

**БИБЛИОТЕЧКА
ПРОГРАММИСТА**

Ю. М. БЕЗБОРОДОВ

Сравнительный курс языка PL/1



БИБЛИОТЕЧКА ПРОГРАММИСТА

Ю. М. БЕЗБОРОДОВ

СРАВНИТЕЛЬНЫЙ КУРС ЯЗЫКА PL/1 (на основе алгола-60)



МОСКВА «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
1980

Сравнительный курс языка PL/1. Безбородов Ю. М. — М.: Наука, Главная редакция физико-математической литературы, 1980.

Курс универсального языка программирования PL/1 ориентирован в основном на читателя, имеющего опыт программирования на языке алгол-60.

В первую очередь, в курсе путем сравнения понятий двух языков излагаются те понятия PL/1, которые близки к понятиям алгола-60. Вследствие этого уже в начале курса читатель получает возможность составлять реальные программы на PL/1 ОС ЕС ЭВМ в пределах тех языковых возможностей PL/1, которые пересекаются с возможностями алгола-60.

ОГЛАВЛЕНИЕ

Предисловие	6
Введение	9
0.1. Общее знакомство с PL/1	9
0.2. О способе изложения	12
0.3. Основные обозначения	14
Упражнения	17
ГЛАВА 1. Сравнение понятий алгола и PL/1	18
1.1. Сравнение основных элементов	18
1.1.1. Символы (18). 1.1.2. Идентификаторы (20).	
1.1.3. Числа (20). 1.1.4. Метки (22). 1.1.5. Перемен-	
ные (22). 1.1.6. Функции (23). 1.1.7. Строки (24).	
1.1.8. Комментарии (24).	
Упражнения	25
1.2. Сравнение выражений	25
1.2.1. Арифметические выражения (25). 1.2.2. Логиче-	
ские выражения (28). 1.2.3. Именующие выражения	
(29). 1.2.4. Условные выражения (30).	
Упражнения	31
1.3. Сравнение операторов	31
1.3.1. Пустые операторы (31). 1.3.2. Операторы пере-	
хода (32). 1.3.3. Операторы процедуры (32). 1.3.4. Опе-	
раторы присваивания (32). 1.3.5. Условные операторы	
(33). 1.3.6. Операторы цикла (34). 1.3.7. Составные	
операторы (35). 1.3.8. Блоки (36). 1.3.9. Помеченные	
операторы (36).	
Упражнения	36
1.4. Сравнение описаний	37
1.4.1. Описания типа (37). 1.4.2. Описания массивов	
(38). 1.4.3. Описания переключателей (39). 1.4.4. Опи-	
сания процедур (40).	
Упражнения	46
	3

1.5. Сравнение программы	46
1.5.1. Сравнение программы (46). 1.5.2. Простейший ввод-вывод (47). 1.5.3. Выход на машину (48).	
Упражнения	48
ГЛАВА 2. Уточнение понятий PL/1	49
2.1. Уточнение основных элементов	49
2.1.1. Арифметические данные (50). 2.1.2. Строчные данные (55). 2.1.3. Проблемные данные и преобразова- ния типа (58). 2.1.4. Меточные данные (59). 2.1.5. Массивы и сечения (61). 2.1.6. Начальные значения (61). 2.1.7. Терминология и формы (64).	
Упражнения	65
2.2. Уточнение выражений	65
2.2.1. Арифметические операции (66). 2.2.2. Строчные операции (70). 2.2.3. Операции над массивами (72). 2.2.4. Формы (72). 2.2.5. Встроенные функции (73). 2.2.6. Псевдопеременные (77).	
Упражнения	78
2.3. Уточнение операторов	79
2.3.1. Терминология (79). 2.3.2. Оператор присваива- ния (80). 2.3.3. Условный оператор (80). 2.3.4. Груп- повой оператор (81). 2.3.5. END-оператор (84).	
Упражнения	84
2.4. Уточнение описаний и блоков	85
2.4.1. Оператор описаний (85). 2.4.2. Блоки (86) 2.4.3. Процедурный блок (90). 2.4.4. Аргументы и параметры (92). 2.4.5. Программа и подпрограммы (97).	
Упражнения	98
ГЛАВА 3. Новые понятия PL/1	100
3.1. Структуры	100
3.1.1. Описания структур (100). 3.1.2. Структурные пе- ременные и квалифицированные имена (105). 3.1.3. Струк- турные выражения (106). 3.1.4. Уточнение форм (109).	
Упражнения	110
3.2. Ситуации и их обработка	111
3.2.1. Ситуации (111). 3.2.2. Включение и выключение ситуаций (112). 3.2.3. Оператор ON (114). 3.2.4. Функции для ситуаций (116). 3.2.5. Формы (117). 3.2.6. Исполни- вание ситуаций при отладке (117).	
Упражнения	119
3.3. Ввод и вывод	119
3.3.1. Операторы ввода-вывода (119). 3.3.2. Ввод-вы- вод списком (122). 3.3.3. Ввод-вывод данными (124).	

3.3.4. Ввод-вывод с редактированием (126).	
3.3.5. Ситуации ввода-вывода (136).	
Упражнения	137
3.4. О других возможностях PL/1	138
ГЛАВА 4. Справочная	140
4.1. Встроенные функции	140
4.1.1. Математические функции (141).	
4.1.2. Арифметические функции (142).	
4.1.3. Строчные функции (144).	
4.1.4. Функции для массивов (147).	
4.1.5. Функции ситуаций (148).	
4.1.6. Разные функции (149).	
4.2. Точность преобразований и выражений	149
4.2.1. Арифметические преобразования (150).	
4.2.2. Строчные преобразования (152).	
4.2.3. Преобразования типа (153).	
4.2.4. Точность выражений (156).	
4.2.5. Сложные преобразования (157).	
4.3. Ситуации	160
4.3.1. Ситуации вычислений (160).	
4.3.2. Ситуации ввода-вывода (161).	
4.3.3. Ситуации, определяемые программистом (162).	
4.3.4. Ситуации системной реакции (162).	
4.4. Обозначения и формы	163
4.4.1. Список обозначений (163).	
4.4.2. Сравнительные формы алгола и PL/1 (165).	
4.4.3. Формы PL/1 (169).	
4.4.4. Список атрибутов (174).	
4.4.5. Ключевые слова PL/1 (175).	
4.4.6. Символы ЕС ЭВМ (178).	
Ответы к упражнениям	180
Литература	188
Предметный указатель	189

ПРЕДИСЛОВИЕ

В последние полтора десятилетия основным, а в начале и единственным, алгоритмическим языком в Советском Союзе был алгол-60, который использовался как для обучения основам программирования, так и для решения вычислительных задач на ЭВМ. Однако в последние годы после появления ЭВМ Единой системы (ЕС ЭВМ) все большее распространение получает PL/1 (Programming Language — язык программирования), также принадлежащий к классу алгоритмических языков. В отличие от алгола-60, предназначенного для решения задач численного анализа, PL/1 является универсальным языком, позволяющим, кроме того, формулировать решение коммерческих задач и задач обработки символьной и двоичной информации, задач АСУ. Можно, в частности, считать, что PL/1 объединяет возможности таких широко распространенных языков как алгол, фортран и кобол. На машинах Единой системы PL/1 заменяет алгол в практической работе, вытесняя его, что можно объяснить не только универсальностью PL/1 и другими его достоинствами, но и крупными недостатками транслятора с алгола-60 в ЕС ЭВМ (неэффективность транслятора, неудобства входного языка и особенно операторов ввода-вывода, невозможность стыковки с другими алгоритмическими языками, отсутствие пакетов прикладных программ).

Настоящий курс языка PL/1 имеет целью облегчить программистам переход в своей работе с других машин на ЕС ЭВМ и в первую очередь рассчитан на тех программистов, которые имеют опыт программирования на алголе-60 и желают в короткий срок ознакомиться с основными возможностями PL/1 и научиться программировать на этом языке для ОС ЕС ЭВМ. В курсе сделана попытка при обучении программированию на PL/1 самым существенным образом использовать те навыки и тот опыт программирования на алголе, которые имеются у читателя. Правда, при этом от читателя потребуются помимо практического опыта еще и знакомство с терминологией и синтаксисом алгола-60 ([1, 2]).

В 1-й главе курса дается сравнение понятий алгола-60 и PL/1, которое проводится таким образом, чтобы дать читателю представление о совпадении и различии конструкций алгола-60 (в эталонной версии) и PL/1 (для ОС ЕС ЭВМ), имеющих одни и те же функциональные возможности. Используемый метод сравнения позволит читателю быстро начать составлять программы на PL/1 в пределах тех его языковых возможностей, которые пересекаются с возможностями алгола. В конце 1-й главы приводятся сведения, которые помогут читателю подготовить свою программу для трансляции и счета на машине.

Во 2-й главе понятия PL/1, введенные в 1-й главе, уточняются и, кроме того, вводятся новые, которые являются как бы логическим развитием понятий алгола. Знакомство с этой главой позволит читателю использовать при программировании средства PL/1, близкие к алгольным, а также поможет ему в сложных случаях применения понятий PL/1, введенных в 1-й главе.

В 3-й главе излагаются понятия PL/1, не имеющие прямых аналогий с алгольными; без них картина основных средств PL/1 не была бы полной и не было бы возможным составление эффективных программ в достаточно широкой области применения. В конце главы перечислены те возможности PL/1, которые остались незатронутыми в этом курсе и с которыми читатель может ознакомиться по имеющейся литературе [3, 4, 5, 10] или, для наиболее сложных понятий, по документации для ОС ЕС ЭВМ [6, 7, 8, 9].

В 4-й, справочной главе дается сводка встроенных функций и преобразований PL/1, таблицы синтаксических форм PL/1 (одна из них в сравнении с алголом) и список используемых обозначений, а также другие справочные материалы.

Все главы (кроме последней) сопровождаются упражнениями, ответы к которым приведены в конце книги (иногда дается один из возможных ответов). Решение упражнений и разбор примеров, данных при изложении основного материала, играют самую существенную роль в изучении средств языка и приобретении навыков в составлении программ на PL/1.

Наряду с примерами и упражнениями, другим средством для контроля правильности и полноты понимания изучаемых понятий языка должны стать приводимые синтаксические формы, которые выражают некоторые свойства этих понятий более четко и наглядно, чем при использовании только словесных описаний. Разбор и, в дальнейшем, обращение к этим формам, должны препятствовать появлению в программах синтаксических ошибок.

Отметим еще раз, что курс ориентирован, в основном, на программистов-практиков, которым необходимо не столько ознако-

миться с особенностями языка или изучить все его понятия, сколько достаточно быстро получить возможность составлять на нем реальные программы. Учитывая, кроме того, что у читателя предполагается известный опыт программирования на алголе, языке в определенном отношении весьма близком к PL/1, автор считает, что некоторая краткость в изложении (компенсированная примерами, формами и упражнениями) будет желательной для таких читателей.

В настоящем курсе язык PL/1 описывается на уровне конкретного представления как входной язык транслятора версии F для ОС ЕС ЭВМ.

Автор признателен всем, кто в той или иной мере содействовал написанию и изданию этой книги. Особую благодарность автор выражает Н. П. Трифонову за детальную и конструктивную критику рукописи, а также Т. Н. Мошонкиной и А. Х.-М. Алдакову за многочисленные и полезные обсуждения и постоянную помощь в работе.

Ю. М. Безбородов

ВВЕДЕНИЕ

0.1. Общее знакомство с PL/1

Для начала знакомства с PL/1 и общего сравнения основных его понятий с понятиями эталонного алгола-60 приведем на обоих языках примеры программ, выполняющих одни и те же вычисления (см. примеры 1 и 2).

В программе алгола вводится исходное значение переменной n , характеризующей верхнюю границу массива S , для которого затем также вводятся исходные значения. Элементы введенного массива проверяются на выполнение некоторого соотношения: если оно не выполняется, то печатается номер элемента и проверка массива заканчивается; в любом случае печатается проверяемый массив (для отчетности). Несмотря на некоторую искусственность, пример имеет то достоинство, что в нем присутствуют все 8 типов операторов алгола и все основные виды описаний и выражений.

Сравним две приведенные программы.

1. Программа в PL/1, как и в алголе, представляет собой последовательность описаний и операторов, но в отличие от алгола является не блоком, а описанием процедуры с заголовком специального вида.

2. Практически все операторы и описания алгола имеют свой аналог в PL/1, который отличается, как правило, лишь ключевыми словами (ограничителями): BEGIN; или DO; вместо **begin**, FLOAT вместо **real**, BY вместо **step**, = вместо := и т. п.; CALL добавлено перед оператором процедуры, имя которой вынесено в качестве метки при ее описании; описания могут быть объединены (за словом DECLARE); опускается DO; , когда оно соответствует **begin** для составного оператора, являющегося телом процедуры или оператора цикла. Кроме того, в PL/1 каждый оператор (и описание) оканчивается точкой с запятой, в то время как в алголе точка с запятой разделяет операторы (и описания), а не входит в них.

Пример 1. Программа на алголе.

begin

integer n ; *ininteger* (0, n);

BLOK: begin array $S[1:n]$;

real f, g ; integer i ;

procedure TWO (x, y, z);

real x, y, z ;

begin $y := -2/3 \times \sin(x \uparrow 2.3)$;

$z := 2_{10} - 4/\ln(x + 0.2)$

end;

inarray (0, S);

CYCL: for $i := 1$ step 1 until $n - 1$ do

begin TWO ($S[i], f, g$);

if $f > S[i+1] \wedge g \leq S[i+1]$ then

else begin *outinteger* (1, i);

go to OFF

end

end;

OFF: *outarray* (1, S)

end BLOK

end

3. Выражения алгола и их элементы также имеют свой аналог в PL/1, который отличается лишь видом букв, стандартных идентификаторов, знаков операций и других ограничителей: ** вместо \uparrow , & вместо \wedge , E вместо $_{10}$ и т. п.

Таким образом, для большинства основных понятий алгола в PL/1 имеются эквиваленты, используя которые можно составлять, по крайней мере, простые программы на PL/1. Но при этом, конечно, нужно знать все ограничения и особенности применения таких эквивалентов в PL/1 (например, неожиданно оказывается, что в PL/1 выражение $1 + 1/3$ или указатель функции TWO (2.5, F, G) в приведенной программе были бы ошибочными).

Вместе с тем, некоторые понятия алгола не имеют своего аналога в PL/1. Ниже перечисляются такие понятия и возможности алгола (звездочками отмечены те из них, которые отсутствуют также и в стандартном подмножестве (Subset'e алгола-60 [2]):

- условные выражения;
- элемент списка цикла типа **while**;
- целые в качестве меток^(*);
- возможность неполной совокупности спецификаций формальных параметров^(*);
- фактические параметры, вызываемые по наименованию и отличающиеся от переменных^(*);

Пример 2. Программа на PL/1.

```
PROG: PROCEDURE OPTIONS (MAIN);  
    DECLARE N FIXED; GET LIST (N);  
    BLOK: BEGIN; DECLARE S(1:N) FLOAT;  
            DECLARE (F, G) FLOAT, I FIXED;  
    TWO: PROCEDURE (X, Y, Z);  
        DECLARE (X, Y, Z) FLOAT;  
        Y = -2/3 * SIN (X ** 2.3);  
        Z = 2E-4 / LOG (X + 0.2);  
    END;  
        GET LIST (S);  
    CYCL: DO I=1 BY 1 TO N-1;  
        CALL TWO (S(I), F, G);  
        IF F > S(I+1) & G <= S(I+1) THEN;  
            ELSE DO; PUT LIST (I);  
                GO TO OFF;  
            END;  
        END;  
    END;  
        OFF: PUT LIST (S);  
    END; /* BLOK */  
END;
```

— список формальных параметров для вызова по значению;
— операции \div , \supset , \equiv .

С другой стороны, в PL/1 имеется множество понятий и возможностей, которые отсутствуют в алголе. Это объясняется, например, тем, что PL/1 разрабатывался как универсальный язык, в то время как алгол предназначается лишь для решения задач численного анализа. Ниже дается краткая характеристика возможностей PL/1, отсутствующих в алголе.

1. В PL/1 помимо работы с данными типа **real**, **integer**, **Boolean** можно оперировать с комплексными, двоичными, строчными данными; имеются меточные (именующие) переменные.

2. Кроме массивов в PL/1 имеются и *структуры*, которые являются объединением данных с различными характеристиками (массивы и простые переменные, строчные и числовые, вещественные и комплексные и т. п.). Возможны операции не только над скалярными величинами, как в алголе, но и целиком над массивами и над структурами.

3. Программа в PL/1 может состоять из нескольких независимо транслируемых частей (процедур), одна из которых является главной (MAIN). Кроме локальных величин, автоматически размещаемых в памяти при входе в блок, в PL/1 допускаются вели-

чины, размещение которых в памяти осуществляется по специальным операторам, а также величины, к которым имеется доступ из любых частей программы.

4. Имеются разнообразные и мощные средства ввода-вывода (включающие работу и с внешней памятью), которые, вместе с тем, очень просты при работе с данными в стандартных форматах.

5. Программист имеет возможность при возникновении в ходе выполнения программы аварийных или других особых состояний машины (ситуаций) задать и осуществить необходимые ему действия.

6. В PL/1 введены машиннозависимые средства, использование которых требует знания особенностей конкретной ЭВМ (например, внутреннего представления данных). Отметим, что поскольку ниже язык PL/1 описывается на уровне конкретного представления для ОС ЕС ЭВМ, то ряд сведений, например, количественные ограничения, естественно, будут иметь машиннозависимый характер.

7. Заметим также, что в языке PL/1 широко используется принцип умалчивания, в соответствии с которым в программе допускаются неописанные или не полностью описанные идентификаторы. Транслятор автоматически приписывает этим идентификаторам заранее определенные «умалчиваемые» свойства, зависящие, например, от контекста, в котором встречаются идентификаторы.

0.2. О способе изложения

На рис. 1 условно показано соответствие между языковыми возможностями, имеющимися в алголе (малый круг) и в PL/1 (большой круг).

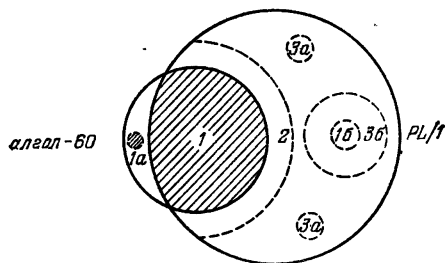


Рис. 1. Соответствие возможностей алгола и PL/1.

Область 1, принадлежащая обоим кругам, содержит те возможности PL/1, которые есть и в алголе и изложение которых удобно вести, сравнивая их между собой. Область 1а содержит те возможности алгола, которых, строго говоря, нет в PL/1, и которых следует избегать при разработке детальных алгоритмов

для программ на PL/1. Но некоторые возможности алгола из этой области могут быть реализованы в PL/1 с помощью других, имеющихся в нем средств.

В 1-й главе курса будут изложены возможности PL/1, соответствующие области *I* и частично области *1a*; кроме того, читатели ознакомятся с простейшим вводом-выводом в PL/1 (область *1б*), что позволит им составлять простые программы и решать их на машине. Понятия PL/1 вводятся в 1-й главе курса путем сравнения их с соответствующими понятиями алгола-60, причем в качестве сравниваемых понятий PL/1 часто будут браться не истинные понятия PL/1, а ограниченные по своим возможностям таким образом, чтобы они наиболее точно соответствовали функциональным возможностям понятий в алголе. Такой подход позволит выявить среди средств PL/1 такие, которыми привык пользоваться программист, работающий на алголе. Кроме того, это сократит различие между конструкциями PL/1, отвечающими этим *упрощенным* понятиям, и конструкциями алгола до минимума, и позволит нам сформулировать результат сравнения понятий алгола и PL/1 в максимально простой и наглядной форме. Часто достаточно будет привести примеры, и соответствие между понятиями двух языков станет очевидным.

Если не оговаривается противное, то предполагается, что все возможности и свойства рассматриваемого понятия алгола (известные читателю) сохраняются и для соответствующего понятия PL/1.

Сравнение понятий алгола и упрощенных понятий PL/1 строится, как правило, по единой схеме и разделяется на строгое и нестрогое.

Нестрогое сравнение отмечает лишь главные различия в соответствующих понятиях и основную роль при этом будут играть примеры.

Строгое сравнение проводится на основе синтаксических форм для понятий сравниваемых языков. Для подчеркивания имеющихся у форм общих свойств их сравнение дается в виде формальных *правил перевода* конструкций алгола в соответствующие конструкции PL/1. (Следует, однако, отметить, что визуальное сравнение форм обычно дает более ясное представление о синтаксисе понятий PL/1, чем словесная формулировка перевода, которую следует рассматривать скорее как вспомогательное средство для тех, кто еще не привык к используемым в формах обозначениям или забыл синтаксис понятий алгола.) Здесь же приводятся ограничения и особенности и даются более сложные примеры.

Область 2 содержит те возможности языка PL/1, которых нет в алголе, но которые весьма близки к алгольным и являются их модификацией или расширением. Изложение таких средств PL/1

ведется во 2-й главе путем уточнения тех понятий, которые были введены в 1-й главе. При этом уточнении прослеживается логическая связь новых понятий (или новых возможностей) PL/1 с аналогичными понятиями алгола.

Из оставшихся возможностей PL/1, не вошедших в области 1 и 2, в 3-й главе рассматриваются лишь некоторые, связанные с вводом-выводом (3б), имеющие аналогии с алгольными возможностями или необходимые при практическом программировании (3а).

0.3. Основные обозначения

Обозначения, используемые в книге, преследуют цель с максимальной наглядностью выявить соответствия и различия в синтаксисе основных понятий двух языков. Вследствие этого они очень лаконичны и составляют значительное количество, что поначалу будет затруднять их запоминание. Но так как введенные обозначения будут встречаться сначала в конструкциях алгола, хорошо знакомых читателю, то это должно помочь быстро привыкнуть к ним.

Кроме обозначений, приводимых ниже, в синтаксических формах используются *литералы*, т. е. один символ или сочетание символов, допустимых в конкретном языке. В алголе таковыми являются, например, буквы, цифры и ограничители вида $+$, $:=$, **begin**, **procedure** и т. д. В PL/1 им соответствуют буквы, цифры, спецзнаки и ключевые слова $+$, $=$, **BEGIN**, **PROCEDURE** и т. п.

Конструктивные обозначения. Здесь приводятся обозначения, которые используются в формах для отражения конструктивных свойств основных понятий языка, для указания порядка следования элементов в определяемой конструкции. Большинство из этих обозначений являются стандартными для документации ЕС ЭВМ.

\sim — «соответствует», «это».

$|$ — «или».

$\{ \}$ — выбор (альтернатива). Например:

$$\begin{aligned} \left\{ \begin{array}{c} x \\ y \\ z \end{array} \right\} &\sim \{x|y|z\} \sim x \text{ или } y \text{ или } z \\ x\{y|z\} &\sim \{xy|xz\} \end{aligned}$$

Иногда эти скобки будут употребляться и просто для выделения какой-либо группы элементов конструкции в качестве одного ее элемента (см. ниже).

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \sim [x|y|z] \sim x \text{ или } y \text{ или } z \text{ или пусто}$$

$$[x] y \sim \{xy \mid y\}$$

$$(x) y (z) \sim \{xyz \mid xy \mid yz\}$$
$$[x] \, y \, [z] \sim \{xyz \mid xy \mid yz \mid y\}$$
$$x \langle y \rangle z \sim xz$$
$$z \dots \sim \{z | zz | zzz | zzzz \langle \text{и т. д.} \rangle\}$$

$$yz \dots \sim \{yz | yzz | yzzz \text{ (и т. д.)}\}$$

$$x\{yz\} \dots \sim \{xyz \mid xyzyz \mid xyzyzyz \text{ (и т. п.)}\}$$

$$\left\{ \begin{matrix} y \\ z \end{matrix} \right\} \dots \sim \left\{ \left\{ \begin{matrix} y \\ z \end{matrix} \right\} \middle| \left\{ \begin{matrix} y \\ z \end{matrix} \right\} \left\{ \begin{matrix} y \\ z \end{matrix} \right\} \left\{ \begin{matrix} y \\ z \end{matrix} \right\} \left\{ \begin{matrix} y \\ z \end{matrix} \right\} \left\{ \begin{matrix} y \\ z \end{matrix} \right\} \langle n \text{ т. п.} \rangle \right.$$

То есть, повторение (...) является более старшей «операцией» чем выбор ({ } или [] или ()).

$$yz \dots \sim \{y | yzy | yzyzy \langle \text{и т. п.} \rangle\}$$

$$yz \dots \sim y [zy] \dots$$

$$y, \dots \sim \{y | y, y | y, y, y \text{ (и т. п.)}\}$$

$$xy, \dots \sim \{xy \mid xy, y \mid xy, y, y \text{ (и т. п.)}\}$$

→ — преобразование. Будет, в частности, использоваться для указания на факт преобразования конструкции одного языка в конструкцию другого языка (алгола в PL/1).

*) Этот термин (option—выбор, optional—необязательный) может использоваться и в дальнейшем, например, для характеристики тех возможностей операторов PL/1, которые могут быть при анализе опущены.

то они будут подчеркиваться. Например, квадратные скобки как символы алгола имеют вид [].

Обозначения понятий. Ниже приведены обозначения для основных понятий языков алгола и PL/1, используемые в 1-й главе курса; остальные обозначения будут даваться по ходу изложения. Все обозначения собраны в п. 4.4.1, куда и следует обращаться в необходимых случаях.

Понятия алгола и соответствующие им по функциям понятия PL/1 обозначаются одними и теми же полужирными буквами, но для алгола они даются *курсивом*. Терминология для таких «соответствующих» понятий PL/1 используется сначала, как правило, алгольная; уточнение терминологии дается позднее, во второй главе.

- a** и **a**—фактический параметр (actual parameter),
- d** и **d**—описание (declaration),
- e** и **e**—выражение (expression),
- f** и **f**—функция, указатель функции (function),
- i** и **i**—идентификатор (identifier),
- l** и **l**—метка (label),
- n** и **n**—число (number),
- p** и **p**—формальный параметр (parameter),
- s** и **s**—оператор (statement),
- v** и **v**—переменная (variable).

К этим ведущим обозначениям могут добавляться модифицирующие обозначения, которые отражают некоторые дополнительные свойства основных понятий.

- A** и **A**—арифметический (Arithmetical). Например, **eA**—выражение арифметическое, **fA**—функция арифметическая;
- E** и **E**—вещественный (rEal). Например, **fE**—вещественная функция (допускается и **fAE**);
- G** и **G**—логический, булевский (loGical). Например, **vG**—булевская переменная;
- I** и **I**—целый (Integer). Например, **nI**—целое число;
- L** и **L**—меточный, именующий (Label). Например, **iL**—идентификатор метки, **eL**—именующее выражение;
- P** и **P**—относящийся к процедуре (Procedure). Например, **iP**—идентификатор процедуры;
- M** и **M**—относящийся к массиву, имеющий размерность (diMension). Например, **iM**—идентификатор массива;
- V** и **V**—относящийся к переменной (Variable). Например, **iV**—идентификатор переменной.

Договоримся ведущее обозначение писать слитно с модификаторами; порядок написания модификаторов после ведущего обозначения не закрепляется.

Кроме перечисленных понятий и модификаторов могут использоваться и другие понятия или модификаторы, имеющие вид группы русских слов, соединенных дефисом. Например:

программа, строка-букв, *s*-цикла

Упражнения.

Используя введенные обозначения, выполнить следующие упражнения для понятий алгола.

- 0.1. Определить цифру в алголе.
- 0.2. Обозначив цифру алгола через *z*, определить целое.
- 0.3. Считая, что *x*—буква, а *z*—цифра, определить идентификатор.
- 0.4. Определить непомеченный составной оператор.
- 0.5. Определить блок.

ГЛАВА 1

СРАВНЕНИЕ ПОНЯТИЙ АЛГОЛА И PL/1

Приступая к сравнению алгола-60 с PL/1, читателю следует освежить в своей памяти терминологию, принятую в алголе, синтаксис и семантику его понятий. Нужно иметь в виду, что затруднения, возникающие при чтении 1-й главы, могут быть вызваны пробелами в знаниях, касающихся алгола, а не сложностью понятий PL/1.

Сравнение понятий алгола и PL/1 проводится в следующем порядке: основные элементы, выражения, операторы, описания, программа.

1.1. Сравнение основных элементов

1.1.1. Символы.

Символы PL/1 делятся на буквы, цифры и специальные знаки (ограничители).

Буква PL/1:

$\alpha \sim \{A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|$
 $R|S|T|U|V|W|X|Y|Z|\text{Q}|\text{@}|\#\}$

Среди букв PL/1 по сравнению с набором букв эталонного языка алгол-60 отсутствуют строчные латинские буквы, но имеются 3 дополнительные «буквы»: Q — знак денежной единицы, @ — коммерческое «at», $\#$ — номер (конечно, употребление этих «букв» и в другом смысле не возбраняется).

Цифра PL/1:

$v \sim \{0|1|2|3|4|5|6|7|8|9\}$

Цифры в PL/1 обычные; но ноль при написании его на бланках и при печати обычно перечеркивается (\emptyset) для отличия его от буквы O.

Спецзнак PL/1:

$\omega \sim \{+|-|*|/|<|=|>|\&|!|!|.|.:|;|()|'|_|_|_|%|?\}$

Среди спецзнаков обращает на себя внимание отсутствие знаков \times , \div , \uparrow , \leq , \geq , \neq , $[,]$ и знаков логических операций (кроме \neg). Правила замены этих знаков на спецзнаки PL/1 будут даваться по ходу сравнения понятий двух языков.

Вместо ограничителей алгола, имеющих вид подчеркнутых (выделенных полужирным шрифтом) английских слов, в PL/1 применяются *ключевые слова*, с частью которых мы уже встречались во введении. В PL/1 они не зарезервированы, то есть такие же слова могут употребляться и в другом смысле, например, в качестве идентификаторов PL/1.

Таким образом, ограничителям алгола соответствуют спецзнаки и ключевые слова PL/1.

Пробел в PL/1 (представленный знаком $_$) отнесен к спецзнакам и, таким образом, употребление его в PL/1 не произвольное, как в алголе, а подчиняется определенным правилам, которые будут даваться по ходу определения понятий PL/1. Пока мы будем руководствоваться следующим общим правилом.

Правило. Ключевые слова, идентификаторы (п. 1.1.2), числа (п. 1.1.3), строки (п. 1.1.7) должны отделяться друг от друга по крайней мере одним пробелом. Пробелы внутри ключевых слов не допускаются (исключение GO TO). Перед или после спецзнака можно ставить один или более пробелов; исключения будут даны в ходе изложения.

Формы. Ниже приводятся формы для символов алгола и PL/1, обозначенных соответственно через σ и σ . Здесь и в дальнейшем сначала дается форма (иногда формы) алгола, а под ней формы PL/1.

$$\sigma \sim \{\alpha | v | \omega | \lambda\}$$

$$\sigma \sim \{\alpha | v | \omega\}$$

Формы для букв, цифр и ограничителей алгола здесь, ради краткости, не приводятся, и читатель может найти их в 4.4.2. О логических значениях PL/1, соответствующих алгольным значениям ($\lambda \sim \{\text{true} | \text{false}\}$), будет сказано позднее.

В PL/1 в некоторых случаях разрешается использование любых символов, имеющихся на устройствах ввода-вывода конкретной ЭВМ. Мы примем, что «символами ЭВМ» являются:

$$\xi \sim \{\sigma | \text{Б} | \text{Г} | \text{Д} | \text{Ж} | \text{З} | \text{И} | \text{Й} | \text{Л} | \text{П} | \text{У} | \text{Ф} | \text{Ц} | \text{Ч} | \text{Ш} | \text{Щ} | \text{Ъ} | \text{Ы} | \text{Э} | \text{Ю} | \text{Я}\}$$

То есть в символы ЭВМ помимо основных символов входят заглавные буквы русского алфавита, не совпадающие по начертанию с латинскими.

Сформулируем правило перевода (транслитерации) букв и цифр алгола в буквы и цифры PL/1. На эти правила мы будем ссылаться в дальнейшем изложении.

Перевод. Заглавные и строчные латинские буквы алгола

заменяются на заглавные; цифры алгола остаются неизменными (при написании на бланке нуль переходит в 0). Если в каком-нибудь блоке алгола используются одни и те же заглавные и строчные буквы, то при замене строчных букв на заглавные можно рекомендовать добавлять еще и дополнительные буквы PL/1 (O, @, #).

Примеры *).

$D \rightarrow D \quad 9 \rightarrow 9$

$d \rightarrow D@ \quad 0 \rightarrow 0$

1.1.2. Идентификаторы.

Идентификаторы PL/1 отличаются только тем, что в них наряду с буквами и цифрами может присутствовать и знак разбивки «_» (в PL/1 ЕС допускается и «—»), но не на первом месте. Кроме того, идентификаторы PL/1 не должны содержать пробелов (вместо них можно употреблять знак разбивки). Сравните, например:

SUM и SUM, I5 и L5,

TEMPERATURA PЕCHИ и TEMPERATURA _ PЕCHИ

Формы.

$i \sim \alpha[\alpha|v] \dots$

$i \sim \alpha[\alpha|v|_] \dots \langle \text{Max} = 31 \rangle$

Перевод. Перевод идентификатора сводится к замене составляющих его символов (см. п. 1.1.1). Пробелы, присутствующие среди символов идентификатора алгола, можно заменить на знаки разбивки (_). При этом нужно следить, чтобы в идентификаторе было не более 31 символа.

Примеры.

COR 10 \rightarrow COR _ 10

V47b \rightarrow V47 B@ или V47B

OCOST _ _ STEEL _ # 12

1.1.3. Числа.

Числа в PL/1 практически отличаются от чисел в алголе только тем, что вместо i_0 пишется буква E и невозможны числа вида $E+5$ (вместо i_0+5). Сравните, например:

71.235 и 71.235

$-0.083_{i_0} + 12$ и $-0.083E + 12$

$-_{i_0} - 7$ и $-1E - 7$

500 и 500

.9 и .9

Однако смотрите ниже ограничения.

Формы.

$nI \sim [\pm] v \dots$

$n \sim [\pm] (\tilde{nI}) (\cdot \tilde{nI}) (i_0 [\pm] \tilde{nI})$

*) В соответствии с принятой традицией буквы и цифры в алголе даются *курсивным* шрифтом, а в PL/1 — обычным прямым.

$\langle \tilde{n}l \sim v. \dots \rangle$

$nl \sim [\pm] v \dots$

$n \sim [\pm] (\tilde{n}l) [.] (\tilde{n}l) [E [\pm] \tilde{n}l]$

$\langle \tilde{n}l \sim v \dots \rangle$

Здесь: nl — целое,

$\tilde{n}l$ — целое без знака.

Ломаные скобки () показывают, что в числе алгола должна обязательно быть хотя бы одна из трех конструкций, заключенных в такие скобки. В PL/1 должна обязательно присутствовать хотя бы одна из двух конструкций, заключенных в те же скобки. То есть допустимы числа с формой $[\pm] \tilde{n}l.[E [\pm] \tilde{n}l]$, например + 49. или 88.E—5

Перевод. Для перевода числа следует заменить i_0 на E и в случае, если число имеет вид $[\pm]_{i_0} [\pm] \tilde{n}l$, дополнительно перед E поставить цифру 1.

Ограничения.

1. Число PL/1 не должно содержать пробелов внутри себя.

2. Предельными значениями в ЕС будем пока считать (уточнения см. в 2.1.1):

— для целых чисел — $\pm 10^5$,

— для чисел с порядком — $\pm 10^{75}$ (в мантиссе не более 6 цифр, в порядке числа не более двух цифр),

— для нецелых чисел без порядка — $\pm 10^5$ (но не более 15 цифр).

З а м е ч а н и е.

В PL/1, как и в алголе, числа без дробной части и порядка имеют тип *целый* (Integer), но точка в их изображении может присутствовать (например, — 56.). В PL/1 числами, соответствующими вещественному типу (*real*) в алголе, являются числа с порядком; они имеют тип *с плавающей точкой*, или сокращенно — *плавающий* (Float), обозначаемый через E. Числа с дробной частью, но без порядка, которые в алголе имеют тип *real*, в PL/1 имеют тип *с фиксированной точкой*, или сокращенно — *фиксированный* (Fixed) с диапазоном изменения, значительно меньшим, чем у чисел с порядком.

В дальнейшем в 1-й главе мы будем иметь дело только с переменными и функциями, значения которых представляются числами типа *целый* или *плавающий*, с диапазонами изменения, указанными выше.

Примеры.

Целое число: 125.

Число с фиксированной точкой: 125.0

Число с плавающей точкой: 125E0

Неправильные числа:

22E+003 (в порядке более двух цифр),

12.34567890123456 (больше 15 цифр),

3.3E+79 (порядок превышает 75),

12.345 678 901 (пробелы внутри числа).

1.1.4. Метки.

Метки в PL/1 имеют практически тот же вид, что и в алголе, если не считать того, что в PL/1 отсутствуют метки, имеющие вид целых без знака.

Формы.

$$l \sim \{iL | \tilde{n}l\}$$
$$l \sim iL$$

Перевод. Перевод метки, являющейся идентификатором, осуществляется в соответствии с п. 1.1.2.

Примеры.

$l1 \rightarrow L@1$, $M2 \rightarrow M2$,

$one\ or\ two \rightarrow ONE_OR_TWO$

1.1.5. Переменные.

Переменные PL/1 являются обобщением переменных в алголе. Пока (уточнения см. в 2.1) мы будем считать, что они отличаются только видом индексных скобок (круглые вместо квадратных). Сравните, например:

$V1$ и $V1$

$S[7; K-2]$ и $S(7, K-2)$

Смотрите, однако, ниже ограничение.

Формы.

$$v \sim \{iV | iM[eA, ..]\}$$
$$v \sim \{iV | iM(eA^+, ..)\}$$
$$\langle eA^+ \sim \{eA1 | FLOOR(eAE + 0.5)\} \rangle$$

В форме алгола подчеркнутые квадратные скобки являются индексными скобками. Через iV обозначен идентификатор простой переменной (лучше было бы iS —идентификатор скаляра, где S —*Scalar*). Значком плюс у eA мы отметили тот факт, что соответствие арифметических выражений алгола и PL/1 в данном случае имеет специфический характер (см. ниже ограничение). Функция $FLOOR$ выделяет целую часть аргумента.

Перевод. Для перевода переменной следует перевести ее идентификатор (п. 1.1.2) и расположить за ним разделенные запятыми переведенные арифметические выражения (п.1.2.1), составляющие список индексов, который нужно заключить в круглые скобки вместо квадратных (если он есть).

Ограничения.

Если индексное выражение eA имеет тип *real*, то оно должно быть заменено на $FLOOR(eA + 0.5)$. Это объясняется тем, что

в PL/1 преобразование от вещественного типа к целому производится отбрасыванием дробной части (и отрицательной), а не взятием ближайшего целого, как в алголе *). Значения индексов могут меняться в пределах от -32767 до $+32767$ ($\pm 2^{15} \mp 1$).

Примеры.

$P3 [M, T [N, 3]] \rightarrow P3 (M, T (N, 3))$

$DELTA [R/Q] \rightarrow DELTA (FLOOR (R/Q + 0.5))$

1.1.6. Функции.

Указатели функций в PL/1 и алголе по внешнему виду практически не отличаются: лишь идентификаторы стандартных (*встроенных*) функций *arctan*, *ln*, *entier* заменяются соответственно на ATAN, LOG, FLOOR, а остальные — на ABS, EXP, SIN, COS, SQRT, SIGN (об особенностях функции FLOOR см. сноску к 1.1.5). Сравните, например:

$FUN1 (P, 2.5, Q [I])$ и $FUN1 (P, 2.5, Q (I))$

$TU (X + 1, \ln (t))$ и $TU (X + 1, LOG (T))$

При обращении к обычным (нестандартным) функциям имеются различия в задании фактических параметров, обсуждаемые в 1.4.4.

Формы.

$f \sim iP [(a, ..)] \quad \langle, \sim \{, | \} \alpha...: \{ \} \rangle$

$f \sim iP [(a, ..)] \quad \langle, \sim \{, |, /*\&...*/ \} \rangle$

В комментариях к форме для алгола отражен тот факт, что в качестве ограничителя параметра вместо запятой может быть конструкция вида:

) строка-букв: (

Перевод. Для перевода указателя функции нужно перевести идентификатор процедуры-функции (п. 1.1.2), учитывая указанное выше соответствие для стандартных функций, и расположить за ним разделенные запятыми переведенные фактические параметры (п.1.2 и 1.4.4), заключив весь список в круглые скобки (если он есть). Если ограничитель параметра вместо запятой имеет вид

) строка-букв: (

его можно заменить на

, /*строка-символов-ЭВМ*/

*) Строго говоря, предложенный способ получения ближайшего целого не совсем правилен, так как функция FLOOR в отличие от *entier* для аргументов с плавающей точкой выдает результат также с плавающей точкой, то есть, может быть, неточный. Поэтому, если бы в ЕС ЭВМ функция FLOOR для рассматриваемого диапазона давала результат с недостатком, то предложенная форма была бы неправомерна. Читателю следует избегать до освоения материала 2-й главы использования вещественных выражений в тех местах, где предполагаются целые значения.

Примененная конструкция для ограничителя PL/1 поясняется в п.1.1.8.

Примеры.

$S(S0) \text{ time: } (T) \text{ velocity: } (V)$
→ $S(S0, / * \text{ВРЕМЯ} * / T, / * \text{СКОРОСТЬ} * / V)$
 $\arctan(\text{sqrt}(R2)) \rightarrow \text{ATAN}(\text{SQRT}(R2))$

1.1.7. Строки.

Строки PL/1 в основном отличаются тем, что в них вместо кавычек используются апострофы, которые внутри строки всегда удваиваются. Кроме того, в строках PL/1 фигурируют символы конкретного представления, а не только основные символы, как в эталонном алголе. Например:

$'this \text{ is a } 'string''$
→ $'THIS \text{ IS A } " STRING''$
или → $'ЭТО " СТРОКА''$

Формы.

$g \sim '[\sigma | g]. \dots' < '[\{\sigma. \dots\} | '[\sigma. \dots]] \dots' >$
 $g \sim '[\xi | \dots]' < '[\{\xi. \dots\} | '[\xi. \dots]] \dots' >$

Из наборов символов σ и ξ здесь исключаются соответственно кавычки ('') или апостроф ('). Для строки алгола здесь дано рекурсивное определение. В комментариях приведены частные формы строк для лучшего выявления их соответствия.

Перевод. Для перевода строки нужно заменить составляющие ее символы (или их группы) на соответствующие им в PL/1, внешние кавычки—на апострофы, а внутренние—каждую на 2 апострофа. (Так как строки в алголе используются лишь как фактические параметры для процедур ввода-вывода и процедур кодов, то «соответствие» может быть весьма условным.)

Примеры.

$" " \rightarrow " " " "$
 $' \wedge - 'AND', \vee - 'OR' \rightarrow ' \& - " И", ! - "ИЛИ"'$
 $' \sqcup ' \sqcup - \sqcup ЭТО \sqcup АПОСТРОФ'$

Для последней строки аналога в алголе нет.

1.1.8. Комментарии.

Форма.

комментарии-PL/1 $\sim / * \xi \dots * /$

Комментарии в PL/1 содержат любые (кроме сочетания $*/$) символы конкретного представления и могут быть вставлены в любое место программы PL/1, где может стоять пробел, например, рядом с любым спецзнаком PL/1 (но не в строках). В частности, они могут стоять в тех местах, где располагаются комментарии в алголе: после точки с запятой (;), **begin** (BEGIN;) или **end** (END;).

Примеры.

```
BEGIN; /* НАХОЖДЕНИЕ ЭКСТРЕМУМА */  
END; /** ИЛИ */  
F1 /*ФУНКЦИЯ */ (A, B+1)  
T /* ПЕРЕМЕННАЯ */ (C, D-1)
```

Упражнения.

Написать для следующих конструкций алгола соответствующие им конструкции в PL/1.

- 1.1.1. *НОМЕР 6* 1.1.2. *B01 [i1, 11]*
1.1.3. $-1.34_{10}10$ 1.1.4. 10^8
1.1.5. *.000 125 439 786* 1.1.6. *entier (T [cos (R0)])*

1.2. Сравнение выражений

В алголе имеются 3 типа выражений: арифметическое, булевское (логическое), именуемое. В этом разделе мы проведем их сравнение с аналогичными выражениями PL/1.

1.2.1. Арифметические выражения.

Если ограничиться рассмотрением только простых арифметических выражений алгола, не содержащих условий (условные выражения рассматриваются в 1.2.4), то практически можно считать, что с точностью до замены одних ограничителей (\times , \uparrow) на другие (*, **) они совпадают с арифметическими выражениями PL/1. Сравните, например:

$$2 \times U - Q (S + C/4) \uparrow 3.3 \quad \text{и} \quad 2 * U - Q (S + C/4) ** 3.3$$

Имеются, однако, некоторые особенности (см. ниже).

Формы.

$$eA \sim \Phi_A \{vA | fA | n | + | - | \times | / | \div | \uparrow | (|)\}$$

$$eA \sim \Phi_A \{vA | fA | n | + | - | * | / <? > | / <? > | ** <? > | (|)\}$$

Здесь мы не приводим явной формы для арифметического выражения алгола ввиду ее сложности, а просто обозначаем форму через Φ_A с фигурными скобками, в которых перечислены составляющие ее компоненты. Формы показывают, что имеется однозначное соответствие между всеми элементами арифметических выражений двух языков и их расположение в выражениях подчиняется одним и тем же правилам: Φ_A одно и то же. Комментарий $<?>$ указывает на имеющиеся особенности в соответствии компонент выражений.

Перевод. Для перевода простого арифметического выражения алгола, не содержащего условий, следует перевести содержащиеся в нем числа, переменные и функции (п.п. 1.1.3, 1.1.5 и 1.1.6), заменить \times на *, \uparrow на ** (пробелы между звездочками не допускаются), оставить неизменными остальные ограничители (кроме \div), расставить все в прежнем порядке, и учесть следующие особенности.

Особенности.

1. Операция \div алгола сводится к операции / в PL/1 следующим образом:

$$x \div y \rightarrow \text{TRUNC}(x/y)$$

Функция TRUNC отсекает дробную часть аргумента.

2. В отличие от алгола несколько подряд идущих операций возведения в степень в PL/1 выполняются справа налево, поэтому при переводе выражений вида $x \uparrow y \uparrow z$ следует расставить дополнительные скобки: $(x**y)**z$

В то время как в алголе возведение в степень $(x \uparrow y)$ в случае, если оба операнда имеют тип целый, дает при $y \geq 0$ результат типа целый, в PL/1 практически можно считать (уточнения будут даны в 2.2.1 и 4.2.4), что целый результат получается, только если оба операнда — целые числа (степень должна быть целым без знака); иначе результат будет иметь тип плавающий или фиксированный (см. 1.1.3). Таким образом, следует придерживаться следующего правила.

Правило. Если в PL/1 выражение, которое должно быть целым, содержит операцию **, оба операнда которой не являются одновременно целыми числами, то результат такой операции должен быть округлен до ближайшего целого, например, по FLOOR ($eA + 0.5$) — см. 1.1.5.

Например:

$$3 \uparrow 5 \rightarrow 3**5$$

$$K \uparrow 3 \rightarrow \text{FLOOR}(K**3 + 0.5)/* \text{ДЛЯ ЦЕЛОГО } K *$$

$$PQ^R = P \uparrow (Q \uparrow R) \rightarrow P** (Q**R) = P** Q**R$$

Для особых значений основания и показателя (равных, например, нулю или отрицательных) вычисление степени производится так же, как и в алголе; в частности $x**(-n) \rightarrow 1/x**n$ для целого положительного числа n .

3. В то время, как в алголе операция деления для целых операндов дает результат типа **real**, в PL/1 получается результат с фиксированной точкой, а не с плавающей. Кроме того, как уже упоминалось выше (1.1.3), числа с точкой, но без порядка, имеют в PL/1 не плавающий тип, а фиксированный, диапазон представления значений которого значительно меньше. Поэтому, если в выражении встречаются числа с точкой, но без порядка, или операция деления над целыми или фиксированными операндами, то в PL/1 для предупреждения возможного переполнения мы будем ориентироваться на одно из следующих правил, которые основываются на материале 2-й главы (см. 2.2.1 и 4.2.4).

Правило 1 (осторожное). Если в программе на алголе результат некоторой конкретной операции при вычислении выражения имел бы тип **real**, то в PL/1 оба операнда этой операции

не должны быть одного из следующих видов: 1) иметь целый тип или 2) являться числом без порядка. В противном случае необходимо от одного из операндов взять функцию `FLOAT` или написать ему, если он число, нулевой порядок, то есть:

$x \otimes y \rightarrow \{x \otimes \text{FLOAT}(y) \mid \text{FLOAT}(x) \otimes y \mid x \otimes y \text{E}0 \mid x \text{E}0 \otimes y\}$
(здесь \otimes — любая арифметическая операция, кроме возведения в степень).

Обозначив через J и K операнды (в частном случае, переменные) целого или фиксированного типа, через R — вещественного (исключая числа без порядка), будем, например, иметь:

$R \otimes 3.2 \rightarrow R \otimes 3.2$ но $J \otimes 3.2 \rightarrow J \otimes 3.2 \text{E}0$

$6.1 \otimes R \rightarrow 6.1 \otimes R$ но $6.1 \otimes J \rightarrow 6.1 \text{E}0 \otimes J$

$J \otimes K \rightarrow J \otimes K$ но $J/K \rightarrow \text{FLOAT}(J)/K$

Правило 2 (слегка рискованное). Преобразование к плавающей точке для целых и фиксированных операндов в PL/1 необходимо: (1) если один из таких операндов является частным от деления целых или фиксированных операндов, причем хотя бы один из последних является целым или фиксированным числом, или (2) если в результате выполнения операций по правилам целочисленной арифметики, без отбрасывания младших цифр (даже нулей), составляющих дробную часть, может получиться результат с количеством цифр более 15. Отметим, что частное от деления целых фиксированных операндов всегда имеет 15 цифр; при этом, если операндами являются целые переменные или функции (но не выражения!), то целая часть частного будет содержать 5 знаков, а дробная — 10.

Допустимы, например, следующие замены (I, J, K — целые переменные, не превышающие 10^6):

$1/3 \rightarrow 1/3, J/K \rightarrow J/K, R + 1/3.2 \rightarrow R + 1/3.2,$

$1.2345 \times 6.7890 \times 2.3456 \rightarrow 1.2345 * 6.7890 * 2.3456$

$1.2345678 + 98765.0 \rightarrow 1.2345678 + 98765.0$

Но $25 + 1/3, J + 1/3.2, J + 5/K, 1.23450 * 6.78900 * 2.34560, 0.00123456789 + 12345.67$ могли бы дать в PL/1 переполнение (для последних двух примеров легко убедиться, что для получения точного результата вычисляемых выражений необходимо более 15 цифр). Для предупреждения переполнения следовало бы прибегнуть к первому правилу:

$25 + 1/3 \text{E}0, J + 1 \text{E}0/3.2, J + \text{FLOAT}(5/K),$

$1.23450 * 6.78900 \text{E}0 * 2.34560, 0.00123456789 + 12345.67 \text{E}0$

Примеры. Пусть в алголе выражение имеет вид:

$A[I] + 4/I - K + 8.4/3 \times A[I \div K]$

Если A типа `real`, то в соответствии с первым (осторожным) правилом следует в PL/1 написать

$A(I) + 4 \text{E}0/I - K + 8.4/3 \text{E}0 * A(\text{TRUNC}(I/K))$

По второму правилу приведение операндов к плавающей форме было бы необязательным (напомним, что операции одного ранга, кроме **, выполняются слева направо). Но если A имеет тип **integer**, то и по второму правилу необходимо было бы такое же преобразование.

1.2.2. Логические выражения.

Если здесь также ограничиться рассмотрением только простых логических (булевских) выражений алгола, не содержащих условий (условные выражения рассматриваются в 1.2.4), то с точностью до замены одних ограничителей на другие можно считать, что они совпадают с логическими выражениями PL/1.

Сравните, например:

$$\neg (A + B \geq -5 \vee Q \wedge \text{false})$$

c

$$\neg (A + B > -5! Q \&'0'B)$$

Однако, имеются некоторые особенности (см. ниже).

Формы.

$$eG \sim \Phi_G \{ \vee G | fG | \text{true} | \text{false} | \neg | \wedge | \vee | \supset | = | () \\ | eA | < | > | \leq | \geq | = | \neq \} \\ eG \sim \Phi_G \{ \vee G | fG | '1'B | '0'B | \neg \langle ? \rangle | \& | ! | \langle ? \rangle | = \langle ? \rangle | () \\ | eA | < | > | < = | > = | = | \neg = \}$$

Здесь Φ_G вместе с содержимым фигурных скобок (сравните с 1.2.1) обозначает форму для логического выражения алгола и PL/1. Как видим, имеется однозначное соответствие между компонентами логических выражений двух языков (не считая некоторых особенностей — см. ниже). Пробелы между знаками «составной» операции типа \neg = не допускаются, так же, как и в логических значениях PL/1 ('1'B и '0'B). Эталонное обозначение операции ИЛИ — $|$ (вместо !).

Перевод. Для перевода простых логических выражений, не содержащих условий, следует перевести переменные и указатели функций (п.п. 1.1.5 и 1.1.6), заменить **true** и **false** соответственно на '1'B и '0'B, а \leq , \geq , \neq , \wedge , \vee , \equiv на $< =$, $> =$, $\neg =$, $\&$, $!$, $=$, оставить неизменными остальные ограничители и поставить все на те же места, учитывая, кроме того, следующие особенности.

Особенности.

1. Отрицание в PL/1 является операцией того же старшинства, что и возведение в степень; это не соответствует ее старшинству в алголе и может потребовать расстановки дополнительных скобок. Например: $\neg z5 > 0 \rightarrow \neg (Z5 > 0)$

2. Операции \equiv нет в PL/1, но можно заменить выражение $x \equiv y$ на $x = y$ или на $(x) = (y)$ (с учетом старшинства соседних операций).

3. Операции \supset нет в PL/1, поэтому можно рекомендовать заменять выражение $x \supset y$ на $\neg x \vee y$, учитывая старшинство операций.

4. Приведем здесь для сравнения старшинство (ранги) арифметических и логических операций двух языков.

Алгол:	PL/1:
1. \uparrow	1. $^{**}, \neg$
2. $\times, /, \div$	2. $*, /$
3. $+, -$	3. $+, -$
4. $<, >, \leq, \geq, =, \neq$	4. $<, >, <=, >=, =, \neg =$
5. \neg	5. $\&$
6. \wedge	6. $!$
7. \vee	
8. \supset	
9. \equiv	

Примеры.

$\neg B1 \equiv K=5 \wedge \text{true} \supset B3 \vee F1(B2)$
 $\rightarrow \neg B1 = (\neg(K=5 \&'1'B) ! B3 ! F1(B2))$

1.2.3. Именующие выражения.

Практически можно считать, что простые именующие выражения алгола, не содержащие условий (условные выражения рассматриваются в 1.2.4), совпадают с «меточными выражениями» PL/1 (с точностью до вида индексных скобок в указателе переключателя). Сравните, например, именующие выражения в следующих операторах перехода:

go to M и GO TO M ;
 go to $S[N-1]$ и GO TO $S(N-1)$;

Формы.

$$eL \sim \{I \mid iML[eA] \mid (eL)\}$$

$$eL \sim \{I \mid iML(eA^+) \mid eL\}$$

Через iML обозначен идентификатор переключателя, который трактуется здесь как «меточный массив». О eA^+ см. в 1.1.5.

Перевод. Для перевода именующего выражения следует перевести идентификатор метки или переключателя (п. 1.1.2), перевести индексное выражение (п. 1.2.1) и заключить его вместо квадратных в круглые скобки (в случае, если выражение eA имеет тип *real*, его нужно заменить на $FLOOR(eA+0.5)$ —см. 1.1.5), опустить круглые скобки, заключающие именующее выражение (если они есть).

Примеры.

go to $PKL[\sin(n \times \pi/2) + n]$
 \rightarrow GO TO $PKL(FLOOR(SIN(N * PI/2) + N + 0.5))$

1.2.4. Условные выражения.

В PL/1 условные выражения отсутствуют, поэтому на них не следует рассчитывать при разработке алгоритмов программ. Здесь мы дадим некоторые рекомендации по замене условных выражений алгола на условные операторы и операторы присваивания алгола или PL/1 (последние будут рассматриваться в 1.3).

Условное арифметическое выражение. Условное выражение вида

if eG then eA_1 else eA_2

может быть заменено на условный оператор

if eG then $v := eA_1$ else $v := eA_2$,

который нужно поставить перед оператором, содержащим условное выражение. Переменная v должна быть дополнительно описана, и ею нужно заместить условное выражение в соответствующем месте. (Предложенный способ, однако, может вызвать серьезные трудности, если выражения eA_1 и eA_2 имеют разные типы или если придется учитывать влияние побочного эффекта, имеющего место и в PL/1).

Пример.

$RO := Y \uparrow (\text{if } X < 0 \text{ then } 1 \text{ else } 2)$

заменяется на

integer IIII;

.
if $X < 0$ then IIII:=1 else IIII:=2;

$RO := Y \uparrow IIII$

Здесь и ниже предполагается, что дополнительно описываемые идентификаторы (IIII и т. п.) не встречались ранее в соответствующем блоке программы.

Условное логическое выражение. Замена выражения вида

if eG then eG_1 else eG_2

производится аналогично замене арифметического выражения.

Пример.

$BOB := (\text{if if } A \text{ then } B \text{ else } C \text{ then } \ln(X) \text{ else } \ln(X+1)) \uparrow 2$

заменяется на

Boolean BBB4; real RRR2;

.
if A then BBB4:=B else BBB4:=C;

if BBB4 then RRR2:=ln(X)
else RRR2:=ln(X+1);

$BOB := RRR2 \uparrow 2$

Условное именуемое выражение. Для того чтобы провести замену условного именуемого выражения способом, аналогичным для арифметического и логического выражений, необходимо иметь меточные (именующие) переменные, которых в алголе

нет. Но в PL/1 такая возможность имеется, и поэтому допустима, например, следующая замена:

go to if $I < 0$ then M else $P[S+5]$

заменяется на

DECLARE VL # 1 LABEL;

IF $I < 0$ THEN VL # 1 = M; ELSE VL # 1 = P(S+5);

GO TO VL # 1;

Дадим общие формы для такой замены.

Ф о р м а для алгола:

<...> if eG then eL_1 else eL_2 <...>

Ф о р м а для PL/1:

DECLARE vL LABEL;

< >

IF eG THEN vL = eL_1 ; ELSE vL = eL_2 ;

<...> vL <...>

К использованным возможностям PL/1 мы еще вернемся во 2-й главе.

Упражнения.

Написать на PL/1 следующие выражения (арифметические переменные с идентификаторами, начинающимися с i, j, k, n или I, J, K, N , будем считать целыми, остальные вещественными — FLOAT).

$$1.2.1. \frac{R_1}{R_2} \frac{(\sqrt[3]{\beta} - e^{3 \cdot 10^{-3} \cdot x^{0,3}} + R_2 \cos(\sqrt[3]{\pi y})) x^{1/7}}{2(1-R)}$$

$$1.2.2. \frac{1/q - 3,3jk + 2/k}{2/i + 84i + 2/q} - \left(5,6^4 - \frac{7,3}{i+j+k} \right) \cdot q$$

$$1.2.3. A \vee a \equiv x + 0.1 < y \supset \text{false}$$

$$1.2.4. x \times y \neq 0 \wedge x \uparrow y \geq z \equiv x < 0 \supset \neg x < y \wedge b$$

1.3. Сравнение операторов

В алголе имеется 8 типов операторов, и в этом разделе будет проведено сравнение с операторами PL/1 для каждого из них. Сначала мы займемся немеченными операторами и в первую очередь основными.

Заметим, что в алголе точка с запятой не является частью оператора, а служит для отделения одного оператора от другого. В PL/1 точка с запятой входит в состав любого оператора и ограничивает его. Таким образом в PL/1:

$\hat{s} \sim \hat{s};$

где \hat{s} — оператор PL/1 без точки с запятой.

1.3.1. Пустые операторы.

Пустой оператор PL/1 не отличается от оператора алгола (если не считать присутствия точки с запятой в операторе PL/1).

Формы.

s-пустой $\sim \langle \text{пусто} \rangle$

s-пустой $\sim \langle \text{пусто} \rangle$;

Перевод. Для перевода пустого оператора следует к оператору алгола добавить ; (точку с запятой).

Пример пустого оператора можно найти во введении.

1.3.2. Операторы перехода.

Оператор перехода PL/1 практически не отличается от оператора алгола (если не считать замены **go to** на **GO TO**). Сравните: **go to M1** и **GO TO M1**;

Формы.

s-перехода $\sim \text{go to } eL$

s-перехода $\sim \left\{ \begin{array}{l} \text{GO TO} \\ \text{GOTO} \end{array} \right\} eL$;

Перевод. Для перевода оператора перехода нужно заменить **go to** на **GO TO** (или **GOTO**), перевести именующее выражение (п. 1.2.3) и в конце поставить точку с запятой.

Примеры операторов перехода PL/1 были даны в 1.2.3.

1.3.3. Операторы процедуры.

Если не обращать пока внимания на некоторые особенности в задании фактических параметров (п. 1.4.4), то можно считать, что оператор процедуры PL/1 отличается только тем, что в начало оператора добавляется слово **CALL** («вызов»). Сравните:

SUM (A, B, C) и **CALL SUM (A, B, C)**;

Формы.

s-процедуры $\sim \text{IP} [(a, ..)] \langle, \sim \{, | \} \alpha...: (\rangle$

s-процедуры $\sim \text{CALL IP} [(a, ..)]; \langle, \sim \{, |, /*\xi...*/ \rangle$

Перевод. Для перевода оператора процедуры нужно после ключевого слова **CALL** расположить в том же порядке, разделив теми же ограничителями, переведенный идентификатор процедуры (п. 1.1.2) и фактические параметры, переведенные по правилам разделов 1.1, 1.2 и 1.4.4, а в конце поставить точку с запятой. Ограничитель параметра вида

) строка-букв:(

можно, учитывая п.п. 1.1.8, заменить на

,/* символы-ЭВМ */

или на

/* символы-ЭВМ */,

Примеры.

SUM (A, B) RESULT:(C) \rightarrow

CALL SUM (A, B, /*РЕЗУЛЬТАТ:*/C);

1.3.4. Операторы присваивания.

Оператор присваивания PL/1 является обобщением оператора алгола, но пока мы будем считать, что он отличается только

заменой знаков присваивания на другие знаки. Сравните, например:

$P:=Q:=R:=S+1.5$ и $P, Q, R=S+1.5$;

Имеется, однако, ограничение (см. ниже).

Формы.

$$\begin{aligned} s\text{-присваивания} &\sim \left\{ \begin{array}{l} [\mathbf{v}A:=]\dots\mathbf{v}A:=\mathbf{e}A \\ [\mathbf{v}G:=]\dots\mathbf{v}G:=\mathbf{e}G \end{array} \right\} \\ s\text{-присваивания} &\sim \left\{ \begin{array}{l} [\mathbf{v}A,]\dots\mathbf{v}A=\mathbf{e}A+; \\ [\mathbf{v}G,]\dots\mathbf{v}G=\mathbf{e}G; \end{array} \right\} \end{aligned}$$

О $\mathbf{e}A+$ см. 1.1.5.

Перевод. Для перевода оператора присваивания следует перевести переменные левой части (п. 1.1.5), разделить их запятыми (вместо $:=$) и за последней переменной (вместо знака присваивания $:=$) поставить знак равенства $=$, затем перевести арифметическое или логическое выражение (п.п. 1.2.1 или 1.2.2) и добавить в конце точку с запятой.

Ограничение. В случае, если список левой части содержит переменные целого типа, а арифметическое выражение имеет тип **real**, то правую часть оператора присваивания для PL/1 нужно округлить до ближайшего целого по $\text{FLOOR}(\mathbf{e}A+0.5)$. При этом следует помнить, что, как мы приняли ранее для 1-й главы (см. 1.1.3), целые переменные могут меняться только в интервале от -10^5 до $+10^5$.

Примеры.

$I:=J:=K/L \rightarrow I, J=\text{FLOOR}(K/L+0.5)$;

$D2[I, J+1]:=\text{false} \vee P[I] \rightarrow D2(I, J+1)='0' \text{BIP}(I)$;

1.3.5. Условные операторы.

Можно считать, что условный оператор PL/1 совпадает с оператором алгола (с точностью до замены строчных букв в ограничителях алгола на заглавные и добавления точек с запятой); уточнение дается в 2.3.3. Сравните:

if $S=1$ **then go to** $L2$ и **IF** $S=1$ **THEN GO TO** $L2$;

или

if $X < Y$ **then** $M:=Y$ **else** $M:=X$

и

IF $X < Y$ **THEN** $M=Y$; **ELSE** $M=X$;

Формы.

s -условный \sim **if** $\mathbf{e}G$ **then** s'_1 [**else** s_2]

s -условный \sim **IF** $\mathbf{e}G$ **THEN** s_1 [**ELSE** s_2]

или s -условный \sim **IF** $\mathbf{e}G$ **THEN** \hat{s}_1 ; [**ELSE** \hat{s}_2];

Штрих отмечает тот факт, что в алголе после **then** не может стоять условный оператор (а иногда и оператор цикла); хотя в PL/1 такого ограничения нет, мы пока (до 2.3.3) будем его придерживаться. Как видим, перед **ELSE**, если оно есть, всегда должна стоять точка с запятой (принадлежащая оператору, относящемуся

к THEN). Точка с запятой в конце условного оператора фактически принадлежит не условному оператору, а оператору, входящему в условный; таким образом, оказывается, что условный оператор не имеет «своей» точки с запятой.

Перевод. Для перевода условного оператора нужно перевести логическое выражение (п. 1.2.2) и операторы, входящие в условный оператор (рассматриваемые в настоящем разделе), и расставить ключевые слова IF, THEN, ELSE на соответствующих местах.

Примеры.

if $\neg A$ then else if $F < G + 1$ then $R := 9$ else M
 \rightarrow IF $\neg A$ THEN; ELSE IF $F < G + 1$

THEN $R = 9$; ELSE CALL M ;

1.3.6. Операторы цикла.

В операторе цикла PL/1 отсутствует элемент списка цикла, совпадающий с элементом типа пересчета алгола (**while**); похожий на **while** элемент списка цикла и другие возможности обсуждаются в п. 2.3.4. В PL/1 в качестве тела цикла может фигурировать несколько операторов, что освобождает от необходимости использовать в этом случае составной оператор (п. 1.3.7). Оканчивается оператор цикла всегда на END;. В остальном можно считать, что оператор цикла PL/1 совпадает с оператором алгола (с точностью до замены ограничителей алгола на другие ключевые слова и знаки PL/1); ниже приведены некоторые ограничения. Сравните:

1) for $i := 1, 5, 8, 9$ step 2 until 25 do

begin $RO := RO + \sin(i \times C)$; $TAU := TAU + \cos(i \times D)$ end
 и DO $I = 1, 5, 8, 9$ BY 2 TO 25;

$RO = RO + \sin(I * C)$; $TAU = TAU + \cos(I * D)$; END;

2) for $k := 1$ step 1 until N do $A[k] := k + 1$

и DO $K = 1$ BY 1 TO N ; $A(K) = K + 1$; END;

Формы.

s -цикла \sim for $vA := \left\{ \begin{array}{l} eA_1 \text{ step } eA_2 \text{ until } eA_3 \\ eA_4 \text{ while } eG \\ eA_5 \end{array} \right\}, \dots \text{ do } s$

s -цикла \sim DO $vA = \left\{ \begin{array}{l} eA_1^+ \text{ BY } eA_2^+ \text{ TO } eA_3^+ \\ \langle ? \rangle \\ eA_5^+ \end{array} \right\}, \dots; s \dots \text{ END};$

О eA^+ см. в 1.1.5.

Перевод. Для перевода оператора цикла, не содержащего элемент списка цикла типа **while**, следует перевести параметр цикла (п. 1.1.5), арифметические выражения, входящие в элементы списка цикла (п. 1.2.1), и оператор, являющийся телом цикла (см. 1.3), заменить ограничители алгола **for**, **:=**, **step**, **until**, **do** соответственно на **DO**, **=**, **BY**, **TO**, точку с запятой (;) и оставить на месте запятые списка цикла; в конце оператора цикла поставить **END**;

Ограничения.

1. Следует осторожно использовать для параметра цикла, являющегося переменной типа целый, выражения вещественного типа в заголовке цикла, так как соответствующее преобразование из вещественного в целый в PL/1 отличается от алгольного (см. 1.3.4 и 1.1.5).

2. Следует иметь в виду, что когда $eA_2^+ = 0$ и $eA_1^+ > eA_3^+$, то в PL/1 цикл не будет выполняться ни разу, в то время как в алголе это задавало бы бесконечный цикл.

Примеры.

```
for i := 1 step 1 until N do
  for j := 1 step 1 until N do
    C[i, j] := A[i, j] + B[j]
  → DO I=1 BY 1 TO N;
    DO J=1 BY 1 TO N;
      C(I, J) = A(I, J) + B(J);
    END;
  END;
```

1.3.7. Составные операторы.

Составной оператор PL/1 фактически совпадает с соответствующим оператором алгола (с точностью до замены ограничителей). Сравните:

```
begin R := 4.5; go to MET3 end
и DO; R = 4.5; GO TO MET3; END;
```

Формы.

s-составной ~ begin s; .. end

s-составной ~ DO; s ... END;

или s-составной ~ DO; \hat{s} ; ..; END;

Перевод. Для перевода составного оператора следует перевести входящие в него операторы (раздел 1.3), располагая их в том же порядке (и опуская разделяющие их точки с запятой), а в начале и в конце всей группы операторов поставить соответственно DO; и END; (вместо begin и end).

Замечание. Это правило перевода (как и все остальные) не рассчитано на практическое использование для перевода программ с алгола на PL/1, а служит лишь для наиболее конкретного указания на совпадения и различия, имеющиеся в сравниваемых формах (об этом было сказано и во введении). В противном случае его следовало бы видоизменить, чтобы не приходилось сначала отбрасывать точки с запятой, а затем опять приписывать, обращаясь к правилам перевода операторов, входящих в составной.

Примеры.

```
1) begin t := q := 10 + 10; comment MNOGO; ZET(t) end
→ DO; T, Q = 1E + 10; /* МНОГО */ CALL ZET(T); END;
```

```

2) for  $i := 1$  step 1 until  $N$  do
    begin  $S := S + a[i]$ ; if  $a[i] < 0$  then  $p := i$  end
→ DO I=1 BY 1 TO N;
    DO; S=S+A(I); IF A(I) < 0 THEN P=I; END;
    END;
или → DO I=1 BY 1 TO N;
    S=S+A(I); IF A(I) < 0 THEN P=I;
    END;

```

1.3.8. Блоки.

Можно считать, что блок PL/1 совпадает с блоком алгола (разница лишь в «лишних» точках с запятой). Пример блока PL/1 можно найти во введении; там он помечен меткой BLOK. Уточнения блока даются в 1.4.4 и 2.4.2.

Формы.

```

s-блок ~ begin {d;} ... s; .. end
s-блок ~ BEGIN; d ... s ... END;
или s-блок ~ BEGIN;  $\hat{d}$ ; ..;  $\hat{s}$ ; ..; END;

```

Здесь используется тот факт, что все описания PL/1 оканчиваются на точку с запятой, то есть $d \sim \hat{d}$.

Перевод. Для перевода блока следует перевести составляющие его описания и операторы (разделы 1.4 и 1.3), располагая их в том же порядке (и опуская разделяющие их точки с запятой); в начале блока поставить BEGIN; , а в конце END; (см. п. 1.3.7).

1.3.9. Помеченные операторы.

Помеченные операторы PL/1 по своему формату не отличаются от алгольных. Сравните:

$M:N2:V1 := 5$ и $M:N2:V1 = 5$;

Формы. Дадим общие формы для операторов алгола и PL/1:

```

s ~ [l:] ... s
s ~ [l:] ... s ~ [l:] ...  $\hat{s}$ ;

```

Перевод. Для перевода оператора алгола следует перевести все его метки, если они есть, по правилу перевода п. 1.1.4, оставив на месте двоеточия (:), и добавить справа непомеченный оператор, переведенный по правилам пунктов 1.3.1 + 1.3.8.

Упражнения.

1.3.1. Написать на PL/1 следующий оператор алгола:

```

if  $X > Y \wedge X > Z$  then  $M := X$  else
    if  $Y > X \wedge Y > Z$  then  $M := Y$  else  $M := Z$ 

```

1.3.2. Написать на PL/1 следующий оператор алгола:

```

for  $j := 1$  step 1 until JK do
    begin  $P[j] := A[N]$ ;
        for  $i := 1$  step 1 until N do
             $P[j] := P[j] \times X[j] + A[N-i]$ 
        end
    end

```

1.3.3. Запрограммировать на PL/1 следующий алгоритм:

1°. Положить $y_n = \frac{1}{2} + \frac{1}{2} z$.

2°. Положить $y_{n+1} = \frac{1}{2} \left(y_n + \frac{z}{y_n} \right)$.

3°. Если $|y_{n+1} - y_n| \geq \varepsilon$ то к 4°, иначе к 6°.

4°. Положить $y_n = y_{n+1}$.

5°. Перейти к 2°.

6°. Конец вычислений (результат: y_{n+1}).

1.3.4. Запрограммировать на PL/1 следующий алгоритм:

1°. Для $i = 1, 2, \dots, N$ выполнить 2° ÷ 7°.

2°. Положить $u = x_i$, $n = 0$, $fun_i = x_i$.

3°. Положить $n = n + 1$.

4°. Если $|u| \leq \delta$, то к 7°, иначе к 5°.

5°. Положить $u = -\frac{ux_i^2}{2n(2n+1)}$.

6°. Положить $fun_i = fun_i + u$; перейти к 3°.

7°. Увеличить i ; при $i > N$, перейти к 8°.

8°. Конец вычислений (результат: fun).

1.3.5. Запрограммировать на PL/1 вычисление факториала:

$$n! = 1 \times 2 \times \dots \times n$$

1.3.6. Запрограммировать на PL/1 умножение матрицы на вектор:

$$c_i = \sum_{j=1}^N a_{ij} \times b_j, \quad i = 1, 2, \dots, N.$$

1.4. Сравнение описаний

В алголе имеются описания четырех видов: типа, массива, переключателя, процедуры. В этом разделе будет дано их сравнение с описаниями PL/1.

1.4.1. Описания типа.

Описание типа для простых переменных в PL/1 аналогично соответствующему описанию в алголе; оно фактически отличается только порядком следования описателей и списка идентификаторов, а также добавлением скобок и ключевого слова DECLARE. Сравните, например:

integer I1, K2, HOMEP

и **DECLARE (I1, K2, HOMEP) FIXED;**

real RO, LAMBDA и **DECLARE (RO, LAMBDA) FLOAT;**

Boolean PRIZNAK и **DECLARE (PRIZNAK) BIT (1);**

Смотрите, однако, ниже замечание.

Формы.

$$dV \sim [\text{own}] \left\{ \begin{array}{l} \text{real} \\ \text{integer} \\ \text{Boolean} \end{array} \right\} iV, ..$$

$$dV \sim \left\{ \begin{array}{l} \text{DECLARE} \\ \text{DCL} \end{array} \right\} (iV, ..) [\text{STATIC}] \left\{ \begin{array}{l} \text{FLOAT} \\ \text{FIXED} \\ \text{BIT (1)} \end{array} \right\};$$

Перевод. Для перевода описания типа нужно за ключевым словом DECLARE (или DCL) разместить в скобках список переведенных идентификаторов простых переменных (п. 1.1.2), а затем поставить ключевые слова PL/1, соответствующие описателям алгола: **own**, **real**, **integer**, **Boolean** соответствуют **STATIC**, **FLOAT**, **FIXED**, **BIT (1)**; в конце ставится точка с запятой (;).

З а м е ч а н и я.

1. Предельными значениями для переменных в PL/1 ранее (см. 1.1.3) мы договорились считать $\pm 10^5$ (**FIXED**) и $\pm 10^{75}$ (**FLOAT**); точность представления для **FLOAT** — 6 десятичных цифр; уточнения будут даны в 2.1.1.

2. Если список идентификаторов состоит из одного идентификатора, то скобки ставить не обязательно.

3. Описания различных типов могут быть объединены через запятую за единственным описателем DECLARE.

Примеры.

DECLARE (I1, K2) FIXED, PRIZNAK BIT (1),
(RO, LAMBDA) FLOAT, PEM FIXED;
own real tau → DECLARE TAU STATIC FLOAT;

Обозначения. В дальнейшем описатели типа будем обозначать через τ и τ , то есть:

$$\tau \sim \left\{ \begin{array}{l} \text{real} \\ \text{integer} \\ \text{Boolean} \end{array} \right\} \quad \text{и} \quad \tau \sim \left\{ \begin{array}{l} \text{FLOAT} \\ \text{FIXED} \\ \text{BIT (1)} \end{array} \right\}$$

• Тогда

$$dV \sim [\text{own}] \tau iV, ..$$

$$dV \sim \text{DECLARE} (iV, ..) [\text{STATIC}] \tau;$$

1.4.2. Описания массивов.

Описание массивов аналогично соответствующему описанию в алголе, только в качестве описателя массива используются круглые скобки, заключающие список граничных пар и следующие сразу за списком идентификаторов. Кроме того, добавляются скобки, заключающие списки идентификаторов и общий список всех сегментов описания (если списки содержат более одного члена). Сравните:

Boolean array Q [1:M, 1:N]

и **DECLARE Q(1:M, 1:N) BIT(1);**

array A [1:50], B [-4:+4]

и DECLARE (A (1:50), B (-4:+4)) FLOAT;

integer array C, D, E [-100:0], R1, R2 [0:5, 0:5]

и DECLARE ((C, D, E) (-100:0), (R1, R2) (0:5, 0:5)) FIXED;

Формы.

$dM \sim [own \tau | \tau] \text{ array } \{iM, .. [\{eA_1:eA_2\}, ..], ..$

$dM \sim \text{DECLARE} (\{iM, ..\} (\{eA_1^+:eA_2^+\}, ..), ..) [\text{STATIC}] \tau;$

В форме алгола внутренние фигурные скобки выделяют граничные пары, подчеркнутые квадратные скобки являются индексными скобками, внешние фигурные скобки заключают сегмент массивов. В случае, когда τ означает *real*, оно может не указываться. О eA^+ см. в 1.1.5.

Перевод. Для перевода описания массивов нужно перевести идентификаторы массивов и арифметические выражения для граничных пар (п. 1.1.2 и 1.2.1), расположив их в том же порядке и разделяя теми же ограничителями, но заменяя квадратные скобки на круглые. Кроме того, списки идентификаторов с общими границами следует заключить в дополнительные круглые скобки, так же как и общий список всех идентификаторов вместе с их граничными парами. Слева от этих последних скобок нужно написать *DECLARE*, а справа — ключевые слова, соответствующие описателям типа и описателю *own*; описатель *FLOAT* здесь опускать нельзя (что разрешается для *real*).

Особенности.

1. Если среди граничных пар в алголе встречаются выражения типа *real*, то они должны быть округлены до целого значения по $FLOOR(eA + 0.5)$ — см. 1.1.5.

2. Для «собственных» массивов PL/1 (STATIC) границы могут быть только целыми десятичными числами.

З а м е ч а н и я.

1. Если упомянутые списки содержат лишь по одному члену, то скобки можно не ставить.

2. Если нижняя граница равна единице, то ее можно не писать, то есть $W [1:K] \rightarrow W (K)$.

3. Описание массивов может быть объединено через запятую с другими описаниями (за общим *DECLARE*).

Примеры.

DECLARE A (15) FLOAT, D FIXED, B BIT (1);

array F [1:1+K↑4] → DECLARE

F (FLOOR (1.5+K**4)) FLOAT;

1.4.3. Описания переключателей.

В случае, когда переключательный список содержит только метки, между описаниями переключателя алгола и PL/1 имеется достаточно прямая аналогия. Сравните:

switch $P := L1, L3, M$

и **DECLARE** $P(3)$ **LABEL INITIAL** ($L1, L3, M$);

Цифра 3 в скобках указывает на количество элементов в переключательном списке.

Формы.

$dML \sim \text{switch } iML := l, .. \langle m \text{ элементов} \rangle$

$dML \sim \text{DECLARE } iML(m) \text{ LABEL INITIAL } \langle l, .. \rangle;$

Перевод. Для того чтобы перевести описание переключателя, следует перевести идентификатор переключателя (п. 1.1.2) и в скобках за ним указать количество меток в переключательном списке. Затем перевести метки (п. 1.1.4), входящие в переключательный список, заключив его в скобки, и расставить все в указанном в форме PL/1 порядке, добавив указанные ключевые слова.

В случае, если в переключательный список для алгола входят указатель переключателя или условное именуемое выражение, то перевод такого описания весьма громоздок, но возможен. Ниже приведены соответствующие формы.

Формы.

$dML \sim \text{switch } iML_0 := \{l \mid iML [eA]$

$\mid \text{if } eG \text{ then } eL_1 \text{ else } eL_2\}, .. \langle m \text{ элементов} \rangle$

$dML \sim \text{DECLARE } iML_0(m) \text{ LABEL INITIAL } (\{l \mid * \mid *\}, ..);$

$\{\langle \text{пусто} \rangle \mid iML_0(k) = iML(eA^+);$

$\mid \text{IF } eG \text{ THEN } iML_0(k) = eL_1; \text{ ELSE } iML_0(k) = eL_2; \}...$

Здесь k — текущий номер в списке элемента, не являющегося меткой; в списке PL/1 ему соответствует *.

Нижние строчки формы для PL/1 задают дополнительные операторы, которые должны быть добавлены соответственно для каждого элемента списка, если он не является меткой; количество таких операторов равно количеству звездочек в списке для PL/1. Примененные здесь конструкции PL/1 объясняются в 2.1.4.

З а м е ч а н и е. Если арифметические выражения в индексных скобках переключателя имеют тип *real*, они должны быть округлены до целых значений по $\text{FLOOR}(eA + 0.5)$ — см. 1.1.5.

Примеры.

switch $Q := L0, \text{ if } X > 1 \text{ then } L2 \text{ else } P[n], L1, P[V - 2.1]$

$\rightarrow \text{DECLARE } Q(4) \text{ LABEL INITIAL } (L0, *, L1, *);$

IF $X > 1$ **THEN** $Q(2) = L2$; **ELSE** $Q(2) = P(N)$;

$Q(4) = P(\text{FLOOR}(V - 1.6));$

1.4.4. Описания процедур.

В PL/1 описание процедур состоит из трех основных частей, которые имеются и в алголе; заголовок процедуры, спецификации формальных параметров и тело процедуры. Список значений в PL/1 отсутствует: вызов по значению или наименованию осуществляется

в зависимости от конкретного вида фактического параметра при обращении к процедуре.

В PL/1, как и в алголе, не требуется полная совокупность спецификаций, но, чтобы пока не углубляться в имеющиеся при этом различия, мы будем считать, что все параметры в PL/1 должны быть специфицированы (уточнения см. в 2.4).

Как видно из следующих примеров, описания процедуры в алголе и PL/1 имеют много общего. Сравните:

```
1) procedure MAXP (x, y, RES); integer x, y, RES;
    if x > y then RES := x else RES := y
MAXP:PROCEDURE (X, Y, RES);
    DECLARE (X, Y, RES) FIXED;
    IF X > Y THEN RES = X; ELSE RES = Y;
END;
/* RES = МАКСИМУМ ИЗ X И Y */
```

Процедура PL/1 всегда оканчивается на END;

```
2) real procedure MAXF (x, y, z); real x, y, z;
    begin real R;
        if x > y then R := x else R := y;
        if z < R then MAXF := R else MAXF := z
    end
MAXF:PROCEDURE (X, Y, Z) RETURNS (FLOAT);
    DECLARE (X, Y, Z) FLOAT, R FLOAT;
    IF X > Y THEN R = X; ELSE R = Y;
    IF Z < R THEN RETURN (R);
    ELSE RETURN (Z);
END; /* MAXF = МАКСИМУМ ИЗ X, Y и Z */
```

Как видим, в случае если телом процедуры является блок, ключевое слово BEGIN; ставить не нужно, а описания переменных, локализованных в этом блоке, можно объединить со спецификациями (можно было бы написать и DECLARE (X, Y, Z, R) FLOAT). Операторы RETURN (R); и RETURN (Z); соответствуют операторам присваивания в алголе, в левой части которых находится идентификатор процедуры-функции.

```
3) procedure MAXM (A) VEKTOР:(C) ORDER:(N);
    integer array A, C; integer N;
    begin integer i, j;
        for i := 1 step 1 until N do
            begin C[i] := A[i, 1];
                for j := 1 step 1 until N-1 do
                    if A[i, j+1] > C[i]
                        then C[i] := A[i, j+1]
                end
            end
        end
    end
```

end

```

MAXM:PROCEDURE (A, /* ВЕКТОР */ C, /* ПОРЯДОК */ N);
  DECLARE A(*, *) FIXED, C(*) FIXED;
  DECLARE (N, I, J) FIXED;
  DO I=1 BY 1 TO N;
    DO; C(I)=A(I, 1);
      DO J=1 BY 1 TO N-1;
        IF A(I, J+1) > C(I)
          THEN C(I)=A(I, J+1);
      END;
    END;
  END;
END;

```

/* C(I)=МАКСИМУМ ПО J ДЛЯ A(I, J), $1 \leq I, J \leq N$ */

В PL/I с помощью звездочек нужно указывать размерность специфицируемых массивов *).

Теперь, после сравнения примеров простейших процедур, перейдем к рассмотрению общих синтаксических форм и имеющихся ограничений.

Формы.

$dP \sim [\tau] \text{ procedure } iP [(p, ..)];$

[value $p, ..$]; [\tilde{d} ;] ... s

$dP \sim iP:PROCEDURE [(p, ..)] [RECURSIVE] [RETURNS(\tau)];$

[<?>] [\tilde{d}] ... s ... END;

$\tilde{d} \sim t \dots p, ..$

$\tilde{d} \sim \text{DECLARE } \{(p, ..) t \dots | \{pt \dots\}, ..\};$

$t \dots \sim \left\{ \begin{array}{l} \text{real} \\ \text{integer} \\ \text{Boolean} \\ [\tau] \text{ array} \\ \text{label} \\ \text{string} \\ \text{switch} \\ [\tau] \text{ procedure} \end{array} \right\}$	$t \dots \sim \left\{ \begin{array}{l} \text{FLOAT} \\ \text{FIXED} \\ \text{BIT (4)} \\ (*, ..) \tau \\ \text{LABEL} \\ \text{CHARACTER (*)} \\ (*) \text{ LABEL} \\ \text{ENTRY } [(\{t^1 \dots\}, ..)] \\ \text{[RETURNS}(\tau^1)] \end{array} \right\}$
--	---

Через \tilde{d} и \tilde{d} мы обозначили спецификации параметров (которые можно считать «неполными описаниями»). Через $t \dots$ обозначены сочетания описателей (или описатели) алгола, встречающиеся в спецификациях, а через $t \dots$ соответствующие им сочетания

*) Трудно удержаться от того, чтобы не отметить, что вместо четырех END;, оканчивающих процедуру, в данном случае достаточно написать лишь END MAXM; (см. 2.3.5).

описателей PL/1, которые в дальнейшем мы будем называть *атрибутами*.

Количество звездочек в спецификации массива для PL/1 указывает на его размерность. Список сочетаний атрибутов ($t^1...$), стоящий в скобках после ENTRY, задает атрибуты значений для соответствующих формальных параметров процедуры, которая специфицируется; RETURNS (τ^1) задается в случае, если специфицируется процедура-функция (единичка введена для отличия этих атрибутов от атрибутов основной процедуры — $t...$ и τ).

Второй вариант формы \tilde{d} для PL/1 применяется при не вполне совпадающих атрибутах формальных параметров (различные размерности у специфицируемых массивов, разное количество параметров у специфицируемых процедур и т. п.). В этом случае для каждого параметра выписываются его атрибуты, и все это отделяется запятой от спецификации другого параметра.

Перевод. Для перевода описания процедуры следует перевести входящие в нее идентификаторы процедуры и формальных параметров (п. 1.1.2); перевести оператор, составляющий тело процедуры (п. 1.3), заменить спецификаторы алгола на соответствующие им в PL/1 (см. формы), а также добавить или переставить местами некоторые компоненты описания в соответствии с формами (или примерами). Если процедура рекурсивная, то в заголовке добавляется RECURSIVE. Если переводится процедура-функция, то в заголовке описатель τ заменяется на RETURNS (τ) и ставится в конец заголовка. Оператор присваивания, в левой части которого стоит идентификатор функции, вида

$$iP := \{eA \mid eG\}$$

заменяется на оператор

$$\text{RETURN}(eA^+ \mid eG);$$

О eA^+ см. 1.3.4 и 1.1.5.

Ограничение. Предполагается, что в процедуре алгола оператор присваивания значения функции является последним выполняемым оператором: оператор RETURN в PL/1 прекращает выполнение процедуры. Например, замена тела процедуры-функции MAXF1 вида

$$\text{MAXF1} := y; \text{ if } x > y \text{ then MAXF1} := x$$

на

$$\text{RETURN}(Y); \text{ IF } X > Y \text{ THEN RETURN}(X);$$

была бы неправильной и потребовалось бы изменение реализуемого алгоритма.

Особенности обращения к процедурам. В том же блоке, где производится описание процедуры, должно быть добавлено

«описание входа» следующего вида:

DECLARE iP ENTRY [({t...},...)] [RETURNS (τ)];

Здесь RETURNS (τ) необходимо только для процедуры-функции. Количество t... в списке после ENTRY должно быть равно количеству параметров процедуры iP, а сами атрибуты t... должны в точности совпадать с атрибутами соответствующих формальных параметров. С алгольной точки зрения такое описание дублирует как описание имени процедуры, так и спецификацию формальных параметров. О случаях, когда такое описание является обязательным, будет сказано в 2.4.4.

Если фактический параметр в операторе процедуры (CALL) или в указателе функции (см. 1.3.3 и 1.1.6) является идентификатором, то для вызова его по значению он должен быть заключен в скобки (см. ниже пояснения к MAXP).

Для того чтобы фактический параметр вызывался по наименованию, он должен быть идентификатором или переменной (может быть, и с индексами), и его атрибуты должны совпадать с атрибутами соответствующего формального параметра в спецификации (или в ENTRY, что то же самое). Это является ограничением по сравнению с алголом.

З а м е ч а н и е. В PL/1, таким образом, фактическим параметром может быть идентификатор, заключенный в скобки. Напомним, что фактическими параметрами в алголе и PL/1 могут быть, кроме того, выражения, идентификаторы и, в некоторых случаях, строки.

Формы.

$$a \sim \{e \mid g \mid l\}$$
$$a \sim \{e \mid g \mid i \mid (i)\}$$

Ограничения.

1. Идентификаторы стандартных функций ABS, FLOOR, SIGN не могут являться аргументами при обращении к процедурам.

2. В случае, когда для формального параметра типа целый задается аргумент типа вещественный, нужно позаботиться о его преобразовании по FLOOR (eA + 0.5) — см. 1.1.5.

Примеры. Дадим примеры описаний ENTRY и обращений к приведенным выше описаниям процедур:

1) DECLARE MAXP ENTRY (FIXED, FIXED, FIXED);

/* DECLARE R1 FIXED, Y1 FLOAT; */

CALL MAXP (25, Y1, R1);

Аргументы 25 и Y1 передаются по значению, так как их атрибуты не совпадают с атрибутами параметров; R1 передается по наименованию. Если бы R1 было описано как FLOAT, то оно вызывалось бы по значению и не изменилось бы после выполнения процедуры.

Если бы тело процедуры MAXP имело вид:

```
IF X > Y THEN Y = X; RES = Y;
```

то, чтобы процедура не меняла значения второго параметра, и в случае, когда он описан с FIXED, необходимо было бы задать

```
CALL MAXP (25, (Y1), R1);
```

```
2) DECLARE MAXF ENTRY (FLOAT, FLOAT, FLOAT)  
    RETURNS (FLOAT);
```

```
/* DECLARE (Y2, Z2) FIXED, X2 FLOAT; */
```

```
X2 = MAXF (3E-5, Y2, Z2 + MAXF (X2, Y2, Z2));
```

Только аргумент X2 передается по наименованию, остальные — по значению.

```
3) DECLARE MAXM ENTRY ((*, *) FIXED,  
    (* ) FIXED, FIXED);
```

```
/* DCL A1 (1:20, 1:20) FIXED, C1 (1:20) FIXED,  
    N1 FIXED; */
```

```
CALL MAXM (A1, C1, N1);
```

Все аргументы передаются по наименованию. Если бы C1 было описано как FLOAT, то результат обращения к процедуре был бы неправильный (C1 передавалось бы по значению и не менялось бы).

Пример. В заключение приведем пример, когда формальным параметром является идентификатор процедуры (функции).

```
/* ВЫЧИСЛЕНИЕ ИНТЕГРАЛА ПО ФОРМУЛЕ  
    ТРАПЕЦИИ */
```

```
INTEGRAL: PROCEDURE (A, B, N, F, INT);
```

```
    DECLARE (A, B, INT) FLOAT, N FIXED,
```

```
    F ENTRY (FLOAT) RETURNS (FLOAT);
```

```
    DECLARE (S, X) FLOAT, I FIXED
```

```
    H = (B - A) / (N - 1); S = 0.5 * (F (A) + F (B)); X = A;
```

```
    DO I = 1 BY 1 TO N - 2; X = X + H;
```

```
    S = S + F (X); END;
```

```
    INT = H * S;
```

```
END;
```

```
G: PROCEDURE (X) RETURNS (FLOAT); DCL X FLOAT;
```

```
    RETURN ((X + 5) ** 3 - SQRT (X / (X + 2))); END;
```

```
DECLARE G ENTRY (FLOAT) RETURNS (FLOAT),
```

```
    INTEGRAL ENTRY (FLOAT, FLOAT, FIXED,
```

```
    ENTRY (FLOAT) RETURNS (FLOAT), FLOAT);
```

```
DECLARE (P, RESULT) FLOAT;
```

```
    . . . . .  
    CALL INTEGRAL (0, P, 20, G, RESULT);
```

```
/* ИНТЕГРАЛ ОТ ФУНКЦИИ G НА ОТРЕЗКЕ
```

```
    [0, P] ПО 21 ТОЧКЕ */
```

Упражнения.

1.4.1. Написать на PL/1 описание следующей процедуры:

```
procedure CAB (A, B, C, N);  
  value N; integer N; array A, B, C;  
  begin integer I, J;  
    for I := 1 step 1 until N do  
      for J := 1 step 1 until N do  
        C[I, J] := A[I, J] × B[J]  
      end  
    end
```

1.4.2. Написать на PL/1 следующее описание:

```
real procedure FACT (N);  
  value N; integer N;  
  if N = 0 then FACT := 1  
  else FACT := N × FACT (N - 1)
```

1.4.3. Оформить решение упражнения 1.3.2 в виде описания процедуры и обратиться к ней.

1.4.4. Оформить решение упражнения 1.3.3 в виде описания процедуры-функции и обратиться к ней.

1.4.5. Написать на PL/1 процедуру упорядочивания по возрастанию массива из M целых чисел.

1.4.6. Написать на PL/1 процедуру-функцию, принимающую логическое значение соответствующее **true**, если два и только два арифметических значения из трех заданных равны между собой, и **false** — в противном случае.

1.5. Сравнение программы

1.5.1. Сравнение программы.

Программа в алголе, как известно, — это блок или составной оператор. Программа в PL/1 — это специального вида процедура. Примеры соответствующих друг другу программ на алголе и PL/1 были приведены во введении.

Формы.

программа-алгола \sim **begin** [d;]... s;.. **end**

программа-PL/1 \sim **īP**: PROCEDURE OPTIONS (MAIN);
[d]... s... **END**;

Перевод. Для перевода программы алгола нужно перевести по правилам разделов 1.4 и 1.3 составляющие ее описания и операторы (опуская разделяющие их точки с запятой) и вместо первого **begin** поставить указанную в форме конструкцию, где **īP** — идентификатор, присвоенный программе. Тильда отмечает тот факт, что идентификатор является «внешним» и не может содержать более 7 символов.

Примеры.

```
MAX: PROCEDURE OPTIONS (MAIN);  
  DECLARE (X, Y, Z) FLOAT;  
  GET LIST (X, Y, Z);  
  IF X > Y & X > Z THEN PUT LIST (X); ELSE  
    IF Y > X & Y > Z THEN PUT LIST (Y);  
      ELSE PUT LIST (Z);  
END;
```

Эта программа вводит (GET) три числа, находит максимальное из них и печатает (PUT) его.

Операторы ввода и вывода описываются в пункте 1.5.2.

1.5.2. Простейший ввод-вывод.

Ввод с перфокарт осуществляется оператором

```
GET LIST ({v | iM},..);
```

где *v* — переменная (простая или с индексами). Если задается идентификатор массива (*iM*), то вводится весь массив.

Вводимые значения перфорируются на перфокартах и имеют вид чисел или логических значений, определенных в 1.1.3 и 1.2.2. Они разделяются между собой одним или несколькими пробелами и для массивов задаются по строкам. Например, для программы приведенной в п. 1.5.1 можно было бы для *X*, *Y* и *Z* задать:

```
123.89E — 4 1 1 1 45.6 7
```

Вывод на печать осуществляется оператором

```
PUT LIST ({eA | eG | g | iM},...) [SKIP];
```

где *eA* и *eG* — выражения, которые в частном случае могут быть числами или переменными, а *g* — строка (см. 1.1.7), печатаемая без окаймливающих апострофов, с заменой двойных внутренних апострофов на одинарные. Если задан идентификатор массива (*iM*), то выдается весь массив. Ключевое слово *SKIP* задает печать с новой строки АЦПУ; в противном случае выводимые значения будут печататься сразу за предыдущими. Печать арифметических и логических значений производится в 5 столбцов; строки PL/1 в зависимости от их длины могут занимать несколько столбцов.

Строки используются при выводе для печати заголовков или поясняющего текста. Например, в п. 1.5.1 после оператора ввода можно добавить оператор

```
PUT LIST ('МАКСИМУМ ИЗ ТРЕХ ЧИСЕЛ',
```

```
  X, Y, Z) SKIP;
```

который с новой строки в первых 4 столбцах напечатает текст и значения *X*, *Y*, *Z*; максимум будет напечатан в 5-м столбце.

Примеры ввода и вывода можно найти и во введении (п. 0.1). Подробнее операторы ввода-вывода излагаются в 3.3 (в частности, см. 3.3.2).

1.5.3. Выход на машину.

Для выхода на машину нужно составить *задание*. Задание для трансляции программы с PL/1 и ее выполнения на машине имеет следующий вид:

```
//имя-задания JOB
//          EXEC PL1LFCG
//PL1L.SYSIN DD *
      текст-исходной-программы

/*
//GO.SYSIN  DD *
      исходные-данные-для-GET

/*
//
```

Имя (идентификатор) задания имеет не более 8 символов.

Для перфорации задание записывается, обычно на 72-позиционном бланке. Сочетания // или /* для управляющих карт должны находиться в первых двух позициях. Имя задания и стандартные имена для DD должны начинаться с 3-й позиции и не должны содержать пробелов. Пробелы обязательно должны окаймлять JOB, EXEC и DD.

Исходная программа записывается на бланках, начиная с любой позиции, кроме первой. Исходные данные могут начинаться с любой позиции.

Пожелание. Теперь Вы можете не только писать программы на PL/1, но и составлять задания для выполнения их на машине. Желательно перед переходом ко второй части написать и, по возможности, выполнить на машине несколько учебных программ.

Упражнения.

1.5.1. Решение упражнения 1.3.4 оформить в программу PL/1.

1.5.2. Решение упражнения 1.3.6 оформить в программу PL/1.

ГЛАВА 2

УТОЧНЕНИЕ ПОНЯТИЙ PL/1

Вторая глава курса посвящена уточнению некоторых понятий PL/1, введенных в 1-й главе. Основной порядок изложения тот же, что и в 1-й главе: основные элементы, выражения, операторы, описания.

При изложении синтаксиса уточняемых и новых понятий PL/1 во 2-й и 3-й главах, наряду со словесным описанием по-прежнему будут использоваться формы, введенные в 1-й главе для сравнения понятий алгола и PL/1. Приводимые формы не следует рассматривать как эталонные, поскольку, во-первых, большинство форм восстановлено по словесному описанию, изложенному в документации [6, 7, 8, 9], и, во-вторых, в курсе ради простоты и наглядности форм, иногда даются упрощенные формы, не отражающие всех вариантов рассматриваемых конструкций или имеющих ограничения (в большинстве случаев для отражения некоторых особенностей используется тильда или черта — см., например, выше \tilde{n} , \tilde{d}). Цифры 0, 1, 2, 3 в индексах (верхних или нижних), используются в формах для нумерации нескольких одинаковых понятий встречающихся в одной и той же форме или на разных уровнях ее детализации (см. выше eA_1 и eA_2 , t^1). Таблица форм, приведенная в 4.4.3, позволяет обозреть все понятия и конструкции PL/1, изложенные в настоящем курсе.

Терминология в этой главе для новых понятий, в основном, ориентирована на принятую в документации ЕС ЭВМ.

2.1. Уточнение основных элементов

Как известно, в алголе-60 величины могут принимать значения трех типов: целый, вещественный, булевский, которым соответствуют описатели **integer**, **real**, **Boolean**. В PL/1 подобных, описателей (которые мы еще в 1-й части договорились называть

атрибутами), характеризующих свойства значений величин, значительно больше, и, кроме того, они могут сочетаться друг с другом, что отражает большое разнообразие видов данных в PL/1. Напомним, что атрибуты используются при описании переменных, функций и других величин PL/1.

2.1.1. Арифметические данные.

Атрибуты масштаба. Вещественным величинам алгола, имеющим тип **real**, при сравнении мы ставили в соответствие величины PL/1, описываемые атрибутом **FLOAT** (плавающий). Этот атрибут характеризует значения, которые представляются в форме с плавающей точкой (в виде мантиссы и порядка), и точность их представления (как и в алголе) ограничена. Абсолютный диапазон изменения таких величин в ЕС ЭВМ — примерно от 10^{-75} до 10^{+75} .

Числа, соответствующие **FLOAT**, должны иметь в своем изображении порядок (с предшествующей буквой **E**) — см. 1.1.3; наличие точки в изображении числа не определяет атрибут **FLOAT**.

Величинам алгола типа **integer** мы ставили в соответствие величины PL/1, описываемые атрибутами **FIXED** (фиксированный). Этот атрибут характеризует значения, которые представляются в форме с фиксированной точкой, и их представление (в отличие от **FLOAT**) является точным. Диапазон изменения таких величин зависит от других дополнительных атрибутов, но он значительно меньше, чем у **FLOAT**. В отличие от описателя **integer**, атрибут **FIXED** может характеризовать в PL/1 не только целые значения, но и значения с дробной частью, для чего он должен быть дополнен атрибутом точности (см. ниже).

Числа с фиксированной точкой не должны иметь в своем изображении десятичного порядка, но могут иметь точку.

Примеры чисел:

FIXED	FLOAT
—1000	—1E3
7.785	778.5E—2
—4.	—4.E0

В дальнейшем атрибут **FIXED** обозначается через **F**, а **FLOAT** — через **E** (**Exponential** — экспоненциальный, показательный). Эти атрибуты называются *атрибутами масштаба*. Обозначение **I**, по-прежнему будет использоваться в качестве характеристики целых значений.

Атрибуты основания. В PL/1 можно указать основание счисления, в котором должны представляться значения величин. Десятичному основанию соответствует атрибут **DECIMAL** (сокращенно **DEC**), двоичному — **BINARY** (сокращенно **BIN**); в дальнейшем эти атрибуты — *атрибуты основания* — обозначаются соответственно через **D** и **B**. Все величины, использованные в приме-

рах 1-й части, задавались с десятичным основанием. Атрибут BINARY применяется обычно вместе с FIXED в случаях, когда необходимо производить вычисления с данными, имеющими точное двоичное (но не десятичное) представление. Кроме того, можно рекомендовать для эффективности работы программы использовать BINARY FIXED для переменных, которые являются индексами.

При записи констант с двоичным основанием число или его мантисса изображаются в двоичном виде, а двоичный порядок (если он есть) задается целым десятичным числом; в конце числа ставится буква В (без пробела).

Примеры двоичных чисел:

FIXED	FLOAT
-110111B	101E-4B
0.1010B	-1.1E68B
+1.B	.10E+128B

Формы для двоичных чисел (ср. с 1.1.3):

$nB \sim [\pm](\bar{n}B)[.](\bar{n}B)[E[\pm]\bar{n}ID]B$

$\langle \bar{n}B \sim \{0 | 1\}... \rangle$

$\langle \bar{n}ID \sim v... \rangle$

В порядке не может быть более трех цифр, а диапазон его изменения — примерно ± 250 . В двоичном числе с фиксированной точкой не может быть более 31 цифры.

Атрибуты моды. Все величины PL/1, с которыми мы имели дело до сих пор, имеют атрибут REAL, характеризующий действительные значения, противопоставляемые комплексным *), которые имеют атрибут COMPLEX (сокращенно CPLX). Ниже эти атрибуты, называемые *атрибутами моды*, обозначаются соответственно R и C.

Комплексное значение состоит, как обычно, из действительной и мнимой частей. Мнимое число, которое служит для изображения мнимой части комплексного значения, представляет собой любое действительное (REAL) число, непосредственно (без пробела) сопровождаемое буквой I. Комплексное число состоит из действительного и мнимого чисел, соединенных знаком + или —, который может быть окаймлен пробелами **). Атрибуты частей комплексного числа могут различаться, в то время как атрибуты частей комплексного значения имеют одинаковые атрибуты.

Примеры комплексных чисел:

-25.25+96.50I, -635-12I, 18.5+.9E+4I
+1.01B-0.10BI, 10B+1BI, 0+1E-32BI

*) В алголе real противопоставлялось integer.

**) В дальнейшем мы встретимся со случаями, когда пробелы внутри комплексного числа будут запрещаться.

Комплексные и мнимые числа будем обозначать через nC , а вещественные через nR .

Сочетание атрибутов. Три пары рассмотренных атрибутов могут задавать 8 различных видов арифметических данных:

REAL	DECIMAL	FIXED	—1000, 0.09375	RDF
REAL	DECIMAL	FLOAT	—1E3, 93.75E—03	RDE
REAL	BINARY	FIXED	1100100B	RBF
REAL	BINARY	FLOAT	110.0100E4B	RBE
COMPLEX	DECIMAL	FIXED	1.5+2.0I	CDF
COMPLEX	DECIMAL	FLOAT	4.11E—5+0.88E4I	CDE
COMPLEX	BINARY	FIXED	0.10B—0.01BI	CBF
COMPLEX	BINARY	FLOAT	—1E2B+1E—2BI	CBE

Здесь помимо возможных сочетаний атрибутов и их сокращенных обозначений приведены примеры констант, представляющих соответствующие значения. Порядок атрибутов не закрепляется и, например, при описании переменной можно задать

FIXED REAL DECIMAL

Атрибут точности. Точность представления и диапазон изменения описываемой величины задается *атрибутом точности*.

Ф о р м а: $t\text{-точности} \sim (\bar{n}RDI [, nRDI]) < \sim (p [, q]) >$

Например: (5), (6, 4), (5, 0), (4, —2), (4, 6).

Первое целое беззнаковое десятичное число (обозначаемое обычно через p) определяет минимальное количество цифр, которое должно отводиться транслятором для представления значения описываемой величины (или его мантиссы). Второе число (может быть со знаком), обозначаемое через q , задается только для значений с атрибутом FIXED и определяет масштабный множитель, то есть положение точки среди цифр, представляющих значение: точка устанавливается перед q -й цифрой, считая справа. Если q для FIXED не задано, то полагается $q=0$, что определяет целые значения (обозначаемые ранее через I — Integer).

Для десятичных (DECIMAL) величин имеется в виду количество десятичных цифр, а для двоичных (BINARY) — количество двоичных цифр. Например, атрибут точности (4) может определять такие значения:

—9999, 93.75E—03, 0020, 0001E2B, —0001B+1010BI

Атрибут (4, 1) может относиться к следующим значениям:

—999.9, 002.0, 000.1B—101.0BI

Допускается случай $p < q$. При этом $q-p$ старших разрядов значения предполагаются равными нулю (и в памяти машины не хранятся).

В случае, если $q < 0$, то положение точки фиксируется с п р а в а от $(p-q)$ -й цифры значения. При этом $abs(q)$ младших цифр значения предполагаются равными нулю (и в памяти машины не

хранятся). Например, указанные ниже атрибуты точности могут определять такие значения:

(4, 5) — .09999, .00020, .00001B — .01010BI

(4, —3) — 9999000, 0020000, 0001000B—1010000BI

Атрибут точности должен следовать в описании за одним из введенных ранее атрибутов для арифметических данных. Например:

DECLARE A FLOAT REAL BINARY (8),

C2 FIXED COMPLEX (3, 2) DECIMAL;

Для комплексной величины атрибуты относятся как к мнимой, так и к действительной части комплексного значения.

Атрибуты по умолчанию. Если некоторая величина не описана или для нее в описании не задан ни один арифметический атрибут (включая и атрибут точности), то в соответствии с правилами умолчания, принятыми в PL/1, атрибуты для такой величины устанавливаются в зависимости от 1-й буквы ее идентификатора: если идентификатор начинается с одной из букв I, J, K, L, M, N, то принимается

REAL BINARY FIXED,

иначе полагается

REAL DECIMAL FLOAT.

Если при описании указан хотя бы один из этих атрибутов, то недостающие выбираются из REAL, DECIMAL, FLOAT (а не из COMPLEX, BINARY, FIXED). Но в случае если указан атрибут точности, соответствующий данным с фиксированной точкой (с *q*), то устанавливается атрибут FIXED, а не FLOAT.

Если атрибут точности не задан, то для ЕС ЭВМ он выбирается в соответствии со следующей таблицей, в которой, кроме того, приводятся максимальные значения, допустимые для атрибута точности (*h*).

	по умолчанию максимум (<i>h</i>)	
DECIMAL FIXED	(5, 0)	(15, <i>q</i>)
BINARY FIXED	(15, 0)	(31, <i>q</i>)
DECIMAL FLOAT	(6)	(16)
BINARY FLOAT	(21)	(53)

$$\left. \begin{matrix} (15, q) \\ (31, q) \end{matrix} \right\} -128 \leq q < 127$$

Замечание. В 1-й главе, таким образом, было установлено соответствие между

integer и REAL DECIMAL FIXED (5, 0),

real и REAL DECIMAL FLOAT (6).

Тем самым для **integer** задавалась точность, соответствующая пяти десятичным знакам, а для **real** — 6 знакам. Если такая точность не устраивает, то атрибут точности нужно задать явно.

Например:

DECLARE (I, J, K) FIXED (8, 0), (P, Q, R) FLOAT (10);

Атрибуты констант. В то время как, например, для переменных атрибуты определяются в описаниях, константы сами задают свои атрибуты. Буква I в конце числа задает COMPLEX (в противном случае—REAL); буква B в конце числа задает BINARY (иначе DECIMAL); буква E в середине числа задает FLOAT (иначе FIXED). Количество цифр в числе или его мантиссе и положение точки среди них задают значения для p и q в атрибуте точности.

Таким образом:

100000 задает REAL DECIMAL FIXED (6, 0)

.1 задает REAL DECIMAL FIXED (1, 1)

0.1000E15BI задает COMPLEX BINARY FLOAT (5)

Преобразование арифметических данных. В алголе имеется 2 типа арифметических данных и 2 вида преобразований: из integer в real и из real в integer; последнее осуществляется по *entier* ($x + 0.5$).

В PL/1 имеется 8 видов арифметических данных (см. выше) и, таким образом, допускается 56 видов преобразований.

Но все эти преобразования состоят из 6 элементарных арифметических преобразований:

REAL \leftrightarrow COMPLEX или $R \leftrightarrow C$

FIXED \leftrightarrow FLOAT или $F \leftrightarrow E$

DECIMAL \leftrightarrow BINARY или $D \leftrightarrow B$

Эти преобразования и рассматриваются ниже. Пока мы ограничимся лишь кратким знакомством с указанными преобразованиями. Более подробные сведения об арифметических преобразованиях, касающиеся, например, точности преобразованного значения, приведены в 4.2.1.

Преобразование моды ($R \leftrightarrow C$). При преобразовании комплексного значения в действительное ($C \rightarrow R$) берется действительная часть комплексного значения. Преобразование действительного значения в комплексное ($R \rightarrow C$) сводится к добавлению нулевой мнимой части.

Например:

$-12.4 + 09.0I \rightarrow -12.4$

$4.560 \rightarrow 4.560 + 0.000I$

Преобразование масштаба ($F \leftrightarrow E$). Преобразование из фиксированной точки в плавающую и обратно делается по обычным правилам и сводится (для $F \rightarrow E$) к разделению значения на порядок и мантиссу или (для $E \rightarrow F$)—к определению положения точки среди цифр значения.

Например:

$83.56 \rightarrow 8.356E1$

$10.100E2B \rightarrow 1010.0B$

Преобразование основания ($D \leftrightarrow B$). Преобразование из одного основания в другое заключается в переводе значения из одной системы счисления в другую.

Например:

$$0.5 \rightarrow 0.1B$$

$$1.1E - 3B \rightarrow 1.875E - 1$$

Следует отметить, что при преобразовании основания происходит перевычисление атрибута точности. Считается, что каждая десятичная цифра «в среднем» соответствует $3.32 \left(\approx \frac{\log_2 x}{\log_{10} x} \right)$ двоичным цифрам (с округлением до большого целого). Поэтому для случая, когда значение не представляется точно в другой системе счисления возможна значительная потеря точности.

Например, оказывается, что десятичное число 0.1 после преобразования к двоичному основанию будет иметь значение 0000.0001B, что равно 0.0625; а 0.1000 преобразуется к значению 0000.00011001100110B, равному примерно 0.09998. Таким образом незначащие нули константы могут играть существенную роль: они указывают на требуемую точность представления и преобразования. Точность преобразования переменной зависит от присланного ей атрибута точности. К этому вопросу мы еще вернемся в 2.2.1.

Приведение точности ($p^0 \rightarrow p$). Преобразование значения с одной точностью к другой заданной, производимое, обычно, при присваивании будем называть *приведением*. Оно сводится к добавлению нулей (двоичных или десятичных) или к отсеканию «лишних» цифр. Отсекание младших цифр производится без округления. В частности, так же происходит и приведение к целым значениям, например, при вычислении индексов. Отсекание старших цифр (возможное только при приведении к FIXED) может привести к ошибке типа SIZE («диапазон»), если среди отсекаемых цифр есть ненулевые. Например, в принятых обозначениях:

для ED (6) \rightarrow ED (4) производится $-7.62548E4 \rightarrow -7.625E4$

$$FD (4,2) \rightarrow FD (2,0)$$

$$12.95 \rightarrow 12$$

$$FD (4,2) \rightarrow FD (4,3)$$

$$12.25 \rightarrow 2.250 \text{ (ошибка)}$$

$$FB (3,1) \rightarrow FB (8,2)$$

$$-01.1B \rightarrow -000001.10B$$

2.1.2. Строчные данные.

В PL/1 имеется 2 основных вида строчных данных: *символьно-(литерно)-строчные* и *битово-строчные*. В отличие от алгола, в PL/1 для работы с символьными строками помимо строчных констант (см. 1.1.7) введены строчные переменные и массивы, определены строчные операции и функции. Битово-строчные данные можно считать обобщением логических данных алгола: они

являются последовательностью логических значений, представленных единицами и нулями.

Символьные строки. Атрибут символьно-строчных величин — CHARACTER (сокращенно CHAR); мы будем обозначать его через H.

Общая форма для символьно-строчных констант имеет следующий вид (ср. с 1.1.7):

$gH \sim [(\bar{n}DI)]' [\xi]' \dots' <\xi \text{ — кроме апострофа}$

Строчная константа заключается в апострофы и состоит из символов конкретного представления — символов ЭВМ (исключая апостроф) и из пар апострофов. Символы ЭВМ (см. 4.4.6) могут отличаться на разных экземплярах ЭВМ. Целое беззнаковое десятичное число в круглых скобках задает коэффициент повторения заданной строки.

Например:

'LAMARA' (2)'ПА' 'COS (X + Y) = '

'В_СТРОКЕ_ "СТРОКА" _ИЛИ_ "СТРОКА" '

Для того, чтобы получить значение, которое представляет строчная константа, нужно отбросить внешние апострофы, а каждую пару апострофов заменить одним; полученную таким образом совокупность символов повторить указанное в коэффициенте число раз. Значениями приведенных выше строк будут:

LAMARA ПАПА COS (X + Y) =

В_СТРОКЕ_ "СТРОКА" _ИЛИ_ "СТРОКА"

Выделим частный случай символьных строк — **числовые строки**, значением которых может быть лишь число (вещественное или комплексное), окаймленное, может быть, пробелами; пробелы внутри комплексного числа в этом случае не допускаются.

Например:

(3)'19' '191919'

'5.45 + 4.30I' ' _1000.111E — 13В_ _ '

З а м е ч а н и е. Несмотря на то, что строчная константа ограничивается апострофами, она должна отделяться пробелами от соседних идентификаторов, ключевых слов, констант.

Битовые строки. Значением битово-строчных величин является последовательность двоичных цифр 0 и 1 — **битов**. Битово-строчные — **битовые** — величины характеризуются атрибутом BIT; мы его будем обозначать через T.

Ф о р м а для битовых констант:

$gT \sim [(\bar{n}DI)]' [0 | 1] \dots' B$

Пробелы в битовой константе не допускаются; $\bar{n}DI$ — коэффициент повторения строки (см. выше).

Например:

(12)'1'B, '0'B, '1001'B, "B

Значениями приведенных констант являются:

11111111111, 0, 1001; значение последней строки — пусто.

См. выше замечание о пробелах.

Атрибут длины. Атрибут длины характеризует количество символов (или битов) в строчной величине. Атрибут длины обязательно задается при описании строчных величин (переменных, функций) и ставится вслед за BIT или CHARACTER (или CHAR); константы сами определяют свою длину.

Форма: t-длины $\sim (eA) \langle \sim (l) \rangle$

Длина обозначается обычно через l и имеет максимальное значение равное $32767 (2^{15} - 1)$.

Для строчных констант, приведенных выше (символьных и битовых), атрибуты длины таковы:

(6), (4), (9), (32); (6), (6), (10), (16); (12), (1), (4), (0),

Пример.

DECLARE LIT CHAR (6),

(BITOW1, B2) BIT (J+1);

Значениями переменной LIT являются последовательности из 6 символов; значения переменных BITOW1 и B2 — последовательности из $J+1$ (J должно быть определено во внешнем блоке) битов.

Преобразование строк. Преобразование строк состоит в изменении их вида или их длины.

Преобразование вида. ($H \leftrightarrow T$). Преобразование символьной строки в битовую ($H \rightarrow T$) возможно лишь в том случае, если первая содержит только символы 0 и 1, которые становятся битовыми цифрами. При преобразовании битовой строки в символьную ($T \rightarrow H$) каждый бит преобразуется в символ. Таким образом, например:

'101'B \rightarrow '101'

'011' \rightarrow '011'B

Приведение длины ($l^0 \rightarrow l$). Приведение к заданной длине осуществляется (например, при присваивании) отбрасыванием «лишних» правых знаков строки или добавлением к строке справа пробелов для (CHAR) или нулей (для BIT). Например, для $H(5) \rightarrow H(7)$ производится 'TARAN' \rightarrow 'TARAN□□'

$T(6) \rightarrow T(4)$

'101011'B \rightarrow '1010'B

$T(6) \rightarrow T(8)$

'101011'B \rightarrow '10101100'B

Переменная длина. Имеется возможность с помощью атрибута VARYING (сокращенно VAR) задать для строчных величин не фиксированную, а переменную длину. В этом случае атрибут длины будет иметь смысл максимального значения для длины. Текущая длина строчной переменной, для которой задан атрибут VARYING, равна длине строчного значения, присвоенного этой переменной в последний раз; вначале длина равна нулю.

Пример.

DECLARE S CHAR (5), KOR BIT (3),

VS CHAR (8) VARYING;

VS='+'; KOR='1'B; S='TIR';

/* VS='+', KOR='100'B, S='TIR' */

VS=S; S=KOR; KOR='10101'B;

/* VS='TIR', S='100', KOR='101'B */

2.1.3. Проблемные данные и преобразования типа.

Арифметические данные и строчные данные образуют единый класс данных, которые называются *проблемными* данными (точнее их следовало бы называть «обрабатываемыми» данными). В классе проблемных данных помимо ранее введенных преобразований определены и преобразования *типа* — между арифметическими и строчными данными.

Преобразования типа. Здесь мы кратко рассмотрим 4 вида преобразований типа, не касаясь сложных вопросов, в частности вопроса о точности или длине преобразованного значения. Более подробные сведения приведены в 4.2.3, куда и следует обращаться по мере необходимости.

Как мы увидим позднее (п.п. 4.2.3 и 4.2.5), преобразования типа могут давать весьма неожиданные результаты, и поэтому, по возможности, их следует избегать, тем более, что они (как, впрочем, и остальные преобразования) сильно снижают эффективность программы. Преобразования типа приходится использовать обычно в том случае, когда над одними и теми же данными необходимо производить операции различного типа: арифметические, строчные, логические.

Битовый в арифметический ($T \rightarrow A$). Битовая строка преобразуется в положительное целое двоичное число (которое затем может быть подвергнуто необходимым арифметическим преобразованиям). Например:

'01011'B \rightarrow 01011B

Арифметический в битовый ($A \rightarrow T$). Арифметическое значение, теряя знак и дробную часть, преобразуется сначала к положительному целому двоичному числу, а затем побитно — в битовую строку. Например:

11011.01B \rightarrow '11011'B

—9.85 \rightarrow '1001'B

Символьный в арифметический ($H \rightarrow A$). Символьная строка преобразуется в арифметическое значение только в том случае, если строка является числовой. Преобразование заключается в выделении числового значения, содержащегося в символьной строке. В некоторых случаях могут отбрасываться мнимая и дробная части выделенного значения; кроме того могут производиться и другие

арифметические преобразования. Например:

'□□—8135□□□' → -8135

'□1.3—2.4□□' → 1

Арифметический в символьный (A → H). Преобразование арифметических данных к символьным строкам имеет специфический характер, поскольку применяется, как правило, при выводе данных на печать. Каждая десятичная цифра арифметического значения (может быть предварительно переведенного из двоичного основания) преобразуется в соответствующий символ. Длина полученной строки зависит от атрибутов преобразуемого значения. Например:

+3126.40 → '□□3126.40'

-0.17350E5 → '-1.73500E+04'

Представление проблемных данных в ЕС ЭВМ. Для тех, кто знаком со способами представления данных в ЕС ЭВМ, может быть полезна следующая информация:

FIXED DECIMAL—представляются в десятичном упакованном формате и занимают $entier(p/2) + 1$ байтов, то есть фактическая точность при четном p будет равна $p + 1$;

FIXED BINARY—представляются в фиксированном (двоичном) формате и занимают 2 (при $p \leq 15$) или 4 байта, то есть фактическая точность будет равна 15 (для $p \leq 15$) или 31—(для $15 < p \leq 31$); константы занимают всегда 4 байта;

FLOAT DEC и **FLOAT BIN**—представляются в одном и том же плавающем (двоичном) формате и занимают 4 (при $p \leq 6$ для DECIMAL и при $p \leq 21$ для BINARY) или 8 байтов. То есть можно считать, что фактическая точность будет равна 6 (или 21) для $p \leq 6$ (или $p \leq 21$) или 16 (или 53) для $6 < p \leq 16$ (или $21 < p \leq 53$);

COMPLEX—представляются двумя вещественными значениями и занимают соответственно двойное количество байтов;

CHARACTER—каждый символ строки занимает 1 байт;

BIT—в одном байте размещается 8 битов (следующий элемент битового массива располагается, начиная с текущего бита в байте).

В других трансляторах возможно другое представление проблемных данных.

2.1.4. Меточные данные.

Меточные данные представляют класс данных управления программой. Те метки, с которыми мы имели дело в 1-й главе, являются меточными константами; они могут быть значениями меточных переменных. Меточные переменные имеют атрибут LABEL, и они могут появляться в операторе перехода и в левой части оператора присваивания, в правой части которого может находиться меточная константа или меточная переменная.

Пример.

```
DECLARE (M, N) LABEL;
```

```
    K=0;
```

```
L1: IF K $\nabla$ =0 THEN M=L3; ELSE M=L2;
```

```
    N=M; GO TO N; /* ПЕРЕХОД НА L3 ИЛИ L2 */
```

```
L2: K=1; GO TO L1;
```

```
L3: K=2;
```

Меточная переменная M принимает значения L3 или L2; переменная N принимает те же меточные значения.

Меточные переменные могут быть и с индексами; в описаниях они снабжаются описателем массива — атрибутом размерности (см. ниже). Примеры их использования были даны в 1-й главе при рассмотрении переключателя алгола (п.п. 1.2.4 и 1.4.3). В PL/1 меточные массивы («переключатели») могут быть многомерными с произвольными граничными парами.

Для оптимизации и контроля работы программы можно в описании меточных переменных вслед за атрибутом LABEL перечислить те метки, которые допустимы в качестве значений описываемой переменной. Например:

```
DECLARE (W, V (0:2)) LABEL (L1, L2, L);
```

Переменные W и V могут принимать только значения L1, L2, L; другие метки-значения будут ошибочными. Если такого списка меток нет, то разрешаются любые меточные значения.

Помимо использования обычных меток, в PL/1 операторы можно метить «массивными» метками, имеющими вид переменной с индексами: **iML (nD1,...)**; идентификатор меточного массива (**iML**) должен быть описан.

Пример.

```
DECLARE KLM (-1 : + 1, 2:3) LABEL;
```

```
    S=0; T=2;
```

```
    GO TO KLM (S, T); /* ПЕРЕХОД НА KLM (0, 2) */
```

```
KLM (0, 2): GO TO KLM (S-1, 3); /* ПЕРЕХОД НА  
                                KLM (-1, 3) */
```

```
KLM (-1, 3):
```

Форма для метки:

```
1 ~ {iL | iML (nD1,...)}
```

Здесь **iL** — идентификатор, который вместе со следующим за ним двоеточием приписан к какому-либо оператору программы. Эти обычные («скалярные») метки — **iL** — можно считать константами, поскольку они служат для представления значений меточных переменных, а массивные метки следует трактовать как переменные с индексами.

2.1.5. Массивы и сечения.

Массивы. И проблемные переменные, и меточные могут быть снабжены атрибутом размерности (diMention), который в терминологии алгола мы называли описателем массива.

Ф о р м а: $tM \sim (\{[eA_n:] eA_n\}, \dots)$

Если нижняя граница отсутствует, она полагается равной единице. Атрибут размерности нужно располагать сразу вслед за описываемым идентификатором или за скобками, заключающими список описываемых идентификаторов.

П р и м е р:

```
DECLARE A (0:20, 0:9),  
(B, C) (30, 30) COMPLEX FIXED (5, 4),  
D (N) BIT (5) VARYING,  
K (S * 2) LABEL;
```

Граничными парами для массивов A, B, C, D, K соответственно являются: (0:20, 0:9), (1:30, 1:30), (1:30, 1:30), (1:N), (1:S * 2); переменные N и S должны иметь значения при входе в блок.

В PL/1 для действий целиком над массивами введены «*массивные*» переменные (или *переменные над массивами*), которые задаются идентификатором массива. В частности, идентификатор массива может являться операндом в выражениях (см. 2.2.3) или стоять в левой части оператора присваивания (см. 2.3.2). Например, оператор $B = C * B$; выполняет поэлементное умножение матриц B и C, описанных выше.

Сечения. В PL/1 имеется возможность использовать специальные массивы—*сечения*, которые являются частями обычных массивов. Для образования таких массивов нужно у переменной с индексами вместо каких-либо индексов-выражений поставить звездочки, которые будут указывать на то, что данный индекс принимает все значения от нижней границы до верхней (определенных в описании). С сечениями можно производить те же действия, что и с обычными массивами (см. ниже 2.2.3 и 2.3.2).

П р и м е р.

```
DECLARE (R, T) (10, 10) FLOAT, Z (10) FIXED;
```

```
.....  
Z = T (*, 1); R (I, *) = R (J, *) - R (K, *);
```

Массиву Z присваиваются значения 1-го столбца матрицы T; *i*-й строке матрицы R присваивается разность *j*-й и *k*-й строк.

2.1.6. Начальные значения.

С помощью атрибута INITIAL («начальный») переменным можно присвоить начальные значения (они будут присваиваться при входе в блок, содержащий соответствующее описание).

П р и м е р.

```
DECLARE (X, Y) FIXED INITIAL (0),
```

(B, A) (0:4) FLOAT (8) INITIAL (1E-4, 1E-5, (3) 0),
 M (20, 10) FLOAT INITIAL ((50) *, (50) 0,
 (10) (0,1,2,3,4,5,6,7,8,9));

Простые переменные X и Y обнуляются; первым двум элементам массивов A и B присваиваются 10^{-4} и 10^{-5} , а остальные три обнуляются; первые 50 элементов (5 строк) массива M пропускаются, следующие 5 строк обнуляются, очередным десяти элементам каждой из последних 10 строк присваиваются значения от 0 до 9.

Как видно из примера, присваивание начальных значений массиву производится по строкам; число в скобках перед константой или списком констант задает кратность их повторения при присваивании, что позволяет сократить перечень начальных значений; звездочка * указывает на пропуск присваивания для очередного элемента массива.

Приведем упрощенные формы атрибута начальных значений:

— для простой переменной:

$t \sim \text{INITIAL}[(c | *)]$

— для массива:

$t \sim \text{INITIAL}[\{[(eA)] \{c | * | ((c | *), \dots)\}, \dots]$

Здесь eA обозначает любую константу — арифметическую, строчную, меточную; арифметическое выражение задает кратность повторения (если $eA \leq 0$, то следующие за ним константы пропускаются).

Пример (для строчных переменных).

DCL A BIT(3) INIT('1'B),

B CHAR (6) INIT((5)'A')VAR,

C (12) CHAR (8) INIT ((6) (4)'NO');

/* A = '100'B, B = 'AAAAA',

C(1), C(2), C(3), C(4), C(5), C(6) = 'NONONONO' */

В атрибуте INITIAL массива C первое число в скобках — это кратность повторения начальных значений, а второе число — коэффициент повторения строки. Если задано только одно такое число, то оно считается за повторитель строки. То есть:

(3)'A' это 'AAA'

(3) (1)'A' это 'A', 'A', 'A'

Если в INITIAL задано начальных значений меньше, чем элементов в массиве, то последним из них ничего не присваивается; если же больше, — то последние значения в INITIAL не используются. Атрибут начальных значений может занимать любое место среди других атрибутов, но всегда после атрибута размерности (если он есть).

Пример.

DECLARE (J1 (40), J2 (20)) CPLX INIT ((30) 1 + 1I) FLOAT;

Атрибут INITIAL, так же как и атрибуты CPLX и FLOAT,

относится равноправно и к J1, и к J2. Поэтому последние 10 элементов массива J1 останутся «неинициализированными», но все элементы массива J2 получают исходные значения.

Метки (меточные константы), указываемые в атрибуте INITIAL для инициализации меточных переменных, должны быть известны внутри блока, где находится такое описание. Примеры инициализации меточных массивов были даны еще в 1-й главе при рассмотрении переключателей (см. 1.2.3 и 1.4.3). Дадим еще пример описания меточных массивов, который аналогичен приведенному в 2.1.4, но не использует массивных меток:

```
DECLARE KLM (-1: +1, 2: 3) LABEL  
INITIAL (K1, K2, L1, L2, M1, M2),  
(S INITIAL (0),  
T INITIAL (2)) FIXED BINARY;
```

```
GO TO KLM (S, T); /* ПЕРЕХОД НА L1 */  
L1: GO TO KLM (S-1, 3); /* ПЕРЕХОД НА K2 */  
K2:
```

Для присваивания значений переменным S и T мы воспользовались атрибутом INITIAL.

Атрибут INITIAL не может быть использован при описании массивных меток. Сами массивные метки нельзя применять для инициализации меточных переменных, поскольку они не являются константами (см. 2.1.4).

Наиболее общим способом инициализации является CALL-опция атрибута INITIAL. В этом случае вместо списка присваиваемых значений, заключенных в скобки, задается CALL-оператор (без точки с запятой) для вызова процедуры, в которой и производится присваивание нужных значений переменным. Выполнение процедуры производится при входе в блок, то есть при «размещении» переменных в памяти.

Ф о р м а:

t-начальных-значений ~ INITIAL CALL IP (a,...)

Например:

```
DECLARE B (0:15) INITIAL CALL IP (Q);
```

Предполагается, что Q определено во внешнем блоке. Описание процедуры может иметь, например, такой вид:

```
IP: PROCEDURE (X); DECLARE X FLOAT;
```

```
DO I=0 TO 15; B (I)=Q+I; END;
```

```
END;
```

Описание входа для процедуры IP (см. 1.4.4):

```
DECLARE IP ENTRY (FLOAT);
```

Для простоты можно считать, что порядок размещения и инициализации переменных соответствует порядку их описания в блоке (слева направо и сверху вниз). Например, допустимо следующее описание:

DECLARE N INITIAL (15), A (— N: +N) INITIAL ((2 * N + 1) 0),
 Q INITIAL (25), B (0: N) INITIAL CALL IP (Q);

На самом деле допускается и следующий порядок описания: A, B, Q, N, поскольку транслятор восстанавливает необходимый порядок размещения и инициализации переменных, если это возможно. Недопустимым является описание:

DECLARE N INITIAL CALL IP (Q), Q INITIAL CALL IP (N);

или

DECLARE A (—5: +5) INIT ((B (0)) 0), B (0: A (5)) INIT ((20) 1);

так как размещение переменных N и Q, A и B взаимосвязано.

Отметим, что с помощью CALL-опции в принципе можно инициализировать и другие переменные, отличные от той, к которой относится CALL-опция в описании, но при этом бывает важен порядок описания таких переменных.

Форма. В заключение дадим общую форму для атрибута начальных значений, учитывающую CALL-опцию и тот факт, что коэффициент кратности может быть многократно вложенным:

$t \sim \text{INIT} [\text{IAL}] \{(\Delta) | \text{CALL IP} (a, \dots)\}$

$\Delta \sim \{[(eA)] \{c | * | (\Delta)\}, \dots\}$

Эта форма мало наглядна в силу своей рекурсивности, и поэтому на практике удобнее обращаться к упрощенным формам, приведенным выше.

2.1.7. Терминология и формы.

Итак, данные PL/1 подразделяются на 2 основных класса: проблемные (обрабатываемые) и меточные (управляющие); проблемные данные в свою очередь делятся на арифметические и строчные.

Значениями проблемных данных являются числа и строки, меточных — простые метки (меточные константы).

Таким образом:

$v \sim \{vA | vG | vL\} \sim \{vO | vL\}$

$c \sim \{n | g | iL\} \sim \{cA | cG | cL\}$

$g \sim \{gH | gT\} < cG \sim \{cH | cT\} >$

Обозначения:

G — строчный (string); раньше G определяло «логические» (то есть битовые) данные PL/1, которые можно считать частным случаем строчных данных;

O — обрабатываемый, проблемный (problem);

c — константа (constant).

По своей организации данные можно подразделить на скаляры и массивы. Значение скалярной переменной — одна константа; значение «массивной» переменной — упорядоченное множество констант. То есть:

$v \sim \{vS | vM\}$

$vS \sim \{iS | iM (eAS, \dots)\}$

$vM \sim \{iM | iM (\{eAS | (*), \dots\})\}$

Здесь

S — скалярный (Scalar),

eAS — скалярное выражение (см. **eA** в 1-й главе),

iS — идентификатор скаляра (в 1-й главе — **iV**).

Переменная с индексами (как и простая переменная), введенная в 1-й главе, является примером скалярной переменной, так как определяет лишь одно значение. В форме для **vM** вторая альтернатива определяет массив-сечение (среди его индексов должна быть хотя бы одна звездочка). Видимо, яснее было бы:

iM (**eAS**, **|*,|** ... ***[, eAS|, *,|** ...)

В заключение приведем формы для чисел **PL/1**:

$n \sim \{nR | nC\}$

$nR \sim \{nRD | nRB\}$

$nC \sim \{nR | \lfloor \dots \rfloor \{\pm\} \lfloor \dots \rfloor \bar{n}RI | nRI\}$

$nRD \sim \{nRDF | nRDE\}$

$nRB \sim \{nRBF | nRBE\}$

$nRDF \sim [\pm] (\bar{n}ID) [.](\bar{n}ID)$

$nRBF \sim [\pm] (\bar{n}IB) [.](\bar{n}IB) B$

$nRDE \sim nRDF E [\pm] \bar{n}ID$

$nRBE \sim [\pm] (\bar{n}IB) [.](\bar{n}IB) E [\pm] \bar{n}ID B$

$\bar{n}ID \sim v \dots$

$\bar{n}IB \sim \{0 | 1\} \dots$

Кроме того примем:

$nID \sim [\pm] \bar{n}ID$

$nIB \sim [\pm] \bar{n}IB$

$nI \sim \{nID | nIB\}$

(точнее было бы **nRID**, **nRIB**, **nRI**).

Упражнения.

2.1.1. Какие атрибуты будут иметь переменные, описанные следующим образом:

DECLARE A FIXED, C BINARY, E FLOAT,
G DEC (7), K (4), N CPLX;

2.1.2. Определить атрибуты следующих констант:

а) 1., 1B, 1.0, 01B, '1.', '01'B, '1.'B, '1E0';

б) 1E0B1, 11, '1+0I', '1B1'.

2.1.3. Массиву **M** (15, 15) присвоить по **INITIAL**:

- а) единицы для главной диагонали, а остальным элементам — нули;
- б) единицы для первого и последнего столбца, а остальные элементы пропустить;
- в) единицы для побочной диагонали, а остальным элементам — нули.

2.2. Уточнение выражений

В **PL/1**, строго говоря, не различаются арифметические и логические выражения, так как благодаря преобразованиям типа допускаются логические операции над арифметическими данными

и арифметические операции над строчными данными. Кроме того, в PL/1 нет понятия именуемого выражения, и термин «выражение» определяет выражение, компонентами которого являются только проблемные данные.

Основные операции в выражениях PL/1 те же, что и в алголе, и они были введены в 1-й главе (см. 1.2.1 и 1.2.2); ниже мы отметим лишь те особенности, которые свойственны операциям в PL/1. Дадим сразу сводку всех операций PL/1 в порядке их старшинства (о некоторых операциях разговор еще впереди),

1. $**$, \neg ; $+$ и $-$ (одноместные)
2. $*$, $/$
3. $+$, $-$ (двуместные)
4. $!!$
5. $<$, $< =$, $>$, $> =$, $=$, $\neg =$, $\neg >$, $\neg <$
6. $\&$
7. $!$

Операции PL/1 по типу получаемого результата могут быть разделены на арифметические и строчные. Последние делятся на операции сравнения, логические операции и операцию сцепления ($!!$).

2.2.1. Арифметические операции.

Одноместные операции. В PL/1 одноместные («префиксные») арифметические операции ($+$ и $-$) отнесены к операциям самого старшего ранга наряду с $**$ и \neg ; допускается сочетание одноместных арифметических операций с другими операциями. Например:

$-- A, A*-B, C**+-D**E$

Порядок выполнения нескольких идущих подряд одноместных операций — с п р а в а н а л е в о, то есть приведенные выражения эквивалентны следующим: $--(-A), A*(-B), C**(+(-(D**E)))$.

З а м е ч а н и е. Таким образом, порядок выполнения операций в выражениях типа $-A*B$ в PL/1 другой, чем в алголе, но это не должно привести к различию в результатах при вычислении подобных выражений в двух языках. Отметим, что как и в алголе $-A**B$ вычисляется справа налево.

Приведение операндов. Перед выполнением арифметической операции [строчные операнды преобразуются к арифметическому типу ($H \rightarrow A, T \rightarrow A$); результат операции всегда имеет тип арифметический. Если арифметические атрибуты операндов (может быть, уже после преобразования типа) различаются, то производятся дополнительные преобразования операндов для приведения их к одинаковым атрибутам (кроме атрибута точности). При этом в соответствии с правилами приведения атрибутов могут выполняться только следующие преобразования:

$R \rightarrow C$ (REAL \rightarrow COMPLEX)
 $F \rightarrow E$ (FIXED \rightarrow FLOAT)
 $D \rightarrow B$ (DECIMAL \rightarrow BINARY)

Таким образом, можно считать, что атрибуты COMPLEX, FLOAT, BINARY являются «старшими» атрибутами, а REAL, FIXED, DECIMAL — «младшими», и приведение атрибутов заключается в преобразовании к старшим атрибутам. Напомним, что приведение осуществляется только для различающихся атрибутов операндов и в указанном выше порядке. Общие атрибуты, определенные из правил приведения, и являются атрибутами результата (для операции ** имеется исключение из этого правила: если показатель степени является целым беззнаковым числом, то его преобразование не делается).

Примеры.

DECLARE C1 COMPLEX FIXED BINARY
 INITIAL (1.01B + 0.01BI);
 $C1 * 1E1B = (1.01B + 0.01BI) * 1E1B$
 преобразуется в
 $(1.01E0B + 0.01E0BI) * (1E1B + 0E0BI)$
 $K1 + 0.75 \rightarrow K1 + 0000.1100000B$
 $JOB - 0.8 \rightarrow JOB - 0000.1100B$
 (= JOB - 0.75)

$CBF * RBE \rightarrow CBE * CBE$
 $RBF + RDF \rightarrow RBF + RBF$

Предполагается, что K1 и JOB объявлены по умолчанию и имеют атрибуты BINARY FIXED. Количество двоичных цифр у преобразованных операндов 0.75 и 0.8 определено по коэффициенту 3.32 (см. 2.1.1 и 4.2.1). Последний пример демонстрирует потерю точности при преобразовании десятичного значения (числа) с фиксированной точкой, не представимого точно в двоичной системе.

Для предотвращения потери точности при преобразовании к другому основанию следует величины с фиксированной точкой задать с нужной точностью (в константах должно быть нужно-количество цифр) или, если допустимо, перейти к вычислениям с плавающей точкой. Например, в предыдущем примере выражение можно было бы написать так:

$JOB + 0.8000$ или $JOB * 0.8E0$

Для случаев $K1 + 0.75$ или $JOB * 134$ подобные трансформации были бы несобязательны, так как 0.75 и 134 и в двоичном виде представляются точно.

Формально и при операциях над значениями с плавающей точкой может происходить потеря точности, но фактически она не происходит, так как в ЕС ЭВМ такие значения и в десятичном

и в двоичном основании представляются в машине в одном и том же виде (см. 2.1.3), и никакого преобразования поэтому вообще не производится (но значение точности перевычисляется и используется в дальнейшем).

З а м е ч а н и е. Если приводимое к другому основанию число является правой частью оператора присваивания, то оно может быть задано с любой точностью, так как в этом случае точность преобразования определяется не точностью, с которой задана константа, а атрибутом точности переменной в левой части. Например, для

```
DECLARE (K2 BINARY (15, 14),  
        K3 BINARY (31, 30)) FIXED;  
K2, K3=0.8;
```

переменная K2 получит значение равное 0.8 с точностью 14 двоичных цифр, а K3—30 двоичных цифр после запятой, что соответствует примерно 4 и 9 десятичным цифрам.

Точность выражения. Для результата любой арифметической операции определяется атрибут точности, который зависит от вида операции и атрибутов операндов. Атрибут точности (как и остальные атрибуты) для промежуточного и окончательного результатов выражения определяется на стадии трансляции, а на стадии выполнения программы получающиеся результаты приводятся к этим, заранее вычисленным атрибутам точности.

Для операций над данными с плавающей точкой точность результата полагается равной максимуму из точностей двух операндов. Например:

$1.2345E5 * 1.11E-4 \rightarrow 1.3702E1$

так как

$ED(5) * ED(3) \rightarrow ED(5)$

(Фактически, принимая во внимание представление данных с плавающей точкой в ЕС ЭВМ, можно считать, что в данном случае точность, как операндов, так и результата, будет равна 6 десятичным знакам (см. 2.1.1)).

При определении точности результата операций над данными с фиксированной точкой в PL/1 исходят из того, что результат операции должен быть представлен абсолютно точно (так же как абсолютно точно представлены и операнды). Поэтому младшие цифры результата не могут отбрасываться или округляться, они должны сохраняться все полностью (именно так поступают, например, в бухгалтерских расчетах с копейками). Что же касается сохранности старших разрядов, то предполагается, что программист должен сам позаботиться о том, чтобы общее количество цифр в результате не превышало максимально допустимого в машине. В противном случае, возникает ошибка типа

FIXEDOVERFLOW (фиксированное переполнение). Напомним, что в ЕС ЭВМ для **FIXED DECIMAL** максимальная точность принимается 15, а для **FIXED BINARY** — 31.

Таким образом, при работе с данными в форме с фиксированной точкой требуется постоянный контроль за количеством цифр в результатах операций. Например, оказывается, что при вычислении значений следующих выражений (см. также 1.2.1) получится переполнение, так как точные результаты для этих выражений занимают более 15 цифр:

$$1234567.89 - 0.123456789, \\ 1.23456 * 1.11111 * 1.78900$$

Для предупреждения возможных переполнений полезно ознакомиться с формулами определения атрибута точности результата операций с фиксированной точкой, которые использует транслятор при своей работе. Формулы приведены в 4.2.4; здесь мы дадим некоторые примеры их применения и пояснения. Под арифметическими примерами даны атрибуты операндов и результата; последние вычислены по формулам из 4.2.4.

$$1) \begin{array}{r} + \quad 999.99 \\ \quad 1.109 \\ \hline 1001.099 \end{array}$$

$$2) \begin{array}{r} * \quad 99.99 \\ \quad 2.0001 \\ \hline 0199.989999 \end{array}$$

$$\text{FD}(5, 2) + \text{FD}(4, 3) \rightarrow \text{FD}(7, 3) \quad \text{FD}(4, 2) * \text{FD}(5, 4) \rightarrow \text{FD}(10, 6)$$

$$3) 4.0/0.3 = 13.33333333333333 \quad 4) 9.9 ** 2 = 098.01$$

$$\text{FD}(2, 1)/\text{FD}(2, 1) \rightarrow \text{FD}(15, 13) \quad \text{FD}(2, 1) ** 2 \rightarrow \text{FD}(5, 2)$$

Из примеров можно заметить, что точность p (а также и масштаб q) результата операций сложения и умножения полагается равной максимально возможной точности, которая, в принципе, может получиться, когда заданная операция будет выполняться над операндами с заданными атрибутами точности. Для операции возведения в степень при определении точности учитывается величина показателя (а не его точность), который при этом предполагается целым числом: иначе степень вычисляется с плавающей точкой. Для операций умножения и возведения в степень добавлена еще единица на случай возможного округления при преобразовании основания. Для операции деления, которая может давать и неточный результат, ему всегда приписывается максимально допустимая в машине точность, а на целую часть результата отводится максимальное количество цифр, которое когда-либо может получиться при фактическом делении операндов с указанными атрибутами точности.

Из приведенных правил определения точности следует, например, что выражение $123 + 4.0/0.3$ вызовет переполнение, так как для получения «точного» результата потребуется более 15 цифр:

$$123 + 4.0/0.3 = 123 + 13.333333333333 = \\ = 136.333333333333 \rightarrow 36.333333333333$$

так как

$$FD(3) + FD(2, 1)/FD(2, 1) \rightarrow FD(3) + FD(15, 13) \rightarrow FD(15, 13),$$

то есть для целой части значения отводится только две цифры (15 — 13 = 2).

Правильный результат можно получить, если написать
 $123 + 04.0/0.3$ или $123 + 4.0/0.30$;

тогда получим:

$$123 + 4.0/0.30 = 123 + 013.333333333333 = \\ = 136.333333333333$$

так как

$$FD(3) + FD(2, 1)/FD(3, 2) \rightarrow FD(3) + FD(15, 12) \rightarrow FD(15, 12)$$

Можно избежать переполнения, если перейти к вычислениям с плавающей точкой (вспомните о правилах п. 1.2.1). Другие примеры можно найти в 4.2.4 и 4.2.5.

З а м е ч а н и е. Комплексная константа, входящая в выражение, сама трактуется как выражение, состоящее из двух операндов, соединенных знаком операции сложения или вычитания (и разделенных, может быть, пробелами). Таким образом, точность константы $-2.1 + 3.4I$, рассматриваемой как выражение, равна не (2, 1), а (3, 1); константа-выражение $2.1 \sqcup + \sqcup \sqcup \sqcup 1.0E1BI$ имеет атрибуты FLOAT BINARY (7). Кроме того, оказываются законными, например, следующие выражения: $-2.1 + - - 3.4I$ и $++ + 3.4I + -2.1$.

Это замечание не относится к комплексным константам, фигурирующим в строках, в атрибуте INITIAL и в некоторых других конструкциях PL/1.

2.2.2. Строчные операции.

Операции сравнения. В PL/1, помимо введенных ранее в 1-й главе операций сравнения, имеются еще две $\sqsupset >$ и $\sqsupset <$, эквивалентные $< =$ и $> =$. Операции сравнения PL/1 являются обобщением операций алгола: с их помощью можно сравнивать не только арифметические операнды, но и строчные.

Результат операции сравнения всегда имеет атрибуты BIT (1): истина соответствует '1'В, ложь — '0'В.

Операндами операций сравнения могут быть данные трех типов: арифметические, символично-строчные и битово-строчные (А, Н, Т). В зависимости от типов операндов различаются 3 типа сравнений:

- 1) *Битовое* — если оба операнда типа Т ($T \leq T$);
- 2) *Символьное* — если один из операндов типа Н, а другой — Н или Т ($H \leq H, H \leq T$);

3) *Алгебраическое* — если один из операндов типа А ($A \leq A$, $A \leq H$, $A \leq T$).

Битовое сравнение производится побитно слева направо (до первого несовпавшего бита); предварительно более короткая строка дополняется справа нулями.

Литерное сравнение производится посимвольно слева направо (до первого несовпавшего символа). Если одна из строк битовая, она преобразуется к символьной ($T \rightarrow H$); кроме того, более короткая строка дополняется справа пробелами. Фактически, сравнение производится над 16-ричными (двоичными) кодами, представляющими символы в ЕС ЭВМ (см. 4.4.6). В соответствии с принятой кодировкой мы получаем, что:

'+' < '—', 'Z' < 'I', 'H' < 'N', 'Q' < '@', 'Ю' < 'Б'

При алгебраическом сравнении, если операнды имеют разные типы, то включаются преобразования $H \rightarrow A$ или $T \rightarrow A$. Кроме того, операнды приводятся к общим атрибутам аналогично операндам арифметических операций, то есть $R \rightarrow C$, $F \rightarrow E$, $D \rightarrow B$. Для комплексных операндов допустимы лишь операции $=$ и \neg .

Примеры истинных выражений:

4.6 > 100.1B	92.6E — 2 < .927
'2.6' > 1.91	110B > '100'B
3 — 2I \neg = 3 + 2I	'1100'B = '11'B
'3 — 2I' > '3 + 2I'	'KONEC' > 'КОНЕЦ'

Логические операции. Как уже упоминалось в 1-й главе, в PL/1 имеется лишь 3 логические операции: отрицание (\neg), И (&), ИЛИ (!).

Логические операции в PL/1 являются обобщением алгольных: их операндами могут быть не только элементарные логические значения '1'B и '0'B, но и их совокупности — битовые строки. Если заданы операнды другого типа, то они приводятся к битовому ($A \rightarrow T$, $H \rightarrow T$). Если длины операндов не совпадают, то более короткая строка дополняется справа нулями. Результат логической операции, определяемый побитно, имеет атрибут BIT. Если один из операндов VARYING, то и результат имеет атрибут VARYING.

Пример.

```
DECLARE (D, A) BIT (1) INIT ('0'B),
        C BIT (8) INIT ('11000'B) VAR,
        (F, B) FIXED (3, 2) INIT (5.25);
```

```
D =  $\neg$  A ! B > F & C;
```

```
/*  $\neg$  A = '1'B, B > F = '0'B,
```

```
B > F & C = '0'B & '11000'B = '00000'B;
```

```
D = '1' B ! '00000' B = '10000'B = '1'B */
```


Замечание. Если значением логического (битового) выражения, обозначаемого ранее через **eG** (теперь правильное было бы **eGT** или просто **eT**), стоящего после ключевого слова **IF** в условном операторе (или после **WHILE** в групповом операторе — см. 2.3.4), является битовая строка с длиной больше единицы, то битовой строке в этом случае соответствует '1'В (истина), если в ней есть хотя бы одна единица (например, '0101'В или '100'В); в противном случае ей соответствует '0'В (ложь).

Операция сцепления. Операция сцепления (конкатенация) изображается в PL/I ЕС двумя восклицательными знаками—!! (эталонное изображение—||). Операция сцепления производится над строками—она присоединяет справа к первому операнду второй и образует новую строку. Длина новой строки равна сумме длин операндов. Если один из операндов **VARYING**, то и результат будет иметь атрибут **VARYING**. Если операнды битовые, то и результат—битовый; в противном случае операнды преобразуются к символьной строке ($T \rightarrow H, A \rightarrow H$), и результат—символьный.

Пример.

```
DECLARE (S, RES) CHAR (10) VARYING,
        PR CHAR (2) INIT ('HET');
```

```
S='ХОРОШО'; RES=PR!!S;
```

```
/* PR='HE', RES='HEХОРОШО' */
```

2.2.3. Операции над массивами.

В PL/I можно производить арифметические и другие операции целиком над массивами. Для этого в качестве операнда операции нужно задать идентификатор массива или его сечение. Если второй операнд также идентификатор массива (или сечение), то операция выполняется над соответствующими элементами двух массивов; предполагается, что массивы (сечения) имеют одинаковые атрибуты размерности. Если второй операнд скалярная («элементная») величина (константа, простая переменная или переменная с индексом), то операция производится над скаляром и всеми элементами массива. Результатом операции в обоих случаях является массив, который, например, оператором присваивания может быть присвоен одному или нескольким массивам (см. 2.3.2). При выполнении операций над массивами могут осуществляться различные преобразования данных, изложенные ранее.

Пример.

```
DECLARE (A, B) (1:2, 1:3) INIT (1, 2, 3, 4, 5, 6);
```

```
A=B+1+A; /* A=3, 5, 7, 9, 11, 13 */
```

2.2.4. Формы.

В зависимости от вида переменных, входящих в выражения, последние называются скалярными или массивными; признаком

массивного выражения является вхождение в него хотя бы одной массивной переменной (или массивной функции).

Итак:

$$e \sim \{eS \mid eM\}$$

$$eS \sim \Phi \{vSO \mid fSO \mid cO \mid \omega O\}$$

$$eM \sim \Phi \{vMO \mid \bar{f}MO \mid eS\}$$

Здесь:

fSO — функция, принимающая скалярные проблемные значения;

$\bar{f}MO$ — массивная функция, то есть функция, значением которой является массив проблемных значений; массивными функциями могут быть только встроенные функции (\bar{f}) (см. 2.2.5);

ωO — совокупность ограничителей, которые могут фигурировать в выражении.

Можно дать и более строгие формы для выражений:

$$eS \sim \left\{ vSO \mid fSO \mid cO \mid \left\{ \pm \right\} eS \mid eS \otimes eS \mid (eS) \right\}$$

$$eM \sim \left\{ vMO \mid \bar{f}MO \mid \left\{ \pm \right\} eM \mid eM \otimes eM \mid eM \otimes eS \mid eS \otimes eM \mid (eM) \right\}$$

где \otimes обозначает любую операцию кроме \neg ; формы имеют рекурсивный характер.

З а м е ч а н и е. Так как арифметические и строчные выражения составляют единый класс, то в дальнейшем под eA и eG , используемыми и в 1-й главе, понимаются выражения, приводимые перед их использованием к арифметическим или строчным значениям. Точно так же и eAI или просто eI обозначают выражения, значения которых приводятся к целым, а eGT или eT к битово-строчным.

Понятие меточного выражения ($eL \sim \{1 \mid vL\}$) в дальнейшем использоваться не будет.

2.2.5. Встроенные функции.

В выражениях для выполнения действий над проблемными данными могут быть использованы *встроенные* — всегда доступные, стандартные — функции. Встроенные функции разделяются на следующие группы: математические, арифметические, строчные, функции для действий над массивами и разные функции.

Ниже приводится краткий обзор, имеющихся в PL/1 встроенных функций и даются простейшие примеры их использования *). Если читателя интересуют какие-либо из встроенных функций, то он сможет более детально познакомиться с ними, обратившись

*) Ради простоты, значения функций в примерах иногда, там где это не существенно, даются с уменьшенной точностью и в десятичной системе вместо двоичной.

в 4.1.1÷4.1.5, где функции даются по алфавиту внутри соответствующих групп.

Встроенные функции, в отличие от невстроенных, могут принимать в качестве своего значения целый массив значений (массивные функции) и определять своим вхождением массивные выражения.

Только идентификаторы математических функций могут являться аргументами при вызове процедур (функций).

Математические функции. С тригонометрическими функциями можно работать как в радианах (SIN, COS, TAN, ATAN), так и в градусах (SIND, COSD, TAND, ATAND); имеются гиперболические функции (SINH, COSH, TANH, ATANH). Помимо вычисления натурального логарифма (LOG) можно вычислять и логарифмы по основаниям 10 и 2 (LOG10, LOG2).

Аргументы математических функций и их значения приводятся к плавающему масштабу, причем точность значения функции соответствует точности аргумента.

$$\begin{aligned} \text{SIND } (45) &= 7.0\text{E}-1, & \text{SIND } (0045) &= 7.071\text{E}-1, \\ \text{LOG10 } (5) &= 6\text{E}-1, & \text{LOG10 } (50.000) &= 1.6990\text{E}0. \end{aligned}$$

Ряд функций определен и для комплексных аргументов.

Арифметические функции. Как правило, атрибуты значения функций зависят от атрибутов их основных аргументов. Среди встроенных функций имеются функции для осуществления преобразования к заданному масштабу (FIXED, FLOAT); основанию (DECIMAL, BINARY), моде (REAL, COMPLEX), точности (PRECISION); многие преобразования осуществляются к требуемой точности, указываемой в аргументах.

$$\begin{aligned} \text{BINARY } (11.5, 8, 2) &= 001011.10\text{B}, \\ \text{PRECISION } (56.78, 4, 1) &= 056.7 \\ \text{COMPLEX } (-3, 9\text{E}-1) &= -3\text{E}0 + 9\text{E}-1\text{I}, \\ \text{FIXED } (1.68\text{E}1, 5, 2) &= 016.80 \end{aligned}$$

Функция FIXED может быть использована для приведения значений с плавающим масштабом к целому типу. Но при этом следует помнить, что FIXED просто отсекает дробную часть:

$$\text{FIXED } (-1.68\text{E}1) = -16 \text{ (точнее } -00016).$$

Для получения ближайшего целого значения можно предложить следующую формулу:

$$\text{FIXED } (x + \text{SIGN } (x) * 0.5).$$

Но по сравнению с *entier* ($x + 0.5$) в алголе эта формула будет давать расхождение (на единицу) для $x = -0.5, -1.5, -2.5$ и т. д. Кроме того, см. ниже особенности функции SIGN.

Функции ADD, MULTIPLY и DEVIDE служат для получения результата арифметических операций (+, *, /) с требуемой точностью:

ADD (12.34, 89.66, 8, 4) = 0102.0000,

MULTIPLY (1.26, 3E0, 2) = 3.7E0

DEVIDE (1E3, 3, 8) = 3.333333E2

Функции FLOOR, CEIL, TRUNC служат для получения одного из «соседних» с аргументом целых значений (но для аргумента с атрибутом FLOAT и результат — FLOAT):

FLOOR (3.8) = 3, CEIL (3.8) = 4, TRUNC (3.8) = 3;

FLOOR (−3.8) = −4, CEIL (−3.8) = −3, TRUNC (−3.8) = −3.

Но FLOOR (3.8E0) = 3.0E0, а TRUNC (3.8765E0) = 3.0000E0; выше точнее было бы написать 03, 04 и т. п.

Функции SIGN и ABS знакомы нам по алголу, но первая имеет двоичное значение, а последняя определена и для комплексных величин:

SIGN (−4.8) = −1B,

ABS (3−4I) = 5.

По функциям CONJG и IMAG можно получить сопряженное значение или выделить мнимую часть:

CONJG (−3−4I) = −3+4I, IMAG (3−4I) = −4

Для округления значений используется функция ROUND:

ROUND (56.78, 1) = 56.80, ROUND (56.78, −1) = 60.00

Положительный остаток от деления двух чисел можно найти по функции MOD:

MOD (19, 5) = 4, но MOD (−19, 5) = 1

Минимум и максимум находится по функциям MIN и MAX:

MAX (MIN (1, 2), MIN (−3, +4), MIN (5, −6)) = 1

Строчные функции. Для образования строк из требуемых строчных компонент предназначены функции REPEAT, STRING, LOW, HIGH. Для ST = 'ЛЯ—' получаем:

REPEAT (ST, 3) = 'ЛЯ—ЛЯ—ЛЯ—ЛЯ—'

Компонентами для STRING являются все элементы заданного массива (или структуры):

при DECLARE UM (2, 2) BIT (2) INIT

('01'B, '11'B, '00'B, '10'B);

функция STRING (UM) получит значение '01110010'B

Компонентами для LOW и HIGH служат «минимальный» и «максимальный» символы ЭВМ (в шестнадцатеричной кодировке: 00 и FF).

Функции BIT и CHAR служат для преобразования строк к указанному виду и к требуемой длине:

BIT ('110+48', 3) = '110'B

Функция UNSPEC преобразует аргумент к битовой строке, содержащей его двоичное представление в ЭВМ:

UNSPEC ('IBM') = '110010011100001011010100'B

I B M

UNSPEC (-125) = '0001601001011101'B

$\underbrace{\hspace{1.5cm}}_1 \underbrace{\hspace{1.5cm}}_2 \underbrace{\hspace{1.5cm}}_5 \underbrace{\hspace{1.5cm}}_{-}$

Функция LENGTH определяет текущую длину строки (UM см. выше):

LENGTH (STRING(UM)) = 8

С помощью BOOL можно выполнить любую из возможных 16 логических операций (в примерах — \supset и \equiv), определяемую третьим аргументом:

BOOL ('10101'B, '01100'B, '1101'B) = '01110'B

BOOL ('10101'B, '01100'B, '1001'B) = '00110'B

Функция TRANSLATE используется для замены одних символов в строке на другие. Например:

TRANSLATE ('ПАПА', 'М', 'П') = 'МАМА'

А для WHOD = '+, ., Q;' и WIHOD = '., P.'

TRANSLATE ('+12,345.67Q;', 'WIHOD, WHOD) =
= '.,12.345,67P.'

Здесь строки 'М' и 'П', WHOD и WIHOD задают *таблицу преобразования*.

Для определения вхождения строки или символов, ее составляющих, в другую строку служат соответственно функции INDEX и VERIFY.

Для STR = 'NO,ONE,ON,OT' получаем:

INDEX (STR, 'ON') = 4, а INDEX (STR, 'NOT') = 0

— строка 'ON' входит в STR, начиная с 4-го символа, а строка 'NOT' в STR не содержится;

VERIFY ('NOT', STR) = 0, а VERIFY ('NEIN', STR) = 3

— все символы строки 'NOT' входят в STR, но 3-й символ строки 'NEIN' не найден в STR.

Функция SUBSTR выделяет требуемую часть строки:

SUBSTR (STR, 5, 2) = 'NE'

SUBSTR (STR, 8) = 'ON,OT'

Функции для массивов. С помощью функций LBOUND, HBOUND, DIM можно узнать текущие значения граничных пар или «протяженность» одной из размерностей указанного массива. Например, для

DECLARE AR (0:5, 10) FIXED;

получим:

LBOUND (AR, 2) = 1, HBOUND (AR, 2) = 10,

DIM (AR, 1) = 6, HBOUND (AR, 1) = 5.

Функции SUM, PROD, ANY, ALL вычисляют сумму, произведение, логическую сумму или логическое произведение всех элементов массива. Например, для UM, описанного выше:

ALL (UM (1, *)) = '10'B

SUM (UM) = 6E0

Функция POLY вычисляет значение полинома с заданными коэффициентами. При

```
DECLARE X1 REAL (8) INIT (0.2),  
        A1 (3) REAL (4) INIT (2, 3, 1);  
/* ПОЛИНОМ 2 + 3 * X + 1 * X ** 2 */
```

получаем POLY (A1, X1) = 2.64E0

Разные функции. Из других функций отметим функции DATE и TIME, которые позволяют узнать дату и текущее время дня.

2.2.6. Псевдопеременные.

Указатели ряда встроенных функций могут фигурировать в качестве переменных левой части оператора присваивания (а также в качестве параметра заголовка цикла). Такие функции в этом случае называются *псевдопеременными*, так как им [как бы присваиваются значения правой части. Фактически же «принимают» значения переменные, являющиеся аргументами таких функций, а сами функции лишь указывают на «принимающие поля» в переменной-аргументе или на способ приема значения. Аргументами псевдопеременных могут быть и переменные-массивы, но не могут быть псевдопеременные.

Псевдопеременные описаны вместе с соответствующими встроенными функциями (см. 4.1). Ниже приводятся примеры использования псевдопеременных.

Псевдопеременная COMPLEX позволяет разделить комплексное значение на составные части, которые трактуются как действительные величины. Псевдопеременные IMAG, REAL служат для изменения мнимой или действительной частей значения комплексной переменной. Например, если $VC = 1 + 2I$, то

по операторам:	получаем:
COMPLEX (VA, VB) = VC;	VA = 1, VB = 2,
IMAG (VC) = -3;	VC = 1 - 3I,
REAL (VC) = -8 + 4I;	VC = -8 + 2I

По SUBSTR можно заменить указанную часть строки. Например, для

```
STR = 'NO, ONE, ON, OT'
```

по оператору

```
SUBSTR (STR, 5, 2) = 'WN';
```

получаем

```
STR = 'NO, OWN, ON, OT';
```

но для

```
SUBSTR (STR, 5) = 'WN';
```

получаем

```
STR = 'NO, OWN □□□□□□'
```

С помощью STRING можно всем компонентам массива (или структуры) присвоить значения компонент строки, «разобрав» для этого заданную строку на части, соответствующие по длине компонентам массива. Например, при

DECLARE UN (2, 2) BIT (2);

по оператору

STRING (UN) = '01110010'B;

получим

UN (1, 1) = '01'B, UN (1, 2) = '11'B, UN (2, 1) = '00'B,

UN (2, 2) = '10'B.

Псевдопеременная UNSPEC позволяет присвоить арифметическим или строчным переменным значение, заданное в двоичном виде конкретного представления данных на ЭВМ (с помощью битовой строки). При

DECLARE H1 CHAR (3), PAK FIXED (1),
I1 BINARY FIXED (5, 2);

по операторам:

получим:

UNSPEC (H1) = '111011001011110011101011'B;	H1 = 'ЖДУ'
UNSPEC (PAK) = '10001101'B;	PAK = -8
UNSPEC (I1) = '0000000000001001'B;	I1 = 010.01B

Упражнения.

2.2.1. Для A = '0100'B, B = '1001'B, C = '0011'B определить значения следующих выражений:

а) A||C||'0010'B;

б) C||BOOL (A, B, '0001'B);

2.2.2. Определить значения следующих выражений:

а) 8.27 > '1000.01'; б) 8.27 > '1000.01B';

в) 1000B > '1000.01'B; г) 1000B > '10';

д) '108' > '1000'; е) '1000'B > '10'B;

ж) ''B = '┐'; з) '1 + 1' = 2.

2.2.3. С помощью сечений и функции SUM вычислить:

а) $S = \sum_{i=1}^n a_i b_i$;

б) $c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}; \quad i, j = 1, \dots, n.$

2.2.4. Пусть строка CH содержит число с плавающей точкой; заменить у порядка знак минус, если он есть, на +.

2.2.5. Переставить местами первые и последние элементы строки P (1/2 элементов строки, где l — длина строки).

2.2.6. Присвоить битовой переменной T значение, соответствующее истине, если:

а) каждый элемент массива А не меньше соответствующего элемента массива В;

б) в массиве А имеется элемент больший 10^6 .

2.2.7. Составить процедуру-функцию, которая получает логическое (булево) значение *ложь*, если среди соответствующих столбцов матриц А и В есть такие, что все элементы столбца матрицы А меньше соответствующих элементов столбца матрицы В; в противном случае функция получает значение *истина*.

2.3. Уточнение операторов

2.3.1. Терминология.

Операторы *) PL/1 часто называют по определяющим их ключевым словам: GO TO-оператор, CALL-оператор, IF-оператор и т. п. DO-оператором называется начало оператора цикла вплоть до первой точки с запятой—то, что по аналогии с алголом-60 можно назвать заголовком цикла. Конструкция DO; в составном операторе также называется DO-оператором. (Оператор цикла и составной оператор являются частным случаем группового оператора—см. ниже 2.3.4.) Конструкции BEGIN; и END; это—BEGIN-оператор и END-оператор. Под PROCEDURE-оператором понимается заголовок процедуры в прежней терминологии.

Описание процедуры в новой терминологии—это оператор, именуемый *процедурным* блоком (PROCEDURE-блок), а обычный блок называется *начальным* блоком (BEGIN-блок) или *простым* блоком. И другие описания по терминологии PL/1 являются оператором: DECLARE-оператором (оператор описания).

Блоки и групповой оператор («группа») в PL/1, строго говоря, являются не операторами, а последовательностью операторов:

групповой-оператор \sim s-DO s ... s-END

простой-блок \sim s-BEGIN s ... s-END

процедурный-блок \sim s-PROCEDURE s ... s-END

Ключевое слово через дефис идентифицирует соответствующий оператор.

В дальнейшем, наряду с новыми терминами используются и старые (алгольные) там, где это не может вызвать двусмысленности. Оператором будем называть как основной («простой») оператор PL/1, содержащий только одну точку с запятой, так и совокупности основных операторов (условный оператор, блок, группа). Оператор процедуры называется оператором вызова процедуры; под процедурой будем подразумевать процедурный блок. Наряду с термином «идентификатор» употребляется и «имя».

*) В документации—*утверждения*.

2.3.2. Оператор присваивания.

В списке левой части оператора присваивания («назначения») PL/1 в отличие от алгола могут быть переменные с разными атрибутами (например, с FIXED и FLOAT, REAL и CHARACTER, COMPLEX и BIT). Значение выражения правой части перед присваиванием приводится к атрибутам каждой переменной. Порядок вычисления индексных выражений в списке левой части и выполнения операций присваивания тот же, что и в алголе:

1) слева направо вычисляются индексные выражения у переменных левой части;

2) вычисляется выражение правой части;

3) значение правой части присваивается слева направо переменным левой части.

В левой части оператора могут фигурировать и идентификаторы массивов, а в правой его части в этом случае — выражения над массивами и (или) скалярами. Смешивание в левой части оператора массивных переменных и обычных скалярных не допускается. Все массивные переменные и в левой, и в правой частях оператора должны иметь одинаковые атрибуты размерности.

Пример.

```
DECLARE (A, B, C, D) (1:20) FLOAT (8);
```

```
B=0; C=B; D, A=C-B*3E-6;
```

Все элементы массива B получают значение нуль. Элементы массива C получают значения соответствующих элементов массива B. Массивам D и A присваиваются значения массивного выражения (см. 2.2.3).

Предполагается, что выполнение операций над массивами производится в порядке увеличения индексов по строкам. Пример:

```
DECLARE (A, B) (2, 3) FIXED INIT (5, 1, 8, 2, 0, 3),
```

```
F FIXED INIT (2);
```

```
B=A * F; /* B=10, 2, 16, 4, 0, 6 */
```

```
B=B * B (1, 2); /* B=20, 4, 64, 16, 0, 24 */
```

В левой части оператора могут находиться и псевдопеременные (\tilde{W}) (см. 2.2.6), а в правой — меточные переменные и метки (в том случае, если в левой части заданы меточные переменные — см. 2.1.4).

Ф о р м а.

$$s\text{-присваивания} \sim \left\{ \begin{array}{l} \{vS | \tilde{fWS}\}, \dots = \{eS | vSL | 1\}; \\ \{vM | \tilde{fWM}\}, \dots = \{vM | eS | vML | vSL | 1\}; \end{array} \right\}$$

2.3.3. Условный оператор.

В PL/1 в условном операторе после THEN может стоять любой оператор, в том числе условный оператор или оператор цикла, что не допускается в условном операторе алгола (см. 1.3.5).

Например:

```
IF A < 0 THEN DO J=1 BY 1 TO N;  
IF C(J)=0 THEN R=R+1; END; ELSE R=-1;
```

или

```
IF M > 1 THEN IF N > 1 THEN  
IF L=0 THEN L=1; ELSE M=1; ELSE K=1;
```

Как видим, для случая оператора цикла никаких двусмысленностей, возможных для аналогичных примеров в алголе, не возникает, так как END; отмечает конец оператора цикла. Что же касается условного оператора после THEN, то здесь может потребоваться дополнительное соглашение для того, чтобы определить, к какому из THEN относится ELSE. В PL/1 принято, что ELSE соответствует тому THEN в операторе, которое будет первым слева от него, если пропустить THEN парные с другими ELSE и те THEN, которые находятся в операторе цикла, составном операторе или блоке. В нашем примере ELSE K=1 соответствует второму слева THEN. Для задания другого соответствия THEN и ELSE нужно было бы использовать составной оператор:

```
IF M > 1 THEN DO; IF N > 1 THEN  
IF L=0 THEN L=1; ELSE M=1; END; ELSE K=1;
```

Для выделения соответствующих THEN и ELSE их удобно располагать одно под другим или в одной и той же строчке. Например,

```
IF M > 1 THEN  
IF N > 1 THEN IF L=0 THEN L=1; ELSE M=1;  
ELSE K=1;
```

2.3.4. Групповой оператор.

Групповой оператор (группа) является обобщением составного оператора и оператора цикла (в терминах алгола).

Общая форма группового оператора имеет следующий вид:

$$s \sim \text{DO} \left\{ \begin{array}{l} [\text{WHILE } (eGT)] \\ vS = \left\{ eA_1 \left[\begin{array}{l} \text{BY } eA_2 [\text{TO } eA_3] \\ \text{TO } eA_3 [\text{BY } eA_2] \end{array} \right] [\text{WHILE } (eGT)] \right\}, \dots \end{array} \right\};$$

s...END;

Выражения везде предполагаются скалярными.

Составной оператор получается из верхней строки формы, когда в ней отсутствует элемент с WHILE. Если этот элемент присутствует, то он задает условие выполнения группы операторов, стоящей между точкой с запятой и END;. Группа операторов выполняется, «если» или «до тех пор пока» логическое (битово-строичное) выражение eGT имеет значение истина ('1'B).

Например, в

DO WHILE (X < M); X = X * 10; Y = Y + 1; END;
 операторы присваивания будут выполняться пока X < M; если X ≥ M уже перед выполнением группового оператора, то операторы присваивания не выполнятся ни разу.

Нижняя строка формы определяет усложненный оператор цикла, в котором могут быть переставлены местами или отсутствовать ключевые слова BY и TO (вместе с eA) или добавлено WHILE (с eGT).

Для того, чтобы разобраться в работе группового оператора, дадим эквивалент одного элемента списка цикла, называемого в документации «спецификацией», в виде последовательности основных операторов PL/1.

$$v_1 = eA_1; \left[v_2 = \left\{ \begin{matrix} 1 \\ eA_2 \end{matrix} \right\}; \right] [v_3 = eA_3;] < \text{BY или TO} > < \text{TO} >$$

v = v₁;

L: [IF v₂ > 0 & v > v₃ ! v₂ < 0 & v < v₃
 THEN GO TO NEXT;] < TO >

[IF eGT THEN; ELSE GO TO NEXT;] < WHILE >

s...

[v = v + v₂; GO TO L;] < BY или TO >

NEXT:

В комментариях указывается, в каких случаях оператор в квадратных скобках будет иметь место в последовательности операторов.

Напомним для сравнения аналогичные последовательности операторов для элементов списка цикла алгола (в виде операторов PL/1).

типа «прогрессии»

типа «пересчета»

v = eA₁;

L2: v = eA₁;

L1: IF eA₂ > 0 & v > eA₂ ! eA₂ < 0 &

IF eGT THEN;

v < eA₃

THEN GO TO NEXT;

ELSE GO TO
 NEXT;

s

v = v + eA₂;

s

GO TO L2;

GO TO L1;

NEXT:

NEXT:

Обратимся теперь к операторам, реализующим групповой оператор. Присваивания v₂ и v₃ (см. первую строку операторов в эквиваленте для PL/1) служат для того, чтобы значения шага и конечного значения (см. eA₂ и eA₃) не менялись в ходе выполнения оператора цикла. (В эталонном алголе они могут меняться,

хотя в большинстве трансляторов добавляются такие же операторы, что и в PL/1, для предотвращения такого изменения). Если элемент с BY отсутствует, то вместо eA_2 предполагается единица.

Окончание цикла в PL/1 (см. оператор с меткой L) имеет ту особенность, что в случае $eA_2 = 0$ и $eA_1 > eA_3$ цикл не будет выполняться ни разу. При отсутствии TO проверки на окончание цикла не будет; выход из цикла должен произойти по WHILE (см. ниже) или по одному из операторов тела цикла (s...).

Если присутствует элемент с WHILE, то производится дополнительная проверка на окончание цикла и выход из него, когда eGT имеет значение ложь.

Если в операторе отсутствует и BY, и TO (см. последние операторы в эквиваленте), то мы получаем цикл типа «арифметического выражения» в алголе, при котором повторения цикла не происходит (WHILE может воспрепятствовать и единственному выполнению тела цикла).

По сравнению с элементом списка цикла типа пересчета в алголе, в PL/1 элемент с WHILE не задает многократность выполнения тела цикла, а служит лишь для сокращения заданного (посредством TO) количества повторений цикла. Изменение параметра цикла при этом производится не произвольным способом, как в алголе, а лишь на заданное приращение (по BY).

В отличие от алгола, в PL/1 значение параметра цикла и после выхода из цикла по исчерпанию определено и равно последнему значению, при котором произошло исчерпание.

Напомним еще раз, что вычисление значений выражений eA_1 , eA_2 , eA_3 и присваивание их переменным v_1 , v_2 , v_3 производится в начале выполнения каждого элемента списка цикла (но значения индексов для индексированного параметра цикла не перевычисляются).

Примеры. Ниже даются примеры DO-операторов (заголовков оператора цикла) с комментариями, поясняющими их работу.

- 1) DO K=1 TO NN; с шагом 1;
- 2) DO J=1 BY 2 WHILE (L \neg K); окончание по WHILE;
- 3) DO V=LOG2 (X) WHILE (X \geq 0); однократное выполнение при условии $X \geq 0$;
- 4) DO V=LOG2(X) TO X**2 WHILE (X > 0); многократное выполнение с шагом, равным единице, и с условием $X > 0$;
- 5) DO WHILE (Z \neg 5+5I); цикл без параметра цикла, многократное выполнение.¹

З а м е ч а н и е. В качестве параметра цикла могут использоваться и комплексные переменные (но условием окончания цикла может быть только WHILE), и даже меточные, в заголовке опе-

ратора цикла следующего вида:

$$\text{DO vLS} = \left\{ \begin{array}{l} \text{vLS} \\ 1 \end{array} \right\} [\text{WHILE (eGTS)}];$$

2.3.5. END-оператор.

Оператор END имеет следующую форму:

$s \sim \text{END } [\tilde{I}]; \tilde{I} \sim \{iL\text{-для-DO} \mid iL\text{-для-BEGIN} \mid iP\}$

Если в операторе присутствует метка (группы или блока) или имя процедуры, то он «закрывает» тот ближайший групповой оператор или блок (простой или процедурный), который помечен данной меткой. При этом закрываются блоки и группы, которые содержатся внутри помеченного оператора, и которые не были еще закрыты своими END-операторами. Таким образом, END-оператор с меткой заменяет несколько подряд идущих END-операторов (см. сноску к процедуре MAXM в 1.4.4).

Если END-оператор, содержащий метку, помечен, то считается, что эта метка (с двоеточием) относится к самому последнему из группы END-операторов, заменяющих исходный END-оператор. Например:

A: END B;

может заменить

END; END; A:END; но не A:END; END; END;

Ограничение. Массивная метка, например, вида B(2,3) в END-операторе не может быть использована.

Упражнения.

2.3.1. Какие значения получают после выполнения следующих операторов встречающиеся в них переменные:

а) DECLARE A(5) FIXED BINARY (4,1), S(5) CHAR (6),

K COMPLEX (4), V(5) BIT (2), T (5,8);

K, A(3), L=3.5—6.8I; S, T=4.3E07;

V, S, T(*, 4)='101'B; A(5), V=0;

б) X, Y, Z=1; IF X < 1 THEN IF Y >= 1 THEN Y=2;
ELSE IF Z=1 THEN X=2;

в) DCL X INIT ('11011'B) BIT(4), Y BIT(5) INIT ('00101'B);
IF Y=X THEN X=Y=X; ELSE Y=X=X!Y;

2.3.2. Какую кратность выполнения задают следующие операторы:

а) DO W=0, W+2 WHILE (W < 10);

б) DO NORA=2.6 BY 0.1 TO 3.5;

в) DCL FCPLX FIXED; DO F=4+5I BY 1I TO 4+10I;

г) DCL I FIXED (1); DO I=0 TO 9;

д) N=27; DO L=N/4 BY -2 WHILE (L > 0); N=N-2; END;

е) N=27; DO L=N/4 BY -2 TO 5 WHILE (L > 0); N=N-2;
END;

ж) N=27; DO WHILE (L > 0); N=N-2; END;

2.3.3. Оформить в виде блока вычисление по следующей формуле:

$$y = e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \dots + \frac{x^n}{n!} + \dots$$

Суммирование производить до тех пор, пока $\left| \frac{x^n}{n!} \right| \geq \varepsilon = 10^{-8}$.

2.4. Уточнение описаний и блоков

2.4.1. Оператор описаний.

Как уже упоминалось в 2.3.1, описания (а также спецификации параметров) в PL/1 называются операторами. Процедура в PL/1 является не описанием, а блоком, и поэтому к ней дальнейшее не относится. Описания в PL/1 имеют единую форму и могут стоять в любом месте блока — простого или процедурного. Но они не могут находиться в IF-операторе или в групповом операторе циклического типа (с WHILE или с параметром цикла). Сфера действия оператора описания — самый внутренний окаймляющий его блок, при входе в который оператор описания и «выполняется». (Можно, таким образом, считать, что оператор описания поднимается транслятором в начало соответствующего блока, где и происходит его выполнение каждый раз при входе в блок).

Форму для оператора описания можно представить в следующем виде:

$d \sim s\text{-описания} \sim \text{DECLARE } \{i \text{ } t... | (i, ..) \text{ } t... | ((i, ..) \text{ } t...) \text{ } t... \langle \text{и. т. п.} \rangle\}, ...;$

В качестве атрибутов t , фигурирующих в форме, могут использоваться любые атрибуты, упоминавшиеся ранее, или те, которые будут введены позже. Вторая и третья альтернативы в форме указывают на то, что общие атрибуты могут быть вынесены за скобки. Такое вынесение («факторизация») может быть проделано многократно, поэтому с использованием рекурсии форма может быть обобщена следующим образом:

$d \sim s\text{-описания} \sim \text{DECLARE } \Delta;$

$\Delta \sim \{\{i | (i, ..) | (\Delta)\} [t...]\}, ..$

Например:

DECLARE SUM FIXED BINARY (15,5), (A, B, C) (8, 8),
(U(0: 9), ((P, Q, R) REAL, (S,T) CPLX) FIXED)
DECIMAL (7);

Оператор описания может быть помечен, но при этом метка играет только роль комментария (переход на нее запрещается). Квадратные скобки у списка атрибутов говорят о том, что возможны описания, например, и такого вида:

TUT: DECLARE N1, (R2), ((P1, P2) (10)) BIT (10);

Напомним еще раз, что атрибут размерности должен стоять сразу вслед за идентификатором или списком идентификаторов, заключенным в скобки (идентификаторы в списке могут быть и вместе с относящимися к ним различными атрибутами); а атрибут точности (длины), наоборот, не может стоять сразу за скобками. Например, следующее описание было бы неверным:

```
DECLARE (X(4) FLOAT, (Y, Z) FIXED) (10) COMPLEX;
```

Но допустимо

```
DECLARE (X FLOAT, (Y, Z) FIXED) (10) COMPLEX;
```

где атрибут (10), считается атрибутом размерности, относящимся к X, Y и Z.

Каждая переменная может быть описана в блоке только один раз, то есть все необходимые атрибуты для переменной должны быть указаны при ее первом и единственном описании.

З а м е ч а н и е. Хотя в PL/1 скалярные переменные с атрибутами REAL FIXED BINARY (15,0) или REAL FLOAT DECIMAL (6) описывать необязательно (см. 2.1.1), а оператор описания можно располагать в любом месте блока, широко пользоваться этими возможностями, отходя от правил алгола, не рекомендуется, так как это будет затруднять проверку больших программ при их отладке.

2.4.2. Блоки.

Назначение блока в PL/1 то же, что и в алголе. Простой и процедурный блоки PL/1 различаются, в основном, лишь способом обращения к ним: вход, выход, передача аргументов.

Вход и выход из блока. Вход в простой блок, как и в алголе, осуществляется только через его начало (после окончания выполнения предыдущего оператора или по оператору GO TO). Вход в процедурный блок производится по оператору CALL или по указателю функции. Возможен возврат в середину блока (как простого, так и процедурного) из вызванной процедуры по метке, переданной процедуре в качестве фактического параметра.

Окончание работы блоков помимо операторов END и GO TO может быть вызвано и оператором STOP;, который служит для прекращения выполнения всей программы. Для процедурных блоков прекращение их работы происходит и по операторам

RETURN; или RETURN (eS);

— соответственно для процедур и процедур-функций; если эти операторы входят в простой блок, принадлежащий процедурному, то его выполнение тем самым также заканчивается.

П р и м е р.

A: BEGIN;

P: PROCEDURE (K, W); DECLARE K FIXED BINARY,
W LABEL;

```

IF K < 0 THEN STOP; /*ОКОНЧАНИЕ
                                ВЫЧИСЛЕНИЙ */
. . . . .
IF K < 5 THEN RETURN; /* ВОЗВРАТ НА END B; */
ELSE GO TO W; /* ВОЗВРАТ НА _МЕТКУ L*/
END P;
. . . . .
B: BEGIN;
. . . . .
  L: I = I + 1;
    CALL P (I, L);
  END B;
END A;

```

Атрибуты сферы и памяти. Величины, описанные в блоке, недоступны из внешнего блока: они определяются («размещаются» в памяти) при входе в блок и автоматически уничтожаются («освобождаются») при выходе из блока. В PL/I таким величинам присваиваются атрибуты AUTOMATIC (автоматический) и INTERNAL (внутренний). Первый атрибут указывает на автоматическое размещение величины в памяти, второй — на локализацию величины внутри блока.

Противоположным по смыслу для INTERNAL является атрибут EXTERNAL (внешний), определяющий величины, которые могут быть известны в различных блоках программы. Для этого те же величины (с совпадающими именами и атрибутами) в других блоках также должны быть описаны (явно или контекстуально — см. ниже) с EXTERNAL. Атрибуты EXTERNAL и INTERNAL называются *атрибутами области действия* или *атрибутами сферы*; допускаются сокращения EXT, INT.

Таким образом, одинаковые имена, описанные в разных блоках с атрибутами INTERNAL, имеют различные непересекающиеся сферы; сфера для одинаковых имен, описанных в разных блоках с атрибутом EXTERNAL состоит из всех этих блоков, то есть является суммой внутренних сфер для этих имен. Напомним, что внешние имена должны содержать не более 7 символов; кроме того, они не могут начинаться с IHE.

Атрибут AUTOMATIC (AUTO) относится к атрибутам *класса памяти*. Другими атрибутами этой группы могут быть STATIC, CONTROLLED, BASED. Величины, описанные со STATIC (статический), размещаются в памяти 1 раз до начала выполнения программы и не уничтожаются при выходе из блока. Поэтому в качестве границ статических массивов или длин статических строк (а также в повторителях атрибута INITIAL) можно при

описании задавать только целые десятичные числа (**nDI**), а не выражения. Если **STATIC** дополняется **INTERNAL**, то такие величины локализуются в соответствующем блоке и являются аналогичными собственным величинам (**own**) в алголе.

Величины с атрибутом **EXTERNAL** по умолчанию предполагаются с атрибутом **STATIC** (**AUTOMATIC** не может быть задан). Для имен с атрибутом **STATIC** по умолчанию предполагается **INTERNAL**, если только это не имя внешней процедуры (см. ниже 2.4.3).

Атрибуты **CONTROLLED** и **BASED** приписываются величинам, размещение которых в памяти и ликвидация осуществляется самим программистом по специальным операторам; такие величины здесь не рассматриваются.

Пример.

```
OC: PROC; DECLARE Z STATIC EXTERNAL FLOAT INIT (0),  
      U STATIC INITIAL (0);  
      Z=Z+1; U=U+1; PUT LIST (Z, U);  
      CALL BC; PUT LIST (Z, U);  
END OC;  
BC: PROCEDURE; DCL Z EXT DEC INIT (0), U INIT (0);  
      Z=Z+1; U=U+1, PUT LIST (Z, U);  
END BC;
```

При первом обращении к процедуре **OC** печатаются следующие пары чисел: 1 и 1, 2 и 1, 2 и 1; при втором обращении — 3 и 2, 4 и 1, 4 и 2; при третьем — 5 и 3, 6 и 1, 6 и 3 и т. д. Это объясняется тем, что переменная **Z** имеет одну и ту же (внешнюю) сферу и значение **Z** наращивается в обеих процедурах, а переменные **U** имеют по умолчанию внутренние сферы, и значение одной из них теряется при выходе из процедуры **BC**, но значение другой сохраняется в процедуре **OC** после выхода из нее, так как там она имеет атрибут **STATIC**.

Напомним, что внешняя переменная **Z** обязана всюду иметь одинаковые атрибуты (в примере **REAL FLOAT DECIMAL (6)**), включая и атрибут **INITIAL** (в одной из наших процедур он мог бы и отсутствовать, поскольку инициализация статических переменных осуществляется только один раз перед началом выполнения программы).

Контекстуальное и неявное описания. Помимо явного описания (по **DECLARE**) в **PL/I** различается еще и *контекстуальное* описание, под которым подразумевается определение вида неописанной величины по тому контексту, в котором встречается эта величина. Например, идентификаторы, стоящие в левой части оператора присваивания, считаются именами скалярных переменных, идентификаторы после ключевого слова **CALL** или с последующими

скобками в выражениях — именами процедур (функций), а не идентификаторами массивов! Переменным и функциям, описанным контекстуально, атрибуты приписываются по правилам умолчания. Кроме того, имя процедуры (функции) считается EXTERNAL STATIC, имя переменной — INTERNAL AUTOMATIC.

Идентификаторы, не описанные ни явно, ни контекстуально, считаются *неявно* описанными как переменные (а не указатели функций без списка аргументов), с атрибутами по умолчанию. Напомним, что метки (обычные, не массивные), описываются явно, своим появлением с двоеточием в качестве «префикса» какого-либо оператора; метки блоков (для процедурного блока — имя процедуры) считаются описанными явно в охватывающем блоке.

Область действия неявного или контекстуального описания — самый внешний блок, содержащий такое описание (исключая, разумеется, блоки, где идентификатор описан явно). Можно заметить, что эта область действия шире области действия для соответствующего (стоящего на том же месте) явного описания.

Пример. Ниже приведены условный пример и схема, поясняющие области действия идентификаторов, описанных различными способами.

A: PROCEDURE;	A	B	B'	C	C'	X	Y	Y'	Z	Z'	R	L	L'
DCL (X, Y) EXT;													
L: X = Z - 1;													
B: BEGIN;													
DCL (Y, Z, B);													
C: PROCEDURE;													
DCL (X, Y) EXT, R;													
L: Y = X; B = Z + R;													
END C;													
Z = Y; CALL C;													
END B;													
Z = C (X);													
END A;													

Имя процедуры A является глобальным по отношению к телу этой же процедуры. Метка блока B неизвестна внутри него, так как внутри блока имя B является именем переменной, объявленной в этом блоке явно. Имя процедуры C известно только в блоке B. Имя C в блоке A (см. C') объявляется контекстуально как имя процедуры-функции, и ему присваиваются по умолчанию атрибуты EXTERNAL и STATIC; то есть должна существовать еще и внешняя процедура (см. ниже 2.4.3) с именем C. Внешнее имя Y известно везде, кроме той части блока B, которая не входит в процедурный блок C. Имя Z объявлено неявно в блоке A и явно в блоке B. Имя метки L объявляется явно и в блоке A, и в блоке C.

2.4.3. Процедурный блок.

Процедурный блок в PL/1 является невыполняемым оператором: при последовательном выполнении операторов он обходится. Как и другие операторы, он может стоять в любом месте программы (кроме тех мест, где не может стоять и оператор описания DECLARE: внутри IF-оператора и в теле группового оператора циклического типа). Выполнение процедурного блока производится только по CALL-оператору или по указателю функции (см. выше 2.4.2).

Процедура (процедурный блок), которая не входит ни в какой другой блок, называется *внешней* процедурой; ее имя получает атрибуты EXTERNAL и STATIC. Везде, где раньше мы говорили о программе PL/1, следовало бы говорить о внешней процедуре: программа в PL/1 может состоять из нескольких внешних процедур (подробнее об этом см. в 2.4.5).

Общая форма для процедурного блока (описания процедуры) имеет следующий вид:

```
d-процедуры ~ {iP:}...PROC [EDURE] [(p, ..)] [RETURNS (t...)]  
[RECURSIVE] [OPTIONS (MAIN)];  
[s | d]...END [i];
```

Процедура может иметь несколько имен и обращаться к ней можно по любому из них; имя внешней процедуры, как и любое внешнее имя, не может содержать более 7 символов (то есть, здесь $iP \sim \{iP | \tilde{iP}\}$ — ср. с 1.5.1).

Опция MAIN указывает на то, что процедура является главной: то есть, именно с нее должно начаться выполнение всей программы.

Опция RETURNS даже для заголовка процедуры-функции является необязательной: атрибуты возвращаемого значения при этом определяются транслятором по умолчанию для имени входа.

Пример.

```
ABS: MOD: PROCEDURE (X, Y);  
      RETURN (SQRT (X * X + Y * Y)); END;
```

При обращении по имени ABS будет возвращаться значение с атрибутами FLOAT DECIMAL (6), а — по имени MOD — с FIXED BINARY (15,0). Атрибуты RETURNS при описании входов ABS и MOD или должны отсутствовать, или содержать в скобках те же атрибуты, что указаны выше.

Дополнительный вход. Помимо основного входа в начало процедуры (по одному из ее имен) в PL/1, в отличие от алгола, допускается вход в середину процедуры по именам ее дополнительных входов, определяемых с помощью невыполняемого (описывающего) оператора входа — ENTRY-оператора, имеющего вид:

```
d ~ s-входа ~ {iP:}...ENTRY [(p, ..)] [RETURNS (t...)];
```

Через IP здесь и в дальнейшем будем обозначать имя как основного, так и дополнительного входов. Параметры ENTRY-оператора (их количество, идентификаторы, атрибуты) могут не совпадать с параметрами в PROCEDURE-операторе. Но при выполнении процедуры в ее операторах не должны встречаться параметры, не определенные для данного входа; операторы ENTRY, встречающиеся по ходу выполнения процедуры, эквивалентны пустому оператору. Опция RECURSIVE в PROCEDURE-операторе относится и к дополнительным входам.

Оператор ENTRY не может стоять во внутреннем блоке процедуры, внутри IF-оператора и в теле оператора цикла.

Атрибут BUILTIN. Как и в алголе, идентификатор какой-либо стандартной функции (например SQRT) может быть описан (явно или контекстуально) и использован в некотором блоке в другом смысле. Для возвращения этому идентификатору смысла имени встроенной функции во внутреннем блоке следует дать описание вида:

```
DECLARE SQRT BUILTIN;  
(BUILTIN — атрибут «встроенный»).
```

Пр и м е р.

```
A:H: PROCEDURE (Q, SQRT, P);  
    DECLARE SQRT (*), P FLOAT;  
    Q=SQRT (P); TAN=1;
```

```
F:B: ENTRY (P);
```

```
    C: BEGIN; DCL SQRT BUILTIN;  
        W=SQRT (P);
```

```
    END C;
```

```
END A;
```

```
    CALL A (U, Z, D); /* U=Z (D); W=SQRT (D)*/  
    CALL B(X); /*W=SQRT (X)*/
```

В процедуре A (или H) идентификатор SQRT объявлен как имя массива, а в блоке C, входящем в процедуру, он является именем встроенной функции. Идентификатор TAN является именем переменной, объявленной контекстуально, а не именем встроенной функции, и обращение к последней в программе будет ошибкой. В комментариях указываются результаты обращения к процедуре по главному или дополнительному входам: Z (D) — переменная с индексом, SQRT (X) и SQRT (D) — указатели функции.

З а м е ч а н и е. В PL/1 обращение к одной и той же процедуре, даже по одному и тому же имени, может производиться как по оператору CALL, так и по указателю функции. При этом должно только выполняться условие, чтобы в первом случае нормальное окончание выполнения процедуры производилось по операторам

END, GO TO или RETURN, а во втором случае — только по RETURN (eS);.

Пример.

```
DCL PF ENTRY (FLOAT) RETURNS (FIXED);  
CALL PF (Z); X=PF (Y);  
PF: PROCEDURE (Q) RETURNS (FIXED); DCL Q FLOAT;  
IF Q < 0 THEN RETURN (1);  
ELSE Q=2;
```

END;

Здесь предполагается, что $Z \geq 0$ а $Y < 0$, и Z имеет умалчиваемые атрибуты. В результате выполнения оператора CALL получаем $Z=2$, а в результате выполнения следующего оператора получаем $X=1$.

2.4.4. Аргументы и параметры.

Формальные и фактические параметры в PL/I называются соответственно *параметрами* и *аргументами*. Помимо тех атрибутов для формальных параметров, которые были даны в 1.4.4, при спецификации (описании) параметров могут использоваться и любые другие, имеющиеся в PL/I, за исключением атрибутов класса памяти, сферы, INITIAL, BUILTIN. Звездочки в скобках, которые фигурировали в спецификациях PL/I (см. 1.4.4), являются атрибутами размерности или длины; звездочки говорят о том, что граничные пары или длина при выполнении процедуры будут определяться по атрибутам передаваемых аргументов. Допускается вместо звездочек задание и десятичных целых чисел, указывающих на конкретные значения граничных пар или длины. Процедура при этом будет более эффективной, но соответствующие фактические параметры должны будут использоваться в качестве своих атрибутов только эти конкретные значения.

З а м е ч а н и е. Отметим, что в отличие от алгола, контекстуальное объявление в PL/I не распространяется на параметры: если параметры не описаны в DECLARE, то они считаются явно описанными в процедурном блоке и всегда как скалярные переменные (с атрибутами по умолчанию). Например, в процедуре

```
BR: PROCEDURE (X, Y);  
CALL Y; GO TO X;  
END BR;
```

операторы, содержащие X и Y ошибочны, так как X и Y считаются описанными как скалярные переменные; кроме того, меточные переменные (см. X) контекстуально не описываются!

Атрибуты входа. Атрибут входа (ENTRY) в описаниях, после которого в скобках перечисляются атрибуты формальных параметров (см. 1.4.4), может быть опущен, если при всех обращениях к процедуре атрибуты фактических параметров (включая и атрибут точности!) совпадают с атрибутами формальных пара-

метров. Если же атрибуты совпадают не для всех параметров, то вслед за атрибутом ENTRY для таких параметров должны быть указаны необходимые атрибуты; пропускаемые параметры отмечаются запятыми, и последние запятые могут быть опущены; факторизация (вынесение за скобки) общих атрибутов здесь не допускается.

Атрибут значения функции (RETURNS), после которого указываются атрибуты возвращаемого значения, может быть также опущен, если при всех обращениях к процедуре-функции и по любому из ее имен атрибуты возвращаемого значения совпадают с атрибутами, которые подразумеваются по умолчанию для имен функции (FIXED BINARY (15,0) или FLOAT DECIMAL (6)).

Вернемся к примерам п. 1.4.4 и дадим минимально необходимые атрибуты при описании имен входов MAXP, MAXF, MAXM, INTEGRAL и G.

1) DCL MAXP ENTRY (FIXED, FIXED); — атрибут FIXED для 3-го параметра опущен, так как аргумент R1 имеет нужные атрибуты;

2) DCL MAXF ENTRY (FLOAT, FLOAT, FLOAT) RETURNS (FLOAT); — атрибут RETURNS для MAXF обязателен, поскольку транслятор по умолчанию для имени функции взял бы FIXED BINARY (15,0);

3) описание входа для MAXM можно опустить целиком (в том числе и атрибут размерности, соответствующий звездочке в скобках при описании параметров), так как все аргументы имеют необходимые атрибуты;

4) DCL INTEGRAL ENTRY (FLOAT,, FIXED,,); — описание входа G опускается. Атрибуты трех параметров, отмеченных запятыми (последние две можно было и не ставить), совпадают с атрибутами аргументов в примере.

Дальнейшее сокращение при описании имен входов привело бы к ошибке при передаче аргументов вызываемой процедуре. Дело в том, что при трансляции обращений к процедуре, как к внутренней, так и к внешней, предполагается, что транслятор не имеет доступа к описанию формальных параметров в этой процедуре (и к атрибутам возвращаемого значения функции). Поэтому при отсутствии описания имени входа аргументы передаются в процедуру с теми атрибутами, которые они имеют. В случае их несоответствия атрибутам формальных параметров никаких преобразований не делается, и поэтому происходит ошибка.

Атрибуты, перечисляемые при описании имени входа, служат для указания на необходимость преобразований аргументов — приведения их атрибутов к атрибутам параметров. (Такие преобразования предусматривает транслятор и при этом печатает предупреждающую диагностику).

Если опустить приведенные выше в примерах описания входов, то это будет ошибкой в силу того, что атрибуты аргументов 25, Y1, 3E—5, Y2, Z2, 0, 1, 20 не совпадают с атрибутами соответствующих параметров, заданных при их описании в процедуре.

Из рассмотренных примеров можно сделать вывод, что опускание атрибутов ENTRY и RETURNS не должно быть рекомендовано к широкому использованию, так как может приводить к труднонаходимым ошибкам.

И атрибут ENTRY, и атрибут RETURNS описывают идентификатор как имя входа. Как следует из вышеизложенного, каждый из этих атрибутов или оба в некоторых случаях могут отсутствовать. Если функция не имеет параметров и контекстуально или явно не объявляется, то ее идентификатор должен быть описан одним из атрибутов входа, например, следующим образом:

DECLARE FUN1 ENTRY;

или

DECLARE FUN2 RETURNS (FLOAT);

Приведем формы атрибутов входа:

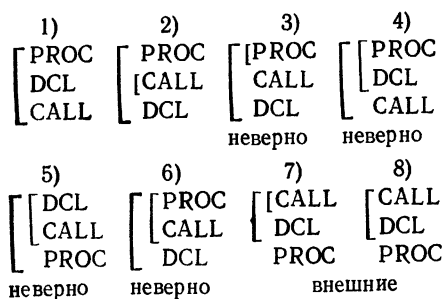
t-ENTRY ~ ENTRY [(t...),...]

t-RETURNS ~ RETURNS (t...)

Вынесение общих атрибутов за скобки (факторизация) для списка атрибутов в ENTRY не допускается.

Для внутренней процедуры описание входа должно находиться в том же блоке, что и сама процедура. Напомним, кроме того, что обращение к процедуре может осуществляться только из того же блока, где находится процедура, или из внутренних блоков. Для внешней процедуры описание ее входа должно находиться в том же блоке, где расположены обращения к ней, или во внешних по отношению к нему блоках.

Ниже приведены схемы взаимных расположений процедуры (PROC), обращения к ней (CALL) и описания имени входа (DCL) внутри блоков; порядок расположения внутри блоков несущественен; квадратные скобки изображают окаймляющие блоки:



Схемы 3 и 4 неверны, поскольку CALL находится во внешнем по отношению к PROC блоке, где имя процедуры, тем самым, неизвестно. Схемы 5 и 6 неверны, так как описание входа находится в блоке внутреннем или внешнем по отношению к тому блоку, где находится процедура. На схемах 7 и 8 даны внешние процедуры.

Можно считать, что в PL/I описание имени входа в операторе DECLARE дополняет описание этого имени, осуществленного посредством PROCEDURE-оператора.

З а м е ч а н и я . 1. В атрибуте ENTRY для параметров, являющихся массивами, в качестве атрибутов может оказаться одна лишь звездочка в скобках. В этом случае всегда предполагается, что другими атрибутами соответствующих параметров являются REAL FLOAT DECIMAL (6), и аргументы будут приводиться к этим атрибутам. Например:

```
DECLARE F ENTRY ((*), (*)), A (5) FIXED,  
                                         B (-9: +9) BINARY;  
CALL F (A, B);
```

Аргументы A и B приводятся к FLOAT DECIMAL (6), и для правильной их передачи параметры в процедуре F должны иметь эти же атрибуты.

2. При несовпадении атрибутов параметров при описании имени входа с атрибутами параметров в теле процедуры никакой диагностики не выдается, и при выполнении произойдет ошибка. То же случится и при несовпадении атрибутов для значения функции в RETURNS-опции и RETURNS-атрибуте. Ошибки может и не быть, если аргументы и параметры имеют одинаковое машинное представление, например, для FIXED BINARY (15) и FIXED BINARY (10) или FLOAT BINARY (30) и FLOAT DECIMAL (12) (см. 2.1.3).

3. Описание входа для встроенной функции не допускается.

З а м е ч а н и я о б а р г у м е н т а х . Если вернуться к форме для аргументов, приведенной в 1.4.4, то теперь она будет выглядеть следующим образом (см. также 2.2.4):

$$a \sim \{e \mid vL \mid (vL) \mid 1 \mid iP \mid (iP)\}$$

Здесь *e* обозначает как скалярное, так и массивное выражение, причем в частном случае оно может быть представлено идентификатором соответствующей проблемной переменной, заключенной, может быть, в скобки. Меточная переменная (*vL*) является скалярной или массивной. Аргументы могут принадлежать как к автоматическому, так и к статическому классу памяти.

Массивные выражения должны соответствовать только массивным параметрам. Скалярные выражения могут соответствовать как скалярным, так и массивным параметрам, но в последнем

случае в атрибуте ENTRY в нужном месте должен быть задан атрибут размерности, и при описании параметра граничные пары для рассматриваемого случая должны иметь вид десятичных целых чисел (но не звездочек).

Для строчных величин, если атрибут изменяемости длины (VARYING или не-VARYING) для аргумента и параметра не совпадает, то в атрибуте ENTRY на соответствующем месте должен быть указан необходимый атрибут параметра (наличие VARYING или его отсутствие вслед за атрибутом длины). Причем, в случае, когда аргумент имеет атрибут VARYING, а параметр описан без VARYING, и вместо максимальной длины задана звездочка (*), то аргумент передается с максимальной объявленной длиной.

Пример.

```
DECLARE S CHARACTER (8) VARYING,
      P ENTRY (CHAR (*));
S='ABCD'; CALL P(S);
P:PROCEDURE (H); DECLARE H CHAR (*), G CHAR (6);
      G=H! 'EF';
END P;
```

Переменная G получает значение ABCD□□, а не ABCDEF, так как аргумент передается в виде ABCD□□□□, и выражение имеет значение ABCD□□□□EF.

Если атрибуты аргумента не совпадают с атрибутами параметра (или если аргумент является выражением, или заключен в скобки), то на стадии трансляции создаются фиктивные («пустые» — DUMMY) аргументы — служебные переменные с атрибутами, совпадающими с атрибутами параметров. При выполнении программы производится приведение аргумента к атрибутам фиктивного аргумента и передача аргумента по значению; в противном случае аргумент передается по наименованию (ср. с 1.4.4).

Общий вход. В PL/1 имеется понятие *общего входа* (GENERIC), которое позволяет программисту при обращении к процедуре не указывать явно имя процедуры или дополнительного входа — оно будет выбрано (на стадии трансляции) в соответствии с количеством фактических параметров или (и) их атрибутами в операторе CALL или в указателе функции.

Пример. Пусть имеются следующие входы:

```
EFC: PROCEDURE (A, B);
      . . . . .
FFC: ENTRY (A, C);
      . . . . .
FC: PROCEDURE (A);
      . . . . .
EC: ENTRY (B);
      . . . . .
```

и следующее описание *)

DECLARE CC GENERIC (EFC ENTRY (FLOAT, FIXED),
FFC ENTRY (FIXED, FIXED),
FC ENTRY (FIXED),
EC ENTRY (FLOAT));

Тогда оператор CALL CC (X, Y) выберет имя входа EFC, если X и Y имеют соответственно атрибуты FLOAT и FIXED; или имя FFC, если X и Y описаны как FIXED и FIXED. Для случая X и Y с атрибутами FLOAT и FLOAT или FIXED и FLOAT возникает ошибка, так как соответствующего имени входа не находится. По оператору CALL CC (Z) произойдет обращение ко входу FC или EC, в зависимости от атрибутов Z (FIXED или FLOAT).

В качестве атрибутов, по которым производится выбор частного входа, могут использоваться любые атрибуты аргументов (параметров) кроме:

- а) атрибута длины (для CHAR или BIT);
- б) значений границ в атрибуте размерности (но количество измерений при этом исследуется);

- в) атрибутов параметров в атрибуте ENTRY.

Никаких преобразований аргументов к атрибутам параметров, указанным при объявлении частных входов, не производится, и нужно внимательно следить за атрибутами задаваемых аргументов (и за точностью!), особенно, если они являются константами или выражениями.

При объявлении частных входов внутри общего они могут быть снабжены другими атрибутами (например, RETURNS), включая GENERIC и BUILTIN; дополнительные объявления для частных входов в других местах не допускаются.

2.4.5. Программа и подпрограммы.

Как уже упоминалось выше, в PL/1 программа может состоять из нескольких внешних процедур — *подпрограмм*, одна из которых является главной (MAIN). Выполнение программы начинается с главной подпрограммы, которая может обращаться (например, по CALL) к другим подпрограммам, которые, в свою очередь, могут обращаться друг к другу. Информационная связь подпрограмм осуществляется с помощью аргументов и параметров или с помощью атрибута EXTERNAL (см. 2.4.2).

Главная процедура также может иметь параметр (только один) который должен быть символьной строкой со следующими атрибутами:

CHARACTER (100) VARYING

*) Не спутайте ENTRY-оператор с атрибутом ENTRY в описании.

Аргумент для этого параметра задается в директиве *) EXEC, например, следующим образом:

/ /имя EXEC PL1LFCLG, PARM.GO='ARGUMENT=25'
Символьная строка может быть задана, разумеется, любая. Все ключевые слова должны быть написаны без пробелов—кроме двух пробелов около EXEC.

В теле главной процедуры заданный аргумент может быть использован любым образом, например, напечатан по PUT LIST:

```
PRMA: PROCEDURE (CTP) OPTIONS (MAIN);  
      DECLARE (CTP) CHAR (100) VARYING;  
      .....  
      PUT LIST (CTP);  
      .....
```

Задание с несколькими подпрограммами имеет следующий вид:

```
/ /PRIM JOB  
/ /SH1 EXEC PL1LFC  
/ /PL1L.SYSIN DD *  
      текст-главной-программы  
/ /SH2 EXEC PL1LFC  
/ /PL1L.SYSIN DD *  
      текст-подпрограммы-1  
/ /SH3 EXEC PL1LFCLG  
/ /PL1L.SYSIN DD *  
      текст-подпрограммы-2  
/ /GO.SYSIN DD *  
      текст-исходных-данных  
/ /
```

В этом задании на первом шаге (SH1) осуществляется трансляция главной подпрограммы; на втором шаге (SH2)—трансляция 1-й подпрограммы, на 3-м шаге (SH3)—трансляция (C—Compile) 2-й подпрограммы, объединение (L—Link) всех трех подпрограмм и выполнение (G—Go) полученной программы с заданными исходными данными. (Карты с признаком окончания вводимой информации (/*) для краткости не задаются—обычно они являются необязательными.)

Упражнения.

2.4.1. Составить процедуру для нахождения действительных корней квадратного многочлена $ax^2 + bx + c$. В случае отрицательного дискриминанта, выдать нулевые значения для корней; предусмотреть дополнительный вход для случая, когда $a=0$.

*) В документации называется *управляющим оператором*.

2.4.2. Найти ошибку в следующем фрагменте программы и исправить ее:

```
PLUS: PROCEDURE (X, Y, Z); DCL (X, Y, Z) FIXED;  
      Z = X + Y; END;
```

```
DECLARE PLUS ENTRY (FIXED, FIXED, FIXED);
```

```
GET LIST (J); CALL PLUS (124, J, K); R = 1.5 + K;
```

2.4.3. Найти ошибку в следующем фрагменте программы и исправить ее:

```
MINUS: PROCEDURE (X, Y); DCL (X, Y) FIXED;
```

```
      RETURN (X - Y); END;
```

```
DECLARE MINUS ENTRY (FIXED, FIXED), L INIT (125);
```

```
      Q = 2.1 * MINUS (L, 2);
```

ГЛАВА 3

НОВЫЕ ПОНЯТИЯ PL/1

В этой главе излагаются некоторые из тех основных возможностей PL/1, которые не затрагивались в первых двух главах, поскольку они не имеют почти никаких аналогий с возможностями алгола, но которые важны при решении задач в весьма широкой области. К таким средствам PL/1 в первую очередь относятся структуры, ситуации и операторы ввода-вывода.

3.1. Структуры

В PL/1, помимо скалярных величин и массивов, допускаются данные, организованные в *структуры*. Понятие структуры можно рассматривать, как результат развития понятия массива. В то время, как массивы являются упорядоченными наборами значений, имеющих одни и те же атрибуты, в структурах могут быть объединены данные как с различными атрибутами, так и различным образом организованные: скаляры, массивы и другие структуры. Кроме того, в отличие от элементов массива, все данные, составляющие структуру, имеют свои имена, которые могут и совпадать в разных структурах. Структура, содержащая другие структуры, может иметь, тем самым, сложное иерархическое строение.

3.1.1. Описания структур

Примеры описаний. Предположим, что нам нужно обрабатывать следующие сведения о сотрудниках некоторого подразделения: фамилия и. о., процент профсоюзного взноса, ежемесячные зарплата и взнос. Если предположить, что количество сотрудников не превышает 100, и воспользоваться известным нам средством — массивами, то необходимые данные можно описать, например, следующим образом:

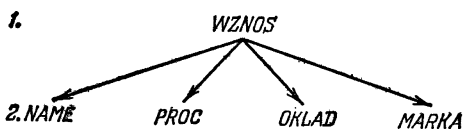
```
DECLARE NAME (100) CHAR (30), PROC (100) FIXED (2),  
        OKLAD (100, 12) FIXED (6, 2),  
        MARKA (100, 12) FIXED (5, 2);
```

Сведения, относящиеся к отдельному сотруднику, характеризуются конкретным значением индекса у соответствующих переменных.

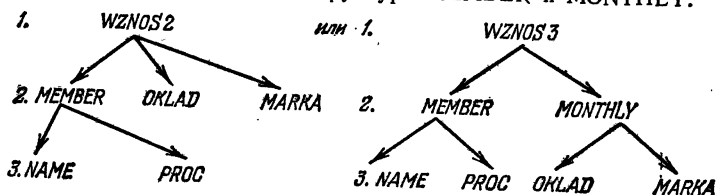
В PL/1, используя понятие структуры, сведения для одного сотрудника можно описать так:

```
DECLARE 1 WZNOS, 2 NAME CHAR (30),
        2 PROC FIXED (2),
        2 OKLAD (12) FIXED (6, 2),
        2 MARKA (12) FIXED (5, 2);
```

Структура WZNOS содержит два скаляра и два массива, с различающимися атрибутами. Целые десятичные числа, отделяемые пробелом (пробелами) от идентификаторов—уровни—определяют подчиненность имен или иерархию имен в структуре, что можно изобразить следующим образом:



Ту же информацию можно было бы организовать и в такие структуры, содержащие подструктуры MEMBER и MONTHLY:



Дадим описание для первой из них:

```
DECLARE 1 WZNOS2, 2 MEMBER, 3 NAME CHAR (30),
        3 PROC FIXED (2),
        2 OKLAD (12) FIXED (6, 2),
        2 MARKA (12) FIXED (5, 2);
```

Для хранения информации, относящейся ко всем сотрудникам, можно воспользоваться еще одним новым понятием—массивом структур со следующим описанием:

```
DECLARE 1 WZNOS4 (100), 2 MEMBER,
        3 NAME CHAR (30), и т. д.
```

(далее см. структуру WZNOS2 или WZNOS3).

Здесь WZNOS4—массив, элементами которого являются структуры.

Вместо структуры MONTHLY, объединяющей два массива, можно также задать массив структур:

..., 2 MONTHLY1 (12), 3 OKLAD FIXED (6, 2),
3 MARKA FIXED (5, 2);

Выбор того или иного способа организации данных зависит от предполагаемого характера их обработки в дальнейшем.

Правила описания. Номер уровня самой старшей — *полной* — структуры должен быть равен единице. Уровни элементов, составляющих некоторую структуру (или подструктуру), могут и не совпадать друг с другом, но они должны быть больше (не обязательно на единицу) уровня структуры, их объединяющей; в противном случае (когда номер уровня некоторого элемента меньше или равен номеру уровня предыдущей подструктуры), считается, что данный элемент принадлежит более старшей структуре.

Таким образом, иерархия имен в структуре задается относительной величиной уровней, а не их абсолютными величинами (которые, нужно заметить, нигде кроме описаний и не используются). Поэтому строение структуры WZNOS3 фактически не изменилось бы, если ее описать, например, следующим образом:

DECLARE 1 WZNOS31, 3 MEMBER, 6 NAME, 4 PROC,
2 MONTHLY, 5 OKLAD, 5 MARKA;

Если читать описание этой структуры слева направо, то уменьшение (по сравнению с предыдущим) номера уровня перед элементом структуры (6 после 3, 5 после 2, а также 3 после 1) говорит о том, что этот элемент является частью непосредственно предшествующего элемента (структуры). А уменьшение номера уровня или его совпадение с предыдущими (4 после 6, 2 после 4, 5 после 5) говорит о том, что начинается новый элемент (подструктура), принадлежащий одной из предыдущих структур (может быть, и самостоятельная (полная) структура, если задан уровень равный единице); отсутствие уровня перед именем также оканчивает описание предыдущей структуры.

При описании структур факторизовать (выносить за скобки) можно не только общие атрибуты элементов, но и уровни структур. Например:

DECLARE 1 WZNOS20,
2 MEMBER, 3 (NAME CHAR (30),
PROC FIXED (2)),
2 (OKLAD FIXED (6, 2),
MARKA FIXED (5, 2)) (12);

но не

DECLARE 1 WZNOS21,
2 (MEMBER, 3 (NAME CHAR (30),
PROC FIXED (2)),
(OKLAD FIXED (6, 2),
MARKA FIXED (5, 2)) (12));

так как к именам NAME и PROC относится как уровень 2, так и 3.

Инициализация структуры при ее описании может быть осуществлена только инициализацией элементов, ее составляющих (то есть, в конечном счете скаляров и массивов). Например:

```
DECLARE 1 WZNOS22,  
        2 MEMBER, 3 NAME CHAR (30) INIT (''),  
        3 PROC FIXED (2) INIT (99),  
        2 (OKLAD FIXED (6, 2), MARKA FIXED (5, 2))  
          (12) INIT ((12) 0);
```

При инициализации элементов, входящих в массив структур, следует учитывать, что количество констант, необходимых для инициализации таких элементов, зависит от атрибута размерности, приписанного массиву структур. Таким образом, приходится считать, что атрибут размерности массива структур (а, может быть, и подструктур) как бы переносится на составляющие его элементы и добавляется к их размерности.

Пример.

```
DCL 1 WZNOS5 (100),  
    2 MEMBER, 3 NAME CHAR (30) INIT ((100) (1)''),  
    3 PROC FIXED (2) INIT ((100) 99),  
    2 MONTHLY (12),  
    3 OKLAD FIXED (6, 2) INIT ((1200) 0),  
    3 MARKA FIXED (5, 2) INIT ((1200) 0);
```

Величины NAME и PROC имеют фактически первую размерность (1:100), а OKLAD и MARKA—вторую (1:100, 1:12).

Имена структур и их элементов могут снабжаться атрибутами памяти или сферы, но атрибуты памяти указываются только для полных структур, а атрибуты сферы—и для подструктур, и для элементных данных, входящих в структуру. Атрибуты соответствующих элементов внешних (EXTERNAL) структур, имеющих одинаковые имена, должны совпадать между собой, но имена элементов могут и различаться. Имена элементов внешних структур по умолчанию считаются INTERNAL, но при изменении их в одном блоке, в другом—они, тем самым, также изменяются. Это можно объяснить тем, что внешние структуры и их элементы размещаются в памяти статически (из EXTERNAL обязательно следует STATIC).

Например, пусть в двух блоках даны следующие описания:

```
A: BEGIN; DECLARE 1 DO EXTERNAL,  
                  2 RE, 2 MI; ... END;  
B: BEGIN; DECLARE 1 DO EXT, 2 FA,  
                  2 SOL FIXED BINARY;
```


Атрибуты FIXED BINARY необходимы, иначе атрибуты для MI и SOL различались бы. Имена переменных, входящих в структуры, имеют атрибут INTERNAL и неизвестны вне своих блоков. Но при изменении RE или MI в одном блоке, FA и SOL также изменятся (об операторе присваивания для структур и их элементов см. в 3.1.3).

Скаляры и массивы называются иногда *элементными* данными при противопоставлении их данным, организованным в структуры. Массивы и структуры называются *агрегатными* данными для противопоставления их скалярам.

Атрибут LIKE. При описании структур, содержащих другие структуры, можно использовать атрибут LIKE («похожий», «подобный»), указывающий на то, что описываемая структура является структурой, подобной той, имя которой задано после LIKE. Подобие состоит в том, что организация новой структуры, имена и атрибуты ее элементов берутся из старой структуры (выражения, входящие в атрибуты, при этом перевычисляются), но атрибуты, относящиеся к имени старой структуры (например, атрибут размерности для массива структур), на новую не переносятся *). Например, для описания структуры, содержащей информацию о комсомольском взносе можно написать:

```
DECLARE 1 KOMS_WZNOS LIKE WZNOS22;
```

Это эквивалентно

```
DCL 1 KOMS_WZNOS,
```

```
2 MEMBER,
```

```
3 NAME CHAR (30) INIT (''),
```

```
3 PROC FIXED (2) INIT (99),
```

и т. д.

Но при описании с помощью LIKE массива структур всегда необходимо задавать атрибут размерности. Например:

```
DECLARE 1 KOMS_WZNOS1 (100) LIKE WZNOS5;
```

Атрибут LIKE используется и при описании подструктур одной и той же или разных структур. Например:

```
DECLARE 1 N, 2 K, 3 I, 3 J, 2 L LIKE K, 2 M LIKE K;
```

эквивалентно

```
DECLARE 1 N, 2 K, 3 I, 3 J, 2 L, 3 I, 3 J, 2 M, 3 I, 3 J;
```

Как видим, в последней структуре фигурируют одинаковые имена (I и J). Это допускается в том случае, если одинаковые имена встречаются в разных подструктурах (или структурах). О том, как ссылаться на различные данные с одинаковыми именами, будет сказано в 3.1.2. Если требуется, то номера уровней

*) Имеются в виду атрибуты, встречающиеся в настоящем курсе.

автоматически подправляются при раскрытии атрибута LIKE. Например,

```
DECLARE 1 NS, 2 E, 3 F, 3 G, 4 H LIKE N;
```

эквивалентно

```
DECLARE 1 NS, 2 E, 3 F, 3 G, 4 H, 5 K, 6 I, 6 J,  
5 L, 6 I, 6 J, 5 M, 6 I, 6 J;
```

При использовании атрибута LIKE нужно учитывать следующие ограничения:

1) та структура, на которую ссылается атрибут LIKE, и ее элементы не должны быть описаны с LIKE. Например, недопустимо 2 M1 LIKE M;

2) добавление элементов к структуре, описанной по LIKE не допускается. Например, недопустимо

```
2 M LIKE K, 4 Z или 2 M LIKE K, 3 Z;
```

3) Структура, на которую ссылается атрибут LIKE, не может содержать массивов с вычисляемыми границами.

3.1.2. Структурные переменные и квалифицированные имена.

Понятие структуры используется для определения понятия структурной переменной.

Структурная переменная задается идентификатором структуры. Но в случае, если эта структура входит в другую структуру, ее идентификатор (имя) должен быть *квалифицирован* (уточнен), то есть дополнен слева через точку именем той структуры, в которую она входит. Например, на приведенные выше структуры или массивы структур можно сослаться следующим образом:

```
WZNOS2, WZNOS2.MEMBER, WZNOS5,  
WZNOS5.MONTHLY, N, N.M
```

Элементы структур, являющиеся скалярами или «скалярными» массивами (в отличие от структурных), определяются также с помощью *квалифицированных* (составных) имен. Например:

```
WZNOS2.OKLAD, WZNOS2.MEMBER.NAME,  
N.K.J, N.L.J, N.M.J
```

Имя некоторой величины считается квалифицированным *полностью*, если в нем указаны имена *всех* (под)структур, содержащих данную величину.

Имена старших структур (все или некоторые из них) в квалифицированных именах могут быть опущены, если это не приведет к двусмысленности, то есть, если в данном блоке не известны другие величины с именами, совпадающими с именами элементов структур. Таким образом, может быть допустимо задать:

```
MEMBER, MONTHLY, OKLAD, M,  
WZNOS2.NAME, K.J, M.J
```

Но, очевидно, что нельзя просто задать J или I (см. выше).

В случае, когда заданное в программе имя переменной можно, в принципе, отождествить с различными величинами, известными в данном блоке, то предполагается, что имя относится к такой величине, для которой указанное имя является квалифицируемым полностью. Например, для

DECLARE 1 A, 2 A, 3 A, 3 B, 4 B, 2 B, 3 C, 3 B;
имена A, A.A, A.A.A будут ссылаться соответственно на структуру, подструктуру, скаляр, но B или B.B являются ошибочными.

При отождествлении заданных квалифицированных имен с величинами программы учитываются и правила локализации имен в блоках. Например, при

DECLARE 1 P, 2 Q, 2 R, 3 S, 3 U;
BEGIN; DECLARE 1 P, 2 T, 3 Q, 3 S;
имя P.S во внутреннем блоке соответствует P.T.S, а имя R.S — определяет P.R.S.

Ссылки на элементы массивов структур (*структурных массивов*) могут осуществляться разными способами:

WZNOS5(1).MONTHLY(6)

или

WZNOS5.MONTHLY(1,6)

WZNOS5(1).MONTHLY(6).OKLAD

или WZNOS5.MONTHLY(1,6).OKLAD

или WZNOS5.MONTHLY.OKLAD(1,6)

Для ссылки на отдельный элемент структуры описанной

DCL 1 X (2, 2), 2 Y (4), 3 Z, 3 W (10);

можно, например, задать

X(1,2).Y(3).W(8) или X(1,2,3).Y.W(8)

или X.Y.W(1,2,3,8)

Последний способ, когда все индексы собраны у элементной переменной, называется «с непрославляемыми индексами» и допускается в различных местах программы без ограничений; первые способы могут быть использованы не всегда (см. например, 3.3.3).

З а м е ч а н и я.

1. Ссылка на элементы массива структур без прославивания индексов соответствует такой трактовке массива структур, когда он представляется в виде структуры, содержащей только «скалярные» подструктуры и массивы (вспомните об инициализации массива структур в 3.1.1).

2. Сечения для массивов структур не допускаются.

3.1.3. Структурные выражения.

В PL/I структурные переменные могут входить в выражения; в этом случае выражения называются *структурными*. Если структурных переменных в выражении несколько, они все должны иметь одинаковое строение (номера уровней, имена и атрибуты элемент-

ных данных могут быть разными), а операции над структурами сводятся к операциям над соответствующими элементами структур. В структурные выражения могут входить и скаляры; при этом операции производятся над скаляром и всеми элементами структуры (ср. с выражениями над массивами). Массивные переменные не могут входить в структурные выражения.

Структурные выражения могут являться правой частью оператора присваивания, в левой части которого находится структурная (псевдо)переменная. Возможны и операторы присваивания со структурной (псевдо)переменной в левой части оператора и скалярным выражением в правой части. При вычислении значений структурных выражений или при присваивании могут происходить различные преобразования данных.

Все сказанное относится и к массивам структур.

Примеры операторов присваивания (описания см. выше):

1) $X = 10;$

Все 176 элементов, содержащихся в массиве структур, получают значение равное десяти.

2) $A.A.A = X(1,2).Y(3).W(8) + X.Y.Z(1,J,K);$

Скалярная переменная A.A.A получает значение скалярного выражения (суммы скалярных элементов массива структур).

3) $P.T = P.R * 0.1;$

Подструктуре присваивается значение структурного выражения; элементы подструктуры P.T получают значения элементов подструктуры P.R, умноженных на скаляр 0.1.

Этот оператор эквивалентен следующей последовательности операторов присваивания:

$P.T.Q = P.R.S * 0.1;$

$P.T.S = P.R.U * 0.1;$

Опция BY NAME. Имеется возможность производить операции и над структурами (массивами структур, подструктурами), имеющими разное строение: опция BY NAME («по имени»), указываемая в операторе присваивания, задает операции над теми элементами структур, заданных в левой и правой частях оператора, которые имеют одинаковые имена — имеются в виду одинаковые квалифицированные имена внутри заданных структур.

Пример.

```
DECLARE 1 ODIN, 2 A, 3 P, 3 Q, 2 B, 3 X, 3 S, 3 F, 2 C,  
        1 DWA, 2 C, 2 A, 3 X, 3 Q, 3 P, 2 B, 3 Q, 3 S,  
        1 TRI, 4 B, 5 S, 5 Q, 3 F, 3 A, 5 X, 5 P, 2 Q;
```

Пусть задан оператор

$ODIN = DWA, BY NAME;$

Определение элементов, имеющих одинаковые имена в структурах ODIN и DWA, можно вести поэтапно. Структура ODIN состоит

из элементов А, В и С, а структура DWA — из С, А и В; причем С в обоих случаях является скаляром. Поэтому получаем следующую последовательность операторов, эквивалентную заданному оператору:

(1) ODIN.C = DWA.C;

(2) ODIN.A = DWA.A, BY NAME;

(3) ODIN.B = DWA.B, BY NAME;

Второй из этих операторов эквивалентен операторам:

ODIN.A.P = DWA.A.P;

ODIN.A.Q = DWA.A.Q;

а третий эквивалентен оператору

ODIN.B.S = DWA.B.S;

Таким образом, приведенный выше оператор эквивалентен четырем операторам, не содержащим опцию BY NAME.

Оператор

ODIN = DWA * TRI, BY NAME;

эквивалентен следующим операторам:

ODIN.A.P = DWA.A.P * TRI.A.P;

ODIN.B.S = DWA.B.S * TRI.B.S;

Элементы Х и Q, встречающиеся в структурах ODIN, DWA и TRI, имеют разные квалифицированные имена и поэтому в операциях не участвуют.

Оператор

ODIN.A, TRI = DWA.B + TRI.B / 2, BY NAME;

эквивалентен оператору

ODIN.A.Q, TRI.Q = DWA.B.Q + TRI.B.Q / 2;

Оператор

ODIN.A = DWA, BY NAME;

был бы ошибочен, поскольку в заданных структурах нет элементов с одинаковыми именами: подструктура ODIN.A состоит из элементов с именами Р и Q, а структура DWA — из элементов С, А и В,

З а м е ч а н и е. Структурные выражения могут являться аргументами процедур, а имена полных структур (не подструктур) — параметрами. Организация структуры-параметра и структуры-аргумента должна быть одинакова. В том случае, если атрибуты соответствующих элементов параметра и аргумента различаются, необходимо дать описание имени процедуры с атрибутом ENTRY (см. 2.4.4), в котором указываются все уровни элементов структуры и необходимые атрибуты. Например, если вторым из 4 аргументов некоторой процедуры PR является структура ODIN (см. выше), то, может быть, понадобится задать:

DECLARE PR ENTRY (, 1, 2, 3, 3, 2, 3 FIXED, 3, 3, 2 CPLX,,);

Атрибут ENTRY необходим также, если аргументом для структурного параметра является скалярная величина.

Элементным данным в структуре-параметре, для которых в атрибуте ENTRY указан только уровень, по умолчанию приписываются атрибуты FLOAT DECIMAL (6) (те же самые, что и для массива, когда в ENTRY заданы одни лишь звездочки — см. 2.4.4).

Отметим, кроме того, что значением одной из встроенных функций может являться структура (или массив структур — см. STRING).

3.1.4 Уточнение форм.

Оператор описаний с учетом описания структур:

$$d \sim \text{DECLARE } \Delta; \Delta \sim \{[\bar{n}ID] \{i | (\Delta)\} [t...]\}, \dots$$

Упрощенная форма:

$$d \sim \text{DECLARE } \{ [\bar{n}ID] i [t...] | (\{[\bar{n}ID] i [t...]\}, \dots) [t...] | [\bar{n}ID] (\{i [t \dots]\}, \dots) [t...]\}, \dots$$

Структурная переменная (U — structure, структурный):

$$vU \sim \{vUS | vUM\}$$

$$vUS \sim [iUS. | iUM (eSI, \dots)] \dots \{iUS | iUM (eSI, \dots)\}$$

$$vUM \sim [vUS.] \dots iUM$$

Здесь vUS и vUM обозначают «структурно-скалярные» и «структурно-массивные» переменные.

Скалярная переменная (может быть, с квалифицированным именем):

$$vS \sim \{iS | iM (eSI, \dots) | [vUS.] \dots vS\}$$

Массивная переменная (может быть, с квалифицированным именем):

$$vM \sim \{iM | iM (\{eSI | (*)\}, \dots) | [vUS.] \dots vM | \{(vUM.) | vUS.\} \dots vS\}$$

Ломаные скобки в 1-м случае указывают на наличие хотя бы одной звездочки при задании сечения (см. 2.1.7), а во 2-м — на наличие хотя бы одного имени массива структур (наряду с именами структур) в квалифицированном имени массива.

Структурное выражение:

$$eU \sim \{eUS | eUM\}$$

$$eUS \sim \Phi \{vUSO | \bar{i}US | eS\}$$

$$eUM \sim \Phi \{vUMO | \bar{i}UM | eS\}$$

Выражение в PL/1:

$$e \sim \{eS | eM | eU\}$$

Оператор присваивания для структур:

$$s \sim \left\{ \begin{aligned} \{vUS | \bar{i}WUS\}, \dots &= \{eUS [, BY NAME] | eS | vSL | I\}; \\ \{vUM | \bar{i}WUM\}, \dots &= \{eUM [, BY NAME] | eS | vSL | I\}; \end{aligned} \right\}$$

В структурные выражения eUS и eUM могут входить структурные переменные, содержащие меточные данные (скалярные и массивные), над которыми, разумеется, не могут выполняться никакие операции, свойственные проблемным данным.

В заключение приведем две таблички, которые помогут установить правильность сочетаний различных видов переменных и выражений в левой и правой частях оператора присваивания (см. $v = e$), или операндов в выражении (см. $e \otimes e$); плюс обозначает допустимость сочетания, минус — нет.

$$v = e;$$

$v \backslash e$	S	M	US	UM
S	+	—	—	—
M	+	+	—	—
US	+	—	+	—
UM	+	—	—	+

$$e \otimes e$$

$e \backslash e$	S	M	US	UM
S	+	+	+	+
M	+	+	—	—
US	+	—	+	—
UM	+	—	—	+

Упражнения.

3.1.1. Описать в виде структуры данные, содержащиеся в следующей таблице.

TABLE

A		B		C	D				
E	F	G			H	I	J	K	L
		M	N						

3.1.2. Восстановить по следующему описанию строение структуры и упростить нумерацию уровней:

DECLARE 1 STRUCTURE, 4 M, 9 P, 8 Q, 6 R, 3 S,
2 T, 4 U, 6 V, 5 W, 3 X, 3 Y, 6 V, 6 W;

3.1.3. Проинициализировать нулями все элементы структур, которым не присвоены начальные значения, и затем проинициализировать структуры, получаемые после раскрытия атрибута LIKE:

a) DECLARE 1 YES (3), 2 X (3), 3 Y, 3Z LIKE P,
1 P (9), 2 Q, 2 R;

b) DECLARE 1 NO, 2 I (3), 3 J, 3 K INITIAL (1, 2, 3), 2 L,
2 U, 3 V, 3 W (2) LIKE I;

3.1.4. Пусть дано следующее описание структур:

DECLARE 1 A, 2 C (5), 3 A (5), 3 C, 2 A,
1 B, 3 B (5), 3 C, 3 D,
1 G, 2 A (5), 3 C (5), 3 A, 2 H LIKE A.C;

Какие из приведенных ниже операторов являются ошибочными и почему? Установить полную квалификацию имен переменных, встречающихся в операторах, и их виды (S, M, US, UM).

- | | |
|---------------------|----------------------------|
| а) $H.C = D$; | б) $B.B = C.C - 7.7$; |
| в) $C.A = C.C$; | г) $B = B.D / 4$; |
| д) $A.C = H$; | е) $G.A = C + D$; |
| ж) $A.C = C.C(1)$; | з) $A.C = C.C$; |
| и) $B = B * H$; | к) $B = A.C(3), BY NAME$; |

3.2. Ситуации и их обработка

Во время выполнения программы PL/1 может возникнуть особое состояние—*ситуация*—в машине (такая, например, как деление на ноль, переполнение и т. п.), которое требует вмешательства операционной системы.

3.2.1. Ситуации.

Ситуации перечислены в 4.3, где даются условия их возникновения, стандартная реакция системы, а также указывается заблокирована («выключена») ли реакция системы на них по умолчанию или нет, и определяется возможность такого выключения. Ниже приводятся примеры на возникновение ситуаций.

```
DECLARE F65 FIXED (6,5) INIT (1.23456),
        XF6 FIXED (6,3) INIT (128.459),
        YE6 FLOAT (6) INIT (1E+40),
        E8 FLOAT (8) INIT (9.8765432E-40),
        B BIT (8), ZH CHAR (6) INIT ('1010'),
        VC (5) COMPLEX INIT (3-8I), N INITIAL (0);
```

YE6*YE6 /*OVERFLOW*/, так как результат $> 10^{75}$;

YE6** -2 /*OVERFLOW*/, так как $x^{**} - n \rightarrow 1/x^{**n}$;

E8/YE6 /*UNDERFLOW*/, так как результат $< 10^{-78}$;

YE6/N или F65/N /*ZERODIVIDE*/, так как $N = 0$;

F65 * F65 * F65 или 110B-1B/11B или $25 + 1/3$ /*FIXEDOVERFLOW*/ (о причинах возникновения переполнения см. в 2.2.1);

F65 = XF6 - 24; /* SIZE*/, так как при присваивании значения правой части, равного 104.459, отсекаются два старших значащих разряда (получается 4.45900). Ситуация не возникнет для оператора $F65 = XF6 - 124$; а также для $F65 = E8$; или $F65 = VC$;

$B = ZH$; /*CONVERSION*/, так как ZH получило значение '1010□□', не преобразуемое к BIT;

$F65 = '1.234567890654321'$; /*CONVERSION*/, так как фиксированное число, выделяемое из строки, неправильное ($p > 15$);

$E8 = VC(10)$; /*SUBSCRIPTRANGE*/, так как значение индекса больше границы, заданной при описании.

Отличием ситуации **FIXEDOVERFLOW** от **SIZE** является то, что **FIXEDOVERFLOW** возникает при выполнении операций в выражении, когда получается результат с фиксированной точкой больше максимально возможного для данного транслятора (15 для **DECIMAL** и 31 для **BINARY**). Ситуация **SIZE** возникает при присваивании значения переменной (в операторе присваивания, в заголовке цикла, при вводе, передаче аргументов), когда старшие значащие цифры теряются из-за несоответствия точности полученного при вычислении значения и объявленной точности для переменной, принимающей значение; ситуация **SIZE** может возникать и при преобразовании к фиксированной точке (см. 4.2).

Стандартной реакцией системы на возникновение ситуаций, связанных с ошибками в программе, являются ситуации **ERROR** и затем **FINISH**, после чего выполнение программы прекращается. Ситуация **ERROR** может возникнуть и при таких ошибках в программе, для которых в PL/I не установлено определенного имени ситуации. Ситуации **ERROR** и **FINISH** введены, в основном, для возможности задания пользователем своих действий в ответ на обнаружение системой ошибки при выполнении его программы (**ERROR**) или для выполнения завершающих действий непосредственно перед окончанием программы (**FINISH**)—см. ниже 3.2.3.

3.2.2. Включение и выключение ситуаций.

В начале выполнения программы возникновение некоторых ситуаций выключено (заблокировано), например: **SIZE**, **SUBSCRIPTRANGE**, **STRINGRANGE**; других—включено (см. 4.3). Программист имеет возможность в ходе выполнения программы включать или выключать ситуации (разрешать или запрещать их возникновение).

Включение и выключение ситуаций осуществляется с помощью *префиксов*, которые имеют вид метки или списка меток, заключенных в обоих случаях в скобки. Идентификаторы таких меток совпадают с названиями ситуаций (для включения) или имеют вид названия ситуации с добавлением в начало частицы **NO** (для выключения). Например:

(**NOUNDERFLOW**, **NOOVERFLOW**): $X = Y * Z * P$;

(**CONVERSION**): $L1: C = A \parallel B$;

(**NOZERODIVIDE**): (**SIZE**): $S, R = Q/I$;

Указанные префиксы блокируют возникновение ситуаций **UNDERFLOW**, **OVERFLOW**, **ZERODIVIDE** и разрешают возникновение ситуаций **CONVERSION** и **SIZE** при выполнении соответствующих операторов.

Следует учитывать, что операторы, при этом, всегда понимаются в смысле PL/I, то есть, например в групповом операторе

(**NOSIZE**): $DO V = S; R = 4.5 + F(V1); END$;

префикс относится только к DO-оператору, то есть к $V=S;$. Префикс условного оператора относится только к выражению, стоящему между IF и THEN, то есть в операторе

```
(NOOFL): IF Q < P **—5 THEN Q=Q * P;
                ELSE Q=Q/P;
```

— только к вычислению $P**—5$.

Если префиксы приписаны к началу блока, то их действие распространяется на все операторы блока и на внутренние блоки (но не относится к процедурам, вызываемым из выполняемого блока и не входящим в него). Но если операторы (или блоки), в свою очередь, имеют префиксы с противоположным смыслом, то они отменяют действие внешних префиксов.

Пример.¹

```
(SIZE): (NOCONV): BEGIN; DCL (X, Y) FIXED,
                (A, B) CHAR (5);
```

```
                . . . . .
```

```
                DO; X=A; Y=B!!C;
```

```
                (CONV): X=C+1;
```

```
                END;
```

```
                END;
```

```
(NOCONV): C: PROCEDURE RETURNS (CHAR (8));
```

```
                . . . . .
```

```
                END C;
```

```
                DECLARE C RETURNS (CHAR (8));
```

Возникновение ситуации CONVERSION заблокировано здесь во всех операторах блока и процедуры, кроме $X=C+1$; ситуация SIZE в процедуре C заблокирована, так как она выключена в нем по умолчанию.

Ситуация CHECK может быть включена только для блоков. В этом префиксе, предназначенном для отладки программ, программист может перечислить идентификаторы переменных, за изменениями которых ему нужно проследить. Например, если к блоку, приведенному выше, приписать префикс (CHECK (Y));, то после выполнения оператора $Y=B !! C$; возникает ситуация CHECK. Идентификатор Y может являться и именем массивной или структурной переменной (может быть и квалифицированным). При присваивании элементу массива печатается весь массив, а для элемента структуры—только этот элемент. При присваивании по INITIAL ситуация не возникает, но возникает при присваивании по псевдопеременной, в операторе ввода, в заголовке цикла. Имя, задаваемое в CHECK, не может быть параметром, но возможна проверка присваивания переменной, являющейся аргументом. Считается, что ситуация CHECK возникает в операторе CALL, если при выполнении процедуры, к которой происходит обращение, какому-либо аргу-

менту, фигурирующему в CHECK-префиксе и передаваемому по наименованию, присваивается значение. Необходимо, кроме того, чтобы при этом выход из выполняемой процедуры осуществлялся нормальным образом (по END или RETURN, но не по GO TO). Независимо от количества присваиваний в процедуре, ситуация возникает только один раз.

Если в префиксе CHECK указать идентификатор метки или входа, то состояние возникает перед выполнением оператора или процедуры с указанной меткой. В случае, если оператор имеет несколько меток, то при переходе на него по метке, не указанной в префиксе CHECK, ситуации не возникает (то же и для имен входов). Для меточной переменной, указанной в CHECK, ситуация возникает, если ее имя встречается в операторе GO TO (а не в операторе присваивания).

Префикс NOCHECK служит для блокирования возникновения ситуации в некотором блоке.

Пример.

```
(CHECK (A,B,L)): BEGIN; DECLARE
    A FLOAT, B FIXED,
    P ENTRY (FLOAT, FLOAT);
P: PROCEDURE (X, Y);
    DECLARE (X, Y) FLOAT;
    X=X+1; Y=Y+1;
    X=X**Y; GO TO L;
M: END P;
    GET LIST (A); AB=8;
    B=5; CALL P (A, B);
L: END;
```

Ситуация CHECK возникает при выполнении оператора GET, оператора B=5; и оператора END, помеченного меткой L; поскольку выход из процедуры P происходит не через END, то ситуация CHECK оператором CALL не вызывается, хотя A и меняется в теле процедуры (переменная B не меняется, поскольку передается в процедуру по значению, а не по наименованию как A). Если бы в операторе GO TO была указана метка M, а не L, то по оператору CALL возникала бы ситуация CHECK из-за изменения A в процедуре.

3.2.3. Оператор ON.

Как упоминалось выше, ситуации ERROR и FINISH, прекращающие выполнение программы, являются обычной реакцией системы на ошибки при выполнении программы, в том числе и на большинство из перечисленных ситуаций. Программист может задать свою реакцию на возникающие ситуации с помощью ON-оператора, имеющего вид:

ON имя-ситуации [SNAP] {s⁰ | SYSTEM;}

В случае возникновения указанной в операторе ситуации, при условии предварительного выполнения такого оператора в программе, будет производиться выполнение оператора s⁰ или стандартное системное действие (SYSTEM); кроме того, может быть сделана аварийная распечатка памяти—SNAP (в шестнадцатеричной системе).

Обычно в качестве оператора s⁰ (называемого ON-единицей) используется оператор ввода-вывода, блок (но без RETURN;), оператор вызова процедуры и т. п.; составной и условный операторы, а также ON-оператор не допускаются. Следует учитывать, что имена в s⁰ считаются относящимися к блоку, где находится ON-оператор, а не к блоку, где возникает ситуация, вызывающая выполнение s⁰. Сам оператор s⁰ всегда трактуется как блок, заключенный мысленно в операторы BEGIN и END; поэтому, если он даже и помечен, то переход на него извне все равно невозможен. После выполнения указанного оператора s⁰, исключая оператор перехода, происходит возврат в точку, где были прерваны вычисления (на вычисление выражения или на выполнение следующего оператора, или на ввод-вывод следующего элемента данных). Для ситуаций ERROR и FINISH происходит возврат на продолжение их обработки, то есть в конечном итоге на прекращение вычислений.

Действие некоторого ON-оператора (то есть задание необходимой реакции на возникающую ситуацию) продолжается до конца блока или до выхода из него по GO TO, или до тех пор, пока не выполнится следующий оператор ON в этом же блоке с тем же именем ситуации. Если этот следующий оператор выполняется во внутреннем блоке или в вызванном процедурном блоке, то после выхода (возврата) из него восстанавливается действие внешнего ON-оператора (или системное действие по умолчанию). Можно восстановить действие внешнего ON-оператора, отменив тем самым действие выполненного ON-оператора во внутреннем блоке, и по оператору:

REVERT имя-ситуации;

Пример.

BEGIN;

BEGIN; A=B/Z; ON ZERODIVIDE; P=Q/Z; END;

BEGIN; F=C/Z; ON ZERODIVIDE GO TO LA; CALL EX;

EX: PROCEDURE; ON ZDIV S=0; S=T/Z * S+T;

ON ZDIV SYSTEM; W=U/Z;

REVERT ZDIV; X=Y/Z;

END EX;

END;

LA: END;

При получении А, в случае деления на нуль, будет выполняться системная реакция на ZERODIVIDE; при получении Р (для $Z=0$) выполнится пустой оператор, Р присвоится неопределенный результат, и вычисления продолжатся; при вычислении F будет выполняться системная реакция (ERROR, FINISH); при делении T/Z—выполнится оператор $S=0$; и вычисления продолжатся; во время получения W произойдет системная реакция на ошибку; при получении X может осуществиться переход на LA (если $Z=0$).

З а м е ч а н и е. По SIGNAL-оператору программист может создать и свою собственную ситуацию, например, ситуацию SOS по оператору SIGNAL SOS;. Оператором ON указывается выполняемое для данной ситуации действие; в противном случае, так как ситуации, созданные программистом, не могут быть выключены, то по оператору SIGNAL выполняется стандартное системное действие.

Пример оператора ON для ситуации, созданной по SIGNAL-оператору:

```
ON GONDITION (SOS) PUT LIST (A, B, C);
```

Имя ситуации, определяемой программистом, получает по умолчанию атрибут EXTERNAL, и поэтому имя ситуации можно использовать в различных блоках программы.

Оператором SIGNAL можно вызвать и любые другие ситуации, что удобно использовать для отладки ON-операторов. Отметим следующую особенность: если для оператора SIGNAL FINISH; не задано ON-оператора, то оператор SIGNAL эквивалентен пустому оператору; но оператор SIGNAL для других ситуаций в подобном случае будет вызывать системную реакцию (если ситуация включена).

3.2.4. Функции для ситуаций.

В ON-операторах при обработке состояний удобно воспользоваться специальными встроенными функциями, которые перечислены в 4.1.5. Все эти функции не имеют параметров.

По функциям ONCHAR и ONSOURCE можно определить символ или строку, при обработке которых возникла ситуация CONVERSION. Используя такую функцию в качестве псевдопеременной, можно исправить ошибочный символ или всю строку. Например:

```
ON CONVERSION BEGIN;  
  IF ONCHAR='␣' THEN ONCHAR='0';  
  ELSE ONSOURCE='0'; END;
```

Если ошибочный символ—пробел, то он заменяется на нуль, иначе вся символьная строка заменяется на нуль, может быть, дополненный справа пробелами; если ошибочный символ не исправляется в ON-операторе, то при повторении преобразования,

автоматически осуществляемом системой, возникает ERROR. (Обратим внимание на следующий особый случай, когда повторное возникновение CONVERSION вызвано числом незаконной длины, содержащимся в преобразуемой (по $H \rightarrow A$) строке — в этом случае ситуация будет повторяться, и произойдет заикливание.)

Функция ONLOC позволяет определить процедуру, в которой возникла обрабатываемая ситуация.

3.2.5. Формы.

Обозначив ситуации через u (situation), получим следующие формы:

— общая форма для оператора PL/1:

$$s \sim \left[\left(\left\{ \begin{smallmatrix} u \\ \text{NO}u \end{smallmatrix} \right\}, \dots \right) : \right] \dots [l:] \dots s \text{ <без рекурсии>}$$

— для оператора REVERT:

$$s \sim \text{REVERT } u;$$

— для оператора SIGNAL:

$$s \sim \text{SIGNAL } u;$$

— для оператора ON:

$$s \sim \text{ON } u \text{ [SNAP] } \{s^0 \mid \text{SYSTEM};\}$$

З а м е ч а н и е. Следует отметить, что не все ситуации могут использоваться в качестве префикса: к таким относятся ситуации, которые не могут быть выключены, например, все ситуации ввода-вывода, ERROR, FINISH (см. 4.3).

3.2.6. Использование ситуаций при отладке.

Ситуации, наряду с операторами печати, являются основным средством при отладке программ. Рассмотрим осуществление с помощью ситуаций трех основных приемов отладки: аварийной печати (dump), печати в узлах (snapshot), слежения (trace).

Для аварийной печати (печати значений переменных в момент возникновения ошибки — ситуации ERROR) используется оператор вида:

ON ERROR оператор-печати

или

ON ERROR BEGIN; операторы-печати END;

В качестве операторов печати можно использовать любые известные операторы (см. 3.3), но в данном случае удобно задать оператор

ON ERROR PUT DATA;

по которому выдаются на печать значения всех переменных, известных в том месте программы, где стоит оператор (см. 3.3.3). Отметим, что для тех переменных, которым до возникновения ошибки ничего не было присвоено (в том числе и по INITIAL), будут печататься случайные значения, и вновь возможно возник-

новение ошибки. Для предупреждения возможного заикливания в этом случае необходимо в качестве ON-единицы задать

BEGIN; ON ERROR SYSTEM; PUT DATA; END;

Для печати в узлах — печати промежуточных результатов в выбранных местах программы — вместо обращения к процедурам печати можно использовать ситуацию CHECK для введенных специально отладочных переменных, например, @DEBUG1, @DEBUG2 и т. д. Включив ситуации CHECK с помощью префиксов вида:

(CHECK (@DEBUGi)):

приписанных к самому внешнему блоку, и задав в нужных местах программы операторы ON вида:

ON CHECK (@DEBUGi) i-я печать;

можно вызвать i-ю печать в нужном месте программы присвоением отладочной переменной произвольного значения или заданием оператора вида:

SIGNAL CHECK (@DEBUGi);

где $i = 1, 2 \dots$

Предлагаемый способ организации печати в узлах удобен тем, что, если отперфорировать префиксы на отдельных перфокартах, то включение и отключение какой-либо отладочной печати производится просто подкладыванием или устранением соответствующей перфокарты, находящейся в начале отлаживаемой программы.

Печать в узлах можно было бы организовать и с помощью ситуации CONDITION (тогда не понадобились бы префиксы и отладочные переменные), но так как отключение таких ситуаций невозможно, то для того, чтобы убрать отладочную печать, пришлось бы удалять или изменять операторы внутри программы, что неудобно.

Слежение за изменением значений некоторых переменных или за последовательностью выполнения помеченных операторов осуществляется с помощью ситуации CHECK, в префиксе которой задаются имена интересующих нас переменных и меток. Сплошная трассировка операторов на некотором участке программы в PL/1 отсутствует. Печать по CHECK происходит каждый раз с новой строки.

Можно осуществить и выборочное слежение за изменением некоторых переменных, выполняя отладочную печать только для определенных значений контролируемой переменной. Например, по

ON CHECK (Q) BEGIN; IF Q=0 THEN PUT

DATA (Q, X, Y); END;

отладочная печать выполняется только при $Q=0$.

В диагностике, которая печатается при возникновении большинства ситуаций, указывается номер оператора, в котором воз-

ника ситуация. Ошибка тем самым быстро локализуется; в противном случае, используя ON-оператор, можно получить дополнительную информацию. Отметим, однако, что при наличии ON-оператора системная диагностика, содержащая определенную полезную информацию, не выдается.

Упражнения.

Пусть имеется следующая программа:

```
P: PROCEDURE OPTIONS (MAIN);
  . . . . .
Q: BEGIN;
  . . . . .
  IF X < Y/Z THEN X=Y/Z; ELSE X=Z/Y;
  S: BEGIN; . . . . . CALL T;
    DO X=Y*Z TO X**Y; Y=Y*Z; END;
  . . . . .
  END S;
END Q;
T:  PROCEDURE;
  . . . . . X=X; . . . . .
  GET LIST (X, Y, Z);
END T;
END P;
```

Дополнить программу таким образом, чтобы она удовлетворяла одному из следующих условий:

3.2.1. В операторе IF и блоке S могла бы возникать ситуация OVERFLOW, а в операторе цикла и в остальных операторах программы P—нет; при возникновении ситуации в операторе IF напечатать Y и Z и нормально окончить выполнение программы, а для ситуаций блока S выполнить системную реакцию.

3.2.2. Каждый раз при присваивании переменной X в блоке Q, исключая присваивание в блоке S, печатать ее новое значение и имя, а в процедуре T печатать значения X, Y и Z.

3.3. Ввод и вывод

Здесь мы ограничимся только рассмотрением ввода исходных данных с перфокарт и вывода результатов на печать. (В PL/I под вводом-выводом подразумевается также обмен информацией с внешней памятью—магнитными лентами и дисками).

3.3.1. Операторы ввода-вывода.

Ввод и вывод осуществляется соответственно операторами GET и PUT, которые имеют следующую общую форму:

```
{GET
PUT} (спецификации-данных) (опции);
```


Спецификации определяют данные, которые участвуют в операциях ввода-вывода, а опции указывают на некоторые дополнительные возможности операторов; необходимо наличие хотя бы одной из упомянутых частей оператора (они могут быть и переставлены).

Имеется 3 способа ввода-вывода данных:

1) *списком* (LIST); этим способом мы пользовались в 1-й главе (см. 1.5.2);

2) *данными* (DATA);

3) *редактированием* (EDIT).

С учетом этих способов и имеющихся опций формы операторов ввода-вывода могут быть представлены в следующем виде:

GET	$\left\{ \begin{array}{l} \text{LIST } (\text{э}, \dots) \\ \text{DATA } [(\text{э}, \dots)] \\ \text{EDIT } \{(\text{э}, \dots) (\Phi, \dots)\} \dots \end{array} \right\}$	[COPY] (SKIP [(eS)]);	
PUT		$\left\{ \begin{array}{l} \text{LIST } (\text{э}, \dots) \\ \text{DATA } [(\text{э}, \dots)] \\ \text{EDIT } \{(\text{э}, \dots) (\Phi, \dots)\} \dots \end{array} \right\}$	(SKIP [(eS)] LINE (eS) PAGE [LINE (eS)]);

В правой части операторов приводятся допустимые опции. Через э, .. обозначен список *элементов данных*. При вводе элементами обычно являются проблемные переменные (или псевдопеременные), которым присваиваются вводимые значения (может быть, после необходимых преобразований), а при выводе — выражения, значения которых выводятся на печать. Наряду со скалярными элементами (переменными, выражениями) могут фигурировать также массивные и структурные. Кроме того, элементом данных может являться и *DO-элемент*, с которым мы познакомимся чуть позже.

Таким образом:

э-ввода $\sim \{vO | \bar{i}WO | DO\text{-элемент}\}$

э-вывода $\sim \{e | DO\text{-элемент}\}$

Некоторые ограничения на вид элементов [будут приведены ниже при рассмотрении конкретных способов ввода-вывода.

Через Φ , .. в формах операторов обозначен *список форматов*, указывающий на способы представления вводимых и выводимых данных на внешнем носителе и на необходимые преобразования. Форматы используются только при редактируемом вводе-выводе; там они и рассматриваются.

В дальнейшем, информация, поступающая на ввод или выдаваемая на печать, называется соответственно *входным* и *выходным потоками*. Входной и выходной потоки предполагаются состоящими из непрерывной последовательности символов, допустимый набор которых и возможные сочетания будут уточняться при рассмотрении каждого способа ввода-вывода.

Линией (LINE) будем называть перфокарту (при вводе) или строчку печати (при выводе). Линия при вводе содержит 80 символов, а при выводе — 120 символов. Порцию из 60 последовательных печатаемых линий называют *страницей* (PAGE). Приведенные числовые характеристики линии и страницы могут быть изменены программистом по специальным операторам, которые здесь не рассматриваются.

Последовательность ввода-вывода. Данные вводятся (выводятся) в порядке, соответствующем их появлению в списке элементов данных. Значения массивов (и массивные выражения) вводятся (выводятся) по строкам, значения структур — в порядке описания их элементов, значения массивов структур — по строкам и с учетом порядка описания элементов структур.

Пример.

```
DECLARE 1 X(3), 2 Y, 2 Z, 1 U, 2 V(3), 2 W(3);
```

```
PUT LIST (X, U);
```

Печать производится в следующем порядке:

X.Y(1), X.Z(1), X.Y(2), X.Z(2), X.Y(3), X.Z(3),

U.V(1), U.V(2), U.V(3), U.W(1), U.W(2), U.W(3).

Для оператора ввода GET LIST (X, U) данные в потоке должны быть расположены в указанном порядке.

DO-элемент. Рассматриваемый ниже DO-элемент используется, в основном, для ввода-вывода некоторых частей массивов в произвольном порядке. Пусть, например, требуется ввести массив по строкам, а вывести его по столбцам. Тогда эта задача может быть решена следующим образом:

```
DECLARE A (0:N, 0:N);
```

```
GET LIST (A);
```

```
PUT LIST (((A (I, J) DO I=0 TO N) DO J=0 TO N));
```

Приведенный оператор вывода эквивалентен следующим операторам PL/I:

```
DO J=0 TO N; DO I=0 TO N; PUT LIST (A(I, J)); END; END;
```

Форма для DO-элемента имеет следующий вид:

$\text{э-DO} \sim (\text{э}, \dots \hat{\text{s-DO}} \langle \text{с параметром цикла} \rangle)$

Здесь в качестве элемента э могут использоваться любые допустимые для конкретного способа ввода-вывода элементы данных, в том числе и сам DO-элемент. В форме $\hat{\text{s-DO}}$ представляет из себя DO-оператор с параметром цикла («заголовок группового оператора») без ограничивающей его точки с запятой; круглые скобки, заключающие DO-оператор, определяют сферу его действия.

В DO-элементе (и в других элементах ввода, например, в форматах или в качестве аргументов псевдопеременной) могут быть использованы и данные, введенные ранее, может быть, и в том же самом операторе ввода. Например:

```

DECLARE B (1:N, 1:N), C (0:N) INITIAL (1);
GET LIST (K, ((B(I, J) DO J=1 TO I),
              C(I) DO I=1 TO K WHILE (C(I-1) > 0)));

```

Осуществляется ввод с заполнением $K(K+1)/2$ элементов матрицы B , находящихся на главной диагонали и ниже ее ($1 \leq I, J \leq K \leq N$), а также массива C ; в случае, если введенный элемент массива C не положителен, ввод прекращается (нулевой элемент массива полагается равным единице).

Замечание. Все, сказанное выше, относится и к вводу-выводу с редактированием (PUT EDIT и GET EDIT), и к выводу данными (PUT DATA) (см. ниже).

Опции операторов ввода-вывода. Выполнение всех опций, кроме COPY, осуществляется перед выполнением операций ввода-вывода.

COPY—контрольная печать вводимых данных. Каждое вводимое значение печатается в том виде, в котором оно пробито на перфокарте. Каждое скалярное значение (включая и элементы массивов и структур) печатается с новой строчки. Опция используется при отладке программ.

SKIP[(eSI)]—пропускается указанное (в eSI) количество линий (перфокарт или строк печати), считая и текущую. Следующая операция ввода-вывода будет выполняться с 1-й позиции требуемой линии. Если e не задано, то предполагается, что $e=1$ (осуществляется переход на начало следующей линии). В случае, если $e \leq 0$, то при вводе полагается $e=1$, а при выводе осуществляется печать с начала текущей строчки и с наложением символов. Если при выполнении SKIP страница исчерпывается, то подводится начало следующей страницы и возникает ситуация ENDPAGE (см. 4.3.2).

LINE (eSI)—подвести линию с заданным (в eSI) номером, считая от начала страницы. Если значение e меньше или равно текущему номеру линии или больше допустимого, то осуществляется переход к началу следующей страницы и возникает ситуация ENDPAGE (см. 4.3.2). При $e \leq 0$ полагается $e=1$. Можно сказать, что опция LINE служит для подвода требуемой линии путем задания ее абсолютного номера, а SKIP—относительного.

PAGE—подвод первой линии следующей страницы; PAGE перед началом печати в программе дает пустую страницу.

3.3.2. Ввод-вывод списком.

Ввод-вывод списком вкратце был уже рассмотрен в 1-й главе. Здесь мы сделаем лишь необходимые уточнения и добавления. Элементы данных (э), допустимые для этого способа, были охарактеризованы в 3.3.1.

Входной поток. Во входном потоке наряду с числами (арифметическими константами) могут находиться и строки (строчные константы), причем комплексные числа не могут содержать пробелы, а в строках не могут быть использованы повторители строки. Числа и строки могут отделяться друг от друга не только пробелами, но и запятой (может быть, вместе с пробелами).

Дадим форму для входного потока:

входной-поток $\sim [\square |,] \dots \{ \tilde{c}0 \{ \square |, \} \dots \} \dots$

$\langle \tilde{c}0 \sim c0$ без (ñID) в g или без \square в nC)

Если между двумя запятыми не содержится никакой константы, то это указывает на пропуск присваивания очередному элементу данных. Допускается размещение одного числа или строки на соседних перфокартах. Напомним, что если атрибуты вводимого значения не совпадают с атрибутами соответствующей переменной, то производятся необходимые преобразования (см. например 4.2).

Выходной поток. Печать символьных строк производится без апострофов (во «внутреннем» виде); битовые строки печатаются с апострофами и буквой В (во «внешнем» виде). Двоичные числовые значения выводятся в десятичном виде.

Фактически, перед засылкой в выходной поток значения, отличные от символьных строк, преобразуются к символьно-строчному виду ($A \rightarrow H$ и $T \rightarrow H$ —см. 4.2); кроме того, в конец строки всегда добавляется один пробел. Таким образом, если обозначить длину этого выводимого строчного значения через l^0 , то количество символов, засланных в выходной поток будет равно:

- для символов— $l^0 + 1$;
- для битов— $l^0 + 4$ (добавляются 2 апострофа и буква В);
- для числовых значений— $l + 1$ (l см. в 4.2.3).

Печатаемая линия (строчка АЦПУ) разделена на 5 полей, по 24 позиции в поле, и печать значения производится всегда с начала очередного поля, с позиций 1, 25, 49, 73, 97 (и, тем самым, в поток после выводимого значения могут вставляться дополнительные пробелы). Если количество выводимых в поток символов превышает 24 и, следовательно, захватывает следующее поле (поля), то очередное значение выводится, начиная с нового поля. Отметим, кроме того, что если выводимое строчное значение не умещается в очередных полях текущей линии, то его остаток печатается на следующей линии; однако, числовое значение в этом случае печатается целиком на новой линии. Например, если выводится массив символьных строк с длиной, равной 24, то в одной линии на печати будет располагаться не 5 элементов массива, а попеременно 3 или 2, поскольку печатаемое значение занимает 2 соседних поля; массив комплексных значений

с точностью более 8 для FIXED или 5 для FLOAT будет печататься в 2 столбца (см. формулы для l в 4.2.3).

Форма для выходного потока:

выходной-поток $\sim [_]\dots\{\tilde{c}0 _ \dots\}\dots$

$\langle \tilde{c}0 \sim \tilde{c}0, \text{ но } gH \sim \xi \dots \rangle$

Пример.

DCL 1 P(2), 3 Q FLOAT, 3 R CHAR (8), 3 S BIT (5),

T(4) FIXED (3) CPLX;

GET LIST (P(1), T(3)) COPY;

/*ВХОДНОЙ ПОТОК: $_ 1E-10 _ \text{'ПРАВЫЙ'}$,

'1010'B, 25, _ _ ,

$_ , 3.5E2 + 2.1E1 _ _ 3.9 \text{'}$ */

/*ПЕЧАТЬ ПО COPY (В СТОЛБЦЕ): $1E-10, \text{'ПРАВЫЙ'}$,

$\text{'1010'B, 25, 3.5E2 + 2.1E1 \text{'}}$ */

/*ВВОД ПО GET: P(1).Q = $1E-10$, P(1).R = 'ПРАВЫЙ _ _ ,

P(1).S = '10100'B ,

P(2).Q = $25E0$,

T(3) = $350 + 021I \text{'}$ */

/*ПЕРЕМЕННЫЕ P(2).R и P(2).S ОСТАЛИСЬ

НЕИЗМЕННЫМИ*/

L = 1; PUT LIST (P.R(L), S(1), T(L+2)) SKIP(2);

/*ПЕЧАТЬ: (ПРОПУСК ДВУХ СТРОЧЕК), ПРАВЫЙ $_ _$,

$\text{'10100'B, 350 + 21I \text{'}}$ */

3.3.3. Ввод-вывод данными.

Входной поток. Во входном потоке перед вводимым значением должна находиться скалярная переменная, к которой относится это значение, и знак равенства, например: $X=2.5$ или $Y(3,8)=1E-4$ или $G.T.P=\text{'1'B}$. Такие «данные» в виде операторов присваивания без точки с запятой отделяются друг от друга пробелами или запятой (может быть, с пробелами); в конце потока данных, относящегося к одному оператору ввода, ставится точка с запятой. Вводимые значения имеют вид проблемных констант $\tilde{c}0$ (без пробелов внутри комплексного числа и без повторителя строки) и могут располагаться на разных перфокартах.

Форма:

входной-поток $\sim \{[_] \dots vSO = [_ \dots] \tilde{c}0 [_ \dots] \} \{[_ \dots |,] \dots$

Имена скалярных переменных, фигурирующих в потоке, должны быть квалифицированы полностью, а для массивных структур они должны иметь непрославляемые индексы, имеющие вид целых десятичных чисел; последнее необходимо и для обычных массивов. Кроме того, имена переменных должны быть известны в том месте программы, откуда производится ввод. Если атрибуты вводимого значения не совпадают с атрибутами заданной перемен-

ной, то производятся необходимые преобразования (см. например 4.2).

Входные данные. При вводе список элементов данных обычно отсутствует, так как переменные, которым присваиваются вводимые значения, отперфорированы вместе со значениями. Если же список данных присутствует, то он используется лишь для контроля ввода: если во входном потоке встретится имя переменной, которое отсутствует среди имен в списке данных, то возникает ситуация NAME (см. 4.3.2).

В списке данных, если он есть, могут фигурировать, причем, в любом порядке, только имена переменных: скалярных, массивных или структурных, если необходимо, то квалифицированных (может быть, и неполностью). То есть, спецификация данных здесь имеет вид:

DATA [(IVO, . .)]

где IVO — имя (в частном случае идентификатор) проблемной переменной:

IV ~ [IU.] . . . {IS | IM | IU}

Ограничение. Имя в списке данных не может быть параметром процедуры.

Пример.

DCL 1 A (8), 2 B, 2 C, D CHAR (8) VAR, E (10) BIT (3);

GET DATA; GET DATA (A, D, F);

/*ВХОДНОЙ ПОТОК: D='BOR' □ □ E (1)='01'B;

A.B (7)=16, A.C (4)=6 □ E (8)='0'B; */

Первый оператор ввода присвоит исходные значения переменной D и первому элементу массива E; второй оператор присвоит значения двум элементам структурного массива A, а при вводе переменной E (8) возникнет ситуация NAME.

Выходные данные. Данными, выводимыми на печать, являются переменные; выражения не могут присутствовать в списке данных. То есть, элемент списка данных имеет вид:

э~{v0 | э-DO}

Если список данных отсутствует, то выводятся все переменные, известные в том месте блока, где находится оператор вывода, и во внешних блоках; даже те переменные, имена которых совпадают с именами, описанными явно во внутреннем блоке. Например, по оператору PUT DATA;, расположенному в блоке, где находится описание, приведенное выше, помимо значений A, D и E будут выведены и переменные известные во внешних по отношению к данному блокам, даже если они имеют имена совпадающие с A, D, E; для массива структур — вывод по непротславляемым индексам.

Выходной поток. Данные в выходном потоке имеют тот же вид, что и во входном потоке (см. выше), но они отделяются друг

от друга только пробелами (может быть и одним); в конце выходного потока для каждого оператора вывода печатается точка с запятой. Напомним, что каждое выводимое значение снабжается при печати его именем (для элементов структур — полностью квалифицированным), а для элементов массивов — кроме того, и индексами. Такая форма выдачи особенно удобна при отладке, хотя и занимает больше места при печати.

Печать данных, как и для PUT LIST (см. 3.3.2), осуществляется в 5 столбцов, начиная с позиций 1, 25, 49, 73, 97; если печатаемое значение занимает и следующий столбец (столбцы), то количество значений в печатаемой строчке соответственно уменьшается. Выводимые значения преобразуются к символьно-строчному виду (см. в 4.2.3 $A \rightarrow H$ и $T \rightarrow H$), но для арифметических данных апострофы не печатаются; значения строчных данных печатаются всегда с апострофами (и с буквой В для битовых данных), то есть во «внешнем» виде (ср. с 3.3.2).

Форма для выходного потока:

выходной-поток $\sim \{vSO = [_ \dots] \tilde{c}O _ \dots \} \dots$;

Пример. Выходной поток, совпадающий со входным в примере, приведенном выше (с точностью до количества пробелов и замены запятых на пробелы), мог бы быть получен по следующим операторам:

PUT DATA (D, E (1));

K=7; J=8; PUT DATA (B (K), A.C (4), E (J));

3.3.4. Ввод-вывод с редактированием.

Элементы данных (э) были охарактеризованы в 3.3.1 (они совпадают с элементами данных при управлении списком).

Существенным отличием ввода-вывода с редактированием от других способов является то, что при вводе значения, находящиеся во входном потоке, а при выводе значения, записываемые в выходной поток, подвергаются преобразованиям, указанным в списке форматов (см. 3.3.1). Каждому элементу данных соответствует свой элемент формата; кроме того, имеются и управляющие форматы, предназначенные для специальной обработки данных в потоках.

Редактирование дает возможность программисту оформлять вводимую и выводимую информацию в виде, наиболее подходящем для каждой задачи, в частности, вводить данные только из определенных полей перфокарт и располагать печатаемые значения в требуемых позициях на странице.

Форматы данных. Сначала мы введем обозначения величин, которые будут использоваться при описании форматов:

w — количество символов, считываемых из входного или записываемых в выходной поток;

\bar{p} — количество цифр в десятичном числе (или мантиссе числа), принадлежащего входному или выходному потокам;

\bar{q} — количество цифр в дробной части десятичного числа (или мантиссы), принадлежащего входному или выходному потокам;

t — определяет масштаб (10^t), на который умножается вводимое или выводимое значение.

Все перечисленные величины *) в нижеследующих форматах могут являться скалярными выражениями, приводимыми к целым действительным значениям, то есть:

$$\{\omega \mid \bar{p} \mid \bar{q} \mid t\} \sim \text{eSI}$$

Выражения вычисляются в момент использования формата, то есть непосредственно перед вводом-выводом соответствующего элемента данных.

(1) *Формат данных с плавающей точкой.*

$$\Phi E \sim E(\omega, \bar{q}[, \bar{p}])$$

При вводе \bar{p} игнорируется. Если \bar{p} не указано, то при выводе полагается $\bar{p} = \bar{q} + 1$, то есть точка устанавливается за первой старшей (и неравной нулю) цифрой значения. Необходимо чтобы выполнялось соотношение $\omega \geq \bar{p} + 6$, иначе возможна ситуация SIZE (ср. с A \rightarrow H в 4.2.3). Шесть дополнительных символов при изображении значения с плавающей точкой (с точностью \bar{p}) отводятся на знак числа, точку, букву E, знак порядка и две цифры порядка. При $\bar{q} = 0$ для положительных значений достаточно $\omega \geq \bar{p} + 4$, так как не печатается ни знак, ни точка. Кроме того, должно быть $\bar{p} \leq 16$ ($\bar{q} \leq 15$), иначе возникает ситуация CONVERSION.

Во входном потоке среди считанных ω символов должно находиться действительное десятичное число (nRD), может быть, окаймленное пробелами, в котором буква E может быть опущена (но знак перед порядком в этом случае необходим). Если все ω символов содержат пробелы, то возникает ситуация CONVERSION.

Форма для ω символов входного потока (ср. с 2.1.7):

$$\text{входной-поток} \sim [_ \dots] \text{ nRDF } \left[\left\{ \begin{array}{l} [E] \{ \pm \} \\ E [\pm] \end{array} \right\} \bar{n}DI \right] [_ \dots]$$

*) В документации они соответственно обозначены через ω , s , d , p . Введение обозначений \bar{p} и \bar{q} указывает на их аналогию с p и q : как p и q характеризуют внутреннюю точность представления данных, так \bar{p} и \bar{q} указывают на точность их представления на внешних носителях (но всегда $\bar{q} \geq 0$ и $\bar{p} > \bar{q}$). Продолжая аналогию, можно было бы и ω обозначить через \bar{l} .

Если мантисса введенного числа содержит точку, то именно она принимается во внимание, а \bar{q} игнорируется.

Ниже приводятся примеры редактируемого ввода значений с плавающей точкой. Здесь и далее даются содержимое потока (в котором обычно указывается более чем w символов), используемый формат, передаваемое отредактированное значение (реальное значение будет определяться атрибутами переменной, принимающей это значение).

В потоке	Формат	Значение
—123456E7890	E (9,5)	—1.23456E7
123.456E7890	E (9,5)	+123.456E7
□—123456E7□□	E (12,3)	—123.456E7
□—123456+7□□	E (11,3)	—123.456E7
□+123456E7□□	E (7,3)	+12.345

В выходной поток выдается округленное до \bar{p} цифр десятичное значение с плавающей точкой, с $p-q$ цифрами перед точкой и с \bar{q} цифрами после нее; знак плюс перед мантиссой опускается. Старшая цифра ненулевого значения всегда отлична от нуля.

Форма для выходного потока (w символов):

выходной-поток $\sim [\square\dots] nRED$

Ниже приводятся примеры редактируемого вывода значений с плавающей точкой.

Значение	Формат	В потоке
+1.23456E4	E (12,5)	□1.23456E+04
—1.23456E4	E (13,4)	□□—1.2346E+04
—1.23456E4	E (10,4)	ОШИБКА (SIZE)
+1.23456E4	E (13,7)	1.2345600E+04
+1.23456E4	E (13,2,4)	□□□□12.35E+03

Пример.

PUT EDIT (P, P, P) (E (10,3), E (13,5), E (12, 4, 6));

Если P равно —214.68, в выходной поток будет выдано:

—2.147E+02□—2.14680E+02—21.4680E+01

По оператору

GET EDIT (P, Q, R, S) (E(11, 4), E (12, 0), E(5, 10), E (7, 1, 8));

если взять в качестве входного потока приведенный только что выходной поток, переменные получают следующие значения:

P = — 2.147E2, Q = — 2.14680E2, R = — 2.14E1,

S = 68.0E+1

Напомним, что при наличии точки в потоке \bar{q} в формате игнорируется.

(2) Формат данных с фиксированной точкой.

$\Phi F \sim F(w[\bar{q}, t])$

Если \bar{q} или t не указаны, то полагается $\bar{q}=0$ или $t=0$.

Во входном потоке среди считанных w символов должно находиться действительное десятичное число с фиксированной точкой ($nRDF$). Если все w символов содержат пробелы, то вводится нулевое значение. Если среди символов потока содержится точка, то \bar{q} игнорируется.

Форма для потока (w символов):

входной-поток $\sim [\square \dots] nRFD [\square \dots]$

Ниже приводятся примеры редактируемого ввода значений с фиксированной точкой.

В потоке	Формат	Значение
—1234.56789	F (8)	—1234.56
—1234.56789	F (5)	—1234
$\square\square 1234.56\square\square$	F (10)	+1234.56
$\square-123456\square\square\square$	F (10, 3)	—123.456
$\square-123456\square\square$	F (10, 3, —2)	—1.23456
$\square-1234.56\square\square$	F (10, 3)	—1234.56

В выходной поток выдается округленное до \bar{q} цифр после точки десятичное значение; незначащие нули и знак + опускаются, а знак минус сдвигается вправо. Чтобы выводимое значение умещалось в поле из w символов, необходимо, чтобы выполнялось условие $w \geq \bar{p} + 3$ (ср. F \rightarrow H п. 4.2.3), иначе возможно возникновение ситуации SIZE. Здесь \bar{p} определяется в зависимости от количества выводимых старших значащих цифр перед точкой и от \bar{q} . Если обозначить через v количество пробелов, заменяющих старшие незначащие нули, то $\bar{p} = p - q - v + \bar{q}$; если v неизвестно, то следует принять $w \geq p - (q - \bar{q}) + 3$.

Форма для потока (w символов):

выходной-поток $\sim [\square \dots] nRFD$

Ниже даются примеры редактируемого вывода значений с фиксированной точкой.

Значение	Формат	В потоке
—1234.56	F (9,2)	$\square-1234.56$
+1234.56	F (10,4)	$\square 1234.5600$
+1234.56	F (9,1)	$\square\square\square 1234.6$
—1234.56	F (6)	$\square-1235$
—1234.56	F (4)	ошибка (SIZE)
—1234.56	F (9,2, —1)	$\square\square-123.46$

Пример.

PUT EDIT (X, Y, Z) (F (6), F (6, 1), F (7,2,1));

Если X, Y и Z равны 214.68, то в выходной поток будет выдано
 $\square\square\square 215\square 214.72146.80$

Если бы было Y = —999.99 или Z = —214.68, то возникло бы SIZE.

Если предположить, что входной поток совпадает с только что полученным выходным, то по оператору

GET EDIT (A (I), B.C, Q (5).S (I).D)

(F (7,1), F (7), F (5,1,2));

получим $A(I) = 21.5$, $B.C = 214.721$, $Q(5).S(I).D = 4680$ (точка в потоке «главнее» \bar{q} в формате). Формат F (8) для A (I) был бы ошибочен (ситуация CONVERSION), так как число имело бы внутри себя пробел (215□2).

(3) Формат комплексных данных.

$\Phi C \sim C(\Phi R[, \Phi R]) \langle \Phi R \sim \{\Phi E | \Phi F\} \rangle$

Формат для комплексного значения содержит два формата для действительных значений, рассмотренных ранее. Первый формат относится к действительной части, а второй — к мнимой части комплексного значения; если задан лишь один формат, то он относится к обеим частям комплексного значения.

Из входного потока считываются два действительных числовых значения в соответствии с указанными двумя форматами и присваиваются действительной и мнимой частям комплексной переменной. Наличие буквы I в мнимой части значения является ошибкой.

Форма для входного потока (2w символов):

входной-поток ~ [□...] nRD [□...] nRD [□...]

Приведем примеры редактируемого ввода для комплексных значений.

В потоке	Формат	Значение
□□12.3□□—45E—2□	C(F(7), E(8,1))	+12.3—4.5E—2I
□□12.3□—456□□□	C(F(7, 2, —1))	+1.23—0.456I

В выходной поток выдаются два действительных числовых значения в виде, описанном выше, то есть без буквы I после мнимой части.

Форма для 2w символов выходного потока:

выходной-поток ~ [□...] nRD [□...] nRD

Приведем примеры редактируемого вывода для комплексных значений.

Значение	Формат	В потоке
—1.234E2+5.678E1I	C(E(10,2))	□—1.23E+02□□5.68E+01
—1.234E2+5.678E1I	C(F(10,2))	□□□—123.40□□□□56.78
—1.234E2+5.678E1I	C(F(5), F(2))	□—12357

Пример.

DECLARE (ROM, DOM) COMPLEX FIXED (3, 1);

PUT EDIT (ROM, —3*ROM) (C(E(10, 3), F(10,3)), C(F(7,1)));

Для ROM=3.4—4.5I выдается:

□3.400E+00□□□□—4.500□□—10.2□□□13.5

Ввод этого потока может быть осуществлен, например, опера-

тором

GET EDIT (ROM, DOM) (C(E(10,3)), C(F(8), F(6)));

(4) *Формат битовых данных.*

$\Phi T \sim B[(w)]$

Во входном потоке среди w символов должно находиться битовое значение (битовая константа без апострофов и буквы В — во «внутреннем» виде), может быть, окаймленное пробелами; иначе возможна ситуация CONVERSION. Задание w обязательно; $w \leq 0$ соответствует пустой строке, которая затем будет приведена к нулевой строке или к пустой при VARYING.

Форма для входного потока (w символов):

входной-поток $\sim [_ \dots] \{0 \mid 1\} \dots [_ \dots]$

Ниже даются примеры редактируемого ввода битовых строк.

В потоке	Формат	Вид значения
1011011	B(3)	'101'B
$_ 1011 _ _$	B(3)	'10'B
$_ 1011 _ _$	B(6)	'1011'B
$_ 1011 _$	B(1)	ошибка
$_ 101234$	B(4)	'101'B

В выходной поток выдается битовое значение (может быть, после преобразований $A \rightarrow T$ или $H \rightarrow T$); если w отсутствует, то количество выдаваемых символов (l) определяется по атрибутам выводимого элемента данных (может быть, и преобразованного). Если $w \leq 0$, то выдается пустая строка. В случае $w < l$ от значения отсекаются правые биты; при $w > l$ дополнение справа производится пробелами (а не нулями).

Форма для выходного потока:

выходной-поток $\sim \{0 \mid 1\} \dots [_ \dots]$

Ниже даны примеры редактируемого вывода битовых строк.

Вид значения	Формат	В потоке
'101'B	B(3)	101
'101'B	B	101
'101'B	B(5)	101 $_ _$
'101'B	B(2)	10

Пример.

DCL SB BIT (5) INIT ('1101'B), RK (3) BIT (2);

PUT EDIT (SB)(B(7)); /*ВЫДАЕТСЯ: 11010 $_ _$ */

Если во входном потоке содержится 11010 $_ _$, то по

GET EDIT (RK)(B(1))(SB) (B(4));

получим $RK(1) = '10'B$, $RK(2) = '10'B$, $RK(3) = '00'B$, $SB = '10000'B$. Формат B(1) относится ко всем элементам массива RK (используемый вид спецификации данных поясняется на стр. 135). Из входного потока для трех элементов массива RK выбираются соответственно 1, 1, 0, а для SB выбирается 10 $_ _$; выбранные значения приводятся затем к атрибутам, указанным в описании.

(5) *Формат символьных данных.*

$\Phi H \sim A[(\omega)]$

Из входного потока вводятся ω символов и присваиваются переменной в списке данных; пробелы являются равноправными символами, то есть входной поток (ω символов) имеет следующую форму:

входной-поток $\sim \xi \dots$

Задание ω обязательно; $\omega \leq 0$ соответствует пустой строке, которая затем приводится к «пробельной» строке или к пустой для VARYING.

Дадим примеры редактируемого ввода символьных строк.

В потоке	Формат	Вид значения
ABCDEFGH	A(5)	'ABCDE'
▢ABCD▢▢▢	A(7)	'▢ABCD▢▢'
▢ABCD▢▢▢	A(2)	'▢A'
12345678	A(2)	'12'

В выходной поток выдается символьное значение (символьная строка во «внутреннем» виде—без удвоения внутренних и без внешних апострофов); если ω отсутствует, то количество выдаваемых символов (l) определяется по атрибутам элемента данных (может быть, и преобразованного). Для $\omega \leq 0$ выдается пустая строка. В случае если $\omega \neq l$, символы отсекаются (или добавляются пробелы) справа.

Форма выходного потока (ω символов):

выходной-поток $\sim \xi \dots [\dots]$

Дадим примеры редактируемого вывода символьных строк.

Вид значения	Формат	В потоке
'ABCD'	A	ABCD
'ABCD'	A(6)	ABCD▢▢
'ABCD'	A(2)	AB
'▢ABCD▢▢'	A(2)	▢A
'▢1234▢▢'	A	▢1234▢▢

Пример.

DCL K FIXED (2) INITIAL (5).

(W, LH INITIAL ('ВЫВОД')) CHAR (5);

PUT EDIT (K+1, LH, 'ВВОД', 'ВВОД') (F(5), A(K+2),
A, A(2));

/*ВЫДАЕТСЯ: ▢▢▢▢6 ВЫВОД▢▢ВВОДВВ*/

Если предположить, что во входном потоке содержится то, что было выдано в выходной, то по

GET EDIT (K, W, SUBSTR (LH, 1, 3)) (F(5), A(K), A(3));
получим: K=6, W='ВЫВОД', LH='▢ВВОД'

Общее замечание. Для арифметических форматов (E и F) выводимое значение располагается у правого края выводимого

поля из w символов, а для строчных (А и В)—у левого; отсекаются соответственно левые или правые лишние символы (для Е и F ими могут быть только пробелы—иначе может возникнуть ситуация SIZE). Для формата С оба действительных выводимых значения располагаются у правого края указанных полей (ср. с выводом для LIST и DATA).

Управляющие форматы. Управляющие форматы не относятся к каким-либо элементам данных: они управляют движением входного или выходного потока между передачей информации, относящейся к соседним элементам данных. Поэтому управляющий формат будет работать только в том случае, если будет работать следующий за ним формат данных.

(6) *Формат пропуска позиций.*

$\Phi \sim X(eSI)$

При вводе пропускается e символов; при выводе—в выходной поток вставляются (печатаются) e пробелов; $e < 0$ эквивалентно $e = 0$.

(7) *Формат установки позиции (колонки).*

$\Phi \sim COLUMN(eSI)$

Устанавливается указанная в eSI позиция в линии. При вводе производится пропуск символов, при выводе—печать пробелов. Если значение e меньше текущего номера позиции, то подводится указанная позиция, относящаяся уже к следующей линии. При $e < 1$ или e больше максимального номера позиции в линии, берется $e = 1$. Можно сказать, что в формате COLUMN указывается номер абсолютной позиции в линии, а в формате X—относительный.

(8) *Формат прогона линий.*

$\Phi \sim SKIP[(eSI)]$

(9) *Формат подвода линии.*

$\Phi \sim LINE(eSI)$

(10) *Формат смены страницы.*

$\Phi \sim PAGE$

Действие последних форматов совпадает с действием соответствующих опций (см. 3.3.1), за исключением того, что вне зависимости от их расположения в операторе опции выполняются всегда до работы любого формата.

Пример. Ввести из колоды перфокарты, начиная с 5-й, и информацию, имеющую атрибуты CHARACTER и FIXED, содержащуюся соответственно в позициях $15 \div 26$ и $50 \div 58$, печатать в колонках $25 \div 36$ и $50 \div 54$ по 40 строчек на странице, начиная с 10-й строчки. Признаком конца вводимой информации являются пробелы в позициях $15 \div 26$.

```

DECLARE A CHAR (12), B FIXED (5), F CHAR (1);
GET SKIP (4); BEG: PUT LINE (10);
DO I=1 TO 40;
    GET EDIT (A, B, F) (COLUMN (15), A(12), X(23),
                                                F(9), X(21), A(1));
    IF A='□' THEN GO TO END;
    PUT EDIT (A, B) (COLUMN (25), A, X(13), F(5));
END;
PUT PAGE; GO TO BEG;
END:

```

С помощью первых операторов GET и PUT, используя опции SKIP и LINE, пропускаются 4 перфокарты и подводится 10-я строчка для печати. Оператор GET в цикле вводит с очередной перфокарты строку и целое число; фиктивная переменная F применяется для того, чтобы сработал формат X(21), так как, в противном случае, он просто пропускается. По оператору PUT выводится введенная информация; переход на следующую строчку осуществляется с помощью формата COLUMN. По окончании цикла осуществляется перевод печати на новую страницу и уход на подвод 10-й строчки. Можно, например, осуществить ввод и следующими операторами:

```

GET SKIP (3); /*ВНЕ ЦИКЛА*/
GET EDIT (A, B) (X(14), A(12), COLUMN (50), F(9)) SKIP(1);
/*В ЦИКЛЕ*/

```

Удаленный формат. В качестве формата может использоваться и ссылка на формат (или список форматов). Ссылка на удаленный формат имеет вид:

ф-удаленный ~R({1|vSL})

Метка или значение меточной переменной отсылают к описательному оператору формата (FORMAT), в котором и содержится список форматов:

d ~s-формата ~ {1:}...FORMAT(ф,...);

Оператор FORMAT причисляется к описаниям, то есть к невыполняемым операторам, но выражения, присутствующие в его форматах, вычисляются не при входе в блок, а при выполнении соответствующих операторов ввода-вывода, точнее, в момент непосредственного использования этих форматов. Оператор формата должен находиться в том же блоке, что и формат типа R, или во внешнем блоке. Необходимо следить за тем, чтобы при многократном использовании удаленных форматов не возникла рекурсивная ссылка на формат.

Удаленный формат удобно применять для замены одного формата на другой в зависимости от некоторого условия (меняя значение меточной переменной) или для использования одного и того же списка форматов в различных операторах ввода-вывода.

Пример.

```
DECLARE G LABEL INITIAL (L1);  
IF P < 0 THEN G=L2;  
PUT EDIT (P, Q, R, S) (F(5), R(G), E(14,6));  
L1: FORMAT (A(15), F(4));  
L2: FORMAT (A(10), F(9));  
PUT EDIT (A, B, C, D) (R (L2), R (L1));
```

В зависимости от величины P значения переменных Q и R выдаются в форматах $A(15)$ и $F(4)$ или $A(10)$ и $F(9)$; значения переменных P и S выдаются соответственно в форматах $F(5)$ и $E(14, 6)$. Переменные A, B, C, D печатаются с преобразованиями соответственно к форматам $A(10), F(9), A(15), F(4)$.

Повторение форматов. Каждый формат данных в списке форматов относится к скалярному значению, которое может являться и элементом массива или структуры. Поэтому при вводе-выводе массивов или структур удобно использовать повторители форматов, которые ставятся перед повторяемым форматом или списком форматов, заключенным в скобки, и имеют вид:

повторитель $\sim \{(eSI) | nDI\}$

Общая форма для формата Φ , указанного в спецификации данных для EDIT, имеет поэтому следующий вид:

$\Phi \sim [(eSI) | nDI] \{\Phi | (\Phi, \dots)\}$ <без рекурсии!>

Если повторитель является числом, то его необходимо отделить пробелом от формата. Если повторитель имеет неположительное значение, то соответствующий формат при выполнении пропускается.

Пример.

```
DCL 1 S, 2 KU (15), 2 V (N), 2 W (N);  
PUT EDIT (S) (15 F(8), (2*N) E (10, 3));
```

Массив KU выводится в формате $F(8)$, массивы V и W в формате $E(10, 3)$.

Другой вариант:

```
PUT EDIT (KU) (F(8)) (V, W) (E(10,3));
```

Здесь используется тот факт, что пара скобок, заключающая формат (или список форматов), сама является повторителем для этого формата. Причем кратность повторения является неопределенной: она обеспечивает форматами все элементы данных, имеющиеся в предшествующем списке данных. После исчерпания элементов данных, находящихся в одном списке, происходит вывод (или ввод) следующего списка элементов данных по форматам, приведенным за этим списком.

Еще один вариант:

```
PUT EDIT ((KU(I), V(I), W(I) DO I=1 TO 15))  
          (SKIP, (M)(F(8), 2 E(10, 3), X(KU(I))));
```


Производится печать в $3 * M$ столбцов, причем $KU(I)$, $V(I)$ и $W(I)$ будут отделяться от следующих трех столбцов пробелами, количество которых определяется только что напечатанным значением $KU(I)$. Предполагается, кроме того, что $N \geq 15$ и $M \leq 4$.

Общее замечание. Возможно смешанное использование всех трех способов управления вводом-выводом для одного потока. При этом следует помнить, что для типа LIST последним введенным (выведенным) символом будет 1-й пробел или запятая, следующая за последним значением в потоке. Для типа DATA таким символом всегда будет точка с запятой. Для EDIT этот символ определяется по w .

3.3.5. Ситуации ввода-вывода.

Описание ситуаций, возникающих при выполнении перечисленных в этом разделе операций ввода-вывода, дано в 4.3.2. Функции, полезные при обработке ситуаций ввода-вывода, приведены в 4.1.5. Здесь мы ограничимся лишь некоторыми замечаниями.

При исчерпании входного потока возникает ситуация ENDFILE. Если это не является ошибкой в программе (например, когда количество данных во входном потоке может меняться и заранее не известно), то для блокировки ситуации ERROR удобно в ON-операторе задать пустой оператор или оператор ухода на обработку окончания ввода, то есть:

ON ENDFILE (SYSIN);

или ON ENDFILE (SYSIN) GO TO метка;

Например, в примере на управляющие форматы, вместо пробы на пробелы, перед 1-м оператором GET можно было задать

ON ENDFILE (SYSIN) GO TO END;

Для определения поля, вызвавшего ситуацию NAME, удобно воспользоваться функцией DATAFIELD.

При возникновении ситуации ENDPAGE, в случае, если в ON-операторе не была выполнена опция (или формат) PAGE, печать продолжается на 61-й строчке, считая от начала той же страницы, и номер строчки продолжает увеличиваться (см. функцию LINENO в 4.1.6), пока не будет выполнен формат (или опция) LINE с меньшим номером или PAGE, которые в этом случае устанавливают номер строчки, равный единице, что делает возможным возникновение в дальнейшем новой ситуации ENDPAGE.

Отметим здесь, что, как правило, на АЦПУ для экономии бумаги устанавливают такую управляющую перфоленту, что по PAGE не происходит подвода бумаги к 61-й строчке (считая от начала страницы), а осуществляется лишь прогон нескольких строчек и, как обычно, номер строчки полагается равным единице. Стандартная управляющая лента вызывала бы и для PAGE, и для

системной реакции на ENDPAGE подвод 73-й строчки (между двумя просечками бумаги АЦПУ — на *бланке* АЦПУ — размещается 72 строчки печати и стандартная страница располагается в первых шестидесяти строчках бланка).

Пример. Ввести и выдать на печать по PUT LIST значения массива A, содержащего 1000 элементов. Выдавать по 50 строчек на странице, начиная с 11-й строчки, и в конце каждой порции, кроме последней, в 62-й строчке печатать слова: СМ. ПРОДОЛЖЕНИЕ.

```
DECLARE A (1000); GET LIST (A);
ON ENDPAGE (SYSPRINT)
  BEGIN; PUT SKIP (1) LIST ('СМ. ПРОДОЛЖЕНИЕ');
  PUT PAGE LINE (11);
END;
PUT PAGE LINE (11) LIST (A);
```

Ситуация ENDPAGE возникает при переходе на 61-ю строчку страницы, поэтому для перехода на 62-ю строчку задается SKIP (1); возможно было и LINE (62), поскольку до выполнения опции PAGE, устанавливающей начало новой страницы, повторное возникновение ситуации ENDPAGE не происходит.

Упражнения.

3.3.1. Вводить по GET LIST с каждой нечетной перфокарты по 3 числа и распечатывать их по PUT LIST в трех средних (из пяти) столбцах, начав с новой страницы, причем, над вторым из трех столбцов дать заголовок ЗНАЧЕНИЯ ПЕРЕМЕННЫХ и подчеркнуть его; для окончания ввода воспользоваться ситуацией ENDFILE.

3.3.2. На перфокартах отперфорировано для ввода по GET DATA некоторое количество целочисленных значений массива $M(1:N)$, отличных от нуля. Требуется ввести эти значения, напечатать их по PUT DATA и, вычислив их произведение, также напечатать, отступив 2 строчки; возникновение ситуаций NAME игнорировать (что позволит обрабатывать перфокарты, на которых отперфорирована информация и для других переменных).

3.3.3. Вводить с перфокарт по GET EDIT значения переменных K, D, C (с атрибутами FIXED(2), CHARACTER(10), FIXED(2)), расположенных соответственно в колонках $11 \div 12$, $K \div K+9$, $69 \div 70$, и пропуская затем C перфокарт ($1 \leq K, C \leq 9$). Печатать значение переменной D в колонках $K \div K+9$, пропуская C строчек и печатая в начале каждой страницы в колонках $75 \div 80$ «PAGE $\square i$ », где $1 \leq i \leq 9$.

3.3.4. Напечатать таблицу значений $\sin x$ и $\cos x$ с точностью 6 знаков после запятой для x на отрезке $(0, \pi/8)$, с шагом 0,01 и с заголовком следующего вида:

10	20	40	60
X	SIN X	COS X	

Числа указывают на позиции, где печатаются вертикальные линии. Значения в таблице должны быть с фиксированной запятой.

3.4. О других возможностях PL/1

В этом разделе будут кратко охарактеризованы те возможности PL/1, которые остались незатронутыми в настоящем курсе.

Управляемая память. В PL/1 программист имеет возможность сам задавать моменты размещения и уничтожения величин в памяти. По оператору `ALLOCATE` указанные величины размещаются в памяти, по оператору `FREE` — уничтожаются; такие величины описываются с атрибутом `CONTROLLED` (управляемые). Повторное размещение той же величины (до ее уничтожения) как бы проталкивает ее в магазин («стек») и делает недоступной для использования до тех пор, пока вытолкнувшая ее величина (ее «тезка») не будет уничтожена.

Величины, характеризующиеся атрибутом `BASED` (базируемые), отличаются от управляемых тем, что имеется возможность доступа ко всем величинам, находящимся в магазине.

Совмещаемые величины. Имеется возможность совместить целиком или частично (для массивов и структур) величины, находящиеся в памяти. Это обеспечивается атрибутом `DEFINED` или с помощью указателя (`POINTER`) — специальной переменной, принимающей адресные значения. Причем совмещение для `DEFINED` может быть достаточно сложным: не обязательно для идущих подряд элементов.

Уплотненные данные. Атрибуты `ALIGNED` и `UNALIGNED` задают «плотное» и «неплотное» размещение величин в памяти. В первом случае за счет экономии памяти время выполнения программы замедляется, а во втором случае — ускоряется. По умолчанию строчные данные уплотнены, а арифметические нет.

Редактирование шаблонами. При вводе-выводе редактирование данных может быть осуществлено и с помощью шаблонов. Например, шаблон `P'(2)Z(3)9V.(2)9'` указывает, что первые две цифры значения должны быть цифрами или пробелами, следующие три — цифрами (от 0 до 9), `V` указывает на местонахождение точки (масштаб), а сама точка — на то, что она должна присутствовать в данном месте значения. Атрибут `PICTURE` устанавливает шаб-

лон для внутренних величин, которые могут участвовать и в выражениях.

Редактирование строк. Форматы ввода-вывода могут быть использованы не только для редактирования потоков, но и для редактирования обычных строчных величин (переменных). Такая возможность определяется с помощью опции STRING в операторах PUT и GET.

Ввод и вывод файлов. С помощью операторов PUT и GET осуществляется работа и с внешней памятью. Для этого в операторе указывается *имя файла*, то есть имя DD-директивы (языка управления заданиями), описывающей внешний набор данных, с которым производится обмен. Например:

```
PUT FILE (MYDISK) EDIT (ARR) (E(12));
```

В операторах ввода-вывода (п. 3.3) по умолчанию предполагаются следующие имена DD:SYSIN и SYSPRINT.

Кроме ввода-вывода потоком (STREAM), операторами READ и WRITE может быть задан обмен записями (RECORD), который производится заданными программистом порциями и выполняется без преобразования данных к символьному виду, что ускоряет обмен.

Мультизадачный режим. Имеется возможность параллельно выполнять независимые части программы с осуществлением синхронизации их выполнения.

Связь с пультом. Программист может по оператору DISPLAY выдать сообщение на операторский пульт и получить ответ оператора (в виде символьной строки).

Препроцессорные операторы. Большинство операторов PL/I могут быть выполнены и на стадии трансляции для редактирования и создания (макрогенерации) текста транслируемой программы.

ГЛАВА 4

СПРАВОЧНАЯ

В этой главе собран справочный материал, посвященный встроенным функциям, преобразованиям, ситуациям. Здесь же приводятся списки атрибутов и символов PL/1, а также обозначений и синтаксических форм.

4.1. Встроенные функции

Ниже дается сводка встроенных функций PL/1, которые разделены на математические, арифметические, строчные, функции для работы с массивами и разные функции. Если не оговорено особо, то аргументами функции являются скалярные или массивные выражения; в последнем случае значением функции будет массив (если массивных аргументов несколько, то они должны иметь одинаковые границы).

Если у функции имеется несколько аргументов, над которыми при вычислении функции производятся какие-либо действия, то эти аргументы преобразуются как операнды выражений (см. 2.2.1), и функция получает результирующие атрибуты (если не указано иное).

Ниже для аргументов встроенных функций используются следующие обозначения:

x, y, z — скалярные и массивные выражения (для псевдо-переменных — скалярные и массивные переменные), то есть $\{x|y|z\} \sim \{eS|eM\}$ (или $\{x|y|z\} \sim \{vS|vM\}$)

l, n, p, q — целые десятичные числа, то есть $\{l|n|p|q\} \sim \sim nID$

$c = a + ib$ — комплексное значение.

Обозначения атрибутов, приписываемые к аргументам x, y, z , будут указывать на требуемое перед вычислением функции преоб-

разование аргументов. Например, xR , yT , zH — преобразованные к REAL, BIT, CHARACTER.

О максимальной точности h см. в 2.1.1.

Примеры обращений к встроенным функциям были даны в 2.2.5 и 2.2.6.

4.1.1. Математические функции.

Аргументы всех функций имеют плавающий масштаб или преобразуются к нему, то есть $x \sim xE$, $y \sim yE$. Значение функции имеет плавающий масштаб; остальные атрибуты совпадают с атрибутами аргумента, в том числе атрибуты точности и размерности. Напомним, что x обозначает и комплексное значение (если не указано обратное).

Идентификаторы математических функций могут быть аргументами при вызове процедур.

ATAN(x) $\arctg(x)$;
главное значение в радианах; для комплексного x (кроме $x = \pm i$) выдается $\text{LOG}((1+ix)/(1-ix))$.

ATAN(xR , xR) $\arctg(x/y)$;
то есть:
если $y > 0$, то $\arctg(x/y)$;
если $x > 0$ и $y = 0$, то $+\pi/2$;
если $x \geq 0$ и $y < 0$, то $\pi + \arctg(x/y)$;
если $x < 0$ и $y = 0$, то $-\pi/2$;
если $x < 0$ и $y < 0$, то $-\pi + \arctg(x/y)$.

ATAND(xR [, yR]) $\arctg(x[/math> / $y])$;
исключается случай 0/0; главное значение в градусах:
 $\text{ATAND}(x[,y]) = (180/\pi) \cdot \text{ATAN}(x[,y])$.$

ATANH(x) $\text{arcth}(x)$;
предполагается, что $\text{ABS}(x) < 1$; для комплексного x (кроме $x = \pm 1 + 0i$) выдается $\text{LOG}((1+x)/(1-x))/2$.

COS(x) $\cos(x)$;
 x — в радианах; для комплексного x выдается $\cos(a)\text{ch}(b) - i \cdot \sin(a)\text{sh}(b)$.

COSD(xR) $\cos(x)$;
 x — в градусах.

COSH(x) $\text{ch}(x)$;
для комплексного x выдается $\text{ch}(a)\cos(b) + i \cdot \text{sh}(a)\sin(b)$.

ERF(xR) функция ошибок: $\text{ERF}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$.

EXP(x)	e^x .
LOG(x)	$\ln(x)$; исключается $x \leq 0$; для комплексного x (кроме $x=0+0i$) выдается: $\text{LOG}(\text{ABS}(x)) + i \cdot \varphi$, где $-\pi < \varphi < +\pi$ ($c=r \cdot e^{i \cdot \varphi}$).
LOG10(xR)	$\log_{10}(x)$; исключается $x \leq 0$.
LOG2(xR)	$\log_2(x)$; исключается $x \leq 0$.
SIN(x)	$\sin(x)$; x —в радианах; для комплексного x выдается $\sin(a)\text{ch}(b) + i \cdot \cos(a)\text{sh}(b)$.
SIND(xR)	$\sin(x)$; x —в градусах.
SINH(x)	$\text{sh}(x)$; для комплексного x выдается $\text{sh}(a)\cos(b) + i \cdot \text{ch}(a)\sin(b)$.
SQRT(x)	$+ \sqrt{x}$; исключается $x < 0$; для комплексного x выдается главное значение: $u \pm iv$, где $u > 0$ или $u=0$ и $v \geq 0$.
TAN(x)	$\text{tg}(x)$; x —в радианах.
TAND(xR)	$\text{tg}(x)$; x —в градусах.
TANH(x)	$\text{th}(x)$.

4.1.2. Арифметические функции.

Значение функции имеет арифметический тип. Аргументы имеют тип арифметический или преобразуемый к нему, то есть $x \sim xA$ и $y \sim yA$. Если не оговорено особо, то те атрибуты значения функции, которые не упоминаются, совпадают с атрибутами основного обрабатываемого аргумента; аргументы могут быть и комплексными.

ABS(x)	Результат: $ x $; для комплексного x выдается $+\sqrt{a^2+b^2}$; точность для FIXED COMPLEX: $(\text{MIN}(h, p+1), q)$.
ADD(x, y, p[,q])	Сложение $x+y$; результат имеет заданную точность ($p \leq h$); для FIXED q обязательно.
BIN[ARY](x[,p[,q]])	Преобразование в двоичное основание, с указанной (иначе см. 4.2.1) точностью.

$\text{CEIL}(xR)$	Минимальное целое не меньше x ; точность для FIXED: $(\text{MIN}(h, \text{MAX}(p-q+1, 1)), 0)$.
$\left\{ \begin{array}{l} \text{COMPLEX} \\ \text{CPLX} \end{array} \right\} (xR, yR)$	Образование $x + iy$; может использоваться как псевдопеременная для разделения комплексного значения на составные части (они становятся вещественными).
$\text{CONJG}(xC)$	Сопряженное значение: $a - ib$.
$\text{DEC}[\text{IMAL}](x[, p[, q]])$	Преобразование в десятичное основание с указанной (иначе см. 4.2.1) точностью.
$\text{DIVIDE}(x, y, p[, q])$	Деление x/y с указанной точностью результата ($p \leq h$); для FIXED q обязательно.
$\text{FIXED}(x[, p[, q]])$	Преобразование к фиксированному масштабу. Когда p и q не заданы, то они берутся по умолчанию: (5,0) или (15,0).
$\text{FLOAT}(x[, p])$	Преобразование к плавающему масштабу. Когда p не задано, то оно берется по умолчанию: 6 или 21.
$\text{FLOOR}(xR)$	Максимальное целое не больше x ; точность для FIXED: $(\text{MIN}(h, \text{MAX}(p-q+1, 1)), 0)$.
$\text{IMAG}(xC)$	Мнимая часть комплексного значения в виде вещественного значения; может использоваться как псевдопеременная для изменения мнимой части значения комплексной переменной.
$\text{MAX}(xR, yR, \dots)$	Максимальное из заданных (не менее двух) значений; если заданы массивы, то результат — массив. Точность для FIXED: $(\text{MIN}(h, \text{MAX}_i(p_i - q_i) + \text{MAX}_i(q_i)), \text{MAX}_i(q_i))$.
$\text{MIN}(xR, yR, \dots)$	Минимальное из заданных (не менее двух) значений; далее как в MAX.
$\text{MOD}(xR, yR)$	Положительный остаток деления x на y ; то есть, минимальное положительное число z , такое что $(x-z)/y$ является целым. Может возникнуть ситуация ZERODIVIDE, а для FIXED может быть и SIZE (см. 4.3.1). Точность для

FLOAT: $(\text{MAX}(p_1, p_2))$; для FIXED: $(\text{MIN}(h, p_2 - q_2 + \text{MAX}(q_1, q_2)), \text{MAX}(q_1, q_2))$.

MULTIPLY($x, y, p[, q]$) Умножение x на y с указанной точностью результата ($p \leq h$); q при FIXED обязательно.

PRECISION($x, p[, q]$) Преобразование в указанную точность.

REAL(xC) Взятие действительной части x ; может использоваться как псевдопеременная для замены действительной части значения комплексной переменной.

ROUND(x, n) Округление. Для FIXED производится на n -й цифре (справа от точки при $n \geq 0$ или слева при $n \leq 0$); отрицательные значения округляются по абсолютной величине. Точность для FIXED: $(\text{MIN}(p+1, h), q)$. Для FLOAT устанавливается в единицу самый правый (31-й или 63-й) двоичный разряд конкретного представления; строчные данные не меняются.

SIGN(xR) То же что и в алголе. Значение имеет атрибуты FIXED BINARY (15,0).

TRUNC(xR) Отсечение дробной части; точность для FIXED: $(\text{MIN}(h, \text{MAX}(p+1-q, 1)), 0)$.

$$\text{TRUNC}(x) = \begin{cases} \text{FLOOR}(x), & \text{при } x \geq 0, \\ \text{CEIL}(x), & \text{при } x < 0. \end{cases}$$

4.1.3. Строчные функции.

Основными аргументами всех функций, если не оговорено противное, являются строчные выражения или преобразуемые к ним, то есть $x \sim xG, y \sim yG, z \sim zG$. Двоичные арифметические данные преобразуются по $A \rightarrow T$, десятичные по $A \rightarrow H$ (см. 4.2.3). Для случая нескольких аргументов, хотя бы один из которых имеет атрибут DECIMAL, а другие — BINARY, последние преобразуются по $B \rightarrow D$ и, затем, $A \rightarrow H$.

BIT ($xT [, l]$) Преобразование x к битовой строке с заданной (иначе см. 4.2.1) длиной.

BOOL (xT, yT, zT) Первые 4 бита строки z (z_1, z_2, z_3, z_4) задают логическую (булевскую) операцию над соответствующими битами строк x и y . Результирующие биты:

$$v_i = \begin{cases} z_1, & \text{для } x_i=0 \text{ и } y_i=0; \\ z_2, & \text{для } x_i=0 \text{ и } y_i=1; \\ z_3, & \text{для } x_i=1 \text{ и } y_i=0; \\ z_4, & \text{для } x_i=1 \text{ и } y_i=1. \end{cases}$$

Перед операцией длины x и y выравниваются. Каждый из аргументов может быть массивным выражением.

CHAR (x [, l])

Преобразование к символьной строке с заданной (иначе см. 4.2.1) длиной.

HIGH (l)

Результат: символьная строка с длиной, равной l ; в ЕС ЭВМ каждый символ будет кодирован в 16-ричной системе как FF.

INDEX (x , y)

Определение номера символа в x , начиная с которого y первый раз входит в x ; если y в x не входит — выдается ноль.

Атрибуты результата: FIXED BINARY (15,0).

LENGTH (x)

Результат: длина строки x в виде FIXED BINARY (15,0).

LOW (l)

Результат: символьная строка с длиной, равной l ; в ЕС каждый символ будет кодирован как 00 (в шестнадцатеричном виде).

REPEAT (x , n)

Сцепление x с собой n раз. Длина результата: $(n+1) \cdot \text{LENGTH}(x)$. Если $n \leq 0$, то x будет результатом (может быть преобразованным).

STRING (vG)

Сцепление в одну строку всех элементов строчной переменной vG , которая предполагается массивной или структурной (для скалярного x он же и будет результатом). Все элементы структуры должны быть строками одного типа; сечение массива не может быть задано в аргументе. Функция используется и как псевдопеременная. В этом случае присваиваемое скалярное значение «разбирается» при присваивании на части, соответствующие элементам vG . Если строки не хватает, то оставшиеся элементы будут пустыми (для VARYING) или заполнятся пробелами.

SUBSTR ($x, yRI [,zRI]$) Выделение из строки x подстроки, начинающейся с позиции y и оканчивающейся на позиции $y+z-1$ (или на последней позиции, если z не задано). Подстрока имеет атрибут **VARYING** и длину, равную z (или $l-y+1$), где l — длина строки x . Выделяемая подстрока должна вся находиться внутри строки x , в противном случае аргументы исправляются автоматически. Сигнализация о неверно заданных аргументах будет производиться, если включена ситуация **STRING-RANGE** (см. 4.3.1). Функция используется и как псевдопеременная для изменения части строки x ; если строка x имеет атрибут **VARYING**, то длина строки при ее изменении не устанавливается, то есть такой строке ранее уже должно быть присвоено какое-либо значение.

TRANSLATE ($xH, yH [,zH]$) Посимвольный перевод x по таблице: z — вход, y — выход (y выравнивается по z). Если z не задано, то в ЕС предполагается строка длиной 256, содержащая символы, закодированные по возрастанию от 00 до FF в 16-ричной системе.

UNSPEC ($x0$) Внутреннее представление значения проблемного выражения в виде битовой строки. Длина строки зависит от атрибутов аргумента:

- для **RBF** (p, q) при $p \leq 15$ равна 16 (для констант — 32), при $p \geq 16$ равна 32;
- для **RDF** (p, q) равна величине $8 * \text{FLOOR}((p+2)/2)$;
- для **RBE** (p) равна 32 ($p \leq 21$) или 64 ($p > 21$);
- для **RDE** (p) равна 32 ($p \leq 6$) или 64 ($p > 6$);
- для **H** (l) равна $8 * l$;
- для **T** (l) равна l ;
- для **C** (p) равна двойной длине соответствующего действительного значения.

Функция используется и как псевдопеременная. При этом присваи-

ваемое значение преобразуется к битовой строке с длиной, зависящей от атрибутов переменной-аргумента (см. выше), и присваивается псевдопеременной прямо в битовой «машинной» форме без приведения к ее атрибутам. Но если псевдопеременная строчная и с VARYING, то длина определяется по присвоенному значению.

VERIFY (x, y)

Проверка того, что все символы x имеются в y — в этом случае выдается нуль. Иначе выдается номер первого слева символа из x , не входящего в y ; атрибуты результата: FIXED BINARY (15,0). Для пустого y выдается 1; но если x пустой, то всегда выдается 0.

4.1.4. Функции для массивов.

Основными аргументами функций являются выражения над массивами или их сечениями, то есть $\{x | y\} \sim eM$; значения функций — скаляры.

ALL (xT) Все строки массива выравниваются по самой длинной (с дополнением нулями справа), после чего производится операция И (&) над соответствующими битами строк. Результат — битовая строка с длиной самой длинной строки.

ANY (xT) Так же как в ALL, но производится операция ИЛИ (|).

DIM (x, n) Размер (протяженность) n -го измерения массива x (верхняя граница — нижняя граница + 1); атрибуты результата: FIXED BINARY (15,0).

HBOUND (x, n) Верхняя граница n -го измерения x ; атрибуты результата: FIXED BINARY (15,0).

LBOUND (x, n) Нижняя граница n -го измерения x ; атрибуты результата: FIXED BINARY (15,0).

POLY (yA, zA) Вычисление значения полинома, образованного по одномерным массивам $y(m:n)$ и $z(k:l)$ следующим образом:

$$y_m + \sum_{j=1}^{n-m} \left(y_{m+j} \times \prod_{i=0}^{j-1} z_{k+i} \right)$$

Для $k-l-1 > m-n$ полагается $z_{k+i} = z_l$ при $k+i > l$. Если z скаляр, то выдается значение

$$\sum_{j=0}^{n-m} y_{m+j} z^j$$

PROD (xA) Произведение всех элементов *x*, преобразованных к плавающей форме; другие атрибуты сохраняются.

SUM (xA) Сумма всех элементов *x*, преобразованных к плавающей форме; другие атрибуты сохраняются.

4.1.5. Функции ситуаций.

Нижеследующие функции предназначены только для использования в ON-операторе или в блоках, им активизированных. Все функции не имеют аргументов.

DATAFIELD Результат: символьная строка переменной длины (≤ 255 в EC), содержащая поле данных, которое вызвало ситуацию NAME; для других ситуаций (не считая ERROR и FINISH после NAME) выдается пустая строка.

ONCHAR Результат: символьная строка длиной 1, содержащая ошибочный символ, вызвавший ситуацию CONVERSION. Если функция применяется после других ситуаций, не связанных с CONVERSION, то выдается пробел. Функция, используемая как псевдопеременная, позволяет исправить неверный символ.

ONCODE Результат: двоичное число точности (15,0), определяющее тип ошибки (тип прерывания). Таблица ошибок здесь не приводится.

ONFILE Результат: символьная строка переменной длины (≤ 31 в EC), содержащая имя SYSIN для ввода или SYSPRINT для вывода, в случае если возникла ситуация ввода-вывода или CONVERSION; для CONVERSION, не связанного с вводом-выводом, выдается пустая строка.

ONLOC В виде символьной строки переменной длины выдается имя входа в процедуру, в которой возникла обрабатываемая ситуация.

ONSOURCE Результат: символьная строка переменной длины (≤ 255 в EC), содержащая ошибочную строку, вызвавшую ситуацию CONVERSION. Если функция применяется после других ситуаций, не связанных с CONVERSION, то выдается пустая строка. Функция, используемая как псевдопеременная, позволяет заменить обрабатываемую строку (она будет подвергнута обработке после выхода из ON-оператора); заменяющее выражение правой части оператора присваива-

ния преобразуется к символьной строке с длиной ранее преобразуемой строки.

4.1.6. Разные функции.

COUNT $\left(\begin{Bmatrix} \text{SYSIN} \\ \text{SYSPRINT} \end{Bmatrix} \right)$	Выдается количество элементов данных, переданных при выполнении соответственно последнего GET или PUT; атрибут результата: FIXED BINARY (15,0).
LINENO (SYSPRINT)	Результат: номер текущей печатаемой линии с атрибутами FIXED BINARY (15,0). Номер, равный нулю, устанавливается по PAGE.
DATE	Выдает символьную строку длины 6, содержащую текущую дату в виде ггммдд (год, месяц, день).
TIME	Выдает символьную строку длины 9, содержащую текущее время дня в виде ччммсстт (час, минута, секунда, миллисекунды).

4.2. Точность преобразований и выражений

Ниже приводится детальное описание всех преобразований данных, имеющих в PL/1. Все преобразования упоминались ранее (2.1.1÷2.1.3). Здесь же основное внимание уделяется определению точности результата при выполнении преобразований и вычислений выражений.

Отметим, что следует различать два случая выполнения преобразований: а) при операции присваивания или б) при вычислении выражений. В первом случае атрибуты, к которым производится преобразование, заданы явно атрибутами переменной левой части оператора присваивания. («Операция присваивания», кроме того, выполняется в операторах DO, RETURN, GET, по INITIAL и при передаче фактических параметров по значению.) Во втором случае атрибуты определяются в зависимости от вида операции и атрибутов операндов (напомним, что из арифметических преобразований, при этом, возможны только $R \rightarrow C$, $F \rightarrow E$, $D \rightarrow B$ — см. 2.2.1). Ниже при указании точности для преобразованного значения будет предполагаться именно этот 2-й случай, если не оговорено обратное. Атрибуты преобразуемой величины будем называть *атрибутами источника*, а атрибуты результата преобразования или операции — *атрибутами мишени (цели)*; некоторые атрибуты мишени могут определяться и форматом при редактируемом вводе-выводе.

Заметим также, что определение атрибутов мишени производится на стадии трансляции, и при этом осуществляется вставка в оттранслированную программу подпрограмм преобразования (и выдача предупреждающей диагностики). Фактическое преобразование конкретных значений к атрибутам мишени осуществляется на стадии выполнения оттранслированной программы. Преобразование констант производится на стадии трансляции, и диагностика при этом, обычно не выдается.

Ниже p° , q° и l° используются для обозначения точности и длины до преобразования, а p , q и l — после преобразования значения; h обозначает максимально допустимую точность (см. 2.1.1).

Предупреждение. Следует считать, что комплексные константы, приводимые в примерах, не являются константами-выражениями (см. 2.2.1), задаваемыми в программе, а представляют собой значения некоторых комплексных переменных, вследствие чего точность таких констант совпадает с точностью их действительной и мнимой частей.

4.2.1. Арифметические преобразования.

R \rightarrow C. Комплексное значение получается из действительного добавлением к последнему нулевой мнимой части с атрибутами действительной части; точность, тем самым, не меняется: $p = p^\circ$, $q = q^\circ$.

Примеры. $-65.40 \rightarrow -65.40 + 00.00I$
 $1.4479E-4 \rightarrow 1.4479E-4 + 0.0000E0I$

C \rightarrow R. Действительное значение получается из комплексного отбрасыванием мнимой части; точность определяется при присваивании (см. ниже $p^\wedge \rightarrow p$)

Примеры. $013.56 + 243.60I \rightarrow 013.56$
 $1001.10B + 0010.01BI \rightarrow 1001.10B$

F \rightarrow E. Значение с фиксированной точкой разделяется на мантиссу и порядок. Точность (p) не меняется: $p = p^\circ$.

Примеры. *) $013.56 \rightarrow 1.3560E1$
 $1101.10B \rightarrow 1.10110E3B$

*) Все примеры, касающиеся значений с плавающей точкой, весьма условны, так как в ЕС ЭВМ и десятичные, и двоичные значения с плавающей точкой представляются в одном и том же «внутреннем кодовом» виде, с 16-ричным порядком и двоичной мантиссой (см. 2.1.3). Значение десятичного порядка (и тем самым положение точки в мантиссе) выбирается в примерах произвольно.

E → F. Значение с плавающей точкой приводится к фиксированной точке; при этом всегда должен быть задан требуемый атрибут точности. Возможно «фиксированное» переполнение (FIXEDOVERFLOW).

Примеры.

Для **ED (6) → FD (6,2)** получим

$$\begin{aligned} & -6.88830E+3 \rightarrow -6888.30 \\ \mathbf{EB (6) \rightarrow FB (5,2)} \quad & 1.10010E2B \rightarrow 110.01B \end{aligned}$$

D → B. При преобразовании значения из десятичного основания в двоичное, двоичная точность находится по следующим формулам:

$$p = \text{MIN} (\text{CEIL} ([1 +] p^\circ * 3.32), h);$$

$$q = \begin{cases} \text{CEIL} (q^\circ * 3.32) & \text{при } q^\circ \geq 0; \\ \text{CEIL} (\text{ABS} (q^\circ * 3.32)) * \text{SIGN} (q^\circ) & \text{при } q^\circ < 0. \end{cases}$$

Единица в p добавляется, если преобразование осуществляется к **FIXED**.

Примеры.

$$0.62500E-1 \rightarrow 0.10000000000000000000E-3B \text{ т. е.}$$

$$\mathbf{ED (6) \rightarrow EB (20)}$$

$$0.10000 \rightarrow 0000.00011001100110011B$$

$$\mathbf{FD (6,5) \rightarrow FB (21,17)}$$

$$0.1 \rightarrow 0000.0001B$$

$$\mathbf{FD (2,1) \rightarrow FB (8,4)}$$

$$.5 \rightarrow 0.1000B$$

$$\mathbf{FD (1,1) \rightarrow FB (5,4)}$$

$$12345.8750000000 \rightarrow .00...0B \text{ (34 нуля)}$$

$$\mathbf{FD (15,10) \rightarrow FB (31,34)}$$

Как видно из примеров, при данном преобразовании возможна потеря точности для значений, которые не представляются точно в двоичном виде (например, $0.1 \rightarrow 2^{-4}$), или —отсечение старших разрядов (ситуация **SIZE**).

З а м е ч а н и е. Для операндов с плавающей точкой также вычисляется точность (p) преобразованного операнда, но эта точность в ЕС ЭВМ учитывается только при дальнейших преобразованиях или при выводе на печать (см. **A → H**, **A → T**). Фактически же никакого преобразования (при **ED → EB**) не производится (см. выше сноску), и точность значений в плавающей форме, являющихся операндами выражения, может быть определена следующим образом:

— для $p_D \leq 6$ (или $p_B \leq 21$) фактическая точность равна 6 (или 21);

— для $p_D > 6$ (или $p_B > 21$) фактическая точность равна 16 (или 53).

Например, можно считать, что

$.1E0 \rightarrow .000110011001100110011E0B$

B \rightarrow D. При преобразовании значения из двоичного основания в десятичное, десятичная точность из двоичной находится по следующим формулам:

$p = \text{MIN} (\text{CEIL} ([1 +] p^\circ / 3.32), h);$

$q = \begin{cases} \text{CEIL} (q^\circ / 3.32) & \text{при } q^\circ \geq 0; \\ \text{CEIL} (\text{ABS} (q^\circ / 3.32)) * \text{SIGN} (q^\circ) & \text{при } q^\circ < 0. \end{cases}$

Единица в p добавляется, если преобразование производится к **FIXED**.

Примеры.

$0.1B \rightarrow 0.5$ то есть **FB** (2,1) \rightarrow **FD** (2,1)

$1.110B \rightarrow 01.7$ — „ — **FB** (4,3) \rightarrow **FD** (3,1)

$10111111B \rightarrow 0191$ — „ — **FB** (8,0) \rightarrow **FD** (4,0)

Как видим, при рассматриваемом преобразовании также возможна потеря точности (например, 1.7 вместо 1.75). О преобразовании значений с плавающей точкой см. выше замечание.

$p^\circ \rightarrow p$. Приведение к заданной точности производится при операции присваивания. Приведение к большей точности сводится к добавлению десятичных или двоичных нулей. Приведение к меньшей точности осуществляется отсечением «ненужных» цифр. Для **FLOAT** отсекаются и добавляются всегда младшие цифры; для **FIXED** могут отсекаться или добавляться и старшие цифры. Младшие цифры отсекаются без округления. При отсечении значащих старших цифр возникает ситуация **SIZE** (если она включена). В ЕС ЭВМ для значений с атрибутами **FLOAT** или **FIXED BINARY** приведение точности фактически производится только в случае перехода от «короткой» (2 или 4 байта) формы представления значения к «длинной» (4 или 8 байтов), или наоборот — см. 2.1.3.

Примеры.

Для **ED** (4) \rightarrow **ED** (6) $3.176E5 \rightarrow 3.17600E5$

ED (4) \rightarrow **FD** (4,3) $0.923E-3 \rightarrow 0.000$

FB (4,2) \rightarrow **FB** (6,1) $01.11B \rightarrow 00001.1B$

FD (6,1) \rightarrow **FD** (4,2) $00068.4 \rightarrow 68.40$

4.2.2. Строчные преобразования.

H \rightarrow T. Символы 0 и 1 преобразуются в биты 0 и 1; если в строке присутствуют другие символы, то возникает ситуа-

ция CONVERSION (при присваивании ситуации не будет, если неверный символ отсекается). Длина строки не меняется *): $l = l^o$.

Примеры. '01011' \rightarrow '01011'B
'0101B' \rightarrow ошибка

T \rightarrow H. Каждый бит строки преобразуется в символ. Длина не меняется *): $l = l^o$.

Пример. '01011'B \rightarrow '01011'

$l^o \rightarrow l$. Приведение к меньшей длине сводится к отбрасыванию лишних правых знаков строки. Приведение к большей длине осуществляется добавлением справа к строке (может быть, уже после преобразования **H \rightarrow T** или **T \rightarrow H**) пробелов (для CHARACTER) или нулей (для BIT).

Примеры.

Для	H (4) \rightarrow H (2)	'TARA' \rightarrow 'TA'
	H (5) \rightarrow H (8)	'TOS□□' \rightarrow 'TOS□□□□□'
но	H (5) \rightarrow H (8) VAR	'TOS□□' \rightarrow 'TOS□□'
	T (4) \rightarrow T (1)	'1010'B \rightarrow '1'B
	T (2) \rightarrow T (5)	'10'B \rightarrow '10000'B

4.2.3. Преобразования типа.

T \rightarrow A. От битовой строки берется не более 31 разряда справа (или ≤ 56 , если требуется преобразование к FLOAT), и они приводятся к атрибутам **BF** (31,0) (или **BE** (31)—при этом отсекаются 25 младших разрядов). Если среди отсекаемых старших разрядов встречаются не нулевые, то возникает ситуация SIZE. Таким образом, всегда $p = 31$.

Примеры.

'1011'B $\rightarrow +0...01011B$ (слева 27 нулей)
'101011...1'B $\rightarrow +11...1B$ (31 единица)
 $\underbrace{\hspace{1.5cm}}_{32 \text{ единицы}}$
'B $\rightarrow +0...0B$ (31 нуль)

A \rightarrow T. Арифметическое значение преобразуется сначала к целому положительному значению с атрибутами **FIXED** ($p^o - q^o, 0$) для значения с фиксированной точкой или к **FIXED** ($p^o, 0$) для значения с плавающей точкой, а затем переводится в двоичную систему (для DECIMAL). Далее, каждая двоичная цифра преобразуется в бит. При этом длина строки равна: $l = p^o [-q^o]$ —для дво-

*) После преобразований, вызываемых некоторыми операциями в выражениях, производится приведение к большей длине.

ичных значений, или $l = \text{CEIL}((p^\circ [-q^\circ]) * 3.32)$ — для десятичных значений. При $p^\circ - q^\circ \leq 0$ получается пустая строка.

Примеры.

+1011.010B → '1011'B
 -12.75 → '0001100'B
 1.275E1 → '0000000001100'B

H → A. Преобразование символьной строки к арифметическому значению заключается в выделении числового значения из строки. Это возможно только в случае, если строка содержит правильную числовую константу (с точностью $p^\circ \leq h$); иначе возникает ситуация **CONVERSION**. Пустая строка преобразуется в нуль. Атрибуты, которые получает выделенное после преобразования значение, не зависят от его первоначальных атрибутов, а зависят только от атрибутов мишени, то есть в данном случае от атрибутов взаимодействующего с ним (через какую-либо операцию) значения. При операции присваивания находящееся в строке значение получает атрибуты переменной, стоящей в левой части оператора.

При арифметических операциях выделенное значение может получить следующие атрибуты:

REAL—мнимая часть выделенного из строки значения всегда отбрасывается;

FLOAT—если атрибут мишени **FLOAT**. Получаемая точность зависит от основания мишени: она равна 15 для **DECIMAL** или 53 для **BINARY**;

FIXED—если атрибут мишени **FIXED**. Получаемая точность зависит от основания мишени и полагается равной (15,0) для **DECIMAL** или (31,0) для **BINARY**, то есть дробная часть выделенного значения отбрасывается.

Если мишень также имеет атрибут **CHARACTER** или если выполняется одноместная операция, то операнды приводятся к **FIXED DECIMAL (15,0)**.

Пример.

DECLARE (H1, H2) CHAR (12) INIT ('1.5E0+1.2E0I'),
 (CF1, CF2) FIXED (2,1) CPLX INIT
 (2.4+3.7I), (E1, E2) FLOAT (2) INIT (4.8E0),
 B1 FIXED BINARY (4,2) INIT (2.25);

Ниже приведены выражения, их атрибуты, полученные значения и предполагаемые значения.

Выражения	Атрибуты	Значения	Вместо
CF1+H1	CFD(15,1)	3.4+3.7I	3.9E0+4.9E0I
B1+H1	RFB(31,2)	3.25	3.75E0+1.2E0I
E1+H1	RED(15)	6.3E0	6.3E0+1.2E0I
-H1	RFD(15,0)	-1	-1.5E0-1.2E0I
E1--H1	RED(15)	5.8E0	6.3E0+1.2E0I
H1+H1	RFD(15,0)	2	3.0E0+2.4E0I

Для операторов:

$$CF2 = H1; , \quad E2 = H1; , \quad H2 = H1;$$

получим:

$$1.5 + 1.2I, \quad 1.5E0, \quad '1.5E0 + 1.2E0I'$$

с атрибутами:

$$CFD(2,1), \quad RED(2), \quad H(12),$$

A → H. Арифметическое значение преобразуется к символьной числовой строке, значением которой будет десятичное числовое значение, равное по величине преобразуемому, и, может быть, окаймленное пробелами. Данное преобразование используется, в основном, при выводе арифметических значений на печать.

Длина и вид полученной строки зависит от атрибутов преобразуемого значения.

F → H. *Случай $p^\circ \geq q^\circ \geq 0$.* Значение преобразуется к виду десятичного числа с фиксированной точкой и располагается в конце строки. Незначащие нули (кроме нуля перед точкой) и знак плюс заменяются пробелами; точка в случае $q^\circ = 0$ не ставится. Длина строки: $l = p^\circ + 3$.

Примеры.

$$-024.97 \rightarrow '\square\square-24.97' \text{ то есть } FD(5,2) \rightarrow H(8)$$

$$.00 \rightarrow '\square 0.00' \quad FD(2,2) \rightarrow H(5)$$

$$1.1100B \rightarrow '\square\square 1.75' \quad FB(5,4) \rightarrow FD(3,2) \rightarrow H(6)$$

Случай $p^\circ < q^\circ$ или $q^\circ < 0$. Значение преобразуется к следующему виду: $nDI F\{\pm\} v[v[v]]$ и располагается в конце строки. Целое десятичное число nDI содержит p° цифр. Одна, две или три цифры в конце служат для изображения порядка числа, полагаемого равным $-q^\circ$ ($-128 \leq -q^\circ \leq 127$). Буква F в середине числа имеет тот же смысл, что и буква E перед порядком числа, и специфицирует этот случай преобразования. Длина строки: $l = p^\circ + 3 + k$, где k — количество цифр в q° .

Примеры.

$$.003445 \rightarrow '\square 3445F-6' \quad \text{для } FD(4,6) \rightarrow H(8)$$

$$-3890000 \rightarrow '-389F+4' \quad FD(3, -4) \rightarrow H(7)$$

$$-0890000 \rightarrow '\square-890F+3' \quad FD(4, -3) \rightarrow H(8)$$

E → H. Значение преобразуется к виду десятичного числа с плавающей точкой в формате: $\{\square | -\} v.v\dots E \{\pm\}$ vv и располагается в конце строки. Мантисса имеет p° цифр, первая из которых не равна нулю (для ненулевого числа); точка ставится после первой цифры. Количество цифр в порядке всегда равно двум. Длина строки: $l \doteq p^\circ + 6$.
Примеры.

.003445E0 → '□3.44500E—03' то есть ED(6) → H(12)
—389E4 → '—3.89E+06' ED(3) → H(9)
110001E1B → '□9.8E+01' EB(6) → ED(2) → H(8)

C → H. Комплексное значение преобразуется к виду десятичного комплексного числа. Действительная часть значения преобразуется, как и в случае **F → H** или **E → H**. Мнимая часть (со знаком) преобразуется тоже, как в случае **F → H** или **E → H**, но полученное мнимое число (с буквой I в конце) располагается в строке сразу вслед за действительной частью. Причем старшие незначащие нули не заменяются пробелами, а уничтожаются, и остальные цифры мнимой части сдвигаются влево (внутри полученного комплексного числа не может быть пробелов). Длина строки: $l = 1 + 2l_R$, где l_R — длина строки для соответствующего действительного значения.

Примеры.

—18.3E1+0.19E—3I → '—1.83E+02+1.90E—04I' т. е.
CDE(3) → H(19)
12.24—00.07I → '□□12.24—0.07I□□'
CDF(4,2) → H(15)
15I → '□□□□0+15I□□'
CDF(2,0) → H(11)

4.2.4. Точность выражений.

Точность (длина) результата операций определяется на стадии трансляции по формулам, приведенным ниже.

Примем $x = x_1 \otimes x_2$. Атрибуты точности (длины) операндов x_1 и x_2 (может быть, уже подвергнутых преобразованиям) и результата x обозначаются:

- для фиксированной точки через $(p_1, q_1), (p_2, q_2), (p, q)$;
- для плавающей точки — $(p_1), (p_2), (p)$;
- для строк — $(l_1), (l_2), (l)$.

Кроме того, обозначим через r количество цифр в целой части значения с фиксированной точкой, то есть: $r = p - q$.

В альтернативе $\left\{ \begin{smallmatrix} 15 \\ 31 \end{smallmatrix} \right\}$ значение 15 принимается для DECIMAL, а 31 для BINARY.

Длина результата логических операций.

Одноместные: $l = l_1$.

Двуместные: $l = \text{MAX}(l_1, l_2)$, $l \leq 32767$.

Точность результата арифметических операций.

Одноместные: $p = p_1$.

Двуместные:

для FLOAT: $p = \text{MAX}(p_1, p_2)$;

для FIXED: $\pm \begin{cases} p = \text{MIN}\left(1 + \text{MAX}(r_1, r_2) + \text{MAX}(q_1, q_2), \left\{\begin{smallmatrix} 15 \\ 31 \end{smallmatrix}\right\}\right), \\ q = \text{MAX}(q_1, q_2); \end{cases}$

$\ast \begin{cases} p = \text{MIN}\left(1 + p_1 + p_2, \left\{\begin{smallmatrix} 15 \\ 31 \end{smallmatrix}\right\}\right), \\ q = q_1 + q_2; \end{cases}$

$/ \begin{cases} p = \left\{\begin{smallmatrix} 15 \\ 31 \end{smallmatrix}\right\}, \\ q = p - r_1 - q_2; \end{cases}$

$\ast\ast \begin{cases} p = (p_1 + 1) \ast x_2 - 1, \\ q = q_1 \ast x_2. \end{cases}$

Последние формулы верны для случая, когда x_1 —FIXED, а x_2 —целое число, большее или равное нулю и, кроме того, $p \leq \left\{\begin{smallmatrix} 15 \\ 31 \end{smallmatrix}\right\}$

(другие атрибуты результата соответствуют атрибутам x_1). Иначе результат получает атрибут FLOAT и полагается:

$p = p_1$, если x_2 —имеет тип целый (другие атрибуты результата те же, что и у x_1);

$p = \text{MAX}(p_1, p_2)$ в остальных случаях.

Примеры (на определение атрибута точности результата).

$T(4) \& T(7) = T(7)$ например $'1001'B \& '1000101'B = '1000000'B$

$ED(3) - ED(6) = ED(6)$ $1.23E0 - 1.2222E0 = 7.78000E-3$

$FD(9,4) + FD(9,6) = FD(12,6)$ $12345.6788 + 111.111111 =$
 $= 012456.789911$

$FB(6,0) + FB(6,0) = FB(7,0)$ $111111B + 000001B = 1000000B$

$FD(9,5) \ast FD(10,9) = FD(15,14)$ $1234.56789 \ast 1.000000001 =$
 $= 4.56789123456789$ (ошибка)

$FD(1,0)/FD(1,0) = FD(15,14)$ $1/4 = 0.250000000000000$

$FD(5,1) \ast\ast 2 = FD(11,2)$ $0020.0 \ast\ast 2 = 000000400.00$

$FD(5,1) \ast\ast 3 = ED(5)$ $0020.0 \ast\ast 3 = 8.0000E3$

$FD(1,0) \ast\ast FD(5,0) = ED(1)$ $2 \ast\ast N = 5E2$, для $N = 00009$

$FD(1,0) \ast\ast ED(5) = ED(5)$ $2 \ast\ast P = 5.1200E2$, для $P = 9.0000E0$

4.2.5. Сложные преобразования.

Выше, при рассмотрении преобразований, были даны примеры преобразований в простейших случаях. Здесь приводятся примеры, в которых требуется осуществить сразу несколько элементарных преобразований. Числовые значения в примерах иногда даются

в сокращенном виде, и поэтому истинные их атрибуты указываются дополнительно с применением условных обозначений.

Пример 1.

DECLARE (A, K) CHARACTER (6);

A = '02.38'; /* A = '02.38□' */

K = 02.38; /* K = '□□□2.3' */

В первом операторе производится преобразование $l^0 \rightarrow l$ или конкретнее $H(5) \rightarrow H(6)$. Во втором операторе осуществляются преобразования $A \rightarrow H$ и $l^0 \rightarrow l$, то есть:

$FD(4,2) \rightarrow H(7) \rightarrow H(6)$

или для значений:

$02.38 \rightarrow '□□□2.38' \rightarrow '□□□2.3'$

Пример 2.

DECLARE B BIT (1), L BIT (6);

B = 3; /* B = '0' B */

L = 3; /* L = '000110' B */

В обоих операторах над числом 3 выполняются преобразования $A \rightarrow T$ (включающие в себя $D \rightarrow B$) и далее $l^0 \rightarrow l$.

Для B: $FD(1) \rightarrow FB(4) \rightarrow T(4) \rightarrow T(1)$

или $3 \rightarrow 0011 B \rightarrow '0011' B \rightarrow '0' B$

Для L: $FD(1) \rightarrow FB(4) \rightarrow T(4) \rightarrow T(6)$

или $3 \rightarrow 0011 B \rightarrow '0011' B \rightarrow '001100' B$

Пример 3.

DECLARE Z BIT (6);

Z = '□2.95□' + '1' B; /* Z = '000000' B */

Перед сложением символьная и битовая строки преобразуются к арифметическим значениям: $H \rightarrow A$ и $T \rightarrow A$. Причем, так как битовая строка преобразуется к FIXED BINARY ($T \rightarrow FB$), то по правилам приведения ($D \rightarrow B$) мишень для первого преобразования будет иметь эти же атрибуты, то есть выполняется $H \rightarrow FB$. Перед присваиванием арифметическое значение преобразуется к битовому, с выравниванием длин: $A \rightarrow T$ и $l^0 \rightarrow l$.

Таким образом:

$H(6) + T(1) \rightarrow FB(31) + FB(31) \rightarrow FB(31) \rightarrow T(31) \rightarrow T(6)$

$'□2.95□' + '1' B \rightarrow 0...010B + 0...01B \rightarrow 0...011B \rightarrow$

$\rightarrow '0...011' B \rightarrow '000000' B$

Пример 4.

DECLARE GH CHAR (5) INIT (2);

GH = GH + 1; /* GH = '□□□□□' */

При инициализации происходит преобразование числа к символьной строке с приведением длины: $A \rightarrow H$ (или точнее $FD(1) \rightarrow H(4)$) и $l^0 \rightarrow l$, то есть $2 \rightarrow '□□□2' \rightarrow '□□□2□'$.

Перед сложением символьная строка преобразуется к арифметическому значению ($H \rightarrow A$), а после сложения — обратно к сим-

вольной строке ($A \rightarrow H$). Таким образом, получаем:

$$\begin{aligned} H(5) + FD(1) &\rightarrow FD(15) + FD(1) \rightarrow FD(15) \rightarrow H(18) \rightarrow H(5) \\ ' \square \square \square 2 \square ' + 1 &\rightarrow 0 \dots 02 + 1 \rightarrow 0 \dots 03 \rightarrow ' \square \dots \square 3 ' \rightarrow \\ &\rightarrow ' \square \square \square \square \square ' \end{aligned}$$

Пример 5.

DECLARE GT BIT (3) INIT ('1' B);

GT = GT + 1; /* GT = '000' B */

При инициализации производится приведение длины битовой константы: $T(1) \rightarrow T(3)$, то есть $'1' B \rightarrow '100' B$.

Перед сложением битовая строка преобразуется к арифметическому значению ($T \rightarrow A$), а после сложения — обратно к битовому ($A \rightarrow T$), то есть

$$\begin{aligned} T(3) + FD(1) &\rightarrow FB(31) + FB(4) \rightarrow FB(31) \rightarrow T(31) \rightarrow T(3) \\ '100' B + 1 &\rightarrow 0 \dots 0100 B + 0001 B \rightarrow 0 \dots 0101 B \rightarrow \\ &\rightarrow '0 \dots 0101' B \rightarrow '000' B \end{aligned}$$

Пример 6.

DECLARE C BIT (2) INITIAL ('1' B), (M, N) CHAR (3)
INIT ('10BIT');

$N = M > C \ \& \ N < 1$; /* $N = '0 \square \square' *$

При инициализации переменных начальные значения приводятся к объявленным атрибутам длины, то есть

для C: $T(1) \rightarrow T(2)$ или $'1' B \rightarrow '10' B$

для M и N: $H(5) \rightarrow H(3)$ или $'10BIT' \rightarrow '10B'$

Перед сравнением битовая строка C преобразуется к символьной с выравниванием длины, то есть $T \rightarrow H$ и $l^0 \rightarrow l$. Символьная строка N преобразуется к арифметическому значению ($H \rightarrow A$), при этом двоичное число, составляющее строку N, преобразуется к десятичному основанию, так как мишень (1) имеет атрибуты FIXED DECIMAL. Результаты операций сравнения имеют атрибуты BIT(1), так же как и для операции логического ИЛИ.

Итак:

$$\begin{aligned} H(3) &> T(2) \ \& \ H(3) < FD(1) \rightarrow \\ \rightarrow H(3) &> H(2) \ \& \ H(3) < FD(1) \rightarrow H(3) > H(3) \ \& \ FD(15) < FD(1) \rightarrow \\ \rightarrow T(1) \ \& \ T(1) &\rightarrow T(1) \rightarrow H(1) \rightarrow H(3) \end{aligned}$$

или

$$\begin{aligned} '10B' &> '10' B \ \& \ '10B' < 1 \rightarrow \\ \rightarrow '10B' &> '10' \ \& \ '10B' < 1 \rightarrow '10B' > '10 \square' \ \& \ 0 \dots 02 < 1 \rightarrow \\ \rightarrow '1' B \ \& \ '0' B &\rightarrow '0' B \rightarrow '0' \rightarrow '0 \square \square' \end{aligned}$$

Напомним еще раз, что вычисление атрибутов для результатов операций и преобразований производится на стадии трансляции по атрибутам операндов, а фактическое осуществление преобразований (как и операций) над конкретными значениями производится при выполнении программы путем обращения к встроенным подпрограммам.

4.3. Ситуации

Ситуация (и ее сокращение)	Условие возникновения	Включено?	Выключается?	Реакция системы	Возврат после ON-оператора
4.3.1. Ситуации вычислений.					
CONVERSION (CONV)	При преобразованиях $H \rightarrow A$ и $H \rightarrow T$ строка содержит недопустимый символ (или незаконное число для $H \rightarrow A$)	да	да	Результат не определен; диагностика; ERROR.	На повторение преобразования (если символ не был исправлен, то — ERROR).
FIXEDOVERFLOW (FOFL)	Значение с фиксированной точкой имеет цифр более 15 (для DECIMAL) или 31 (для BINARY).	да	да	То же	На следующую операцию.
OVERFLOW (OFL)	Значение с плавающей точкой по абсолютной величине больше допустимого (10^{75}).	да	да	То же	То же
SIZE	При присваивании, преобразовании или вводе теряются старшие значащие разряды значения (для FIXED).	нет	да	То же	То же
STRINGRANGE (STRG)	Неверное задание аргументов при обращении к функции SUBSTR.	нет	да	Исправление аргументов; повторение.	См. реакцию системы

SUBSCRIPT RANGE (SUBRG)	Значение индекса выходит за заданные границы.	нет	да	Результат не определен; диагностика; ERROR	На следующую операцию.
UNDERFLOW (UFL)	Абсолютный результат с плавающей точкой меньше допустимого (10 ⁻⁷⁸). При вычитании равных чисел не возникает.	да	да	Результат — нуль; диагностика; продолжение выполнения.	То же
ZERODIVIDE (ZDIV)	Деление на нуль.	да	да	Результат не определен; диагностика; ERROR.	То же
4.3.2. Ситуации ввода-вывода.					
ENDFILE (SYSIN)	Вводимые данные исчерпались.	да	нет	Диагностика; ERROR.	На следующий оператор.
ENDPAGE (SYSPRINT)	Страница исчерпалась (при выводе).	да	нет	Печать с новой строки (с 1-й строки).	Продолжение выполнения оператора вывода.
NAME (SYSIN)	Имя во входном потоке (для DATA), неизвестное в блоке или отсутствующее в списке данных (см. функцию DATA-FIELD).	да	нет	Диагностика; продолжение ввода данных.	На ввод следующего поля данных.
TRANS-MIT ($\left. \begin{matrix} \text{SYSIN} \\ \text{SYS-PRINT} \end{matrix} \right\}$)	Устойчивая ошибка при вводе или печати данных.	да	нет	Диагностика; ERROR.	Продолжение ввода вывода.

Ситуация (и ее сокращение)	Условие возникновения	Включено?	Выключается?	Реакция системы	Возврат после ON-оператора
4.3.3. Ситуации, определяемые программистом.					
CHECK (i,..)	Присваивание переменной с именем i или выполнение оператора (процедуры) с меткой (именем) i.	нет	да	Печать i и его значения (для переменной); на продолжение вычислений.	На продолжение вычислений.
CONDITION (i)	По оператору SIGNAL i; (i считается EXTERNAL).	да	нет	Сообщение (печать i) и продолжение вычислений.	На следующий оператор.
4.3.4. Ситуации системной реакции.					
ERROR	Какая-либо ошибка в программе (в частности см. выше) или SIGNAL ERROR;.	да	нет	Возникновение ситуации FINISH.	См. реакцию системы.
FINISH	Ошибка (см. ERROR), STOP; или SIGNAL FINISH; в программе, RETURN; или END; для главной процедуры.	да	нет	Окончание программы.	См. реакцию системы (при GO TO в ON-операторе или для SIGNAL FINISH; — продолжение вычислений).

4.4. Обозначения и формы

4.4.1. Список обозначений.

Понятия

a	— аргумент, argument
c	— константа, constant
d	— описание, declaration
e	— выражение, expression
f	— функция, function
g	— строка, string
i	— идентификатор, identifier
l	— метка, label
n	— число, number
p	— параметр, parameter
s	— оператор, statement
t	— атрибут, attribute
u	— ситуация, situation
v	— переменная, variable
ф	— формат
э	— элемент данных
α	— буква
ν	— цифра
ξ	— символ ЭВМ
σ	— символ языка
ω	— спецзнак

Модификаторы

A	— арифметический, Arithmetic
B	— двоичный, Binary
C	— комплексный, Complex
D	— десятичный, Decimal
E	{ — вещественный (алгол), rEal — плавающий (PL), Exponential
F	— фиксированный, Fixed
G	{ — логический (алгол), loGical — строчный (PL), strinG
H	— символьный, cHaracter
I	— целый, Integer
L	— меточный, Label
M	— для массива, diMention
O	— проблемный, prOblem
P	— для процедуры, Procedure
R	— действительный (PL), Real
S	— скалярный, Scalar

T	—битовый, biT (в 1-й главе— G)
U	—структурный, strUcture
V	—для переменной, Variable
W	—для псевдопеременной (iW)

С уточнениями

\tilde{c}	—с, но g без повторителя (nID) или nC без L
\tilde{d}, \tilde{t}	—для параметров
\tilde{f}	—встроенная
\tilde{i}	—внешний (≤ 7 символов)
\tilde{l}	— iP и iL для блоков и s-DO
\tilde{n}	—без знака
\hat{d}, \hat{s}	—без «;»
$\tilde{g}, \tilde{\sigma}$	—кроме ' (или ' и ' в алголе)
\tilde{c}	— \tilde{c} , но gH без ' (внешних)

Дополнительные для PL/I

$c = a + ib$	—комплексное значение
$h = \begin{Bmatrix} 15 \\ 31 \end{Bmatrix}$	для FIXED или $\begin{Bmatrix} 16 \\ 53 \end{Bmatrix}$ для FLOAT
m, n	—целые десятичные числа
l	—длина строки
p, q	—точность; $r = p - q$
$\overline{p}, \overline{q}$	—точность в потоке
t	—масштаб (10^t) для EDIT
w	—длина поля для EDIT

Конструктивные обозначения

$\{ \}$	$\{x y z\} \sim \begin{Bmatrix} x \\ y \\ z \end{Bmatrix} \sim x \text{ или } y \text{ или } z$
$[]$	$[x y z] \sim \begin{bmatrix} x \\ y \\ z \end{bmatrix} \sim x \text{ или } y \text{ или } z \text{ или пусто}$
	$[x] y [z] \sim \{xyz xy yz y\}$
$()$	$(x) y (z) \sim \{xyz xy yz\}$
$\langle \rangle$	$x \langle y \rangle z \sim xz$
\dots	$z \dots \sim \{z zz zzz \langle \text{н. т. д.} \rangle\}$
	$yz \dots \sim \{yz yzz yzzz \langle \text{н. т. п.} \rangle\}$
	$\{y z\} \dots \sim \left\{ \begin{Bmatrix} y \\ z \end{Bmatrix} \begin{Bmatrix} y \\ z \end{Bmatrix} \begin{Bmatrix} y \\ z \end{Bmatrix} \begin{Bmatrix} y \\ z \end{Bmatrix} \begin{Bmatrix} y \\ z \end{Bmatrix} \begin{Bmatrix} y \\ z \end{Bmatrix} \begin{Bmatrix} y \\ z \end{Bmatrix} \right\} \langle \text{н. т. д.} \rangle$
\dots	$yz \dots \sim \{y yzy yz yzy \langle \text{н. т. п.} \rangle\} \sim y [zy] \dots$

4.4.2. Сравнительные формы

АЛГОЛ-60		Пункты курса	PL/1
Элементы			
α	$\{A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h i j k l m n o p q r s t u v w x y z\}$ $\{0 1 2 3 4 5 6 7 8 9\}$ $\{\text{true} \text{false}\}$ $\{+ - \times / \div < <= > >= \neq \neg \wedge \vee \supset \equiv \cdot , : ; := _{\text{to}} () [] ' \text{'}' \text{if} \text{do} <\text{и т. д.}>\}$ $\{\alpha v \lambda \omega\}$	1.1.1 (1.2.2) 1.1.1 1	

g	$\langle \{\sigma g\} \dots \rangle \langle \{\{\sigma \dots\} \{\sigma \dots\}\} \dots \rangle$	1.1.7 1.1.8	$\langle \{\{\xi \dots\} \{\xi \dots\}\} \dots \rangle$ /* ξ ...*/ <комментарий>	g
e	Выражения			e
eA	$\Phi_A \{ \sigma A fA n + - \times / \div \uparrow () \}$ $\langle J/K J \otimes 3.8 0.1 \otimes J \rangle$	1.2.1 (1.2.4)	$\Phi_A \{ \sigma A fA n + - * / \langle ? \rangle ** \langle ? \rangle () \} \langle ? \rangle$ $\langle x \div y \rightarrow \text{TRUNC}(x/y) \rangle \langle x \uparrow y \uparrow z \rightarrow (x**y)**z \rangle$ $\langle J/\text{FLOAT}(K) J \otimes 3.8E0 0.1E0 \otimes J \rangle$	eA
eG	$\Phi_G \{ \sigma G fG \text{true} \text{false} \neg \wedge \vee \supset \equiv $ $ eA < \leq = \geq > \neq () \}$	(1.1.5) 1.2.2 (1.2.4)	$\{ eA \text{FLOOR}(eA \cdot 0.5) \}$ $\Phi_G \{ \sigma G fG '1'B '0'B \neg \langle ? \rangle \& ! \langle ? \rangle = \langle ? \rangle$ $ eA < < = = > > \neg = () \}$ $\langle \neg S > 1 \rightarrow \neg(S > 1) \rangle$ $\langle x \supset y \rightarrow \neg x \uparrow y \rangle \langle x \equiv y \rightarrow (x) = (y) \rangle$	eA^+ eG
eL	$\{ l iML[eA] (eL) \}$	1.2.3 ÷ 4	$\{ l iML(eA^+) eL \}$	eL
\otimes	$\langle \uparrow \parallel \times / \div \parallel + - \parallel \leq \parallel \neg \wedge \vee \supset \parallel \equiv \rangle$	1.2.2	$\langle ** \neg \parallel * / \parallel + - \parallel \leq \parallel \& \parallel ! \rangle \langle \text{по рангам} \rangle$	\otimes
s	Операторы			s
s^I s^{II}	$\langle \text{пусто} \rangle$ $\text{go to } eL$	1.3.1 1.3.2	$\langle \text{пусто} \rangle$; $\{ \text{GO} \neg \text{TO} \text{GOTO} \} \neg eL$; $\langle \neg \text{пробел} \rangle$	s^I s^{II}

s^{III}	$iP[(a,...)] <, \sim\{, \} \alpha...: \{ \}$	1.3.3	$CALL \sqcup iP[(a,...)]; <, \sim\{, , /*\xi...*/>$	s^{III}
s^{IV}	$\{ [vA:] =]...vA: = eA^+ \}$	1.3.4	$\{ [vA:]...vA = eA^+; \}$	s^{IV}
s^V	$\{ [vG:] =]...vG: = eG \}$	1.3.5	$IF \sqcup eG \sqcup THEN \sqcup s_1 [ELSE \sqcup s_2] <s \sim \hat{s};>$	s^V
s^{VI}	$\begin{cases} eA_1 \text{ step } eA_2 \text{ until } eA_3 \\ eA_4 \text{ while } eG \\ eA_5 \end{cases} \dots \text{do } s$	1.3.6	$DOLVA = \begin{cases} eA_1^+ \sqcup BY \sqcup eA_2^+ \sqcup TO \sqcup eA_3^+ \\ <\hat{s}> \\ eA_5^+ \end{cases} \dots; s... \text{END};$	s^{VI}
s^{VII}	$\text{begin } s,... \text{ end}$	1.3.7	$DO; s... \text{END}; <s... \sim \hat{s};...>$	s^{VII}
s^{VIII}	$\text{begin } \{d';\}...s; ..\text{end}$	1.3.8	$BEGIN; d...s... \text{END}; <d... \sim \{\hat{d};\}...>$	s^{VIII}
d	$s \sim [l:]...s$	1.3.9	$s \sim [l:]...s \sim [l:]... \hat{s};$	
d	Описания			d
τ	$\{ \text{real} \text{integer} \text{Boolean} \} <\sim\{E I G\}>$	1.4.1	$\{ \text{FLOAT} \text{FIXED} \text{BIT}(1) \} <\sim\{E I G\}>$	τ
dV	$[\text{own}] \tau iV,...$	1.4.1	$\text{DECLARE} (iV,...) [\text{STATIC}] \sqcup \tau;$	dV
dM	$[\text{own } \tau \tau] \text{ array } \{iM,..., [\{eA_1: eA_2\},...]\},...$	1.4.2	$\text{DECLARE}(\{iM,...\}(\{eA_1^+: eA_2^+\},...)) [\text{STATIC}] \sqcup \tau;$	dM
dML	$\text{switch } iML : = l,... <m \text{ элементов}>$ $\text{switch } iML : = \{l iML^0 [eA]\},...$	1.4.3	$\text{DECLARE } iML(m) \text{ LABEL } \sqcup \text{INITIAL}(l,...);$ $\text{DECLARE } iML(m) \text{ LABEL } \sqcup \text{INITIAL}(\{l *\},...);$ $[iML(k) = iML^0(eA^+);]...<k \text{ — номер элемента}>$	dML

dP	$[p^0]$ procedure $iP[(p,...)]$; [value $p,...$] [\tilde{d}];... s $\langle iP := \{eA \mid eG\}$ — в процедуре-функции	1.4.4	<p>DECLARE iP ENTRY[({t...},...)] [RETURN(τ^0)];</p> <p>iP: PROCEDURE[({p,...})][RETURN(τ^0)] [RECURSIVE]; [$\langle ? \rangle$] [\tilde{d}]...s...END; \langle RETURN($\{eA + \mid eG\}$); \langle последний \rangle</p>	dP
\tilde{d}	$t...p,... \langle p \sim t \rangle$ $\left\{ \begin{array}{l} \text{real} \\ \text{integer} \\ \text{Boolean} \\ [\tau] \text{ array} \\ \text{label} \\ \text{string} \\ \text{switch} \\ [\tau^1] \text{ procedure} \end{array} \right\}$		<p>DECLARE $\{(p,...) t... \mid \{p \mid t...\},...\}$; $\langle p \sim i \rangle$</p> $\left\{ \begin{array}{l} \text{FLOAT} \\ \text{FIXED} \\ \text{BIT}(1) \\ (*,...) \tau \langle .. - \text{кол-во измерений} \rangle \\ \text{LABEL} \\ \text{CHARACTER} (*) \\ (*) \text{LABEL} \\ \text{ENTRY}[(\{t^1...\},...)] [\text{RETURN}(\tau^1)] \end{array} \right\}$	\tilde{d}
$t...$				$t...$
a	$\{e \mid g \mid i\}$	1.4.4		a
Программа				
$\text{begin}[\tilde{d};]...s;...end$	1.5.1	iP :PROCEDURE OPTIONS(MAIN); [\tilde{d}]... s ...END;		

4.4.3. Формы PL/1

Формы		Пункты
Символы и комментарии		
$\alpha \sim \{A B C D E F G H I J K L M N O P Q R S T U V W X Y Z \square @ \#\}$ $v \sim \{0 1 2 3 4 5 6 7 8 9\}$ $w \sim \{+ - * / < = > \neg \& ! . : ; () ' _ \% ?\}$ $\sigma \sim \{\alpha v w\}$ $\xi \sim \{\bullet Б Г Д Ж З И Й Л П У Ф Ц Ч Ш Щ Ъ Ы Э Ю Я\}$ комментарии $\sim / * \xi \dots \langle \text{кроме сочетания} * \rangle / *$	1.1.1	4.4.6
Константы		1.1.8
$nI \sim \{nID nIB\}$ $nB \sim [\pm] v \dots \langle \tilde{n}ID \sim v \dots \rangle$ $nRDF \sim [\pm] (\tilde{n}ID) [.] (\tilde{n}ID)$ $nRDE \sim nRDF E [\pm] \tilde{n}ID$ $nR \sim \{nRD nRB\}$ $nRD \sim \{nRDF nRDE\}$ $n \sim cA \sim \{nR nC\}$ $n\tilde{C} \sim \{nR [.] \dots \} \{\pm\} [.] \dots \langle \tilde{n}ID \sim v \dots \rangle$ $gT \sim [(\tilde{n}ID)]' [0 1] \dots 'B$ $gH \sim [(\tilde{n}ID)]' [\xi]' \dots ' \langle \xi \rangle$ — без апострофа $g \sim cG \sim \{gT gH\}$ $\langle g \sim \{[0 1] \dots 'B [\xi]' \dots \} \rangle$ $c \sim \{cA cG cL\}$ $cL \sim iL$ $cO \sim \{cA cG\} \langle \tilde{cO} \sim \{g nR n\tilde{C}\} \langle \tilde{cO} \sim \tilde{cO}, \text{но } gH \sim \xi \dots \rangle$	1.1.3	2.1.1 2.1.7
	1.1.7	2.1.2
Идентификатор, имя, метка		
$i \sim \alpha [\alpha v _] \dots \langle \text{максимум} = 31 \rangle$ $\tilde{i} \sim i$ $\langle \text{максимум} = 7 \rangle$ $iV \sim [iU.] \dots \{iS iM iU\}$ $i \sim \{iL iML(nID, \dots)\} \langle \tilde{i} \sim \{iL\text{-для-DO} iL\text{-для-BEGIN} iP\text{-для-PROCEDURE}\} \rangle$	1.1.2 1.1.4	3.1 2.1.4

<p style="text-align: center;">Переменные</p> $ \begin{aligned} v &\sim \{vS vM vU\} \\ vU &\sim \{vUS vUM\} \\ \{vS vM\} &\sim \{vA vG vL\} \\ vO &\sim \{vA vG\} \\ vS &\sim \{IS IM(eSI, \dots) \{vUS\} \dots vS\} \\ vM &\sim \{iM iM(\{eSI \{*\} \dots\}) \{vUS\} \dots vM \{vUS\} \dots vS\} \\ vUS &\sim \{IUS iUM(eSI, \dots) \dots \{IUS iUM(eSI, \dots)\} \\ vUM &\sim \{vUS\} \dots iUM \end{aligned} $	1.1.5	2.1.7 2.1.5 3.1
<p style="text-align: center;">Функции, аргументы, псевдопеременные</p> $ \begin{aligned} f &\sim \{fS \tilde{f}M\} \\ f &\sim \{f \tilde{f}\} \quad f \sim iP[(a, \dots)] \quad \langle \tilde{f} - \text{встроенная, см. 4.1} \rangle \\ &\quad a \sim \{e I vL iP (a)\} \quad \langle \tilde{f}W - \text{псевдопеременная, см. 2.2.6} \rangle \\ f &\sim \{fA fG \tilde{f}\} \end{aligned} $	1.1.6	2.2.4 2.4.4
<p style="text-align: center;">Выражения</p> $ \begin{aligned} e &\sim \{eS eM eU\} \\ eU &\sim \{eUS eUM\} \\ \{eS eM\} &\sim \{eA eG\} \\ &\quad \left(\begin{array}{l} \text{приводное} \\ \text{к } A \text{ или к } G \end{array} \right) \\ &\quad \langle eI \sim eIR \rangle \\ eS &\sim \{vSO fS \tilde{e}O (eS) \ominus eS eS \otimes eS\} \quad \langle \text{с рекурсией} \rangle \\ eM &\sim \{vMO \tilde{f}M (eM) \ominus eM eM \otimes eM eM \otimes eS eS \otimes eM\} \\ eUS &\sim \{vUSO \tilde{f}US (eUS) \ominus eUS eUS \otimes eUS eUS \otimes eS eS \otimes eUS\} \\ eUM &\sim \{vUMO \tilde{f}UM (eUM) \ominus eUM eUM \otimes eUM eUM \otimes eS eS \otimes eUM\} \\ \langle \ominus \sim \{ \neg + - \} \rangle &\quad \text{— одноместные, } \otimes \text{ — двуместные: все кроме } \neg \rangle \\ \langle \text{операция } \sim \{ \neg + - * / + - ! \} \rangle &\quad \langle \text{по рангам} \rangle \rangle \\ \langle \text{правила приведения: } T \rightarrow H \rightarrow A, R \rightarrow C, F \rightarrow E, D \rightarrow B \rangle \end{aligned} $	1.2	2.2 4.2

Описания			
$d \sim [(\left\{ \begin{smallmatrix} u \\ NO_u \end{smallmatrix} \right\}, \dots)] \dots \{IP:\} \dots PROCEDURE(p, \dots) [RETURNS(tO \dots)] [RECURSIVE];$		1.4.4	2.4.3
$d \sim \{IP:\} \dots ENTRY(p, \dots) [RETURNS(t \dots)]; \quad IP \sim \{IP \tilde{IP}\} \quad p \sim i$	$\{s d\} \dots END(\tilde{i});$		
$d \sim DECLARE \Delta; \Delta \sim \{\tilde{nID}\text{-уровень}\} [i (\Delta)] [t \dots], \dots$	$i \sim \{IS iM iU iP\}$	1.4	2.4.1
$\langle DECLARE \{\tilde{nID}\} [i t \dots] [(\{\tilde{nID}\} [i t \dots], \dots) t \dots], \dots; \langle t \text{— см. 4.4.4} \rangle$			
$t\text{-точности } \sim (\tilde{nID} [, nID]) \langle \sim (p[, q]) \rangle$			
$t\text{-длины } \sim (eSI) \langle \sim (l) \rangle$	$t \sim (\{nID *\}) \langle \text{для параметров} \rangle$		2.1.1
$t\text{-размерности } \sim (\{[eSI:] eSI\}, \dots) \quad \tilde{t} \sim (\{[nID:] nID\}, \dots *, \dots)$			2.1.4
$t\text{-входа } \sim (ENTRY(\{\tilde{nID}\} [t \dots], \dots)) (RETURNS(tO \dots))$		1.4.2	2.1.5
$t\text{-общего-входа } \sim GENERIC \{IP \text{ t-входа}, \dots\}$	$t\text{-подобия } \sim LABEL \text{ iVU}$	1.4.4	2.4.4
$t\text{-начальных-значений } \sim INITIAL \{(\Theta) CALL \text{ iP}(a, \dots)\}; \quad \Theta \sim \{([eSI]) \{c * (\Theta)\}\} \dots$	$t\text{-меточный } \sim LABEL [(cL, \dots)]$		
$\langle INITIAL \{([eSI]) \{c * ([eSI]) \{c *\} \dots\}\} \dots \rangle CALL \text{ iP}(a, \dots) \rangle \rangle$			2.1.6
Операторы			
Общая форма: $s \sim [(\left\{ \begin{smallmatrix} u \\ NO_u \end{smallmatrix} \right\}, \dots)] \dots [!:] \dots s \langle \text{без рекурсии} \rangle \langle u \text{— см. в 4.3} \rangle$		1.3.9	3.2
Основные операторы			
$s^I \sim \langle \text{пусто} \rangle; \quad s^{II} \sim GO \text{ TO } \{I vSL\}; \quad s^{III} \sim CALL \text{ iP}(a, \dots);$		1.3	2.4.4
$s^{IV} \sim \left\{ \begin{array}{l} \{vS \tilde{fWS}\} \dots = \{eS vSL I\}; \\ \{vM \tilde{fWM}\} \dots = \{eM eS vML vSL I\}; \\ \{vUS \tilde{fWUS}\} \dots = \{eUS BY \text{ NAME} eS vSL I\}; \\ \{vUM \tilde{fWUM}\} \dots = \{eUM BY \text{ NAME} eS vSL I\}; \end{array} \right.$		1.3.4	2.3.2

$s^V \sim$ IF eST THEN s_1 [ELSE s_2] $s^{VII} \sim$ DO; { s d}... END[\bar{i}];	1.3	2.3.3 2.3.4
$s^{VI} \sim$ DO { $\{vs fws\} = \left\{ es_i \begin{cases} BY esA_2[TO esR_3] \\ TO esR_3[BY esA_2] \end{cases} [WHILE(est)] \right\}, ... \}$; $s... \text{END}[\bar{i}]$;	1.3.6	2.3.4
$s^{VIII} \sim$ BEGIN; { s d < кроме ENTRY>}... END [\bar{i}]; $s \sim$ RETURN(es0); $s \sim$ STOP;	1.3.8	2.4.2
$s \sim$ ON u [SNAP] { s < кроме IF, DO, ON> SYSTEM; } $s \sim$ REVERT u; $s \sim$ SIGNAL u;		3.2
Операторы ввода-вывода		
$s \sim$ GET $\left(\begin{matrix} LIST(\vartheta,...) \\ DATA[(\vartheta,...)] \\ EDIT\{((\vartheta,...)(\Phi,...)\}...\end{matrix} \right) (SKIP(esI)) [COPY]$;		3.3.1
$s \sim$ PUT $\left(\begin{matrix} LIST(\vartheta,...) \\ DATA[(\vartheta,...)] \\ EDIT\{((\vartheta,...)(\Phi,...)\}...\end{matrix} \right) (SKIP(esI) LINE(esI) PAGE(LINE(esI)))$; ϑ -DO $\sim (\vartheta,...) \hat{\sqcup}$ -DO <c параметром цикла>		
Данные В LIST для GET: $\vartheta \sim \{v0 fwo \vartheta$ -DO} для PUT: $\vartheta \sim \{e0 \vartheta$ -DO}	1.5.2	3.3.2
В потоке [\sqcup ,]...{ $e0$ { \sqcup , }...} [\sqcup]...{ $e0$ [\sqcup]...}		

<p>в DATA для GET: э ~ iVO для PUT: э ~ {vO э-DO} в EDIT для GET: э ~ {vO fWO э-DO} для PUT: э ~ {eO э-DO}</p> <p style="text-align: center;">Форматы</p> <p> $\phi \sim X(eSI) \langle \text{пропуск} \rightarrow \rangle$ $\phi E \sim E(w, \bar{q}, \bar{p})$ В w символах (w + w для C' $\phi \sim COLUMN(eSI) \langle \text{позиция} \rangle$ $\phi F \sim F(w[, \bar{q}, t])$ $[_...]\{nRDE nRDF\} \langle _... \rangle$ $\phi \sim SKIP(eSI) \langle \text{прогон} \downarrow \rangle$ $\phi C \sim C(\phi R[, \phi R])$ $[_...]\{nRDF \langle _... \rangle$ $\phi \sim LINE(eSI) \langle \text{строчка} \rangle$ $\langle \phi R \sim \{\phi E \phi F\} \rangle$ $[_...]\{nRD \langle _... \rangle$ $\phi \sim PAGE \langle \text{страница} \rangle$ $\phi T \sim B[(w)]$ $\langle _... \rangle$ — только для ввода $\phi \sim R(\{I vSL\}) \langle \text{см. FORMAT} \rangle$ $\phi H \sim A[(w)]$ $\langle _... \rangle \{0 1\} \dots [_...]$ $\xi \dots$ d-формата $\sim \{I\} \dots$ FORMAT (φ,...); $\langle \{w \bar{p} \bar{q} t\} \sim eSI \rangle$ Общая форма: $\phi \sim [(eSI) nDI] \{\phi (\phi, \dots)\} \langle \text{без рекурсии!} \rangle$ </p>	<p> $\{[_]\dots vSO = [_]\dots \tilde{cO} [_]\dots\} \{[_]\dots \dots\};$ $\{vSO = [_]\dots \tilde{cO} [_]\dots\} \dots;$ </p>	<p>3.3.3</p> <p>3.3.4</p>
<p style="text-align: center;">Главная процедура</p> <p> $[(\{ \{ \begin{smallmatrix} u \\ NOu \end{smallmatrix} \}, \dots \}) : \dots \{ \tilde{ip}^0 \} \dots \text{PROCEDURE} [(p)] \text{OPTIONS}(\text{MAIN}); \{s d\} \dots \text{END} [\tilde{ip}^0];$ $\langle \text{DECLARE } p \text{ CHARACTER (100) VARYING;} \rangle$ </p>	<p>1.5.1</p> <p>2.4.5</p>	

4.4.4. Список атрибутов.

	часть 1	части 2,3
1. Атрибуты данных:		
1.1. Атрибуты проблемных данных (O):		
1.1.1. Атрибуты проблемных данных, арифметические (A):		
FIXED	1.4.1*)	2.1.1
FLOAT	1.4.1*)	2.1.1
DECIMAL		2.1.1
BINARY		2.1.1
REAL		2.1.1
COMPLEX		2.1.1
(точность)		2.1.1
1.1.2. Атрибуты проблемных данных, строчные (G):		
CHARACTER		2.1.2
BIT	1.4.1*)	2.1.2
VARYING		2.1.2
(длина)	1.4.1*)	2.1.2
1.2. Атрибуты управляющих данных (L):		
LABEL	1.4.3*)	2.1.4
2. Атрибуты входа (P):		
ENTRY	1.4.4	2.4.4
RETURNS	1.4.4	2.4.4
GENERIC		2.4.4
BUILTIN		2.4.4
3. Атрибуты памяти:		
AUTOMATIC		2.4.2
STATIC		2.4.2
4. Атрибуты сферы:		
INTERNAL		2.4.2
EXTERNAL		2.4.2
5. Атрибуты, дополнительные к атрибутам данных:		
(размерность)	1.4.2*)	2.1.5
INITIAL	1.4.3	2.1.6
LIKE		3.1.1

*) См. еще и 1.4.4.

4.4.5. Ключевые слова PL/1.

Ключевое слово	Сокращение	Перевод	Пункты курса
AUTOMATIC	AUTO	автоматический	2.4.2
BEGIN		начало	1.3.8
BINARY	BIN	двоичный	2.1.1, 2.2.5
BIT		битовый	1.4.1, 2.1.2
BUILTIN		встроенный	2.4.3
BY		шаг	1.3.6, 2.3.4
BY NAME	CHAR	по имени	3.1.3
CALL		вызвать	1.3.3
CHARACTER		символьный	1.4.4, 2.1.2
CHECK		проверка	3.2, 4.3.3
COLUMN	COL	колонка	3.3.4
COMPLEX	CPLX	комплексный	2.1.1, 2.2.5
CONDITION	CONV	ситуация	3.2.3, 4.3.3
CONVERSION		преобразование	3.2, 4.3.1
COPY		копировать	3.3.1
DATA		данные	3.3.3
DECIMAL	DEC	десятичный	2.1.1, 2.2.5
DECLARE	DCL	описать	1.4, 2.4.1
DO		выполнить	1.3.6, 1.3.7
EDIT			2.3.4
ELSE		редактирование	3.3.4
END		иначе	1.3.5, 2.3.3
		конец	1.3.6 ÷ 8, 1.4.4, 2.3.4, 2.3.5
ENDFILE		конец данных	3.3.5, 4.3.2
ENDPAGE		конец страницы	3.3.5, 4.3.2
ENTRY		вход	1.4.4, 2.4.3, 2.4.4
ERROR	EXT	ошибка	3.2.1, 4.3.4
EXTERNAL		внешний	2.4.2
FINISH		окончание	3.2.1, 4.3.4
FIXED	FOFL	фиксированный	1.4.1, 2.1.1
FIXEDOVER-FLOW		фиксированное переполнение	3.2, 4.3.1
FLOAT		плавающий	1.4.1, 2.1.1
FORMAT		формат	3.3.4
GENERIC		общий	2.4.4

Ключевое слово	Сокраще- ние	Перевод	Пункты курса
GET		ввести	1.5.2, 3.3.1
GO TO	GOTO	перейти на	1.3.2
IF		если	1.3.5, 2.3.3
INITIAL	INIT	начальный	1.4.3, 2.1.6
INTERNAL	INT	внутренний	2.4.2
LABEL		метка	1.4.3, 2.1.4
LIKE		подобный	3.1.1
LINE		линия	3.3.1, 3.3.4
LIST		список	1.5.2, 3.3.2
MAIN		главная	1.5.1, 2.4.3
NAME		имя	3.3.5, 4.3.2
NOCHECK		без проверки	3.2.2
NOCONVERSION	NOCONV	без преобразования	3.2.2
NOFIXEDOVER- FLOW	NOFOFL	без фиксированно- го переполнения	3.2.2
NOOVERFLOW	NOOFL	без переполнения	3.2.2
NOSIZE		без диапазона	3.2.2
NOSTRINGRANGE	NOSTRG	без границ строки	3.2.2
NOSUBSCRIPT- RANGE	NOSUB- RG	без границ индекса	3.2.2
NOUNDERFLOW	NOUFL	без исчезания	3.2.2
NOZERODIVIDE	NOZDIV	без деления на нуль	3.2.2
ON		при	3.2.3
OPTIONS		дополнения	1.5.1, 2.4.3
OVERFLOW	OFL	переполнение	3.2, 4.3.1
PAGE		страница	3.3.1, 3.3.4
PROCEDURE	PROC	процедура	1.4.4, 2.4.3
PUT		выдать	1.5.1
REAL		действительный	1.5.2, 3.3.1
RECURSIVE		рекурсивный	2.1.1
RETURN		возврат	1.4.4, 2.4.3
RETURNS		возвращает	1.4.4, 2.4.2
			2.4.4
REVERT		вернуть	3.2.3
SIGNAL		имитация	3.2.3
SIZE		диапазон	3.2, 4.3.1
SKIP		прогон	3.3.1, 3.3.4
SNAP		печать памяти	3.2.3

Ключевое слово	Сокраще- ние	Перевод	Пункты курса
STATIC	STRG SUBRG	статический	1.4.1, 2.4.2
STOP		стоп	2.4.2
STRINGRANGE		границы строки	3.2, 4.3.1
SUBSCRIPT- RANGE		границы индекса	3.2, 4.3.1
SYSIN		системный ввод	4.3.2
SYSPRINT		системная печать	4.3.2
SYSTEM		системный	3.2.3
THEN		то	1.3.5, 2.3.3
TO	UFL VAR	до	1.3.6, 2.3.4
TRANSMIT		передача	3.3.5, 4.3.2
UNDERFLOW		исчезновение	3.2, 4.3.1
VARYING		изменяемый	2.1.2
WHILE	ZDIV	пока	2.3.4
ZERODIVIDE		деление на ноль	3.2, 4.3.1
Д о п о л н е н и е			
ALIGNED	CTL DEF	выровненный	3.4
ALLOCAT		разместить	3.4
BASED		базированный	2.4.2, 3.4
CONTROLLED		управляемый	2.4.2, 3.4
DEFINED	PIC PTR	определяемый	3.4
DISPLAY		показать	3.4
FILE		файл	3.4
FREE		освободить	3.4
PICTURE	PTR	шаблон	3.4
POINTER		указатель	3.4
READ		читать	3.4
RECORD		запись	3.4
STREAM	UNAL	поток	3.4
STRING		строка	3.4
WRITE		писать	3.4
UNALIGNED		невывровненный	3.4

4.4.6.Символы ЕС ЭВМ

Символ	Код	Перфорация	Символ	Код	Перфорация	Символ	Код	Перфорация
пробел	40	нет	Ю	B8	12,11,0,8	Я	DD	12,11,9,8,5
[4A	12,8,2	Б	BA	12,11,0,8,2	S	E2	0,2
.	4B	12,8,3	Ц	BB	12,11,0,8,3	T	E3	0,3
<	4C	12,8,4	Д	BC	12,11,0,8,4	U	E4	0,4
(4D	12,8,5	Ф	BE	12,11,0,8,6	V	E5	0,5
+	4E	12,8,6	Г	BF	12,11,0,8,7	W	E6	0,6
	4F	12,8,7	А	C1	12,1	X	E7	0,7
&	50	12	В	C2	12,2	Y	E8	0,8
]	5A	11,8,2	С	C3	12,3	Z	E9	0,9
□	5B	11,8,3	D	C4	12,4	У	EB	11,0,9,8,3
*	5C	11,8,4	E	C5	12,5	Ж	EC	11,0,9,8,4
)	5D	11,8,5	F	C6	12,6	Ь	EE	11,0,9,8,6
;	5E	11,8,6	G	C7	12,7	Ы	EF	11,0,9,8,7
┐	5F	11,8,7	Н	C8	12,8	Ø	F0	0
└	60	11	І	C9	12,9	І	F1	1
/	61	0,1	И	CB	12,0,9,8,3	2	F2	2
	6A	12,11	Й	CC	12,0,9,8,4	3	F3	3
,	6B	0,8,3	Л	CE	12,0,9,8,6	4	F4	4
%	6C	0,8,4	Ј	DI	11,1	5	F5	5

0.1. $\{0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\}$

0.3. $x[x|z] \dots$

0.5. **[*l*:]...begin *d*; .; *s*...end** <МОЖНО И {*d*; }...>

1.1.1. HOMEР 6

1.1.2. B01(I@1, I1)

1.1.3. $-1.34\text{E}10$

1.1.4. 1E8

1.1.5. .000125439786

1.1.6. FLOOR(T(FLOOR(COS(R0)+0.5)))

1.2.1. R1/R2*(SQRT(BETA)—EXP(3E—5*X0.3))**

$$+ R^2 \cdot \cos((3.1416 \cdot Y)^{(1/3)}) \cdot X^{(1/7)/2/(1-R)}$$

1.2.2. $(1/Q - 3.3 * J * K + 2/K) / (2/I + 84E0 * I + 2/Q) * \text{ПО ВТОРОМУ ПРАВИЛУ} / - (5.6 * 4 - 7.3E0 / (I + J + K)) * Q$

1.2.3. $(A@!A) = (\neg(X + 0.1 < Y)! '0' B)$

1.2.4. $(X * Y \uparrow = 0! X ** Y > = Z) = (\neg(X < 0)! \neg(X < Y) \& B)$

1.3.1. IF $X > Y \& X > Z$ THEN $M = X$;

```
ELSE IF Y>X&Y>Z THEN M=Y;
```

ELSE M=Z;

1.3.2. DO J=1 BY 1 TO JK:

DO; P(J)=A(N); DO I=1 BY 1 TO N;

$$P(J) = P(J) * X(J) + A(N - I); \text{ END; END; END;}$$

1.3.3. $YN = 0.5 + 0.5 * Z;$

CYCL: $YN1 = 0.5 * (YN + Z / YN)$:

```
IF ABS(YN1-YN)>=EPS THEN DO; YN=YN1;
                                GO TO CYCL; END;
```

1.3.4. DO I=1 BY 1 TO NN;

U = X(I); N = 0; FUN(I) = X(I);

$$L3:N=N+1; \text{ IF } ABS(U) \leq DELTA$$

THEN GO TO L7;

$$U = -U * X(I) * X(I) / 2 / N / (2 * N + 1);$$

```

    FUN(I)=FUN(I)+U; GO TO L3;
L7:: END;
1.3.5. F=1;
    DO I=2 BY 1 TO N; F=F*I; END;
1.3.6. DO I=1 BY 1 TO N;
    DO; C(I)=0; DO J=1 BY 1 TO N;
        C(I)=C(I)+A(I, J)*B(J);
    END; END; END;
1.4.1. CAB: PROCEDURE (A, B, C, N);
    DECLARE N FIXED; DECLARE ((A, C)(*,*),
        B(*)) FLOAT;
    BEGIN; DECLARE (I, J) FIXED;
        DO I=1 BY 1 TO N;
        DO J=1 BY 1 TO N;
            C(I, J)=A(I, J)*B(J); END; END.
    END;
    END;
1.4.2. FACT: PROCEDURE(N) RECURSIVE RETURNS (FLOAT);
    DECLARE N FIXED.
    IF N=0 THEN RETURN (1);
        ELSE RETURN (N*FACT(N-1));
    END;
1.4.3. POLY: PROCEDURE (A, P, X, JK, N):
    DECLARE (A, P, X)(*)FLOAT;
    DECLARE (JK, N) FIXED;
    BEGIN; DECLARE (I, J) FIXED;
        дальше см. ответ к 1.3.2.
    END; END;
    DECLARE (A1(1:20), (P1, X1) (1:70)) FLOAT;
    DECLARE (JJ, NN) FIXED;
    DECLARE POLY ENTRY((*)FLOAT,
        (*)) FLOAT, (*))FLOAT, FIXED, FIXED);
    .....
    CALL POLY (A1, P1, X1, (JJ), (NN));
1.4.4. KOREN: PROCEDURE (Z,EPS) RETURNS (FLOAT);
    DECLARE (Z, EPS) FLOAT;
    BEGIN; DECLARE (YN, YN1) FLOAT;
        дальше см. ответ к 1.3.3.
    RETURN(YN1);
    END; END; /*KOREN*/
    DECLARE (RAZNICA, P) FLOAT;
    DECLARE KOREN ENTRY (FLOAT, FLOAT)
        RETURNS (FLOAT);
    .....
    RAZNICA=KOREN((P), 1E-7)-SQRT(P);

```

```

1.4.5. POR: PROCEDURE (MASS, M);
      DECLARE MASS (*)FIXED; DECLARE M FIXED;
      BEGIN; DECLARE(I, J) FIXED;
            DECLARE R FIXED;
            DO J=1 BY 1 TO M-1;
              DO I=1 BY 1 TO M-J;
                IF MASS(I)>MASS(I+1) THEN
                  DO; R=MASS(I+1);
                     MASS(I+1)=MASS(I);
                     MASS(I)=R; END;
                END; END;
      END; END;

1.4.6. RAWNO2: PROCEDURE(A,B,C) RETURNS(BIT(1));
      DECLARE(A,B,C)FIXED;
      IF A=B & A $\neg$ =C ! A=C & A $\neg$ =B !
                                     B=C & B $\neg$ =A
      THEN RETURN ('1'B);
      ELSE RETURN ('0'B);
      END;

1.5.1. SINUS: PROCEDURE OPTIONS (MAIN);
      DECLARE (I, N, NN) FIXED,
              (U, DELTA) FLOAT;
      GET LIST(NN);
      BEGIN; DECLARE (X, FUN)(NN) FLOAT;
      GET LIST(X);
      дальше см. ответ к 1.3.4.
      PUT LIST(X) SKIP;
      PUT LIST(FUN) SKIP;
      PUT LIST(NN, DELTA) SKIP;
      END;
      END;

1.5.2. AHAB: PROCEDURE OPTIONS(MAIN);
      DECLARE(I, J, N) FIXED; GET LIST (N);
      BEGIN; DECLARE (A(N, N), B(N), C(N)) FLOAT;
      GET LIST (A, B); PUT LIST (A, B);
      дальше см. ответ к 1.3.6.
      PUT LIST (C);
      END;
      END;

2.1.1. A FIXED DECIMAL REAL (5),
      C BINARY FLOAT REAL (21);
      E FLOAT DECIMAL REAL (6),
      G DECIMAL FLOAT REAL (7),
      K(4) FIXED BINARY REAL (15),
      N COMPLEX DECIMAL FLOAT (6).

```

- 2.1.2. а) FIXED DECIMAL (1), FIXED BINARY (1),
 FIXED DECIMAL (2, 1),
 FIXED BINARY(2), CHARACTER (2),
 BIT (2), ошибка, CHAR(3);
 б) COMPLEX FLOAT BINARY(1),
 COMPLEX FIXED DECIMAL (1),
 CHARACTER (4), CHARACTER (3).
- 2.1.3. а) INITIAL ((14)(1. (15)0), 1);
 б) INITIAL ((15)(1, (13)*, 1));
 в) INITIAL ((14)0, (15)(1, (13)0), 0);
- 2.2.1. а) '11110011'B; б) '00110000'B;
- 2.2.2. а) '0'B; б) '1'B; в) ошибка (точка); г) '0'B; д) '1'B; е) '0'B;
 ж) '1'B; з) ошибка (в строке не число).
- 2.2.3. а) DCL(A, B)(N); ... S=SUM(A*B);
 б) DCL(A, B, C)(N, N); ... DO I=1 TO N; DO J=1 TO N;
 C(I, J)=SUM(A(I,*)*C(*,J));
 END; END;
- 2.2.4. I=INDEX(CH, 'E');
 IF SUBSTR(CH, I+1, 1)='—'
 THEN SUBSTR (CH, I+1, 1)='+';
- 2.2.5. L=LENGTH(P); DO M=1 TO L/2;
 S=SUBSTR (P, L-M+1, 1);
 SUBSTR (P, L-M+1, 1)=SUBSTR
 (P, M, 1);
 SUBSTR (P, M, 1)=S;
 /* DCL S CHAR (1) */ END;
- 2.2.6. а) T=ALL (A<B); б) T=ANY (A>1E5);
- 2.2.7. COMPR: PROCEDURE (A, B) RETURNS (BIT (1));
 DECLARE (A, B) (*, *) FLOAT,
 C(LBOUND (A, 2) : HBOUND
 (A, 2)) BIT (1);
 DO J=LBOUND (A, 2) TO HBOUND (A, 2);
 C(J)=ALL (A(*, J)<B(*, J)); END;
 RETURN (ANY(C));
 END;
- 2.3.1. а) K=3.500E0-6.800E01, A(3)=011.1B,
 L=+000000000000011B; 2-й оператор ошибочен из-за
 размерностей S и T; массив V='10'B, массив S=
 '101111', 5 элементов сечения T(*, 4)=5.00000E0;
 4-й оператор ошибочен из-за размерностей A(5) и V.
 б) X, Y, Z=1: в) X='1101'B, Y='10101'B;
- 2.3.2. а) 2 раза; б) 10 раз или бесконечное повторение для
 NORA, с атрибутами по умолчанию; в) ошибка—ТО для
 комплексного параметра цикла; г) бесконечное повторе-

ние, так как при $I=9$ для $I=I+1$ получается $I=0$ (ситуация SIZE); д) 3 раза; е) 1 раз; ж) бесконечное повторение (для $L > 0$) или ни разу.

2.3.3. BEGIN; DECLARE (Y, P) FLOAT (8);

Y=0; P=1;

DO N=1 BY 1 WHILE (ABS (P) >= 1E-8);

Y=Y+P; P=P*X/N;

END; /*X ОПИСАН ВО ВНЕШНЕМ БЛОКЕ*/

END;

2.4.1. KOR: PROCEDURE(A, B, C, U, V);

DECLARE(A, B, C, U, V) FLOAT;

D=B*B-4*A*C; U, V=0;

IF D < 0 THEN RETURN;

S=SQRT(D);

U=(-B+S)/2/A; V=(-B-S)/2/A; RETURN;

KOR2: ENTRY(B, C, U, V); U=-B/C; V=0;

END;

2.4.2. Переменные J и K по умолчанию имеют атрибуты FIXED BINARY (15, 0), а параметры—FIXED DECIMAL (5, 0), поэтому K вызывается по значению и меняться не будет. Следует описать K с атрибутами FIXED DECIMAL, или заменить K на другую переменную с такими атрибутами.

2.4.3. Атрибуты значения функции, предполагаемые по умолчанию: FIXED BINARY (15, 0), и поэтому 2.1 преобразуется к значению равному 2.0625 (см. п. 2.2.1). Следует в заголовке процедуры и в ее описании добавить опцию и атрибут RETURNS (FIXED) или задать 2.1 с большей точностью.

3.1.1. DECLARE 1 TABLE, 2 A, 3 E, 3 F, 2 B, 3 G, 4 M, 4 N,
3 H, 2 C, 2 D, 3 (I, J, K, L);

3.1.2. DECLARE 1 STRUCTURE, 2 M, 3 (P, Q, R), 2 S,
2 T, 3 U, 4 (V, W), 3 X, 3 Y,
4 V, 4 W;

3.1.3. а) DECLARE 1 YES(3), 2 X(3), 3 Y INIT ((9)0),
3 Z LIKE P,

1 P(9), 2 (Q, R) INITIAL ((9)0);

DECLARE 1 (YES(3), 2 X(3), 3 Y INIT ((9) 0),

3 Z, 4 (Q, R) INIT(0),

1 P(9), 2 (Q, R) INIT ((9)0);

б) DECLARE 1 NO, 2 I(3), 3 J INIT ((3)0),

3 K INIT (1, 2, 3),

2 L INIT (0),

2 U, 3 V INIT (0), 3 W(2) LIKE I;

DECLARE 1 NO, 2 I (3), 3 J INIT ((3)0),

3 K INIT (1, 2, 3),

2 L INITIAL (0),
 2 U, 3 V INIT (0), 3 W(2),
 4 J INIT ((2)0);
 4 K INIT (1, 2);

- 3.1.4. а) $G.H.C = B.D$; $vS = vS$;
 б) $B.B = A.C.C - 7.7$; $vM = vM - n$;
 в) $A.C.A = A.C.C$; $vM = vM$; ошибка, так как массивы
 разной размерности (2 и 1);
 г) $B = B.D/4$; $vUS = vS/n$;
 д) $A.C = G.H$; $vUM = vUS$; ошибка, так как сочетание мас-
 сива структур и структуры недопустимо;
 е) ошибка, так как имя С имеет несколько смыслов. Пра-
 вильно $A.C$, тогда $G.A = A.C + B.D$; $vUM = vUM + vS$;
 ж) $A.C = A.C.C(1)$; $vUM = vS$;
 з) $A.C = A.C.C$; $vUM = vM$; ошибка, так как сочетание мас-
 сива структур и массива недопустимо;
 и) $B = B * G.H$; $vUS = vUS * vUS$; ошибка, так как подструк-
 туры В и Н имеют разное строение;
 к) $B.C = A.C.C(3)$; (с учетом опции BY NAME); $vS = vS$;

3.2.1. (NOOVERFLOW):

P: PROCEDURE OPTIONS (MAIN);

 Q: BEGIN;

 ON OVERFLOW BEGIN; PUT LIST (Y, Z);
 GO TO KON; END;
 (OVERFLOW): IF $X < Y/Z$
 THEN (OFL): $X = Y/Z$; ELSE (OFL): $X = Z/Y$;
 REVERT OVERFLOW; /*ВОЗВРАТ К СИСТЕМ-
 НОЙ РЕАКЦИИ БЛОКА Р */
 (OVERFLOW): S: BEGIN; ... CALL T;
 (NOOVERFLOW): DO $X = Y * Z$ TO $X ** Y$;
 (NOOFL): $Y = Y * Z$; END;
 /*ДАЛЕЕ БЕЗ ИЗМЕНЕНИЙ */

 KON: END P;

3.2.2. P: PROCEDURE OPTIONS (MAIN);

(CHECK (X)): Q: BEGIN;
 IF $X < Y/Z$ THEN $X = Y/Z$; ELSE $X = Z/Y$;
 (NOCHECK(X)): S: BEGIN; ... CALL T;
 DO $X = Z * Y$ TO $X ** Y$; $Y = Y * Z$;
 END;


```

END S; END Q;
(CHECK (X)): T: PROCEDURE;
    ON CHECK (X) PUT LIST (X, Y, Z);
    ...X = X; ...
    GET LIST (X, Y, Z);
    END T;

```

```

END P;

```

```

3.3.1. BEGIN;

```

```

    PUT PAGE LIST ((47)'␣',
        'ЗНАЧЕНИЯ␣ПЕРЕМЕННЫХ');

```

```

    PUT LIST ((47)'␣', (19)'—') SKIP;

```

```

    ON ENDFILE (SYSIN) GO TO KONEC;

```

```

VVOD: GET LIST (A, B, C); GET SKIP (2);

```

```

    PUT SKIP LIST ((23)'␣', A, B, C);

```

```

    GO TO VVOD;

```

```

    KONEC:END;

```

```

3.3.2. BEGIN; DECLARE M (1:N) FIXED INIT ((N)0),

```

```

    PROD FIXED INIT (1);

```

```

    ON NAME (SYSIN); GET DATA (M);

```

```

    DO I=1 TO N; IF M(I)≠0 THEN

```

```

        DO; PROD=PROD*M(I);

```

```

        PUT DATA (M(I)); END;

```

```

    END;

```

```

    PUT DATA (PROD) SKIP (3);

```

```

END;

```

```

3.3.3. BEGIN; DECLARE I FIXED (1) INIT (0),

```

```

    (K, C)FIXED(2), D CHAR(10);

```

```

    ON ENDFILE (SYSIN) GO TO ST;

```

```

    ON ENDPAGE (SYSPRINT) BEGIN; I=I+1;

```

```

        PUT PAGE EDIT ('PAGE␣', I)

```

```

        (COL (75), A, F(1)); END;

```

```

    SIGNAL ENDPAGE (SYSPRINT);

```

```

    VVOD: GET EDIT (K, D, C) (COL (11), F(2),

```

```

        COL(K), A(10), COL (39), F(2));

```

```

    PUT EDIT (D) (COLUMN (K), A);

```

```

    GET SKIP (C+1); PUT SKIP (C+1);

```

```

        /*МОЖНО И C, И C+1*/

```

```

    GO TO VVOD;

```

```

    ST: END;

```

```

3.3.4. PUT EDIT ((49)'—') (COL (11), A) PAGE;

```

```

    PUT SKIP EDIT ('I', 'X', 'I', 'SIN␣X',

```

```

        'I', 'COS␣X', 'I')

```

```

        (COL (10), A, COL(15), A, COL(20), 5 (A, X(7)));

```

```

    PUT SKIP EDIT ('I', 'I', 'I', 'I', (49)'—')

```

```

(CGL(10), A, COL(20), A, COL(40), A, COL(60),
A, COL(11), A);
PUT EDIT (('I', X, 'I', SIN(X), 'I', COS(X), 'I'
DO X=0 BY 0.01 TO 3.1416/8))
(COL(10), A, X(2), F(4, 2), COL(20), A,
2 (X(5), F(9, 6), X(5), A));
DECLARE X FIXED (7, 6);
/*КОЛИЧЕСТВО ОПЕРАТОРОВ ПЕЧАТИ
МОЖНО ВАРЬИРОВАТЬ*/

```

ЛИТЕРАТУРА

1. Лавров С. С. Универсальный язык программирования (алгол-60). Изд. 3.—М.: Наука, 1972.
2. Жоголев Е. А., Трифонов Н. П. Курс программирования.—Изд. 3.—М.: Наука, 1971.
3. Джермейн К. Программирование на IBM/360.—М.: Мир, 1973.
4. Скотт Р., Сондак Н. ПЛ/1 для программистов.—М.: Статистика, 1977.
5. Лепин-Дмитрюков Г. А. Программирование на языке ПЛ/1 (для ДОС ЕС ЭВМ).—М.: Советское радио, 1978.
6. ЕС ЭВМ ОС ПЛ/1. Описание языка.
Ц51.804.002 Д45.—М.: 1977.
7. ЕС ЭВМ ОС ПЛ/1. Описание языка. Справочник.
Ц51.804.002. Д53.—М.: 1977.
8. ЕС ЭВМ ОС ПЛ/1. Руководство программиста.
Ц51.804.002. Д46.—М.: 1977.
9. ЕС ЭВМ ОС ПЛ/1. Руководство программиста. Справочник.
Ц51.804.002. Д81.—М.: 1977.
10. Бухтияров А. М., Фролов Г. Д., Олюнин В. Ю. Сборник задач по программированию на языке ПЛ/1.—М.: Наука, 1978.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

- Алгол-60 9—46, 49—55, 60, 61,
79, 82, 83, 88, 92, 165—
168
Аргумент 44, 92, 95, 108, 140,
170
— по значению 44, 96
— по наименованию 44, 96
— пустой 96
Атрибут длины 57
— —, переменной 57, 96
— начальных значений 61—64
— общего входа 96
— размерности 61, 86
— точности 52
Атрибуты 43, 85, 174
— входа 92—94
— источника 149
— констант 54
— масштаба 50
— мишени 149
— моды 51
— основания 50
— памяти 87, 103
— по умолчанию 53
— сферы 87, 103
Бит 56
Блок 74, 80, 84, 86
—, вход 86
—, выход 86, 91
— простой (начальный) 79
— процедурный 79, 90
Ввод-вывод 47, 119
— данными (DATA) 120, 124
—, опции 122
—, последовательность 121
— редактированием (EDIT) 120,
126
Ввод-вывод смешанный 136
— списком (LIST) 47, 120, 122
Выражение 25, 66, 73, 166, 170
— арифметическое 25, 65
— комплексное 70
— логическое 28
— массивное 72
— меточное (именующее) 29, 73
— скалярное 65, 72
— структурное 106, 109
—, точность 68, 156
Группа 79, 81, 84
Данные 11
— агрегатные 104
— арифметические 50
— битово-строчные 56
—, виды 52
— меточные 59, 64
— проблемные 58, 64
— —, представление 59
— символично-строчные 56, 64
— строчные 55
— элементные 47, 98
Задание 48, 98
Идентификатор 20, 169
— внешний 46
Имя внешнее 90
— входа 91
— квалифицированное 105, 125
— — полностью 105
Инициализация 63
— структур 103
Комментарии 24, 169
Константы 64, 169
— строчные 56
— числовые (см. число)

Линия 121

Массив 61, 64

- , сечение 61
- структур 101, 106
- Метки 22, 60, 169
- массивные 60, 84

Обозначения 14, 163

Оператор блок 36, 79

- ввода (GET) 47, 119, 172
- вывода (PUT) 47, 119, 172
- вызова процедуры (CALL) 32, 86
- групповой 79, 81
- описания (DECLARE) 79, 85, 109
- перехода (GO TO) 32
- помеченный 36
- присваивания 32, 80, 109
- пустой 31
- составной 35, 81
- условный (IF) 33, 80
- цикла 34, 81
- BEGIN 79
- DO 79
- END 79, 84
- ENTRY 90
- FORMAT 134
- ON 114
- RETURN 43, 86
- REVERT 115
- SIGNAL 116
- STOP 86

Операторы 31, 79, 166, 171

Операции 29, 66

- арифметические 66—69
- —, точность 69, 157
- — одноместные 66
- над массивами 72
- строчные 70
- — логические 71
- — сравнения 70

Описание входа 44, 94

- контекстуальное 88, 92
- массива 38
- неявное 88
- параметров 92
- переключателя 40
- переменной (типа) 37
- процедуры 40, 90
- явное 88, 89

Описания 37, 85

Опция 15

- BY NAME 107

Опция CALL 63

- COPY 122
- LINE 122
- MAIN 90
- PAGE 122, 149
- RECURSIVE 90
- RETURNS 90
- SKIP 122
- SNAP 114
- Отладка 117

Параметр 83, 92

- , спецификация 42
- фактический (см. Аргумент)
- цикла 36, 83

Переменные 22

- массивные 61, 109
- меточные 60
- скалярные 65, 109
- структурные 105, 109

Подпрограмма 97

Поток 120, 172

Преобразование $A \rightarrow H$ 59, 72, 123, 126, 144, 155, 158, 159

- $A \rightarrow T$ 58, 71, 131, 144, 153, 158, 159
- $B \rightarrow D$ 55, 144, 152
- $C \rightarrow H$ 156
- $C \rightarrow R$ 54, 150
- $D \rightarrow B$ 55, 67, 71, 149, 151, 158
- $E \rightarrow F$ 54, 151
- $E \rightarrow H$ 156
- $F \rightarrow E$ 54, 67, 71, 149, 150
- $F \rightarrow H$ 155
- $H \rightarrow A$ 58, 71, 154, 158, 159, 160
- $H \rightarrow T$ 57, 71, 131, 152, 160
- $R \rightarrow C$ 54, 67, 71, 149, 150
- $T \rightarrow A$ 58, 71, 153, 158, 159
- $T \rightarrow H$ 57, 71, 72, 123, 126, 153, 159
- $l^o \rightarrow l$ 57, 153, 158, 159
- $p^o \rightarrow p$ 55, 152

Преобразования 149—156

- арифметические 54, 150
- — масштаба 54
- — моды 54
- — основания 55
- — точности (приведение) 55
- строчные 57, 152
- — вида 57
- — длины 57
- типа 58, 153

Префикс 112, 117

Приведение операндов 66

- точности 55, 68

Пробел 15, 19, 56
Программа 46, 97, 168
Процедура внешняя 90
— главная 97, 173
Псевдопеременная COMPLEX
77, 143
— IMAG 77, 143
— ONCHAR 116, 148
— ONSOURCE 116, 148
— REAL 77, 144
— STRING 77, 145
— SUBSTR 77, 146
— UNSPEC 78, 146
Псевдопеременные 77

Сечение 61, 65
Символ PL/1 16, 19, 169
—, буква 18
—, спецзнак 18
— ЭВМ 19, 178
—, цифра 18
Ситуации 111
— ввода-вывода 136, 161
— вычислений 160
— определяемые программистом
162
— системной реакции 112, 162
Ситуация CHECK 113, 114, 118,
162
— CONDITION 116, 118, 162
— CONVERSION 111, 116, 117,
127, 131, 148, 153, 154, 160
— ENDFILE 136, 161
— ENDPAGE 136, 137, 161
— ERROR 112, 115, 117, 160,
161, 162
— FINISH 112, 115, 117, 162
— FIXEDOVERFLOW 66, 111,
112, 151, 160
— NAME 125, 136, 148, 161
— OVERFLOW 111, 160
— SIZE 55, 111, 112, 133, 143,
151, 152, 153, 160

Ситуация STRINGRANGE 146,
160
— SUBSCRIPTRANGE 111, 161
— TRANSMIT 161
— UNDERFLOW 111, 161
— ZERODIVIDE 111, 115, 143,
161
Скаляр 64
Страница ввода-вывода 121
Строка 24, 47, 56
Структура 100
—, описание 100, 102

Умолчание, принцип 12
Уровень 101

Факторизация 85, 102
Формат 120
— данных 126—132
—, повторитель 135
— удаленный 134
— управляющий 133—134
Функции 23, 170
— встроенные 23, 73, 140—149
— — арифметические 74, 142
— — для массивов 76, 147
— — математические 74, 141
— — разные 77, 149
— — ситуаций 116, 148
— — строчные 75, 144
— массивные 73
— скалярные 74

Число 20, 65
— двоичное 51
— комплексное 51, 70
— с плавающей точкой 21, 50
— с фиксированной точкой 21,
50
— целое 21

Элемент данных 120, 172
— списка цикла 34, 82

Юрий Михайлович Безбородов

СРАВНИТЕЛЬНЫЙ КУРС ЯЗЫКА PL/1

(Серия: «Библиотечка программиста»)

М., 1980 г., 192 стр. с илл.

Редакторы *Н. Н. Васина, О. Ю. Меркадер*

Техн. редактор *Л. В. Лихачева*

Корректор *А. Л. Ипатова*

ИБ № 11548

Сдано в набор 19.10.79. Подписано к печати 17.04.80. Т-08141. Бумага 84×108¹/₃₂, тип. № 1. Литературная гарнитура. Высокая печать. Условн. печ. л. 10,08. Уч.-изд. л. 11,83. Тираж 65 000 экз. Заказ № 1065. Цена книги 80 коп.

Издательство «Наука»

Главная редакция

физико-математической литературы

117071, Москва, В-71, Ленинский проспект, 15

Ордена Октябрьской Революции

и ордена Трудового Красного Знамени

Первая Образцовая типография

имени А. А. Жданова Союзполиграфпрома

при Государственном комитете СССР по делам

издательств, полиграфии и книжной торговли.

Москва, М-54, Валовая, 28

80 коп.

5 В1'66
Б391

