значению X, либо по адресу памяти, где хранится строковая переменная A;
 63) VAL (A) — преобразование строки A в число;
 64) VARPTR (V) — определение адреса переменной V в памяти;
 65) VARPTR (#F) — определение адреса памяти, по которому хранится блок управления файлами для заданного файла;
 66) VARPTR\$ (V) — определение типа переменной V и соответствующего ей адреса памяти в виде трехсимвольного строкового значения;
 67) XLATE (A, B) — перекодировка строки символов A в соответствии со словарем B;
 Данный список стандартных функций не претендует на полноту.

Приложение III. ВСТРОЕННЫЕ ФУНКЦИИ ПЛ/1

1. Математические функции:

- 1) ATAN (X); а) X действительный, результат $\text{Arctg } X$, в радианах; б) X — комплексный, результат комплексный;
- 2) ATAN (X, Y); X, Y действительные, результат: при $Y > 0$ $\text{ATAN} (X/Y)$, при $X > Y$, $Y = 0 \pi/2$, при $X \geq 0$, $Y < 0 \pi + \text{ATAN} (X/Y)$, при $X < 0$, $Y = 0 -\pi/2$, при $X < 0$, $Y < 0 -\pi + \text{ATAN} (X/Y)$, в радианах, возникает ситуация ERROR, если $X = Y = 0$.
- 3) ATAND (X); X действительный, результат $\text{Arctg } X$, в градусах;
- 4) ATAND (X, Y); X, Y действительные, результат $(180/\pi) * \text{ATAN} (X, Y)$, в градусах, возникает ситуация ERROR, если $X = Y = 0$;
- 5) ATANH (X); а) X действительный, результат $\text{Arcth } X$, возникает ситуация ERROR, если $|X| \geq 1$; б) X комплексный, результат $(\ln((1+X)/(1-X)))/2$, возникает ситуация ERROR, если $|X| = 1$;
- 6) COS (X); а) X действительный, в радианах, результат $\cos X$; б) X комплексный, результат комплексный;
- 7) COSD (X); X действительный, в градусах, результат $\cos X$;
- 8) COSH (X); а) X действительный, результат $\text{ch } X$; б) X комплексный, результат комплексный;

9) ERF (X); X действительный, вычисляется интеграл
$$\frac{2}{\pi} \int_0^x e^{-t^2} dt;$$

10) ERFC (X); X действительный, вычисляется функция $1 - \text{—ERF (X)}$;

11) EXP (X); результат e^X ;

12) LOG (X); а) X действительный, результат $\ln X$; б) X комплексный, результат комплексный;

13) LOG10 (X); X действительный, вычисляется десятичный логарифм $\lg X$;

14) LOG2 (X); X действительный, вычисляется логарифм при основании 2, т. е. $\log_2 X$;

Для функций 12) — 14) возникает ситуация ERROR, если $X \leq 0$.

15) SIN (X); а) X действительный, в радианах, вычисляется $\sin X$; б) X комплексный, результат комплексный;

16) SIND (X); X действительный, в градусах, вычисляется $\sin X$;

17) SINH (X); а) X действительный, результат $\text{sh } X$; б) X комплексный, результат комплексный;

18) SQRT (X); а) X действительный, вычисляется \sqrt{X} , возникает ситуация ERROR, если $X < 0$; б) X комплексный, результат комплексный;

19) TAN (X); X в радианах, результат $\text{tg } X$;

20) TAND (X); X в градусах, результат $\text{tg } X$;

21) TANH (X); результат $\text{th } X$.

2. Арифметические функции (здесь приняты обозначения: w — общее количество разрядов в представлении числа, d — количество разрядов после точки; w, d — целые десятичные числа; w — положительное, d — может быть и отрицательным):

22) ABS (X); результат $|X|$;

23) ADD (X, Y, w, d); вычисляется сумма $X+Y$ с точностью (w) в форме с плавающей точкой, (w, d) — с фиксированной точкой;

24) BINARY (X, w, d); X преобразуется в двоичное число с точностью (w) в случае с плавающей точкой, (w, d) — в случае с фиксированной точкой; если w, d опущены, результату присваивается точность аргумента; допускается сокращение BIN;

25) CEIL (X); X действительный, результат — наименьшее целое число, которое больше или равно X;

26) COMPLEX (X, Y); X, Y действительные, формируется комплексное число $X+Yi$; допускается сокращение CPLX;

27) CONJ (X); X комплексный, результат — число, сопряженное X;

28) DECIMAL (X, w, d); X преобразуется в десятичное число по аналогии с функцией BINARY; допускается сокращение DEC;

29) DIVIDE (X, Y, w, d) вычисляется X/Y с разрядностью (w, d); если X, Y — с плавающей точкой, d опускается;

30) FIXED (X, w, d); X преобразуется в число с фиксированной точкой; если w, d опущены, используется принцип умолчания;

31) FLOAT (X, w); X действительный, X преобразуется в число с плавающей точкой; если w опущено, используется принцип умолчания;

32) FLOOR (X); X действительный, определяется наибольшее целое число, которое меньше или равно X;

33) IMAG (X); X комплексный, результат — мнимая часть X;
34) MAX (XZ); XZ — список действительных аргументов, результат — значение наибольшего из аргументов;

35) MIN (XZ); XZ — список действительных аргументов, вычисляется значение наименьшего из аргументов;

36) MOD (X, Y); X, Y действительные, результат — положительный остаток от деления X/Y ;

37) MULTIPLY (X, Y, w, d); результат — произведение $X*Y$ с разрядностью (w, d); если X, Y — с плавающей точкой, d опускается,

38) PRECISION (X, w, d); X преобразуется в число с разрядностью (w, d) в случае фиксированной точки и (w) в случае плавающей точки (здесь d опускается); допускается сокращение PREC;

39) REAL (X); X комплексный, результат — вещественная часть X;

40) ROUND (X, N); N — целая десятичная константа, X — арифметическая переменная или строка цифровых знаков; если аргумент X задан с фиксированной точкой, то X округляется до N-го разряда справа от точки, если X — с плавающей точкой, то использование функции не имеет смысла;

41) SIGN (X); X действительный, результат есть -1, 0, +1 соответственно при $X < 0$, $X = 0$, $X > 0$;

42) TRUNC (X); X действительный, соответствует функции CEIL (X) при $X < 0$ и FLOOR (X) при $X \geq 0$.

3. Функции для обработки строк:

43) BIT (X, N); X — строка или выражение, N — десятичное целое число без знака; X преобразуется в строку битов длины N; если N не задано, длина результата определяется X;

44) BOOL (X, Y, Z); X, Y — переменные типа строки бит, Z — битовая строка длиной 4 бита; результата — строка бит с длиной, равной наибольшей из длин строк X и Y, биты выбираются в зависимости от значений бит в Z;

45) CHAR (X, N); X — строка или выражение, N — десятичное целое число без знака; X преобразуется в символьную строку длины N; если N не задано, длина результата определяется X;

46) HIGH (N); N — десятичное целое число без знака; результат — символьная строка длины N, каждый байт которой есть шестнадцатеричное число FF;

47) INDEX (X, Y); X, Y — строки или выражения; результат — целое двоичное число без знака, указывающее номер позиции в строке X (считая слева), с которой строка Y содержится в X, или нуль, если Y не входит в состав X;

48) LENGTH (X); X — строка или выражение, результат — двоичное целое число, представляющее текущую длину X;

49) LOW (N); N — целое десятичное число без знака; результат — строка символов длины N, каждый байт представляет собой шестнадцатеричное число 00;

50) REPEAT (X, N); X — строка или выражение, N — целое десятичное число; результат — строка, полученная сцеплением строки X N раз; если $N \leq 0$ — строка X;

51) STRING (X); X — имя переменной, массива или структуры; осуществляется сцепление всех элементов в X;

52) SUBSTR (X, M, N); X — строка или выражение, M — скалярное выражение, которое может быть преобразовано в целое число (M может быть массивом, если X — массив), N — целое

десятичное число без знака; результат — часть строки X, которая начинается с позиции M и имеет длину N;

53) TRANSLATE (X, Y, Z); X, Y, Z — строки символов или бит; результат — строка, полученная из X заменой тех ее элементов, которые содержатся в Z, соответствующими элементами из Y;

54) UNSPEC (X); X не может быть переменной типа метки или массивом (может быть переменной типа указателя); X представляется в памяти в форме строки бит;

55) VERIFY (X, Y); X, Y — строки символов или бит; результат — целое двоичное число, равное 0, если каждый элемент X входит также в Y или же X есть пустая строка, равное 1, если X — не пустая, а Y — пустая строка; в других случаях результат равен номеру первого слева элемента из X, который не входит в Y.

4. Функции для обработки массивов:

56) ALL (X); X — массив из строк бит, результат — строка бит, где *i*-й бит равен 1, если в X имеется *i*-й бит каждого элемента и он равен 1, в противном случае *i*-й бит равен 0;

57) ANY (X) — массив из строк бит, результат — строка бит, где *i*-й бит равен 1, если хоть в каком-нибудь из элементов X имеется *i*-й бит и он равен 1, в противном случае *i*-й бит равен 0;

58) DIM (X, N); N — двоичное целое число; результат — двоичное целое число, задающее текущую N-ю размерность массива X;

59) HBOUND (X, N); N — двоичное целое число; результат — двоичное целое число, равное текущей верхней границе массива X по N-му измерению;

60) LBOUND (X, N); N — двоичное целое число; результат — двоичное целое число, равное текущей нижней границе массива X по N-му измерению;

61) POLY (X, Y); X, Y — одномерные массивы; результат — многочлен от X и Y;

62) PROD (X); произведение всех элементов массива X;

63) SUM (X); сумма всех элементов массива X.

5. Функции специального назначения:

64) ADDR (X); X — имя переменной; результат — скалярное значение указателя, которое указывает адрес переменной X в памяти;

65) ALLOCATION (X); X — имя небазированной переменной с управляемым способом размещения; результат — '1'B, если память выделена для X, определенного в данном блоке, в противном случае — '0'B;

66) COUNT (F); F — имя файла с описателем STREAM; результат — двоичное целое число с фиксированной точкой, задающее количество элементов, переданных в последней операции ввода или вывода с файлом F;

67) DATAFIELD; результат — строка символов, содержащая поле данных, вызвавших последнее прерывание по ситуации NAME;

68) DATE; результат — строка символов вида 'YYMMDD', где YY — текущий год, MM — текущий месяц, DD — текущий день;

69) EVENT (X); X — имя переменной типа события; результат — строка битов '1'B или '0'B в зависимости от текущего состояния события с именем X;

70) LINENO (F); F — имя файла с описателем PRINT; результат — двоичное целое число с фиксированной точкой, являющееся текущим номером печатной строки;

71) NULL; определяет нулевое значение указателя;

72) ONCHAR; результат — строка длины 1, содержащая символ, который вызвал последнее прерывание по ситуации CONVERSION;

73) ONCODE; результат — двоичное целое число, определяющее своим значением последнее прерывание (например, константа 0 соответствует ситуации FINISH);

74) ONCOUNT; результат — двоичное значение, равное числу прерываний, включая последнее;

75) ONFILE; результат — строка символов, содержащая имя файла, для которого были выполнены последние ввод или вывод или преобразование, вызвавшие прерывание;

76) ONKEY; результат — строка символов, содержащая ключ записи, вызвавшей последнее прерывание;

77) ONLOC; результат — строка символов, содержащая имя точки входа в процедуру, которая вызвала прерывание;

78) ONSOURCE; результат — строка символов, включающая содержимое поля, обрабатываемого при возникновении последнего прерывания по ситуации CONVERSION;

79) PRIORITY (X); X — имя переменной типа ветви; результат — ветвь с именем X получает приоритет по отношению к ветви, в которой выполняется данная функция;

80) STRING (X); X — упакованная структура, содержащая только символьные строки и (или) строки из цифровых знаков; результат — строка, полученная путем сцепления всех элементов структуры X;

81) TIME; символьная строка из девяти элементов, указывающая текущее время в виде 'HHMMSSTTT', где HH — часы, MM — минуты, SS — секунды, TTT — миллисекунды.

6. Псевдопеременные:

82) COMPLEX (A, B); A, B — идентификаторы, возможно, с различными описателями; результат — действительная часть выражения присваивается A, мнимая — B (например, в результате выполнения оператора COMPLEX (X, Y)=5.1+7.5I; X будет равно 5.1, Y — 7.5;

83) IMAG (C); C — комплексная переменная; результат — мнимая часть выражения присваивается мнимой части переменной C, вещественная часть этой переменной остается без изменения;

84) ONCHAR; результат — символ, являющийся значением этой функции после выполнения оператора присваивания, заменяет символ, при обработке которого возникла ситуация CONVERSION (например, оператор ON CONVERSION ONCHAR='1'; обеспечивает замену непробуемого символа единицей);

85) ONSOURCE; результат — заменяется символьная строка (поле), при обработке которой возникла ситуация CONVERSION;

86) REAL (C); C — комплексная переменная; результат — вещественная часть выражения присваивается вещественной части переменной C, мнимая часть этой переменной остается без изменения;

87) SUBSTR (X, M, N); X, M, N имеют тот же смысл, что и в п. 3, только X не может быть массивом или выражением; результат — значение выражения, преобразованного в строку символов, присваивается подстроке, определенной псевдопеременной

SUBSTR (например, если X есть 'ABCDE', а Y есть 'XYZPQ', то оператор SUBSTR (X, 2, 3)=SUBSTR (Y, 3, 3); заменяет значение X на 'AZPQE';

88) UNSPEC (X); X имеет тот же смысл, что и в п. 3, только X не может быть выражением; результат — выражение преобразуется в строку бит и присваивается переменной X без преобразования в тип X (например, оператор UNSPEC (V)=F; преобразует F в строку бит и присваивает это значение V).

Как уже было отмечено в § 11 гл. III количество встроенных функций существенно зависит от версии языка и типа транслятора, поэтому приведенный список с одной стороны не является исчерпывающим, с другой — содержит такие функции, которые не реализуются в ряде версий. В частности, в версии, называемой подмножеством ПЛ/1, не допускаются встроенные функции, приведенные здесь под номерами: 23, 26, 29, 33, 37, 39, 48, 51, 58, 59, 60, 61, 65, 66, 67, 70, 72, 73, 74, 75, 77, 78.

СПИСОК ЛИТЕРАТУРЫ

Общая

1. Пярнпуу А. А. Программирование на Алголе и Фортране.— М.: Наука, 1978.
2. Пярнпуу А. А. Программирование на алгоритмических языках.— М.: Наука, 1983.
3. Королев Л. Н. Структуры ЭВМ и их математическое обеспечение.— М.: Наука, 1978.
4. Турский В. Методология программирования.— М.: Мир, 1981.
5. Мейер Б., Бодуэн К. Методы программирования.— М.: Мир, 1982.
6. Власов В. К., Королев Л. Н., Сотников А. Н. Элементы информатики.— М.: Наука, 1988.

Главе II

7. Грунд Ф. Программирование на языке ФОРТРАН IV.— М.: Мир, 1976.
8. Хьюз Ч., Флигер Ч., Роуз Л. Методы программирования: курс на основе ФОРТРАНА.— М.: Мир, 1981.

Главе III

9. Пул Л. Работа на персональном компьютере.— М.: Мир, 1986.
10. Кетков Ю. Л. Диалог на языке Бейсик для мини- и микро-ЭВМ.— М.: Наука, 1988.

Главе IV

11. Скотт Р., Сондак Н. ПЛ/1 для программистов.— М.: Статистика, 1977.
12. Аугустон М. И., Балодис Р. П., Барздинь Я. М., Икауниекс Э. А., Калниньш А. А. Программирование на ПЛ/1 ОС ЕС.— М.: Статистика, 1979.

Главе V

13. Системы математического обеспечения ЕС ЭВМ/ Под ред. А. М. Ларионова.— М.: Статистика, 1974.
14. Девич У. Операционные системы.— М.: Мир, 1980.
15. Катцан Г. Язык ФОРТРАН-77.— М.: Мир, 1982.
16. Салтыков А. И., Макаренко Г. И. Программирование на языке ФОРТРАН.— М.: Наука, 1976.
17. Программирование на языке Бейсик-плюс для СМ-4 / В. П. Семик, Б. Р. Мончилович, Д. П. Непчатых и др.— М.: Финансы и статистика, 1982.
18. Язык программирования Бейсик, Библиотека МикроДОС.— М.: МЦНТИ, 1987.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

- Автокод 27
Автоматизированное место 17
Ада 33
Адрес 26
Алгол 30, 32
Алгоритм 21
Алфавит 136, 224
Ассемблер 33
- База данных 17
— знаний 14
— индексов 155
— элементная 9
Базисный Фортран 29, 309, 322
Байт 11
Бейсик 30, 136, 346, 354
Бит 11
Бланк Фортрана 60
Блок 32, 250, 258
— данных 159
— нулевой 35
— общий 91
— обычный 250, 258
— процедурный 250, 258, 259
— системный 14
— наименований 113
Блок-схема 19
Буфер 204, 207, 279
Быстродействие ЭВМ 9
- Ввод данных 284, 286, 296
Величина 40
Версия языка 38
Видеодиск 16
Вывод данных 284, 286, 296
Выражение 30, 46, 136, 245
— арифметическое 47
— индексное 44
— логическое 47
— над массивами 247
— — структурами 248
- Выражение скалярное 245
— строковое 147
— числовое 143
- Граница изменения индексов 53, 237
Группа 250, 254
- Данные 40, 97, 137, 224
Двоичный знак 9
Десятичный порядок 41
Диалог 12
Директива 136
Диск 16
— винчестерский 16
— гибкий 16
Дискет 16
Дисплей 14
Дробь 41
— правильная 41
- Заголовок оператора 51
— цикла 68
Загрузка 36
Задание 304
Задача 305
Замена формального параметра 76, 85
Запись 101, 203, 278, 305
- Идентификатор 44, 140, 228
Имя внешнее 259, 275
— внутреннее 275
— родовое 204
— файла 204
Индекс массива 44, 141, 227
— спецификации 116, 289
Интеллектуальная рабочая станция 17
Интерпретатор 37
Интерпретация 35
Информация 7
Искусственный интеллект 18

Клавиатура 14
 Ключевое слово 40, 51, 137, 148, 228, 231
 Кобол 31
 Код 7
 — объектный 37
 Команда 9, 26, 136
 Комментарий 51, 61, 150, 249
 Компилятор 37
 Компиляция 35
 Константа 40, 138, 225
 — арифметическая 40, 225
 — двойная 226
 — действительная 41, 225
 — десятичная 225
 — комплексная 40, 42, 226
 — логическая 40, 43
 — мнимая 226
 — строковая 138, 225, 226
 — текстовая 43, 138
 — типа метки 242
 — целая 40, 138
 — числовая 40, 138
 — шестнадцатеричная 43
 Коэффициент кратности 54, 96, 113, 115
 — масштабный 126
 — повторения 227, 243, 292, 316
 Курсор 149

Лисп 31

Магнитная лента 15
 Магнитный барабан 15
 — диск 15
 Макрокоманда 35
 Массив 44, 141, 227, 237
 Математическая модель 7, 18
 Метка дисководов 204
 — оператора 43, 136, 148, 242, 305
 Метод логических устройств 306
 Микропроцессор 13
 Модуль абсолютный 306
 — исходный 306
 — объектный 306
 Монитор 35, 36
 Мультипрограммирование 12

Наименование 132
 Начальное значение 54, 243
 Номер оператора 148
 — строки 136
 — файла 205

Обеспечение математическое 10, 303
 — программное 9, 10, 303
 Обработка информации 18
 — — автоматическая 8
 — прерываний 36, 267
 — результатов 20
 Общий блок 91
 Объем оперативной памяти 9
 Объявление типа 45
 — — автоматическое 45
 Оверлей 194
 Операнд 26, 46
 Оператор 29, 50, 136, 148, 230, 250
 — безусловного перехода 183
 — безусловной передачи управления 252
 — ввода 101, 159
 — — последовательного доступа без формата 106
 — — — с форматом 108
 — — прямого доступа без формата 107
 — — — с форматом 109
 — — с клавиатуры 161
 — внешних подпрограмм 82
 — внутренней передачи 294, 319
 — возврата 76, 84, 105, 193, 260
 — — с параметром 85
 — восстановления 160
 — входа 89, 264
 — — в процедуру 264
 — вывода 101
 — — на дисплей 273
 — — — печать 169
 — — — экран 164
 — — последовательного доступа без формата 107
 — — — с форматом 110
 — — прямого доступа без формата 107
 — — — с форматом 111
 — вызова 260
 — — подпрограммы 85, 193
 — длины строки 163
 — задания размеров массива 55, 154, 237
 — — формата 115
 — закрытия файла 205, 208
 — занесения записи 207
 — записи в файл 206
 — заполнения поля 206
 — имитации прерывания 272
 — исключения записи 298, 299
 — исполняемый 51, 149, 230

Оператор конца файла 106
 — начала отсчета индексов 155
 — неисполняемый 51, 149, 230
 — обмена 162
 — обработки матриц 198
 — общих переменных 195
 — объявления данных 232
 — окончания 71, 177
 — описания общих блоков 91
 — — массива 154
 — — типа 151
 — — файлов 102
 — описательный 52
 — освобождения памяти 276, 277
 — останова 71, 177, 257
 — открытия файла 205, 207, 282
 — отладки 311, 324
 — отмены прерывания 272
 — очистки экрана 164
 — перехода 61, 182, 252
 — подвода файла 105
 — поиска записи файла 106
 — полного останова 71
 — прерываний 267
 — присваивания 57, 156, 250
 — — метки 58
 — — начальных значений 95
 — продолжения 69
 — пустой 231
 — размеров массива 55
 — размещения в буфере 298
 — — данных 276, 277
 — режимов 164, 209
 — структуры файла 206
 — сцепления программ 195
 — считывания записи 207
 — — файла 206
 — удаления 156
 — управления 51, 61, 252
 — — файлами 105
 — условного останова 71
 — условный 64, 185, 253
 — — полный 185, 186
 — форматного вывода 166
 — форматов 115, 132, 294
 — формирования списков 113
 — цвета 210
 — цикла 67, 187, 255
 — эквивалентности 99
 Оператор-функция 73, 152
 Операционная система 13, 303, 304
 Операция 9, 26, 47, 143, 245
 — арифметическая 47, 143, 245
 — логическая 47, 48, 145, 246

Операция машинная 26
 — отношения 47, 48, 145, 246
 — строковая 147
 — сцепления 246, 335
 — числовая 143
 Описание данных 52, 151, 232
 — — контекстуальное 232
 — — неявное 52, 232
 — — явное 53, 232
 — массива 55, 154, 237
 — структуры 238
 — типа 53, 151, 233
 — функции 73, 75, 152, 260
 Описатель 232
 — классов памяти 275
 — области действия 259, 275
 — упаковки 275
 — файла 279
 Основная программа 29, 51
 Основной символ языка 40
 Отладка программы 20, 308
 Отношение 48, 145, 246

Память 8
 — архивная 15
 — внешняя 8
 — оперативная 8
 Параллельная работа 8
 Параметр фактический 45, 73, 76, 85, 152, 153, 260
 — формальный 73, 75, 84, 152, 153, 259
 — цикла 67, 187, 253
 Паскаль 33
 Переменная 40, 44, 138, 140, 227
 — базовая 241
 — действительная 45, 141
 — комплексная 45, 233
 — логическая 45
 — простая 40, 44, 141, 227, 232
 — с индексами 40, 44, 141, 237
 — скалярная 141, 227, 232
 — строковая 141
 — типа метки 243
 — — указателя 243
 — целая 45, 141
 Перфокарта 15
 Перфолента 15
 ПЛ/1 32
 Повторное определение 241
 Повышенная точность 309
 Подпрограмма 9, 29, 51, 83, 136, 193
 — библиотечная 72
 — данных 97

Подпрограмма-процедура 83
 Подпрограмма-функция 75
 Поколение ЭВМ 10
 Поле 203
 Полуслово 12
 Последовательный доступ к файлу 203
 Прерывания 267
 Примечания 61, 150, 243
 Программа 8, 19, 21, 51, 148, 250
 — вызывающая 52
 — начальной загрузки 35
 — обрабатывающая 306
 — основная 29, 51
 — отладочная 308
 — рабочая 34
 — сервисная 307
 — системная 306
 — служебная 325
 — управляющая 306
 Программирование 8
 — структурное 33
 Программная единица 51
 Программное обеспечение 9, 10, 303
 Процедура 250
 — главная 250
 Процедура-подпрограмма 262
 Процедура-функция 153, 260
 Процессор 9
 — центральный 9
 Прямой доступ к файлу 203

 Раздел памяти 305
 Размерность массива 44, 154, 237
 Разряд двоичный 9
 Распределение памяти 27, 274
 Расширение имени файла 204
 Редактирование связей 37, 306
 Редактор программы в кодах 36, 307
 — текстов 36
 Режим графический 208
 — немедленной обработки 136
 — отладочный 308
 — программируемый 136
 — разделения времени 12
 — текстовый 208
 — трассировки 181
 Ресурсы задачи 305

 Сборка 37
 Сегмент программы 29, 51
 Сечение массива 238

Система аналого-цифровая 8
 — команд 9
 — компьютеров 14
 — математического обеспечения 303
 — операционная 13, 303
 — программирования 304
 — счисления 25
 — управления 17
 — экспертная 14
 Ситуация ввода и вывода 269
 — вычислительная 268
 — контроля программы 270
 — прерывания 267
 — реакции системы 268
 Слово 9
 — двойное 12
 — ключевое 40, 137, 228
 — машинное 9, 12
 — служебное 40, 137, 228
 Снобол 31
 Спецификация 115
 — преобразования 116
 — управляющая редакционная 116
 — формата 115, 289
 Список ввода-вывода 104, 284
 — граничных пар 238, 336
 — индексов 44, 155, 237
 — спецификаций 115
 Стандартная функция 72, 170, 226
 Стек 194
 Строка 43, 138, 226
 Структура 227, 238
 Структурирование программы 192

 Терминал 34
 Тип данного 138
 Том внешней памяти 305
 Транслятор 29, 34, 303, 309
 Трансляция 18, 306

 Указатель 45, 243
 — длины 52
 — функции 45, 73, 152, 260
 Управление заданиями 305
 — печатью 130, 293
 Управляющий символ 130, 293
 Устройство алфавитно-цифровое печатающее 15
 — арифметическое 8
 — ввода 8
 — вывода 8
 — запоминающее 8, 13, 15

Устройство — оперативное 8
— — постоянное 8
— печатающее 14
— подготовки перфокарт 34
— системное 306
— управления 8

Файл 101, 203, 241, 278, 305
— загрузочный 37
— последовательного доступа 102

— прямого доступа 102

Формат 115, 289

— переменный 318

Фортран 29, 39, 333

Функция 45, 152

— внешняя 72

— встроенная 72, 266

— стандартная 72

Цикл 67, 187, 255

— вложенный 70, 191

— неявный 104

Число 40, 138

— восьмеричное 140

— двоичное 140

— действительное 41, 225

— десятичное 138

— комплексное 42, 226

— целое 40, 139

Шаблон 166, 235

Шаг задания 304

Электронная вычислительная машина 7

— — — аналоговая 7

— — — виртуальная 38

— — — дискретного действия 7

— — — многопрограммная 12

— — — многопроцессорная 13

— — — моделирующая 7

— — — непрерывного действия 7

— — — персональная 8

— — — универсальная 7

— — — цифровая 7

Элемент массива 44, 141, 238

— структуры 240

— языка 39

Ядро операционной системы 36

Язык алгоритмический 28

— входной 38

— машинно-ориентированный 28

— машинный 25

— проблемно-ориентированный 28

— программирования 25

— универсальный 30

— универсальный 30

— эталонный 38

Ячейка памяти 8

Учебное издание

ПЯРНПУУ Аарне Антонович

**ПРОГРАММИРОВАНИЕ НА СОВРЕМЕННЫХ
АЛГОРИТМИЧЕСКИХ ЯЗЫКАХ**

Заведующий редакцией *Т. В. Шароватова*

Редактор *Л. Г. Полякова*

Художественный редактор *Т. Н. Кольченко*

Технический редактор *И. Ш. Аксельрод*

Корректоры *Л. И. Назарова, М. Л. Медведская*

ИБ № 41071

Сдано в набор 09.10.89. Подписано к печати 30.05.90. Формат 84×108/₃₂. Бумага тип. № 2. Гарнитура литературная. Печать высокая. Усл печ. л. 20,16. Усл. кр.-отт. 20,16. Уч.-изд. л. 25,62. Тираж 216 000 экз. Заказ 759. Цена 1 р. 30 к.

Издательско-производственное
и книготорговое объединение «Наука»
Главная редакция
физико-математической литературы
117071 Москва В-71, Ленинский проспект, 15

Набрано и сматрицировано в ордена
Октябрьской Революции и ордена Трудового
Красного Знамени МПО «Первая Образцовая
типография» Государственного комитета СССР
по печати. 113054 Москва, Валовая, 28

Отпечатано в типографии № 2 Госкомиздата
РСФСР. г. Рыбинск 152901, ул. Чкалова, дом 10

lp 30κ.