

АЛЕКСЕЕВ Е. Р., ЧЕСНОКОВА О. В., ПАВЛЫШ В. Н., СЛАВИНСКАЯ Л. В.

ТУРБО ПАСКАЛЬ 7.0



ИДЕАЛЬНЫЙ УЧЕБНЫЙ КУРС

ЧИСЛЕННЫЕ МЕТОДЫ

**Алексеев Е. Р., Чеснокова О. В.,
Павлыш В. Н., Славинская Л. В.**

Турбо Паскаль 7.0



ИЗДАТЕЛЬСТВО

NT Press

Москва, 2004

УДК 004.9
ББК 32.973.26–018.2
А47

Подписано в печать 07.07.04. Формат 70х90 $\frac{1}{16}$.
Усл. печ. л. 19.72. Тираж 5000 экз. Заказ № 4543

Алексеев Е.Р.

А47 Турбо Паскаль 7.0 / Е.Р. Алексеев, О.В. Чеснокова, В.Н. Павлыш, Л.В. Славинская. — М.: ООО «Издательство АСТ»: Издательство «НТ Пресс», 2004. — 270, [2] с.: ил.

ISBN 5-17-026342-2 (ООО «Издательство АСТ»)

В книге приведена методика составления алгоритмов с помощью блок-схем, содержится описание языка программирования Турбо Паскаль версии 7.0. Изложены методы решения нелинейных уравнений, систем линейных алгебраических уравнений, обработки экспериментальных данных, численные методы интегрирования. Рассмотрены практические примеры программирования, ряд работающих модулей для рисования графиков, работы с массивами, превышающими 64 Кб, и упражнения для самостоятельной работы.

Для школьников и студентов, изучающих программирование, а также для всех желающих изучить язык Турбо Паскаль 7.0.

УДК 004.9
ББК 32.973.26–018.2

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельца авторских прав.

Материал, изложенный в данной книге, многократно проверен. Но, поскольку вероятность технических ошибок все равно остается, издательство не может гарантировать абсолютную точность и правильность приводимых сведений. В связи с этим издательство не несет ответственности за возможный ущерб любого вида, связанный с применением содержащихся здесь сведений.

Все торговые знаки, упомянутые в настоящем издании, зарегистрированы. Случайное неправильное использование или пропуск торгового знака или названия его законного владельца не должно рассматриваться как нарушение прав собственности.

© Алексеев Е. Р., Чеснокова О. В.,
Павлыш В. Н., Славинская Л. В., 2004
© НТ Пресс, 2004

Содержание

Введение	8
-----------------------	----------

Глава 1 ▼

Разработка алгоритмов решения задач	11
1.1. Понятие алгоритма	11
1.2. Изображение алгоритма в виде блок-схемы	12
1.2.1. Алгоритмы линейной структуры	14
1.2.2. Алгоритмы разветвленной структуры	15
1.2.3. Алгоритмы циклической структуры	20

Глава 2 ▼

Данные в языке Турбо Паскаль. Операции и выражения	27
2.1. Алфавит языка	27
2.2. Данные в языке Турбо Паскаль	28
2.2.1. Константы языка Турбо Паскаль	28
2.2.2. Переменные языка Турбо Паскаль	28
2.2.3. Типы данных в языке Турбо Паскаль	28
2.3. Операции и выражения в языке Паскаль	31
2.4. Стандартные функции в языке Паскаль	32

Глава 3 ▼

Структура программы на языке Паскаль.

Операторы языка Паскаль	34
3.1. Структура программы на языке Турбо Паскаль	34
3.2. Простейшие операторы языка Паскаль	36
3.2.1. Оператор присваивания	36
3.2.2. Операторы ввода-вывода	36
3.2.3. Форматированный вывод информации	39
3.2.4. Составной оператор	39
3.3. Структурные операторы языка Паскаль	40
3.3.1. Условный оператор if ... then ... else	40
3.3.2. Оператор варианта case	42
3.3.3. Оператор цикла с предусловием while ... do	43
3.3.4. Оператор цикла с постусловием repeat ... until	45
3.3.5. Оператор цикла с параметром for ... do	47
3.3.6. Операторы break, continue, exit, halt	49

3.4. Упражнения по теме «Программирование линейных процессов»	51
3.5. Упражнения по теме «Программирование разветвленных процессов»	51
3.6. Упражнения по теме «Программирование циклических вычислительных процессов»	52
3.7. Упражнения по теме «Программирование циклов с неизвестным числом повторений»	54

Глава 4 ▼

Численное решение трансцендентных

и нелинейных уравнений	55
4.1. Метод половинного деления	56
4.2. Метод хорд	57
4.3. Метод касательных	59
4.4. Метод простой итерации	60
4.5. Упражнения по теме «Решение нелинейных и трансцендентных уравнений»	62

Глава 5 ▼

Массивы. Алгоритмы обработки массивов

5.1. Описание массивов	63
5.2. Операции над массивами	65
5.3. Ввод-вывод элементов массива	65
5.4. Алгоритм нахождения суммы элементов массива	67
5.5. Алгоритм нахождения произведения элементов массива	67
5.6. Алгоритм поиска максимального элемента в массиве и его номера	68
5.7. Алгоритм упорядочивания элементов в массиве	69
5.7.1. Первый способ	69
5.7.2. Второй способ	70
5.8. Удаление элемента из массива	71
5.9. Примеры программ	72
5.10. Упражнения по теме «Массивы»	81

Глава 6 ▼

Обработка матриц в Турбо Паскале

6.1. Ввод-вывод матриц	82
6.2. Алгоритмы и программы работы с матрицами	83
6.3. Упражнения по теме «Работа с матрицами»	94

Глава 7 ▼

Подпрограммы в языке Турбо Паскаль

7.1. Процедуры в языке Турбо Паскаль	96
7.2. Формальные и фактические параметры	98

7.3. Функции в языке Турбо Паскаль	100
7.4. Особенности работы с подпрограммами в Турбо Паскале версии 7.0	102
7.4.1. Открытые массивы	102
7.4.2. Параметры константы	103
7.5. Процедурные типы	103
7.6. Упражнения по теме «Подпрограммы»	106

Глава 8 ▼

Работа с файлами в языке Турбо Паскаль	108
8.1. Описание файловых переменных	108
8.2. Обработка типизированных файлов	109
8.2.1. Процедура assign	109
8.2.2. Процедуры reset, rewrite	109
8.2.3. Процедура close	109
8.2.4. Процедура rename	110
8.2.5. Процедура erase	110
8.2.6. Функция eof	110
8.2.7. Процедуры write, read	110
8.3. Последовательный и прямой доступ к файлам	112
8.3.1. Функция filesize	112
8.3.2. Функция filepos	112
8.3.3. Процедура seek	113
8.3.4. Процедура truncate	115
8.4. Обработка ошибок ввода-вывода	117
8.5. Работа с текстовыми файлами	119
8.6. Упражнения по теме «Работа с файлами в языке Турбо Паскаль»	121

Глава 9 ▼

Динамические переменные и указатели	123
9.1. Динамическая память	123
9.2. Адреса и указатели	124
9.3. Объявление указателей	124
9.4. Выделение и освобождение динамической памяти	125
9.5. Процедуры freemem, getmem. Использование динамических массивов	126
9.6. Массивы больше 64 Кб в Турбо Паскале	128

Глава 10 ▼

Модули в Турбо Паскале	133
10.1. Использование модуля CRT	134
10.1.1. Основные процедуры и функции модуля CRT	134
10.1.2. Вывод псевдографики и спецсимволов	139
10.2. Использование модуля PRINTER	141

Глава 11 ▼

Строки и записи в языке Турбо Паскаль	142
11.1. Строки в языке Турбо Паскаль	142
11.2. Работа с записями	144
11.3. Упражнения по теме «Строки в языке Турбо Паскаль»	149
11.4. Упражнения по теме «Обработка записей»	150

Глава 12 ▼

Графические средства Турбо Паскаля	151
12.1. Краткая характеристика графических режимов	151
12.2. Управление графическими режимами	152
12.3. Некоторые графические процедуры и функции	156
12.4. Вывод текста в графическом режиме	164
12.5. Сохранение и выдача изображений	166
12.6. Построение графика функции на экране дисплея	167
12.7. Упражнения по теме «Графические средства Турбо Паскаля»	177

Глава 13 ▼

Создание личных модулей	178
13.1. Структура модуля	178
13.2. Модуль изображения графиков и поверхностей непрерывных функций	179

Глава 14 ▼

Решение задач линейной алгебры	189
14.1. Решение систем линейных алгебраических уравнений методом Гаусса	189
14.2. Вычисление обратной матрицы методом Гаусса	195
14.3. Вычисление определителя методом Гаусса	198
14.4. Упражнения по теме «Решение задач линейной алгебры»	202

Глава 15 ▼

Метод наименьших квадратов	205
15.1. Постановка задачи	205
15.2. Подбор параметров функций	207
15.2.1. Линейная функция	207
15.2.2. Квадратичная функция	207
15.2.3. Кубическая функция	208
15.2.4. Полином k -й степени	208
15.2.5. Функции, приводимые к линейной	209
15.2.6. Подбор параметров функции $y = ax^b e^{cx}$	209
15.2.7. Уравнение регрессии и коэффициент корреляции	210
15.2.8. Криволинейная корреляция	212
15.3. Примеры	212

Глава 16 ▼

Интерполяция функций	221
16.1. Канонический полином	222
16.2. Полином Ньютона	223
16.3. Полином Лагранжа	225
16.4. Сплайн-интерполяция	225
16.5. Упражнения по теме «Обработка результатов эксперимента»	238

Глава 17 ▼

Численное интегрирование функций	241
17.1. Интегрирование по методу прямоугольников	241
17.2. Интегрирование по методу трапеций	243
17.3. Интегрирование по методу Симпсона	244
17.4. Вычисление интеграла с заданной точностью	245
17.5. Упражнения для самостоятельной работы	246

Приложение ▼

Основные приемы работы в среде Турбо Паскаля	249
Вызов Турбо Паскаля	249
Текстовый редактор	251
Смещение курсора	252
Команды редактирования	252
Работа с блоком	252
Работа с файлами	253
Выполнение и отладка программы	254
Справочная служба Турбо Паскаля	255
Функциональные клавиши	256
Работа с меню	256
Меню FILE	258
Меню EDIT	259
Меню SEARCH	260
Меню RUN	260
Меню COMPILE	261
Меню DEBUG	262
Меню TOOLS	263
Меню OPTIONS	263
Меню WINDOW	264
Меню HELP	264
Директивы компилятора	265

Используемая литература	267
--------------------------------------	-----

Предметный указатель	268
-----------------------------------	-----

Введение

В настоящее время существует множество подходов к изучению программирования. По мнению авторов, на первом этапе изучения необходимо освоить методы составления алгоритмов без привязки к конкретному языку программирования. Одним из наиболее наглядных методов составления алгоритмов является язык *блок-схем*. Об этом свидетельствует и опыт преподавания программирования. Однако, несмотря на огромное количество хороших книг по программированию, появившихся в последнее время, практически неохваченным остался раздел учебной литературы, включающий описание методов составления алгоритмов и блок-схем.

Авторы постарались заполнить этот пробел, написав книгу, которая соединила в себе учебник по алгоритмизации, программированию и основам численных методов. Особое внимание было уделено тем разделам, где описывалось программирование инженерных задач. Удалось ли нам достичь поставленной цели – судить читателю.

В качестве языка программирования был выбран Турбо Паскаль, который представляется нам ясным, логичным и гибким языком, приучающим к хорошему стилю программирования. Освоив его, можно перейти к серьезному профессиональному программированию в среде Borland Delphi.

Книга состоит из 17 глав и приложения.

Первая глава посвящена описанию методики составления блок-схем алгоритмов. Во второй и третьей главах изложены основные элементы языка (переменные, выражения, операторы) Турбо Паскаль.

В четвертой главе рассказывается о методах решения нелинейных уравнений. Этот раздел вычислительной математики позволит читателю приобрести практические навыки программирования.

Пятая и шестая главы посвящены изучению алгоритмов обработки массивов и матриц, а также реализации их в Турбо Паскале. Данные главы совместно с первой являются ключом к пониманию принципов программирования.

В седьмой главе читатель познакомится с подпрограммами, а также с механизмом передачи параметров между ними с использованием открытых массивов и процедурных типов.

В восьмой главе речь пойдет об использовании файлов в Турбо Паскале. Механизм прямого и последовательного доступа к файлам и обработки ошибок ввода-вывода объяснен на практических примерах. Здесь же описана работа с текстовыми файлами.

В девятой главе рассказывается об указателях, работе с динамическими массивами и матрицами. Приведен текст подпрограмм, позволяющих работать с массивами объемом более 64 Кб.

В десятой главе описаны стандартные модули Турбо Паскаля **CRT**, **Printer**. Одиннадцатая глава посвящена работе со строками и записями.

Прочитав двенадцатую главу, читатель узнает о графических средствах Турбо Паскаля и алгоритме построения графиков непрерывных функций на экране дисплея. Здесь же приводятся исходные тексты программ (с подробными комментариями) для изображения графиков непрерывных и разрывных функций.

В тринадцатой главе кратко описан механизм создания собственных модулей. Приведен текст модуля построения графиков функций и поверхностей.

Четырнадцатая глава посвящена алгоритмам решения систем линейных алгебраических уравнений, нахождения определителя и вычисления обратной матрицы методом Гаусса. Приведены подробные блок-схемы алгоритмов и тексты программ с комментариями.

В пятнадцатой и шестнадцатой главах рассказывается об алгоритмах обработки экспериментальных данных. В пятнадцатой главе изложен метод наименьших квадратов, приведены формулы коэффициента корреляции, индекса криволинейной корреляции. В шестнадцатой главе описаны алгоритмы построения интерполяционных кривых (полиномы Ньютона, Лагранжа, канонический, интерполяция сплайнами), даны примеры блок-схем построения всех упомянутых полиномов.

Семнадцатая глава посвящена численному интегрированию.

В приложении описана среда разработчика Турбо Паскаль.

Система обозначений

В книге выделены:

- программные элементы языка Паскаль – моноширинным шрифтом;
- определения, новые понятия – *курсивом*;
- элементы файловой системы, сочетания клавиш, элементы оболочки Турбо Паскаля – **полужирным шрифтом**.

В текстах программ комментарии идут после символов {.



Таким значком в тексте выделена особо важная информация, примечания и предупреждения.



Этим значком отмечена дополнительная информация: комментарии к основному тексту раздела, полезные советы.

В книге приведено большое количество примеров программ с комментариями, на которые рекомендуем обращать внимание. Все программы тестировались авторами, но если читатель обнаружит ошибки, то просим сообщить о них по адресу: teacher@teacher.dn-ua.com. Будем рады получить отзывы и замечания.

Глава

Разработка алгоритмов решения задач

Решение любой задачи на ЭВМ принято разбивать на следующие этапы: разработка алгоритма решения задачи, составление программы решения задачи на алгоритмическом языке, ввод программы в ЭВМ, отладка программы (исправление ошибок), выполнение программы на ПК, анализ полученных результатов.

Эта глава будет посвящена первому этапу решения задачи – разработке алгоритма.

1.1. Понятие алгоритма

Алгоритм – четкое описание последовательности действий, которые необходимо выполнить при решении задачи. Можно сказать, что алгоритм описывает процесс преобразования исходных данных в результаты, так как для решения любой задачи необходимо:

1. Ввести исходные данные.
2. Преобразовать исходные данные в результаты (выходные данные).
3. Вывести результаты.

Разработка алгоритма решения задачи – это разбиение задачи на последовательно выполняемые этапы, причем результаты выполнения предыдущих этапов могут использоваться в качестве исходных данных для последующих. При этом должны быть четко указаны как содержание каждого этапа, так и порядок их выполнения. Отдельный этап алгоритма представляет собой либо более простую, чем исходная, задачу, алгоритм решения которой известен (разработан заранее), либо достаточно простую и понятную без пояснений последовательность действий.

Разработанный алгоритм можно записать несколькими способами:

- на естественном языке;
- в виде блок-схемы;
- в виде R-схемы.

Рассмотрим пример алгоритма на естественном языке:

1. Ввести в компьютер числовые значения переменных a , b и c .
2. Вычислить d по формуле $d = b^2 - 4ac$.
3. Если $d < 0$, то напечатать сообщение «Корней нет» и перейти к п. 4. Ина-

че вычислить корни по формулам $X_1 = \frac{-b + \sqrt{d}}{2a}$, $X_2 = \frac{-b - \sqrt{d}}{2a}$ и напечатать

значения X_1 и X_2 .

4. Прекратить вычисления.

1.2. Изображение алгоритма в виде блок-схемы

Блок-схемой называется наглядное графическое представление алгоритма, когда отдельные его этапы изображаются при помощи различных геометрических фигур, называемых блоками, а связи между этапами (последовательность выполнения этапов) указываются при помощи стрелок, соединяющих эти фигуры. Блоки сопровождаются надписями.

Типичные действия алгоритма изображаются следующими геометрическими фигурами:

- *блок начала-конца алгоритма* (рис. 1.1). Надпись внутри блока: «начало» («конец»);
- *блок ввода-вывода данных* (рис. 1.2). Надпись внутри блока: ввод (вывод или печать) и список вводимых (выводимых) переменных;
- *блок решения или арифметический* (рис. 1.3). Внутри блока записывается действие, вычислительная операция или группа;
- *условный блок* (рис. 1.4). Условие записывается внутри блока. В результате проверки условия осуществляется выбор одного из возможных путей (ветвей) вычислительного процесса. Если условие истинно, то следующим выполняется этап, соответствующий ветви «+»; если условие ложно, то выполняется этап, соответствующий ветви «?».

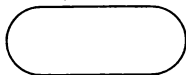


Рис. 1.1 ▼ Блок начала-конца алгоритма

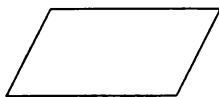


Рис. 1.2 ▼ Блок ввода-вывода данных

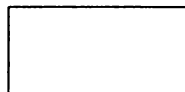


Рис. 1.3 ▼ Арифметический блок

В качестве примера рассмотрим блок-схему алгоритма решения квадратного уравнения (рис. 1.5), описанного в предыдущем подразделе.

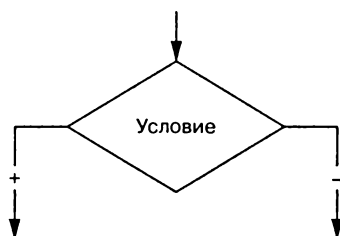


Рис. 1.4 ▼ Условный блок

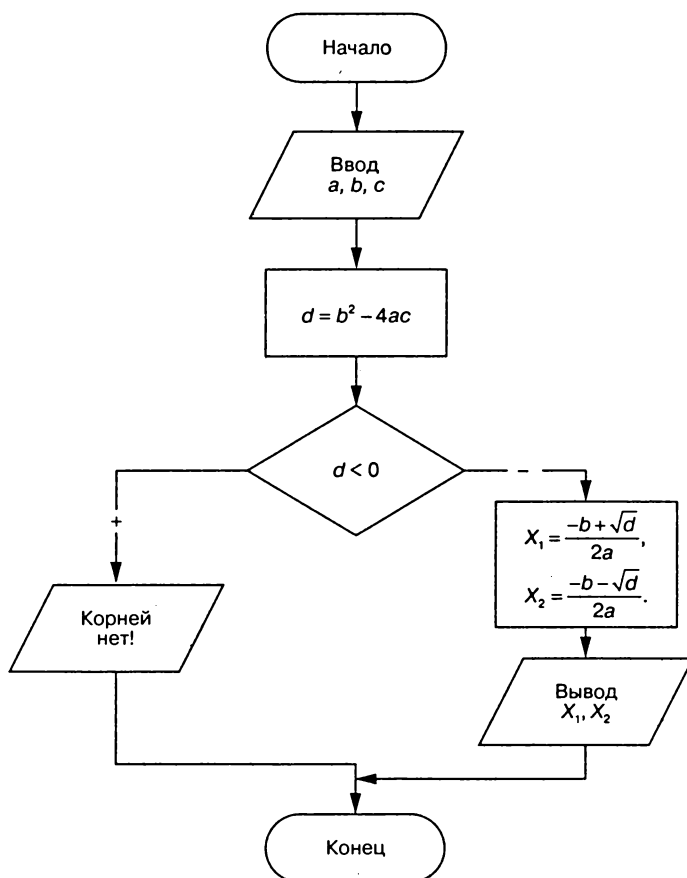


Рис. 1.5 ▼ Блок-схема алгоритма решения квадратного уравнения

1.2.1. Алгоритмы линейной структуры

Линейный алгоритм – это такой, в котором все операции выполняются последовательно одна за другой (рис. 1.6).

Рассмотрим несколько примеров линейных алгоритмов.

ПРИМЕР 1.1. Зная длины трех сторон треугольника, вычислить его площадь и периметр.

Пусть a, b, c – длины сторон треугольника. Необходимо найти S – площадь треугольника, P – периметр. Для нахождения площади можно воспользоваться формулой Герона: $S = \sqrt{r(r-a)(r-b)(r-c)}$, где r – полупериметр.

Входные данные: a, b, c .

Выходные данные: S, P .

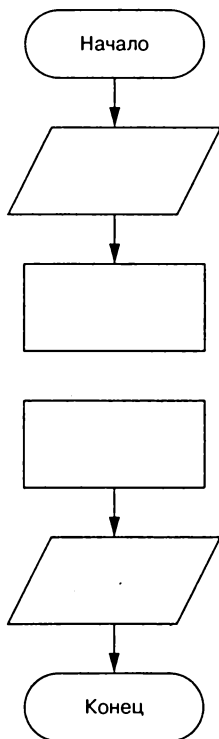


Рис. 1.6 ▼ Размещение блоков в линейном алгоритме

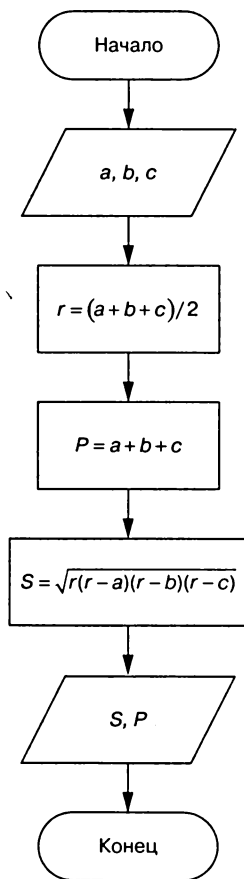


Рис. 1.7 ▼ Алгоритм решения задачи из примера 1.1

Блок-схема алгоритма представлена на рис. 1.7.



В этих блоках знак «=» означает не математическое равенство, а операцию присваивания. Переменной, стоящей слева от оператора, присваивается значение, указанное справа. Причем это значение может быть заранее определенным либо вычисляться при помощи математического выражения. Например, операция $r = (a + b + c) / 2$ – имеет смысл (переменной r присвоить значение $r = (a + b + c) / 2$), а выражение $(a + b + c) / 2 = r$ – бессмыслица.

ПРИМЕР 1.2. Известны плотность и геометрические размеры цилиндрического слитка, полученного в металлургической лаборатории. Найти объем, массу и площадь основания слитка.

Входные данные: R – радиус основания цилиндра, h – высота цилиндра, ρ – плотность материала слитка.

Выходные данные: m – масса слитка, V – объем, S – площадь основания.

Блок-схема представлена на рис. 1.8.

ПРИМЕР 1.3. Заданы длины двух катетов в прямоугольном треугольнике. Найти длину гипотенузы, площадь треугольника и величину его углов.

Входные данные: a , b – длины катетов.

Выходные данные: c – длина гипотенузы, S – площадь треугольника, α , β – углы.

Блок-схема представлена на рис. 1.9.

1.2.2. Алгоритмы разветвленной структуры

Алгоритмы разветвленной структуры применяются, когда в зависимости от некоторого условия необходимо выполнить либо одно, либо другое действие. В блок-схемах разветвленные алгоритмы изображаются так, как показано на рис. 1.10–1.11.

Рассмотрим несколько примеров построения алгоритмов разветвленной структуры.

ПРИМЕР 1.4. Известны коэффициенты a , b и c квадратного уравнения. Вычислить корни квадратного уравнения.

Входные данные: a , b , c .

Выходные данные: x_1 , x_2 .

Блок-схема представлена на рис. 1.5.

ПРИМЕР 1.5. Заданы коэффициенты a , b и c биквадратного уравнения $ax^4 + bx^2 + c = 0$. Найти корни уравнения. Для решения задачи необходимо заменой $y = x^2$ привести уравнение к квадратному и решить его.

Входные данные: a , b , c .

Выходные данные: x_1 , x_2 , x_3 , x_4 .

Блок-схема представлена на рис. 1.12.

Алгоритм состоит из следующих этапов:

1. Вычисление дискриминанта уравнения d .
2. Если $d \geq 0$, то определяются y_1 и y_2 ; иначе корней нет.
3. Если $y_1, y_2 < 0$, то корней нет.

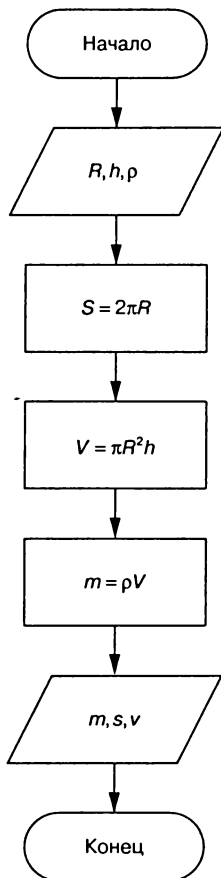


Рис. 1.8 ▼ Алгоритм
примера 1.2

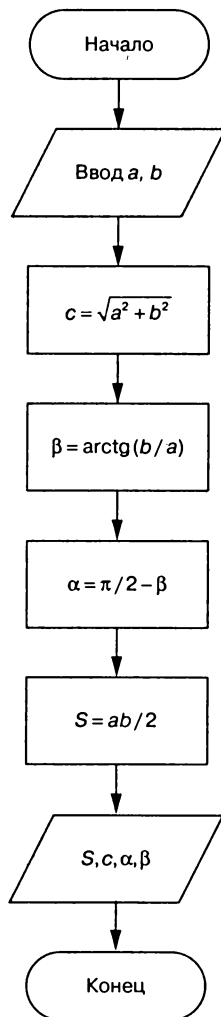


Рис. 1.9 ▼ Алгоритм
примера 1.3

4. Если $y_1, y_2 \geq 0$, то вычисляются четыре корня по формулам $\pm\sqrt{y_1}, \pm\sqrt{y_2}$ и выводятся значения корней.
5. Если условия 3 и 4 не выполняются, то необходимо проверить знак y_1 . Если $y_1 \geq 0$, то вычисляются два корня по формуле $\pm\sqrt{y_1}$. Если же $y_2 \geq 0$, то вычисляются два корня по формуле $\pm\sqrt{y_2}$. Вычисленные значения корней выводятся.

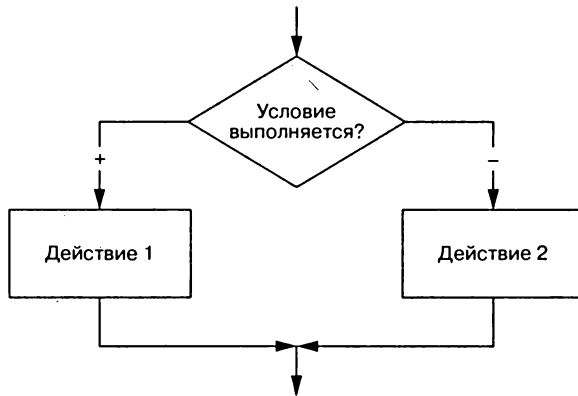


Рис. 1.10 ▼ Фрагмент алгоритма разветвленной структуры



Рис. 1.11 ▼ Пример разветвления структуры алгоритма

Еще раз обратимся к алгоритмам на рис. 1.5, 1.12. Нетрудно заметить, что если a принимает значение 0, алгоритмы не работают (ведь на 0 делить нельзя). Это – *недостаток алгоритмов*. Его можно избежать, если проверять значение переменной a сразу после ввода. Алгоритмы такой проверки приведены ниже. В первом случае (см. рис. 1.13), если введенное значение переменной $a = 0$, выполнение вычислительного процесса сразу же прекращается. Алгоритм, изображенный на рис. 1.14, позволяет при нулевом значении a повторить ввод переменной. Этот процесс будет продолжаться до тех пор, пока a не станет отличной от нуля. Подобные алгоритмы называются циклическими.

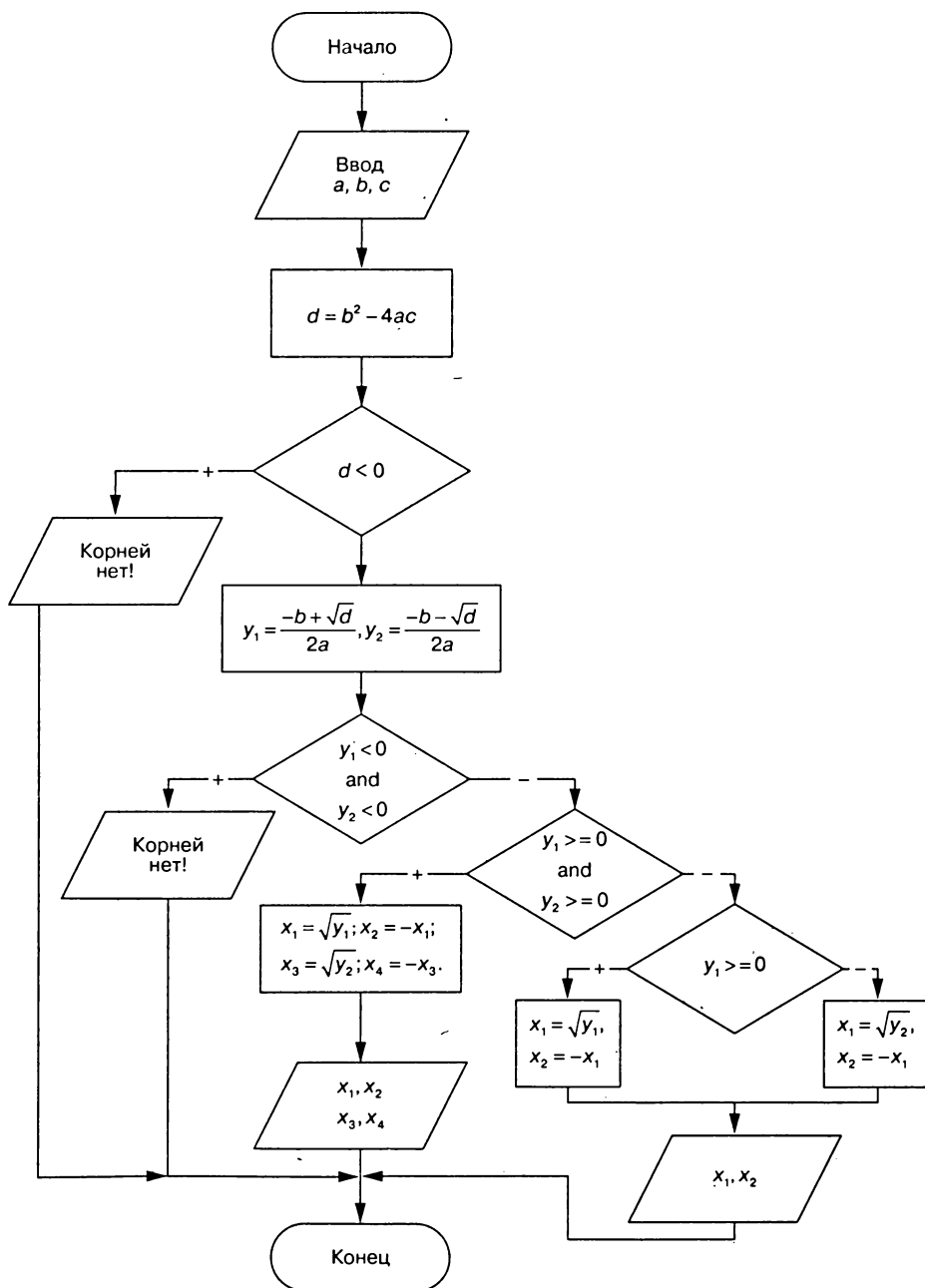


Рис. 1.12 ▼ Алгоритм решения биквадратного уравнения

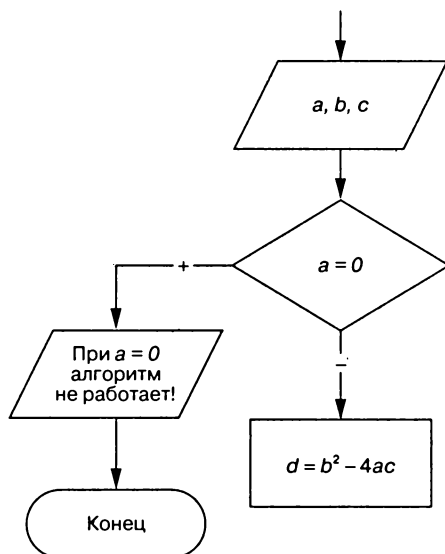
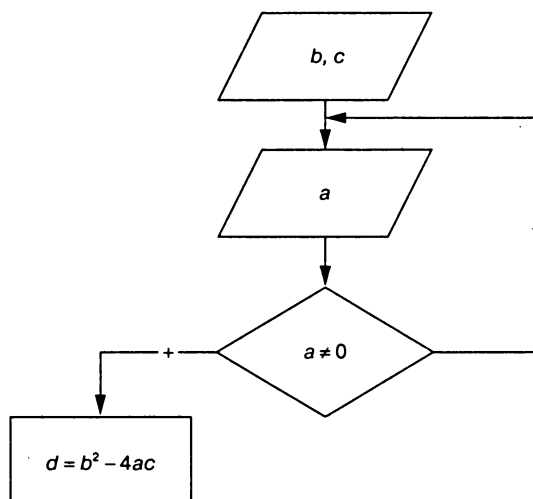


Рис. 1.13 ▼ Проверка входных данных

Рис. 1.14 ▼ Повторение ввода переменной a

! *Перед вычислением значения математического выражения последнее следует проанализировать: при каких значениях переменных выражение имеет смысл. В алгоритме необходимо предусмотреть предварительную проверку переменных на значения, для которых выражение не может быть определено. Если, например, требуется извлечь корень четной степени, то перед вычислением необходимо проверить подкоренное выражение – оно не должно быть отрицательным, а в случае нахождения значения дроби – проверить знаменатель – он не должен быть равным 0. В блок-схеме такие проверки реализуются с помощью условного блока. Отсутствие подобных проверок в программе может привести к критическим ошибкам.*

1.2.3. Алгоритмы циклической структуры

Циклом в программировании называют повторение одних и тех же действий (шагов). Последовательность действий, которые повторяются в цикле, называют телом цикла.

Существует два типа алгоритмов циклической структуры:

- цикл с предусловием (рис. 1.15);
- цикл с постусловием (рис. 1.16).

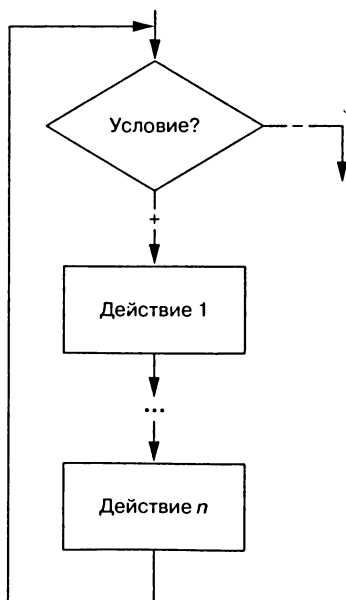


Рис. 1.15 ▼ Алгоритм циклической структуры с предусловием

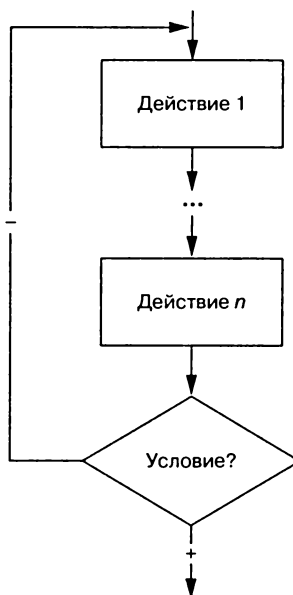


Рис. 1.16 ▼ Алгоритм циклической структуры с постусловием

Рассмотрим, в чем отличие этих типов алгоритмов:

- в цикле с предусловием условие проверяется до тела цикла, в цикле с постусловием – после тела цикла;
- в цикле с постусловием тело цикла выполняется хотя бы один раз, в цикле с предусловием оно может не выполниться ни разу;
- в цикле с предусловием проверяется условие продолжения цикла, в цикле с постусловием – условие выхода из цикла.

Оба эти цикла взаимозаменяемы, какой из них выбрать – зависит от конкретной задачи.

Рассмотрим основные алгоритмы циклической структуры на примерах.

Циклы с известным числом повторений

В этих циклах количество повторений тела цикла можно определить по исходным данным. Например, для каждого значения переменной x – *параметра* цикла (которая принимает значения в диапазоне от xn до xk , изменяясь шагом dx) – нужно вычислить значение переменной y по заданной формуле. Вычисление каждого значения будет происходить при новом повторении цикла. Количество повторений тела цикла можно определить так: $n = (xk - xn) \text{ div } dx + 1$, где div – операция целочисленного деления.

ПРИМЕР 1.6. Вычислить значения y , соответствующие каждому значению x ($xn \leq x \leq xk$, шаг изменения x равен dx), по формуле $y = e^{\sin(x)} \cos(x)$. Вычислить сумму положительных значений y , произведение ненулевых y , количество отрицательных y .

Входные данные: xn, xk, dx .

Выходные данные: множество значений y , S – сумма положительных y , P – произведение ненулевых y , k – количество отрицательных y .

Для каждого значения x вычисляем значение y . Алгоритм нахождения суммы следующий: до тела цикла сумме (S) присваивается значение 0; затем каждое следующее значение y добавляется к значению S и записывается в ту же переменную S . Алгоритм нахождения произведения отличается лишь тем, что переменной, в которой будет накапливаться произведение (P), до выполнения цикла присваивается значение 1. Алгоритм нахождения количества элементов заключается в наращивании переменной k на 1 (до входа в цикл переменная обнуляется).

Блок-схема алгоритма решения этой задачи представлена на рис. 1.17.

ПРИМЕР 1.7. Вычислить значения y , соответствующие каждому значению x ($xn \leq x \leq xk, dx$) по формуле:

$$y = \begin{cases} e^{-x} \sin x, & |x| \leq a \\ e^{-x^2} \cos x, & |x| > a \end{cases}.$$

Найти максимальное и минимальное значение y .

Входные данные: xn, xk, dx, a .

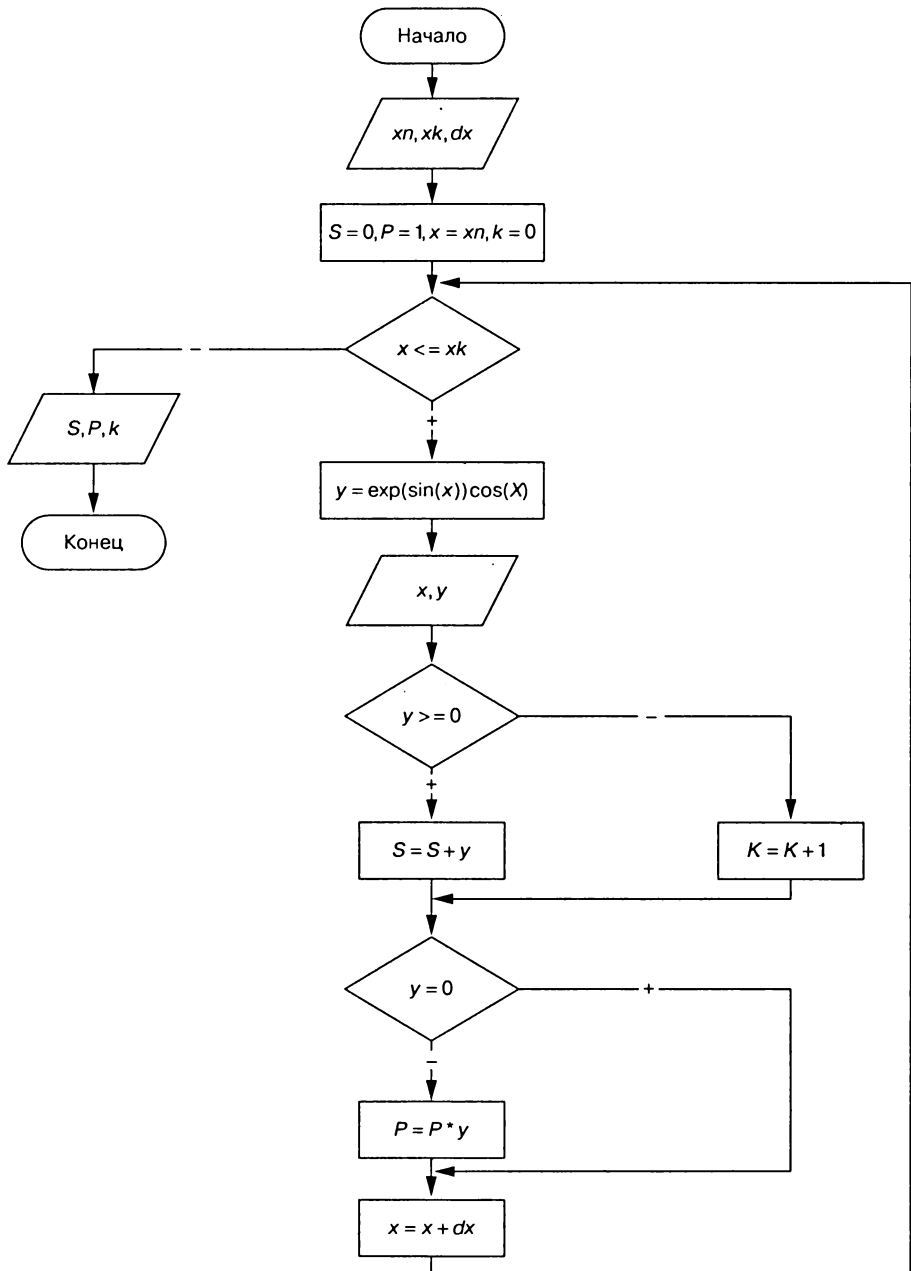


Рис. 1.17 ▼ Блок-схема алгоритма примера 1.6

Выходные данные: множество значений y , y_{\max} – максимальное значение y , y_{\min} – минимальное значение y .

Алгоритм поиска минимального (максимального) значения переменной y можно описать следующим образом. Пусть минимальное (максимальное) значение y хранится в переменной y_{\min} (y_{\max}). До начала цикла этой переменной присваивается очень большое (маленькое) значение, например 10^{20} (-10^{20}). Затем в цикле сравниваются значения y и y_{\min} (y_{\max}), и, если встречается значение y меньшее (большее), чем хранится в y_{\min} (y_{\max}), то его необходимо записать в переменную y_{\min} (y_{\max}). Таким образом, все значения y неявно сравниваются между собой, а по окончании цикла в переменной y_{\min} (y_{\max}) будет храниться y наименьшее (наибольшее) среди значений y .

Блок-схема алгоритма изображена на рис. 1.18.

В рассмотренных примерах можно заранее вычислить количество повторений, зная начальное и конечное значения переменной цикла и шаг ее изменения.

Циклы с неизвестным числом повторений

В этих циклах неизвестно, сколько раз повторится тело цикла (более того, неизвестно, закончится ли цикл вообще). Поэтому вместо конечного значения числа повторений существует условие окончания, то есть цикл заканчивается, если выполняется некоторое условие. Рассмотрим несколько примеров циклов с неизвестным числом повторений.

ПРИМЕР 1.8. Вычислить значения z , которые соответствуют каждому значению x ($x = 1$; $dx = 0,5$) по формуле: $z = \ln x \cdot \sqrt{\frac{x}{x^3 + 1}}$. Считать z до тех пор, пока подкоренное выражение больше или равно 0,02. Определить k – количество вычисленных z .

Входные данные: x , dx .

Выходные данные: множество значений z , k – количество вычисленных z .

Здесь условие окончания цикла явно указано в условии примера. Цикл окончится, когда подкоренное выражение станет меньше 0,02. Количество вычисленных z определяется так же, как и в примере 1.6.

Блок-схема алгоритма изображена на рис. 1.19.

ПРИМЕР 1.9. Вычислить отрицательный корень уравнения $x^3 - x + 0,5 = 0$, используя рекуррентную формулу: $x_{k+1} = \sqrt[3]{x_k - 0,5}$ ($k = 0, 1, 2, \dots$). $x_0 = -1,3$; $\varepsilon = 10^{-4}$. Определить количество итераций (шагов цикла); вычисления прекратить при выполнении условия $|x_{k+1} - x_k| < \varepsilon$.

Входные данные: x_0 , ε (точность ε).

Выходные данные: k – количество итераций; x – корень.

Промежуточные данные: x_1 – последующее значение x .

В данном примере вычисляется множество значений $x_1, x_2, x_3, \dots, x_k$. Их количество заранее неизвестно, но конечное значение переменной x и является

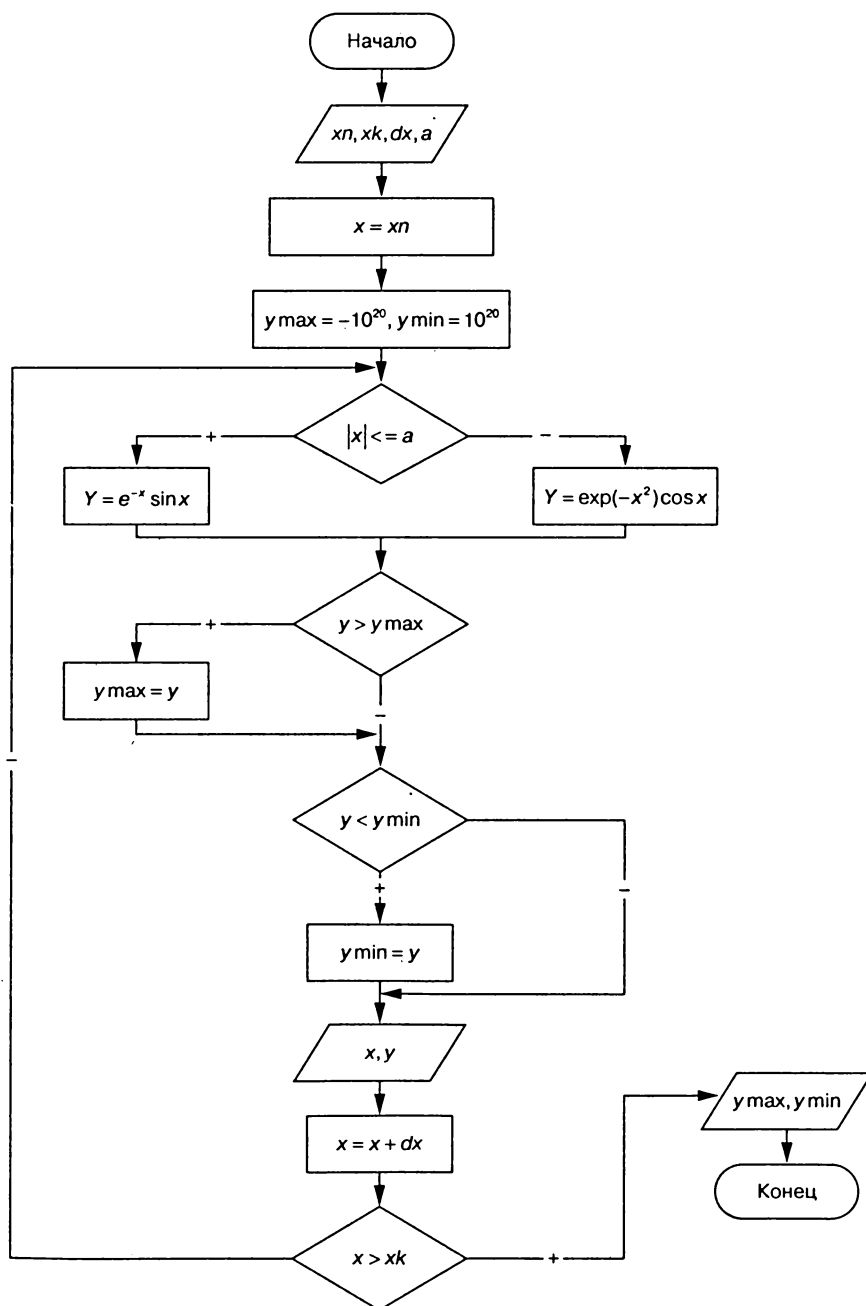


Рис. 1.18 ▼ Блок-схема алгоритма примера 1.7

корнем уравнения. В каждом цикле необходимо сравнивать текущее и предыдущее значения x . Поэтому для расчета можно использовать всего две переменные: x_0 – предыдущее значение и x_1 – текущее значение. Необходимо организовать цикл, который окончится, если модуль разности текущего и предыдущего значений станет меньше заданной точности. В теле цикла на каждом новом шаге в переменную x_0 нужно записывать текущее значение и пересчитывать x_1 по формуле.

Блок-схема алгоритма изображена на рис. 1.20.

ПРИМЕР 1.10. Вычислить значения функции

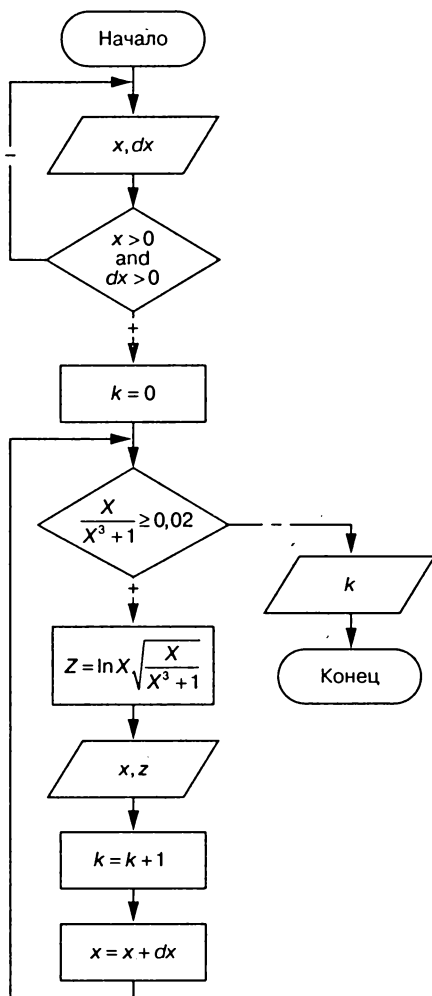


Рис. 1.19 ▼ Блок-схема алгоритма примера 1.8

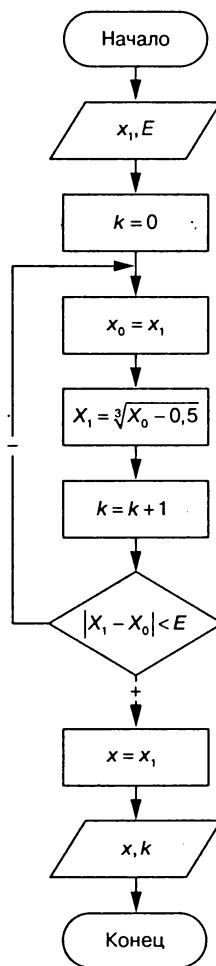


Рис. 1.20 ▼ Блок-схема алгоритма примера 1.9

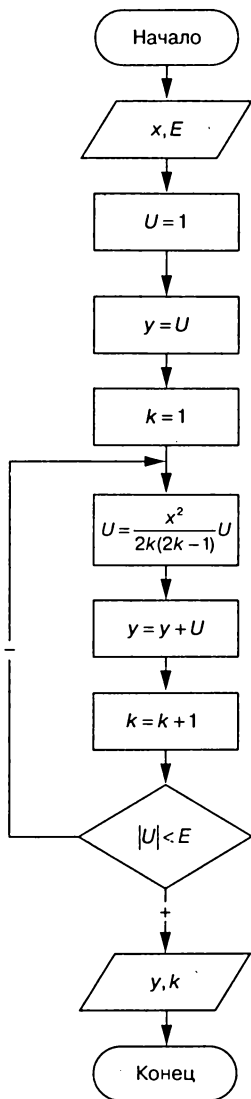


Рис. 1.21 ▼ Блок-схема алгоритма примера 1.10

$y = chx = \sum_{k=1}^{\infty} U_k$, используя рекуррентную формулу

$$U_{k+1} = \frac{x^2}{(2k-1)2k} U_k, (k = 1, 2, 3, \dots).$$

Вычисления прекратить при выполнении условия $|U_{k+1}| < \epsilon$. Определить количество итераций. Начальные значения параметров равны $U_1 = 1$; $x = 5,5$; точность ϵ принять равной 10^{-4} .

Входные данные: U – начальное приближение, x , E – точность (ϵ).

Выходные данные: y – вычисленное значение функции, k – количество итераций.

В отличие от предыдущего примера, здесь достаточно одной переменной U , в которой будет храниться текущее значение y . На каждом шаге циклического алгоритма необходимо пересчитывать значение U и добавлять к нему U . Цикл окончится, как только абсолютное значение U станет меньше E . По окончании цикла выводится y – сумма значений U .

Блок-схема алгоритма изображена на рис. 1.21.

Для решения всех рассмотренных выше примеров будет использован язык программирования Турбо Паскаль.

Глава 2

Данные в языке Турбо Паскаль. Операции и выражения

Язык Паскаль был разработан Николасом Виртом в шестидесятые годы прошлого века как учебный язык для студентов. Современный Турбо Паскаль сохранил простоту и структуру языка, разработанного Виртом. Это достаточно мощное средство программирования, предназначенное для написания программ различной сложности. На Турбо Паскале можно выполнить простые расчеты, составить программы для реализации сложных инженерных задач, обучающие программы, программы-оболочки, тестирующие программы и драйверы.

В книге будет рассматриваться седьмая версия Турбо Паскаля (TP 7.0). Однако практически без изменений большинство программ может быть перенесено в среду Free Pascal. Эта разработка американской фирмы Borland дала развитие языку Object Pascal, который лежит в основе системы визуального программирования для Windows – Delphi. Познакомившись с программированием в TP 7.0, вы сможете самостоятельно продолжить изучение этого языка, а, возможно, и перейти к программированию для Windows с использованием Delphi.

2.1. Алфавит языка

Программа на языке Паскаль может содержать следующие символы:

- латинские буквы A, B, C, ..., x, y, z;
- цифры 0, 1, 2 ..., 9;
- специальные символы +, -, /, =, <, >, [], ., (), ;, :, { }, \$, #, _, @.

В качестве имен программ, типов, констант, переменных, модулей и других объектов языка используются идентификаторы, которые представляют собой

совокупность букв, цифр и символа подчеркивания, начинающуюся с буквы или символа подчеркивания. Идентификатор не может содержать пробел. При написании могут быть использованы как прописные, так и строчные буквы. Каждое имя (идентификатор) должно быть уникальным. Длина имени не ограничена. Если в именах первые 63 символа неодинаковые, то имена считаются различными. Большие и маленькие буквы равнозначны.

2.2. Данные в языке Турбо Паскаль

Для решения задачи в любой программе выполняется обработка каких-либо *данных*, которые могут быть самых различных типов: целые и вещественные числа, символы, строки, массивы. Все данные в языке Паскаль должны быть описаны в начале программы.

Данные языка Паскаль можно разделить на *константы* и *переменные*.

2.2.1. Константы языка Турбо Паскаль

Константы не изменяют своего значения в процессе выполнения программы. Они описываются с помощью служебного слова `const`, за которым идет список имен констант, каждому из которых с помощью символа «`=`» присваивается значение. Одна константа от другой отделяется точкой с запятой, например:

```
Const  
  h=3;  
  b=-7.5;  
  c='abcde';
```

2.2.2. Переменные языка Турбо Паскаль

Переменные могут изменять свое значение в процессе выполнения программы неограниченное число раз. Описание переменных начинается со служебного слова `var`, за которым следуют имена переменных, и через двоеточие указывается их тип, например:

```
var  
  a,b: real;  
  c,d: integer;
```

2.2.3. Типы данных в языке Турбо Паскаль

Типы данных в Паскале можно разделить на *скалярные* и *структурированные*. Существует также возможность вводить собственные типы данных.

В скалярных типах данных можно выделить следующие группы.

Целочисленные типы данных занимают от 1 до 4 байт. Все они представлены в табл. 2.1.

Пример описания переменных целочисленных типов:

```
var  
  a,b:byte;  
  f:word;
```

Таблица 2.1 ▼ Целочисленные типы данных

Тип	Диапазон	Размер в байтах
Byte	0..255	1
Word	0..65535	2
Integer	-32768..32767	2
Shortint	-128..127	1
Longint	-2147483648..2147483647	4

Вещественные типы данных занимают от 4 до 10 байт. Вещественные данные могут быть как с плавающей, так и с фиксированной точкой.

Примеры вещественных чисел:

- с фиксированной точкой: 4.12, 6.05, -17.5489;
- с плавающей точкой: $-3.2E - 6(-3.2 \cdot 10^{-6})$, $-6.42E + 2(-6.42 \cdot 10^2)$.

Все вещественные типы приведены в табл. 2.2.

Таблица 2.2 ▼ Вещественные типы данных

Тип	Диапазон	Мантисса	Размер (в байтах)
Real	2.9E-39..1.7E38	11-12	6
Single	1.5E-45..3.4E38	7-8	4
Double	5.0E-324..1.7E308	15-16	8
Extended	3.4E-49321..1E4932	19-20	10

Пример описания переменных вещественных типов:

```
var
  a,b,c: real;
  d,f: double;
  k: single;
```

Символьный тип данных представляет собой любой символ, который может быть отображен на экране дисплея. Он занимает 1 байт и может быть описан с помощью служебного слова `char`, например:

```
var
  a,b: char;
```

В тексте программы значения переменных и константы символьного типа должны быть заключены в апострофы: 'a', 'b', '+'.

Логический (булевский) тип данных. Данные этого типа могут принимать два значения: `true` (истина) или `false` (ложь).

Например:

```
var
  a,b: boolean;
```

Кроме стандартных скалярных типов в Турбо Паскале существует возможность вводить такие скалярные типы, как перечислимый и интервальный.

Перечислимый тип задается непосредственным перечислением значений, которые может принимать переменная данного типа, например:

```
var
  a,c: (red,blue,green);
  b: (dog,cat);
```

Можно сначала ввести перечислимый тип данных, а затем описать переменные этого типа. Для создания нового типа используется служебное слово `type`:

```
type <имя_типа>=<определение_типа>;
```

Например:

```
type
  color=(red,blue,green);
var
  a,b: color;
```

Интервальный тип данных позволяет задавать две константы, которые определяют границы изменения переменных данного типа. Значение первой константы должно быть меньше значения второй. Сами же они являются целочисленными или символьными, например:

```
var
  a,b,c: -7..4;
  x: 'a'..'c';
```

Как и в случае перечислимого типа, можно предварительно ввести тип данных с помощью служебного слова `type`, а затем описывать переменные данного типа.

Например:

```
type
  x=0..9;
var
  a,b: x;
```

Каждая переменная интервального типа занимает 1 байт.

К *структурированным* типам данных относятся: массивы, строки, записи, файлы, множества.

Массив – совокупность данных одного и того же типа. Число элементов массива фиксируется при описании типа и в процессе выполнения программы не изменяется. Для доступа к элементу необходимо указать имя массива и его номер в квадратных скобках. Для описания массивов используется служебное слово `array`. Описание переменной данного типа имеет следующий вид:

```
<имя_переменной>: array[i..i1,j..j1,...] of <тип_элементов>;
```

где i, i_1 – границы первого индекса массива; j, j_1 – границы второго индекса массива.

Например:

```
var
  a: array[1..10] of integer;
```


Можно сначала определить тип данных массива, а затем описывать переменные этого типа, как и в случае со скалярными типами.

Строки – последовательность символов. При использовании в выражениях строка заключается в апострофы. Ее длина ограничена 255 символами. Для описания переменных строкового типа используется служебное слово `string`, например:

```
<имя_переменной>: string[n],
```

где n – длина строковой переменной; если n не указано, то длина строки равна 255 символам.

Записи и файлы будут рассмотрены ниже.

2.3. Операции и выражения в языке Паскаль

Выражение задает порядок выполнения действий над данными и состоит из операндов (констант, переменных, обращений к функциям), круглых скобок и знаков операций: $a+b*\sin(\cos(x))$. Операции делятся на унарные (например, `c`) и бинарные (например, $a+b$), а также на ниженазванные группы.

Арифметические операции, применяемые в Турбо Паскале, приведены в табл. 2.3.

Таблица 2.3 ▼ Арифметические операции

Операция	Действие	Тип операндов	Тип результата
+	Сложение	Целый, вещественный	Целый, вещественный
-	Вычитание	Целый, вещественный	Целый, вещественный
	Умножение	Целый, вещественный	Целый, вещественный
/	Деление	Целый, вещественный	Целый, вещественный
Div	Деление нацело	Целый	Целый
Mod	Остаток от деления	Целый	Целый
And	«и»	Целый	Целый
Shl	Сдвиг влево	Целый	Целый
Shr	Сдвиг вправо	Целый	Целый
Or	«или»	Целый	Целый
Xor	Исключающее «или»	Целый	Целый
-	Отрицание	Целый	Целый
Not	Логическое отрицание	Целый	Целый

Операции отношения выполняют сравнение двух операндов и определяют истинно выражение или ложно. Их результат – логический. Определены следующие операции отношения: `<`, `>`, `=`, `<=`, `>=`, `<>`.

Пример операций отношения:

```
3.14<>2, 6>4.
```

Операции отношения определены и над символьными переменными и строками:

```
'a'<'b', 'abc'<'abd'.
```

Логические операции выполняются над логическими данными. Определены следующие логические операции (табл. 2.4).

Таблица 2.4 ▼ Логические операции

A	B	Not A	A and B	A or B
t	t	f	t	t
t	f	f	f	t
f	t	t	f	t
f	f	t	f	f

В таблице t – true (истина), f – false (ложь).

В логических выражениях могут использоваться операции отношения, логические и арифметические.

Пример логического выражения:

$(a+x) > (c+d*\cos(y)) \text{ or } (a>b)$

В сложных выражениях порядок, в котором выполняются операции, соответствует приоритету операций. В Паскале приняты следующие приоритеты:

1. Унарные операции.
2. *, /, div, mod, and, shl, shr.
3. +, -, or, xor.
4. =, <>, >, <, >=, <=.

Использование скобок в выражениях позволяет менять порядок вычислений.

2.4. Стандартные функции в языке Паскаль

В Турбо Паскале определены *стандартные функции* над арифметическими операндами (табл. 2.5):

Таблица 2.5 ▼ Некоторые стандартные функции

Обозначение	Тип аргументов	Тип результата	Действие
Abs (X)	Целый, вещественный	Целый, вещественный	Модуль числа
Sin (X)	Вещественный	Вещественный	Функция синус
Cos (X)	Вещественный	Вещественный	Функция косинус
Arctan (X)	Вещественный	Вещественный	Арктангенс
Pi		Вещественный	π
Exp (X)	Вещественный	Вещественный	e^x
Ln (X)	Вещественный	Вещественный	Функция нат. логарифма
Sqr (X)	Вещественный	Вещественный	x^2
Sqrt (X)	Вещественный	Вещественный	\sqrt{x}
Int (X)	Вещественный	Вещественный	Целая часть числа
Frac (X)	Вещественный	Вещественный	Дробная часть числа
Round (X)	Вещественный	Целый	Округление числа X
Trunc (X)	Вещественный	Целый	Отсечение дробной части числа X

Таблица 2.5 ▼ Некоторые стандартные функции (окончание)

Обозначение	Тип аргументов	Тип результата	Действие
Random		Вещественный	Случайное число от 0 до 1
Random (n)	Целый	Целый	Случайное число от 0 до n

Остальные часто встречающиеся функции (тангенс, арксинус и т.д.) моделируются из уже определенных с помощью известных математических соотношений, например:

$$\operatorname{tg}(x) = \frac{\sin(x)}{\cos(x)}.$$

Определенную проблему представляет возведение X в степень n . Если значение степени n – целое, то можно n раз перемножить X или воспользоваться формулой:

$$\begin{cases} X^n = e^{n \ln(X)}, & X > 0 \\ X^n = -e^{n \ln|X|}, & X < 0 \end{cases}.$$

Формула программируется с помощью стандартных функций на языке Паскаль:

- для положительного X – $\exp(n * \ln(x))$;
- для отрицательного X – $-\exp(n * \ln(\operatorname{abs}(x)))$.

Данную же формулу можно использовать для возведения X в дробную степень n , где n – обыкновенная правильная дробь вида k/l , а знаменатель l нечетный.

Если знаменатель l четный, это означает извлечение корня четной степени, следовательно, есть ограничения на выполнение операции.

Таким образом, для программирования выражения, содержащего возведение в степень, надо внимательно проанализировать значения, которые могут принимать X и n , так как в некоторых случаях возведение X в степень n невыполнимо.

Следует также отметить особенность использования функции `random`: перед ее применением необходимо инициализировать генератор случайных чисел, выполнив процедуру `randomize`.

Строковые функции будут рассмотрены при изучении переменных строкового типа в разделе 11.1. С другими функциями можно ознакомиться в книгах по Турбо Паскалю [3, 8–10].

3 Глава

Структура программы на языке Паскаль. Операторы языка Паскаль

В этой главе будут рассмотрены особенности построения программы на Турбо Паскале, в том числе назначение и содержание разделов, синтаксис и применение операторов. Здесь также приведены примеры различных языковых конструкций.

3.1. Структура программы на языке Турбо Паскаль

Программа, написанная на языке Турбо Паскаль, имеет следующую структуру:

- заголовок программы;
- раздел описаний;
- тело программы.

Заголовок программы состоит из служебного слова `program`, имени программы, образованного по правилам использования идентификаторов Паскаля, и точки с запятой, например:

```
program my_prog001;
```

Раздел описаний включает следующие подразделы:

- раздел описания констант;
- раздел описания типов;
- раздел описания переменных;
- раздел описания процедур и функций.

В языке Турбо Паскаль должны быть описаны все переменные типы, константы, которые будут использоваться программой. В стандартном Паскале порядок следования разделов в программе жестко установлен, в Турбо Паскале такого строгого требования нет. В программе может быть несколько разделов описания констант, переменных и т.д.

Структура программы на языке Паскаль:

```
program <имя_программы>;
  const <описания_констант>;
  type <описания_типов>;
  var <описания_переменных >;
begin
  <операторы_языка>;
end.
```

Тело программы начинается со слова `begin`, затем следуют операторы языка Паскаль, реализующие алгоритм решаемой задачи. *Операторы* в языке Паскаль отделяются друг от друга точкой с запятой и могут располагаться в одну строчку или начинаться с новой строки (в этом случае их также необходимо разделять точкой с запятой). Назначение символа «`;`» – отделение операторов друг от друга. Тело программы заканчивается служебным словом `end`. Несмотря на то что операторы могут располагаться в строке как угодно, рекомендуется размещать их по одному в строке, а в случае сложных операторов отводить для каждого несколько строк. Рассмотрим более подробно структуру программы:

```
program <имя_программы>;
  const <описания_констант>;
  type <описания_типов>;
  var <описания_переменных >;
begin
  <оператор_1>;
  <оператор_2>;
  ...
  <оператор_n>
end.
```

В текст программы на Паскале могут быть включены комментарии в фигурных скобках (`{это комментарий}`) или в круглых скобках в сопровождении символа «`*`» (`(* это тоже комментарий *)`). Комментарии не выполняются программой, а служат для пояснения отдельных ее частей. Приведем пример текста программы на Паскале:

```
program one;
const
  a=7;
var
  b,c: real;
begin
  c:=a+2; b:=c-a*sin(a)
end.
```

3.2. Простейшие операторы языка Паскаль

В языке Паскаль есть *простые* и *структурированные* операторы. Рассмотрим сначала первые.

3.2.1. Оператор присваивания

В операторе присваивания слева всегда стоит имя переменной, а справа – значение, например:

```
a:=b;
```

где a – имя переменной или элемента массива,

b – значение как таковое, выражение, переменная, константа или функция.

Типы переменных a и b должны совпадать или быть совместимыми для присваивания, то есть тип, к которому принадлежит переменная b , должен находиться в границах типа переменной a .

В результате выполнения оператора $a:=b$ переменной a присваивается значение b , например:

```
var
  a,b,c,d:real;
begin
  c:=pi/2;
  d:=sin(pi*c)*cos(c)*ln(c);
  a:=(c+d)/(c-d)*exp(-c);
  d:=sqrt(c)*exp(1/9*ln(c));
end.
```

3.2.2. Операторы ввода-вывода

Ввод информации с клавиатуры осуществляется с помощью оператора `read`. Он может иметь один из следующих форматов:

```
read (x1,x2, ...,xn);
```

или

```
readln (x1,x2, ...,xn);
```

где x_1, x_2, \dots, x_n – список вводимых переменных.

Когда в программе встречается оператор `read`, ее действие приостанавливается до тех пор, пока не будут введены исходные данные. Вводимые переменные отделяются друг от друга пробелом или **Enter**.

Так чем же отличаются `read` и `readln`? Оператор `readln` аналогичен оператору `read`, разница заключается в том, что после считывания последнего в списке значения для одного оператора `readln` данные для следующего оператора `readln` будут считываться с начала новой строки. Но следует помнить, что **Enter** переведет курсор на новую строку независимо от того, как именно происходит считывание данных. При введении числовых значений большой разницы между `read` и `readln` нет. При вводе строковых переменных лучше использовать оператор `readln`. Строковые значения вводятся подряд

или отделяются нажатием клавиши **Enter**. Более подробно ввод строковых переменных будет рассмотрен в разделе 8.1.

Пример:

```
program three;
var
  a,b,c:real;
begin
  read(a,b);
  c:=a+b
end.
```

Для *вывода информации* (чисел, строк и булевых значений) на экран дисплея служат операторы `write` и `writeln`. В общем случае операторы `write` и `writeln` имеют вид:

```
write(x1,x2, ...,xn); writeln(x1,x2, ...,xn);
```

где x_1, x_2, \dots, x_n представляют собой список выводимых переменных, констант, выражений (x_1, x_2, \dots, x_n не могут быть перечислимого типа).

Операторы `write` и `writeln` последовательно выводят все переменные на экран дисплея. Если используется оператор `writeln`, то после вывода информации *курсор перемещается на новую строку*.

Вещественные данные выводятся в формате с плавающей точкой. Ширина поля вывода в этом случае составляет 18 символов: `#####E±###`, где `#` – любая десятичная цифра от 0 до 9, например:

```
0.344300000000E-01    0.03443
-5.44317180000E+02    -544.31718
```

Как видно, число, стоящее после `E`, – это степень, в которую необходимо возвести число 10, и затем результат умножить на число, стоящее перед `E`.

Рассмотрим фрагмент программы на Паскале:

```
r:=17.42;
c:=-0.0001342;
k:=12;
writeln(r);
writeln(c);
writeln(k);
```

В результате выполнения этого фрагмента на экране появятся следующие числа:

```
1.7420000000E+01
-1.3420000000E-04
12
```

Попробуйте самостоятельно разобраться, чем отличается вывод в следующих трех программах:

```
var a,b,c:real;          var a,b,c:real;
begin                    begin
a:=174.256;              a:=174.256;
```

```

b:=-13.6671512;          b:=-13.6671512;
c:=24316.1196673;        c:=24316.1196673;
write(a);                write(a);
write(b);                writeln(b);
write(c);                write(c);
end.                     end.

var a,b,c:real;
begin
a:=174.256; b:=-13.6671512; c:=24316.1196673;
writeln(a); writeln(b); writeln(c);
end.

```

После изучения операторов `read` и `write` можно написать несколько простейших программ.

ПРИМЕР 3.1. Заданы длины трех сторон треугольника a , b , c . Вычислить периметр и площадь его. Значения a , b , c ввести с клавиатуры. Блок-схема алгоритма приведена на рис. 1.7.

```

program four;
var
  a,b,c,s,p:real;
begin
  write('a='); readln(a);
  write('b='); readln(b);
  write('c='); readln(c);
  p:=(a+b+c)/2;
  s:=sqrt(p*(p-a)*(p-b)*(p-c));
  writeln('периметр треугольника', 2*p);
  writeln('площадь треугольника', s);
end.

```

ПРИМЕР 3.2. Заданы длины двух катетов в прямоугольном треугольнике. Вычислить длину гипотенузы, площадь треугольника, величины его углов. Блок-схема алгоритма приведена на рис. 1.9.

```

program five;
var
  a,b,c,s,alf,bet: real;
begin
  writeln('введите длины катетов');
  readln(a,b);
  c:=sqrt(sqr(a)+sqr(b));
  alf:=arctan(b/a);
  bet:=pi/2-alf;
  s:=a*b/2;
  writeln('гипотенуза равна', c);
  writeln('площадь треугольника', s);
  writeln('углы треугольника', pi/2,alf,bet);
end.

```


3.2.3. Форматированный вывод информации

В операторах `write` (`writeln`) имеется возможность указать константу (или выражение) целочисленного типа, определяющую ширину поля вывода. Для целых и строковых выражений она указывается через двоеточие после имени выводимой переменной или выражения, например:

```
var
  a: string[15];
  b,c: integer;
  d: word;
begin
  readln(c,b);
  readln(a); readln(d);
  writeln(a:18);
  writeln(c:6,b:5,d:7)
end.
```

При выводе вещественных значений, кроме ширины поля вывода, через двоеточие надо указывать количество позиций, необходимых для дробной части числа. При форматированном выводе вещественных чисел эти числа выйдут в формате с фиксированной точкой.

ПРИМЕР 3.3. Заданы радиус основания и высота цилиндра. Вычислить площадь основания и объем. Блок-схема алгоритма приведена на рис. 1.8.

```
var
  s,v,r,h:real;
begin
  writeln('Введите R и H');
  readln(r,h);
  v:=pi*sqr(r)*h;
  s:=pi*sqr(r);
  writeln(' v=', v:6:2);
  writeln(' s=', s:8:3);
end.
```

Рассмотрим более подробно форматированный вывод вещественного числа `write(a:m:n)`, где m – ширина поля вывода, n – количество знаков в дробной части числа. Если число не помещается в m позиций, то поле вывода расширяется до минимально необходимого. В связи с этим допустимыми являются следующие форматы `a:2:2`, `a:1:2`, `a:0:2`, `a:1:2`.

3.2.4. Составной оператор

Составной оператор – группа операторов, отделенных друг от друга точкой с запятой, начинающихся со служебного слова `begin` и заканчивающихся служебным словом `end`.

```
begin
  <оператор_1>;
  ...
  <оператор_n>
end;
```

Транслятор воспринимает составной оператор как единый.

3.3. Структурные операторы языка Паскаль

К структурным операторам относятся условные операторы и операторы цикла.

3.3.1. Условный оператор **if ... then ... else**.

Условный оператор **if** служит для организации процесса вычислений в зависимости от какого-либо логического условия. Оператор имеет вид:

```
if <условие> then <оператор_1> else <оператор_2>;
```

В качестве условия должно использоваться логическое значение, представленное константой, переменной или выражением, например:

```
a:=2; b:=8;
if a>b then
  writeln('a больше b')
else writeln('a меньше b');
```

Если условие истинно, то выполняется оператор (простой или составной), следующий за словом **then**. Но если условие ложно, то будет выполняться оператор, следующий за словом **else**. Альтернативная ветвь **else** может отсутствовать, если в ней нет необходимости. В таком «усеченном» операторе в случае невыполнения условия ничего не происходит, и управление передается следующему оператору, например:

```
a:=2; b:=8; c:=0;
if a>b then
  begin
    writeln('a>b');
    c:=a+b
  end;
c:=c+12;
writeln('c=', c:2);
```

Условные операторы могут быть *вложены* друг в друга.

```
if <условие> then
  if <подусловие> then
    begin
      ...
    end
  else
```

```

begin
...
end
else
begin
...
end;

```

При вложениях всегда действует *правило*: альтернатива else считается принадлежащей *ближайшему* условному оператору if, имеющему ветвь else. Следовательно, есть риск сделать ошибку, например:

```

. If <условие_1> then
  if <условие_2> then
    <оператор_A>
else
  <оператор_B>;

```

По записи похоже, что <оператор_B> будет выполняться, только если <условие_1> ложно, но в действительности он будет отнесен к <условию_2>. При этом «;» после <оператора_A> только ухудшит положение. Выход таков: нужно представить вложенное условие как составной оператор:

```

.if <условие_1> then
begin
  if <условие_2> then
    <оператор_A>;
end
else
  <оператор_B>;

```

В этом случае для ветви else ближайшим незакрытым оператором if окажется оператор с <условием_1>.

ПРИМЕР 3.4. Написать две программы решения квадратного и биквадратного уравнений. Блок-схемы алгоритмов решения этих задач приведены на рис. 1.5 и 1.12.

Программа решения квадратного уравнения следующая:

```

Program kvadrat;
var
  a,b,c,d,x1,x2: real;
begin
  writeln('Введите коэффициенты квадратного уравнения');
  readln(a,b,c);
  { Вычисление дискриминанта. }
  d:=b*b-4*a*c;
  if d<0 then writeln('Корней нет')
  else
    begin
      x1:=(-b+sqrt(d))/2/a;
      x2:=(-b-sqrt(d))/(2*a);
      writeln('X1=',x1:6:3, ' X2=',x2:6:3)
    end
end.

```

Программа решения биквадратного уравнения такова:

```

Program bikvadrat;
Var
  a,b,c,d,x1,x2,x3,x4,y1,y2: real;
begin
  writeln('Введите коэффициенты биквадратного уравнения');
  readln(a,b,c);
  d:=b*b-4*a*c;
  if d<0 then
    writeln('Корней нет')
  else
    begin
      y1:=(-b+sqrt(d))/2/a;
      y2:=(-b-sqrt(d))/(2*a);
      if (y1<0) and (y2<0) then
        writeln('Корней нет')
      else if (y1>=0) and (y2>=0) then
        begin
          x1:=sqrt(y1);
          x2:=-x1;
          x3:=sqrt(y2);
          x4:=-sqrt(y2);
          writeln('X1=',x1:6:3,' X2=',x2:6:3);
          writeln('X3=',x3:6:3,' X4=',x4:6:3);
        end;
      else if (y1>=0) then
        begin
          x1:=sqrt(y1);
          x2:=-x1;
          writeln('X1=',x1:6:3,' X2=',x2:6:3);
        end;
      else
        begin
          x1:=sqrt(y2);
          x2:=-x1;
          writeln('X1=',x1:6:3,' X2=',x2:6:3);
        end;
      end;
    end;
end.

```

3.3.2. Оператор варианта case

Оператор варианта case необходим в тех случаях, когда в зависимости от значений какой-либо переменной надо выполнить те или иные операторы.

```

case <управляющая_переменная> of
  <набор_значений_1>: оператор_1;
  <набор_значений_2>: оператор_2;
  ...
  <набор_значений_N>: оператор_N
else
  <альтернативный_оператор>
end;

```

Оператор работает следующим образом. Если управляющая_переменная принимает значение из набора_значений_1, то выполняется оператор_1. Если управляющая_переменная принимает значение из набора_значений_2, то выполняется оператор_2. Если управляющая_переменная принимает значение из набора_значений_N, то выполняется оператор_N. Если управляющая переменная не принимает ни одно значение из имеющихся наборов, то выполняется альтернативный_оператор.

Тип управляющей_переменной, которая стоит между служебными словами case и of, должен быть только перечислимым (включая char и boolean) диапазоном или целочисленным. Набор_значений – это конкретное значение управляющей переменной или выражение, при котором необходимо выполнить соответствующий оператор, игнорируя остальные варианты. Ключевое слово else может отсутствовать.

Рассмотрим пример, в котором err (переменная типа word) принимает некоторые целочисленные значения, соответствующие коду ошибки завершения программы. В зависимости от значения переменной err на экран выводится соответствующее сообщение:

```
case err of
  0: writeln('Нормальное завершение программы');
  2,4,6: begin
    writeln('Ошибка при работе с файлом');
    writeln('Повторите действия снова');
  end;
  7..99: writeln('Ошибка с кодом',err);
else
  writeln('Код ошибки=',err,'Смотри описание');
end.
```

Блок-схема алгоритма работы этого оператора приведена на рис. 3.1.

Значения в каждом наборе должны быть уникальны, то есть они могут появляться только в одном варианте. Пересечение наборов значений для разных вариантов является ошибкой.

3.3.3. Оператор цикла с предусловием while ... do

Обращение к оператору while ... do переводится как «пока ... делать» и выглядит так:

```
while <условие> do <оператор>;
```

Оператор, стоящий после служебного слова do и называемый телом цикла, будет выполняться циклически, пока логическое условие истинно. Само условие может быть логической константой, переменной или логическим выражением.

Условие выполнения тела цикла while проверяется до начала каждой итерации. Поэтому, если условие сразу не выполняется, то тело цикла игнорируется, и управление передается оператору, стоящему сразу за телом цикла.

Оператор while ... do предназначен для реализации циклов с предусловием.

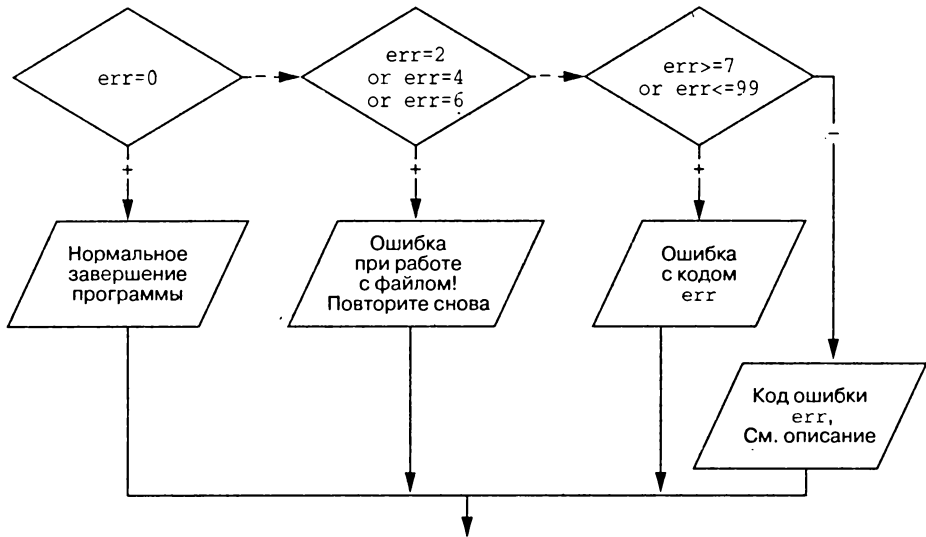


Рис. 3.1 ▼ Алгоритм применения оператора case

При написании циклов с предусловием следует помнить следующее. Во-первых, чтобы цикл имел шанс когда-нибудь закончиться, содержимое его тела должно обязательно влиять на условие цикла. Во-вторых, условие должно состоять из корректных выражений и значений, определенных еще до первого выполнения тела цикла.

ПРИМЕР 3.5. Вычислить значения y , соответствующие каждому значению x ($x_n \leq x \leq x_k$, шаг изменения x равен dx) по формуле $y = e^{\sin(x)} \cos(x)$. Вычислить сумму положительных значений y , произведение ненулевых y , количество отрицательных y (блок-схема приведена на рис. 1.17).

```

var
  x,xn,xk,dx,y,s,p:real;
  k:integer;
begin
  { Ввод исходных данных. }
  write('xn=');readln(xn);
  write('xk='); readln(xk);
  write('dx='); readln(dx);
  s:=0; { Присвоение начальных значений }
  p:=1; { для подсчета суммы s, произведения p }
  k:=0; { и количества k. }
  { Присвоение параметру цикла начального значения. }
  x:=xn;
  { Пока условие истинно, выполнять тело цикла. }
  while x<=xk do
    begin { Начало цикла while. }
      y:=exp(sin(x))*cos(x);
      writeln('x=',x:1:2,' y=',y:1:2);
    end
  end

```

```

    if y>=0 then s:=s+y
    else k:=k+1;
    if y<>0 then p:=p*y;
    x:=x+dx; { Нарастивание параметра цикла. }
end; { Конец цикла while. }
writeln('s=',s:1:2,' p=',p:1:2,' k=',k);
end.

```

ПРИМЕР 3.6. Вычислить значения z , которые соответствуют каждому значению x ($x = 1$; $dx = 0,5$), по формуле: $z = \ln x \sqrt{\frac{x}{x^3+1}}$. Считать z до тех пор, пока подкоренное выражение больше или равно 0,02. Определить k – количество вычисленных z (блок-схема приведена на рис. 1.19).

```

var
  dx,x,xn,z:real;
  k:integer;
begin
  { Ввод исходных данных. }
  write('xn='); readln(xn);
  write('dx='); readln(dx);
  k:=0; { Присвоение начального значения k. }
  x:=xn; { Присвоение начального значения xn. }
  while x/(x*x*x+1)>=0.02 do
  { Пока условие истинно, выполнять тело цикла. }
    begin { Начало цикла while. }
      z:=ln(x)*sqrt(x/(x*x*x+1));
      writeln(' x=',x:1:2,' z=',z:1:2);
      k:=k+1;
      x:=x+dx; { Нарастивание параметра цикла. }
    end; { Конец цикла while. }
  writeln(' k=',k);
end.

```

Значение x изменяется внутри цикла. При этом гораздо безопаснее так писать тело цикла, чтобы оператор, влияющий на условие, был последним в блоке. Это является гарантией от нежелательных переборов.

Тело цикла может содержать другие вложенные циклы.

3.3.4. Оператор цикла с постусловием repeat ... until

В цикле с предусловием предварительной проверкой определяется, выполнять тело цикла или нет, до первой итерации. Если это не соответствует логике алгоритма, то можно использовать цикл с постусловием, то есть цикл, в котором проверяется, делать или нет очередную итерацию, лишь после завершения предыдущей. Это имеет принципиальное значение лишь на первом шаге, а далее циклы ведут себя идентично. Оператор repeat ... until реализует цикл с постусловием. Цикл с постусловием всегда будет выполнен хотя бы один раз.

```

repeat
  <оператор_1>;
  <оператор_2>;

```

```
<оператор_N>;
until<условие>;
```

В цикле `while` подразумевается такой алгоритм: пока условие истинно, выполнять операторы тела цикла. В цикле `repeat` действует другой алгоритм: выполнять тело цикла, пока не станет истинным условие, то есть пока условие ложно, выполняется цикл.

ПРИМЕР 3.7. Вычислить значение y , соответствующее каждому значению x ($x_n \leq x \leq x_k$, dx), по формуле:

$$y = \begin{cases} e^{-x} \sin x, & |x| \leq a \\ e^{-x^2} \cos x, & |x| > a \end{cases}.$$

Найти максимальное и минимальное значения y (блок-схема приведена на рис. 1.18).

```
var
  x, xn, xk, dx, a, y, ymax, ymin: real;
  k: integer;
begin
  { Ввод исходных данных. }
  write('xn='); readln(xn);
  write('xk='); readln(xk);
  write('dx='); readln(dx);
  write('a='); readln(a);
  x:=xn; { Присвоение параметру цикла начального значения. }
  ymax:=-10E20; { Присвоение начальных значений переменным. }
  ymin:=10E20; { Для подсчета минимума и максимума. }
  repeat { Начало цикла. }
    if abs(x)<=a then y:=exp(-x)*sin(x)
    else y:=exp(-sqr(x))*cos(x);
    if y>ymax then ymax:=y;
    if y<ymin then ymin:=y;
    writeln('x=',x:1:2,' y=',y:1:2);
    x:=x+dx; { Наращивание параметра цикла. }
  until x>xk; { Пока условие ложно, выполнять тело цикла. }
  writeln('ymax=',ymax:1:2,' ymin=',ymin:1:2);
end.
```

ПРИМЕР 3.8. Вычислить отрицательный корень уравнения $x^3 - x + 0,5 = 0$, используя рекуррентную формулу: $x_{k+1} = \sqrt[3]{x_k - 0,5}$ ($k = 0, 1, 2, \dots$), $x_0 = -1,3$; $\varepsilon = 10^{-4}$.

Определить количество итераций; вычисления прекратить при $|x_{k+1} - x_k| < \varepsilon$ (блок-схема приведена на рис. 1.20).

```
var
  x1, x0, x, eps: real;
  k: integer;
begin
  write('x0='); readln(x1); { Ввод начального значения x. }
```



```

write('eps='); readln(eps); { Ввод точности вычисления. }
k:=0; { Начальное значение количества итераций. }
repeat { Начало цикла. }
  x0:=x1; { Присвоение переменной x0 k-го значения. }
  x1:=exp(1/3*ln(x0-0.5)); { Вычисление (k+1)-го значения. }
  k:=k+1; { Подсчет количества итераций. }
until abs(x1-x0)<eps; { Пока условие ложно, выполнять тело цикла. }
x:=x1; { Присвоение переменной x значения корня уравнения. }
writeln('x=',x:1:2, ' k=',k);
end.

```

ПРИМЕР 3.9. Вычислить значения функции $y = chx = \sum_{k=1}^{\infty} U_k$, используя рекуррентную формулу $U_{k+1} = \frac{x^2}{(2k-1)2k} U_k$, ($k = 1, 2, 3, \dots$). Вычисления прекратить при $|U_{k+1}| < \epsilon$, определить количество итераций (блок-схема приведена на рис. 1.21).

```

var x,v,y,eps:real;
    k:integer;
begin
  write('x='); readln(x); { Ввод начального значения x. }
  write('eps='); readln(eps); { Ввод точности вычисления. }
  k:=1; { Начальное значение количества итераций. }
  v:=1; { Значение начального приближения. }
  y:=v; { Начальное значение корня. }
  repeat
    v:=sqr(x)*v/(2*k*(2*k-1)); { Расчет по рекуррентной формуле. }
    k:=k+1; { Подсчет количества итераций. }
    y:=y+v; { Вычисление корня. }
  until abs(v)<eps;
  writeln('y=',y:1:2, ' k=',k);
end.

```

3.3.5. Оператор цикла с параметром for ... do

Операторы цикла с пред- и постусловием, несмотря на то что обладают значительной гибкостью, не слишком удобны для организации «строгих» циклов, которые должны быть выполнены заданное число раз. Оператор цикла с параметром используется именно в таких случаях.

```

for <параметр_цикла>:=<начальное_знач.> to <конечное_знач.> do
  <оператор>;
for <параметр_цикла>:=<конечное_знач.> downto <начальное_знач.> do
  <оператор>;

```

Оператор, представляющий собой тело цикла, может быть простым или составным. Параметр цикла, а также диапазон его изменения могут быть только целочисленного или перечислимого типа. Параметр описывается совместно с другими переменными. Шаг цикла for всегда постоянный и равен интервалу между двумя ближайшими значениями типа параметра цикла. Например:

```

var { Описание параметров цикла. }
  i: integer; c:char; b: boolean;
begin { Вывод на печать целых чисел от 1 до 10. }
  for i:=1 to 10 do writeln(i); { Шаг цикла равен 1. }
  { Вывод на печать чисел от 10 до -10. }
  for i:=10 downto -10 do { Шаг цикла равен -1. }
    writeln(i);
  { Вывод на печать латинских символов от a до r. }
  { Параметр цикла изменяется от a до r в алфавитном порядке. }
  for c:='a' to 'r' do writeln(c); end.

```

Выполнение цикла начинается с присвоения параметру стартового значения. Затем следует проверка, не превосходит ли параметр конечное значение. Если результат проверки утвердительный, то цикл считается завершенным, и управление передается следующему за телом цикла оператору. В противном случае выполняется тело цикла, и параметр меняет свое значение на следующее согласно заголовку цикла. Далее снова производится проверка значения параметра цикла, и алгоритм повторяется. В блок-схемах такие циклы изображаются так, как показано на рис. 3.2.

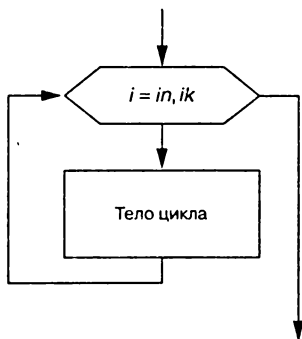


Рис. 3.2 ▼ Блок модификации в цикле с параметром

Здесь i – переменная цикла, in – начальное значение переменной цикла, ik – конечное значение переменной цикла. Блок, в котором указаны все эти значения, называется *блоком модификации*. Шаг изменения переменной цикла равен единице. Алгоритм, изображенный на рис. 3.2, работает следующим образом: переменной цикла присваивается начальное значение, затем значение переменной цикла увеличивается на единицу, и цикл повторяется. Так продолжается до тех пор, пока переменная

цикла не превысит свое конечное значение, после чего цикл заканчивается. Если шаг изменения равен -1 , то в блоке модификации мы напишем: $i := in, ik, -1$.

ПРИМЕР 3.10. Вычислить факториал числа N ($N! = 1 \cdot 2 \cdot 3 \dots N$). Блок-схема приведена на рис. 3.3.

Входные данные: N – целое число, факториал которого необходимо вычислить.

Выходные данные: `factorial` – значение факториала числа N .

Промежуточные данные: i – параметр цикла.

Вводится число N . Переменной `factorial` присваивается начальное значение. Затем организуется цикл, причем в блок-схеме для этого используется *блок модификации*. Здесь параметру цикла i присваивается начальное значение 1, и цикл повторяется до N с шагом 1. Когда параметр i становится больше N , цикл заканчивается, и на печать выводится значение переменной `factorial`, которая была вычислена в теле цикла.

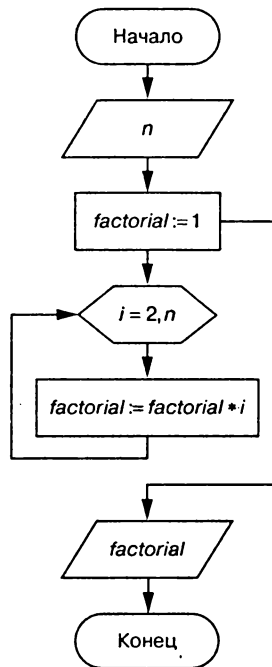


Рис. 3.3 ▼ Алгоритм вычисления факториала

Теперь рассмотрим текст программы вычисления факториала на языке Турбо Паскаль.

```

var
  factorial, n, i:integer;
begin
  write('n='); readln(n);
  factorial:=1; { Стартовое значение. }
  for i:=2 to n do
    factorial:=factorial*i;
    writeln(factorial);
  end.

```

3.3.6. Операторы **break**, **continue**, **exit**, **halt**

Операторы **break** и **continue** введены в язык Турбо Паскаль версии 7.0.

Оператор **break** осуществляет немедленный выход из циклов **repeat**, **while**, **for**. Его можно использовать только внутри циклов.

ПРИМЕР 3.11. Вычислить значение y , соответствующее каждому значению x ($x_1 \leq x \leq x_k$, dx), по формуле: $y = \ln x$.

```

Var
  x,xn,xk,dx:real;
  n,I:integer;
begin
  write('xn=');readln(xn); { Ввести начальное значение. }
  write('xk=');readln(xk); { Ввести конечное значение. }
  write('dx=');readln(dx); { Ввести шаг изменения переменной. }
  x:=xn; { Параметру цикла присвоить начальное значение. }
  while true do
  begin
    y:=ln(x);
    writeln('x=',x:6:3, ' y=',y:6:3);
    x:=x+dx;
    if x>xk then break;
    { Если значение числа x больше xk, то выйти из цикла. }
  end;
end.

```

Оператор `continue` начинает новую итерацию цикла, даже если предыдущая не была завершена. Его можно использовать только внутри цикла. Рассмотрим пример:

```

for i:=1 to n do
begin
  write('c='); readln(c); { Ввести значение c. }
  { Если значение числа x отрицательно, то начать цикл заново. }
  if c<=0 then continue;
  b:=sqrt(c)
end;

```

Оператор `exit` осуществляет выход из подпрограммы.

Оператор `halt` прекращает выполнение программы и возвращает код завершения в операционную систему. Структура оператора:

```
halt [(e)];
```

e – переменная типа `word` (код завершения).

Если e отсутствует, то код завершения 0. Рассмотрим использование операторов `exit` и `halt` на примере вычисления факториала:

```

Var
  f,i,n: integer;
begin
  readln(n); { Ввести значение n. }
  if n<0 then { Если значение числа n отрицательно, то }
  begin
    writeln('n - отрицательное'); { выдать сообщение }
    halt { и выйти из программы. }
  end
  \else
  begin
    f:=1;
    for i:=2 to n do f:=f*i
  end;
  writeln('n!= ', f);
end.

```

3.4. Упражнения по теме «Программирование линейных процессов»

Разработать блок-схему и программу на Турбо Паскале:

1. Заданы два катета прямоугольного треугольника. Найти гипотенузу и углы треугольника.
2. Тело падает с высоты h . Какова его скорость в момент соприкосновения с землей, и когда это произойдет?
3. Известна гипотенуза c и прилежащий угол α прямоугольного треугольника. Найти площадь треугольника.
4. Диагональ квадрата равна d . Вычислить площадь S и периметр P квадрата.
5. Известна диагональ прямоугольника d и угол α между диагональю и большей стороной. Вычислить площадь S прямоугольника.
6. Величины сторон треугольника составляют a , b , c соответственно. Найти углы треугольника α , β , γ .
7. Тело имеет форму параллелепипеда с высотой h . Прямоугольник в основании имеет диагональ d . Известно, что диагонали основания пересекаются под углом α . Найти объем тела V и площадь поверхности S .
8. В треугольнике известен катет a и площадь S . Найти величину гипотенузы c , второго катета b и углов α и β .
9. Площадь квадрата равна S . Вычислить сторону квадрата a , диагональ d и площадь S_1 описанного вокруг квадрата круга.
10. В равнобедренном треугольнике известно основание c и угол при нем α . Найти площадь треугольника S и величину боковой стороны a .

3.5. Упражнения по теме «Программирование разветвленных процессов»

Разработать блок-схему и программу на Турбо Паскале:

1. Задана точка M с координатами (x, y) . Определить месторасположение этой точки в декартовой системе координат (является ли эта точка началом координат, лежит на одной из координатных осей или расположена в одном из координатных углов).
2. Определить, поместится ли равнобедренный треугольник с основанием c и высотой h в прямоугольник со сторонами a и b так, чтобы высота треугольника была параллельна одной из сторон прямоугольника.
3. Выяснить, у какого из трех прямоугольных треугольников площадь больше:
 - гипотенуза c , угол α ;
 - катет a , прилежащий угол β ;
 - высота h , угол γ .

4. Задан параллелограмм со сторонами a , b и углом α между ними. Определить тип параллелограмма (ромб, прямоугольник или квадрат), если это возможно.
5. Известны углы α и β у основания трапеции. Выяснить, если это возможно, тип трапеции (прямоугольная, равнобедренная, прямоугольник).
6. Задан круг с центром в точке $O(x_0, y_0)$ и радиусом R_0 и точка $A(x_1, y_1)$. Определить месторасположение точки по отношению к кругу (находится внутри круга, вне его или лежит на окружности).
7. Определите, пересекаются ли парабола $y = cx^2 + dx + f$ и прямая $y = ax + b$. При положительном ответе найти точки пересечения.
8. Выяснить, пересекаются ли параболы $y = ax^2 + bx + c$ и $y = dx^2 + ex + f$. При положительном ответе найти точки пересечения.
9. Задана окружность с центром в точке $O(x_0, y_0)$ и радиусом R_0 и прямая $y = ax + b$. Определить, пересекаются ли прямая и окружность. При положительном ответе найти точки пересечения.
10. Известны длины отрезков a , b , c и d . Определить треугольники минимальной и максимальной площади, которые можно построить из этих отрезков.

3.6. Упражнения по теме «Программирование циклических вычислительных процессов»

Разработать блок-схему и программу на Турбо Паскале:

1. Вычислить значения y , соответствующие каждому значению x ($x_1 \leq x \leq x_k$,

шаг изменения x равен dx) по формуле $y = \frac{\sqrt[3]{a-x^2} \ln(a+x)}{\sqrt[3]{x^2} + \sqrt[3]{a}}$. Вычислить

сумму, произведение и количество значений y . Контрольный расчет провести при $a = 2,17$; $x_n = -1,5$; $x_k = 0,5$; $dx = 0,2$.

2. Вычислить значения z , соответствующие каждому значению x ($x_1 \leq x \leq x_k$),

шаг изменения x равен dx) по формуле $z = \frac{\sqrt[3]{x^3} + ax}{\ln \sqrt{a^2 + \sqrt{x}}}$. Определить сред-

нее арифметическое вычисленных z . Контрольный расчет провести при $a = 5,27$; $x_n = 1$; $x_k = 10$; $dx = 1$.

3. Вычислить значения t , соответствующие каждому значению x ($x_1 \leq x \leq x_k$),

шаг изменения x равен dx) по формуле $t = \frac{|a - b\sqrt[3]{x}|}{b \ln|a^2 + x|}$. Определить $F = \sum t$.

Контрольный расчет провести при $a = 3,5$; $b = 6,8$; $x_n = -3$; $x_k = 3$; $dx = 0,5$.

4. Вычислить значения t , соответствующие каждому значению x ($x_1 \leq x \leq x_k$),

шаг изменения x равен dx) по формуле $t = \frac{\sqrt[3]{ax}}{a + x \lg(a+x)}$. Вычислить сумму

- положительных значений t , произведение отрицательных t , количество t . Контрольный расчет провести при $a = 1,23$, $x_n = -0,5$, $x_k = 0,5$, $dx = 0,1$.
5. Вычислить значения z , соответствующие каждому значению x ($x_n \leq x \leq x_k$), шаг изменения x равен dx) по формуле $z = \frac{(ax)^2 \sqrt[3]{\frac{1}{(a+x)^2}}}{a \ln(a+x^2)}$. Вычислить $F = \prod_{i < a} z + \sum_{i \geq a} z$ и n – количество вычислительных z . Контрольный расчет провести при $a = 2,62$; $x_n = -3$; $x_k = 3$; $dx = 0,6$.
6. Вычислить значения t , соответствующие каждому значению x ($x_n \leq x \leq x_k$), шаг изменения x равен dx) по формуле $t = \frac{a + \sqrt{ax}}{\sqrt{a} + \ln(a+x)}$. Вычислить сумму значений $t \geq a$, произведение всех значений t , количество отрицательных t . Контрольный расчет провести при $a = 3,72$; $x_n = 1$; $x_k = 3$; $dx = 0,2$.
7. Определить значения t , соответствующие каждому значению x ($x_n \leq x \leq x_k$), шаг изменения x равен dx) по формуле $t = (a+b)^2 \sqrt{\frac{a+x}{b+x}} \ln(a+x)$. Вычислить количество отрицательных значений x . Найти минимальное значение среди вычисленных значений t . Контрольный расчет провести при $a = 6,13$; $b = 3,42$; $x_n = -2$; $x_k = 3$; $dx = 0,5$.
8. Определить значения y , соответствующие каждому значению x ($x_n \leq x \leq x_k$), шаг изменения x равен dx) по формуле $y = \frac{a^2 + x\sqrt[3]{x}}{\sqrt{a} + \sqrt[3]{x}}$. Найти максимальное значение y и среднее значение среди положительных элементов x . Контрольный расчет провести при $a = 2,89$; $x_n = -50$; $x_k = 50$; $dx = 10$.
9. Вычислить значения z , соответствующие каждому значению x ($x_n \leq x \leq x_k$), шаг изменения x равен dx) по формуле $z = a \sqrt[4]{\frac{ax}{\ln^3(a+x)}}$. Определить разницу между минимальным и максимальным значениями z . Контрольный расчет провести при $a = 2,94$; $x_n = 1,5$; $x_k = 5,5$; $dx = 0,4$.
10. Найти значения z , соответствующие каждому значению x ($x_n \leq x \leq x_k$), шаг изменения x равен dx) по формуле $z = \frac{\sqrt[3]{(a^2 - 2ab + x)}}{(a+b)^2 + e^x}$. Определить минимальное значение среди значений $z \leq 0$, максимальное среди $z > 0$ и количество вычисленных z . Контрольный расчет провести при $a = 4,32$; $b = 8,13$; $x_n = -3$; $x_k = 4$; $dx = 0,7$.

3.7. Упражнения по теме «Программирование циклов с неизвестным числом повторений»

Разработать блок-схему и программу на Турбо Паскале:

1. Дано: $q = 3$; $dq = -0,2$. F вычислять по формуле: $F = \sqrt{(1+0,5q)} - \frac{1}{q+1}$. Считать до тех пор, пока подкоренное выражение больше 0. Определить k – количество вычисленных F . Вывести на экран q , F , k .
2. Известно, что $x = 2$; $dx = -0,2$. Z вычислять по формуле: $Z = \frac{1}{\ln(x^2 - 0,5x)}$. Считать Z до тех пор, пока выражение под знаком логарифма больше 0. Найти k – количество вычисленных Z . Вывести на экран k , Z , x .
3. Дано: $a = 1,2$; $x = 1$; $dx = 0,5$. Вычислять Z по формуле: $Z = \frac{a + \sqrt{ax^3 + x}}{\sin x + 3}$. Считать Z до тех пор, пока подкоренное выражение меньше 250. Определить k – количество вычисленных Z . Вывести на экран x , Z , k .
4. Известно, что $a = 5$; $da = -0,5$. Вычислять Z по формуле: $Z = q + \frac{1}{q+1}$, где $q = a^2 - a$. Считать Z до тех пор, пока $q > 0$. Определить k – количество вычисленных Z . Вывести на экран a , q , Z , k .
5. Значение переменной x равно 1, $dx = 0,5$. Вычислять Z по формуле: $Z = q(\cos(3x) + \sin(5x))$, где $q = e^{x-1} + x$. Считать до тех пор, пока $q < 400$. Определить k – количество вычисленных Z . Вывести на экран x , q , Z , k .
6. Дано: $x = 5$; $dx = 1$. Z вычислять по формуле: $Z = y + \sqrt[3]{y} + \sqrt[5]{y}$, где $y = e^{0,2x}$. Считать Z до тех пор, пока $y < 25$. Найти k – количество вычисленных Z . Вывести на экран x , y , Z , k .
7. Вычислить квадратный корень из положительного числа A , используя рекуррентную формулу $X_{k+1} = (X_k + A/X_k)/2$ для $k = 0, 1, 2, \dots$. Определить количество итераций (шагов цикла); вычисления прекратить при $|x_{k+1} - x_k| < \epsilon$. ($x_0 = 1$; $A = 1,8$; $\epsilon = 10^{-4}$).
8. Вычислить корень степени m из числа A , используя рекуррентную формулу $X_{k+1} = ((m-1)X_k + A/X_k^{m-1})/m$ для $k = 0, 1, 2, \dots$. Определить количество итераций. Вычисления прекратить при $|U_{k+1}| < \epsilon$. ($m = 3$, $x_0 = 1$; $A = -1,2$; $\epsilon = 10^{-4}$).
9. Вычислить значение функции синус для некоторого числа $x \in [0; \pi/4]$ по формуле $y = \sum_{k=1}^{\infty} u_k$, используя $U_{k+1} = (-U_k \times X^2)/(2k \times (2k+1))$ для $k = 1, 2, \dots$. Вычисления прекратить при $|U_{k+1}| < \epsilon$, определить количество итераций. Начальные значения параметров равны $x = \pi/7$; $U_1 = x$. Точность (ϵ) принять равной 10^{-4} .
10. Вычислить действительные корни уравнения $x - \sin(x) = 0,25$, используя рекуррентную формулу $X_k = \sin(X_{k-1}) + 0,25$ ($k = 1, 2, \dots$). $x_0 = 1,2$; $\epsilon = 10^{-4}$. Определить количество итераций; вычисления прекратить при $|x_{k+1} - x_k| < \epsilon$.

Глава

Численное решение трансцендентных и нелинейных уравнений

В этой главе рассмотрим реальную алгебраическую задачу – решение нелинейного уравнения. Если алгебраическое или трансцендентное уравнение достаточно сложное, то его корни сравнительно редко удается найти точно. Поэтому большое значение приобретают способы приближенного нахождения корней уравнения и оценки степени их точности. Далее будут рассмотрены несколько численных методов и приведены алгоритмы нахождения корней уравнений. Следует обратить внимание на то, что все эти алгоритмы содержат циклы с неизвестным числом повторений, которые были рассмотрены ранее.

В общем случае аналитическое решение уравнения

$$f(x) = 0 \quad (4.1)$$

можно найти только для узкого класса функций. В большинстве случаев приходится решать уравнение (4.1) численными методами в два этапа. Сначала необходимо *отделить корни уравнения*, то есть найти достаточно тесные промежутки, называемые *интервалами изоляции корня*, – в них содержится только один корень. Далее проводят *уточнение отделенных корней*, то есть находят корни с заданной точностью.

Интервал можно выделить графически, изобразив график функции, или каким-либо другим способом, основанным на следующем свойстве непрерывной функции $f(x)$: если функция непрерывна на интервале $[a, b]$ и на его концах имеет различные знаки, $f(a)f(b) < 0$, то между точками имеется хотя бы один корень. Мы будем считать интервал настолько малым, что в нем находится только один корень. Рассмотрим самый простой способ уточнения корней.

4.1. Метод половинного деления

Метод половинного деления (дихотомии) осуществляется на практике следующим образом. Пусть был выбран интервал изоляции $[a, b]$ (рис. 4.1.). Примем за первое приближение корня точку c , которая является серединой отрезка $[a, b]$. Далее будем действовать по следующему алгоритму:

1. Находим точку $c = (a + b) / 2$;
2. Находим значение $f(c)$;
3. Если $f(a) \times f(c) < 0$, то корень лежит на интервале $[a, c]$, в других случаях он находится на интервале $[c, b]$;

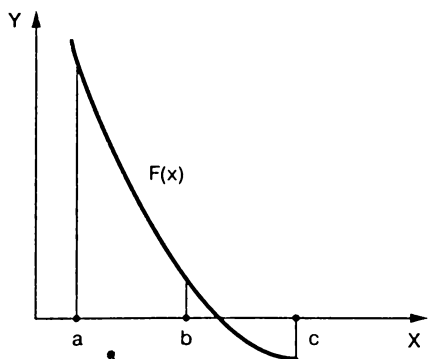


Рис. 4.1 ▼ Графическая интерпретация метода половинного деления

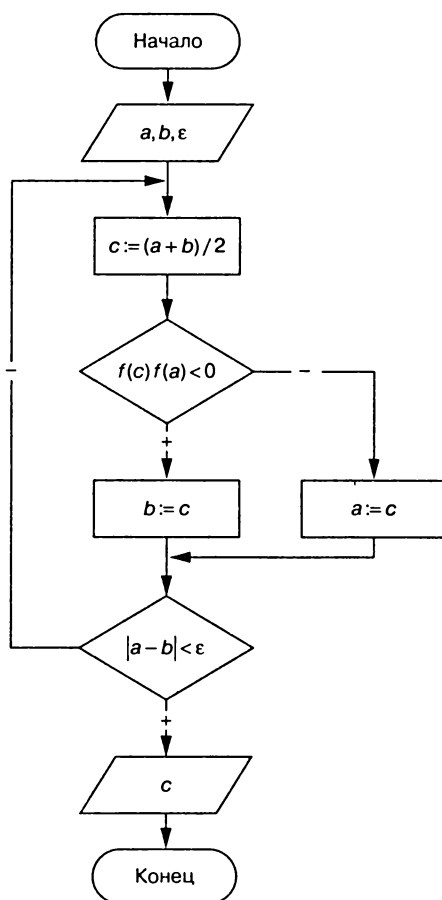


Рис. 4.2 ▼ Алгоритм решения уравнения методом дихотомии

4. Если величина интервала $\leq \epsilon$, то найден корень с точностью ϵ , иначе возвращаемся к п. 1.

Блок-схема алгоритма решения уравнения методом дихотомии приведена на рис. 4.2.

Несмотря на простоту, такое последовательное сужение интервала проводится редко, так как требует слишком большого количества вычислений. Кроме того, этот способ не всегда позволяет найти решение с заданной точностью.

Рассмотрим другие способы уточнения корня, при использовании которых будем требовать, чтобы функция $f(x)$ удовлетворяла следующим условиям на интервале $[a, b]$ [4, 6]:

- функция $f(x)$ непрерывна вместе со своими производными первого и второго порядка. Функция $f(x)$ на концах интервала $[a, b]$ имеет разные знаки $f(a) \times f(b) < 0$;
- первая и вторая производные $f'(x)$ и $f''(x)$ сохраняют определенный знак на всем интервале $[a, b]$.

4.2. Метод хорд

Метод хорд отличается от предыдущего метода тем, что очередное приближение берем не в середине отрезка, а в точке пересечения с осью X (рис. 4.3) – прямой, соединяющей точки $(a, f(a))$ и $(b, f(b))$.

Запишем уравнение прямой, проходящей через точки с координатами $(a, f(a))$ и $(b, f(b))$:

$$\frac{y - f(a)}{f(b) - f(a)} = \frac{x - a}{b - a}; \quad y = \frac{f(b) - f(a)}{b - a} \cdot (x - a) + f(a) \quad (4.2)$$

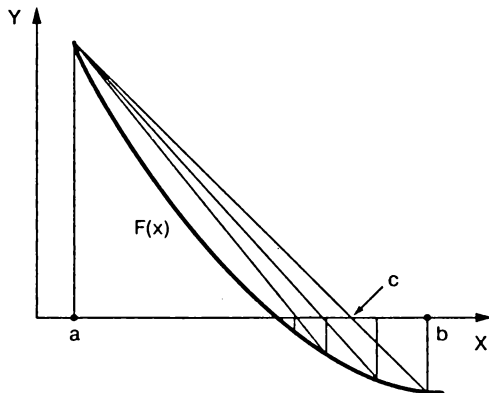


Рис. 4.3 ▼ Графическая интерпретация метода хорд

Прямая, заданная уравнением (4.2), пересекает ось X при условии $y = 0$. Найдём точку пересечения хорды с осью X :

$$y = \frac{f(b) - f(a)}{b - a} \cdot (x - a) + f(a), \quad x = a - \frac{f(a) \cdot (b - a)}{f(b) - f(a)}; \text{ итак, } c = a - \frac{f(a)}{f(b) - f(a)}(b - a).$$

Далее необходимо вычислить значение функции в точке c .

Блок-схема метода хорд представлена на рис. 4.4.

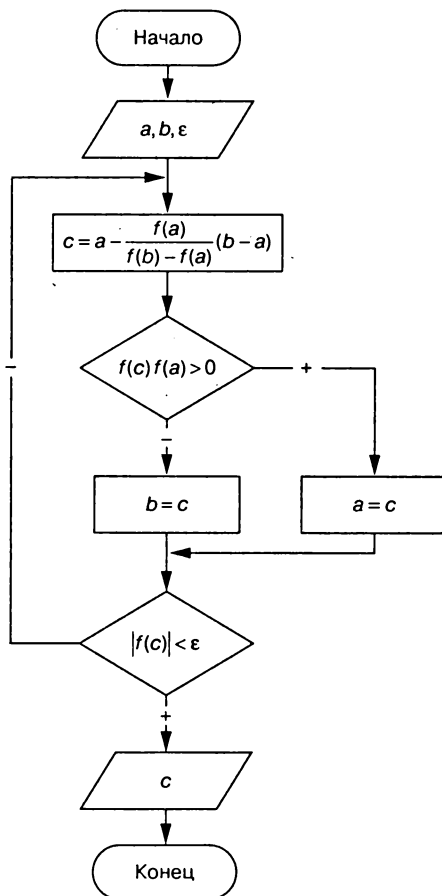


Рис. 4.4 ▼ Алгоритм метода хорд

4.3. Метод касательных

Метод касательных (метод Ньютона) на практике используется следующим образом. В одной из точек интервала $[a, b]$, например в точке c , проведем касательную (рис. 4.5). Запишем уравнение этой прямой:

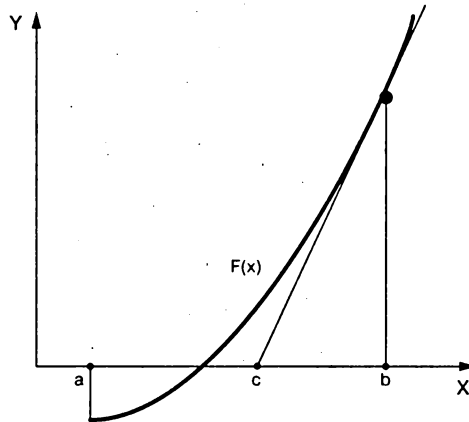


Рис. 4.5 ▼ Графическая интерпретация метода касательных

$$y = kx + m \quad (4.3)$$

Так как данная прямая является касательной, и она проходит через точку $(c, f(c))$, то $k = f'(c)$.

Отсюда следует:

$$\begin{aligned} y &= f'(c)x + m, f(c) = f'(c)c + m, m = f(c) - cf'(c), \\ y &= f'(c)x + f(c) - cf'(c), y = f'(c)(x - c) + f(c). \end{aligned}$$

Найдем точку пересечения касательной с осью X :

$$f'(c)(x - c) + f(c) = 0, \quad x = c - \frac{f(c)}{f'(c)}.$$

Если $|f(x)| < \varepsilon$, то точность достигнута, и точка x — решение; иначе необходимо переменной c присвоить значение x , провести касательную через новую точку c и так продолжать до тех пор, пока $|f(x)|$ не станет меньше ε . Осталось решить вопрос, что выбрать в качестве точки начального приближения c .

В этой точке должны совпадать знаки функции и ее второй производной. А так как нами было сделано допущение, что вторая и первая производные не меняют знак, то можно проверить условие $f(x)f''(x) > 0$ на обоих концах интервала и в качестве начального приближения взять ту точку, где оно выполняется.

Блок-схема метода Ньютона представлена на рис. 4.6.

4.4. Метод простой итерации

Для решения уравнения методом простой итерации необходимо записать уравнение (4.1) в виде $x = \varphi(x)$, задать начальное приближение $x_0 \in [a, b]$ и организовать следующий итерационный вычислительный процесс:

$$x_{k+1} = \varphi(x_k), \quad k = 0, 1, 2, \dots$$

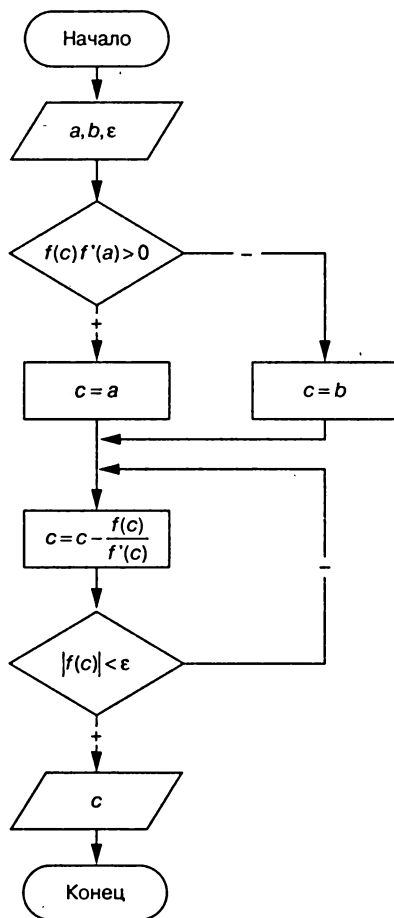


Рис. 4.6 ▼ Алгоритм метода Ньютона

Вычисление прекратить, если $|x_{k+1} - x_k| < \epsilon$ (ϵ – точность).

Если неравенство $|\varphi'(x)| < 1$ выполняется на всем интервале $[a, b]$, то последовательность $x_0, x_1, x_2, \dots, x_n$ сводится к решению x^* , то есть $\lim_{k \rightarrow \infty} x_k = x^*$.

Значение функции $\varphi(x)$ должно удовлетворять условию $|\varphi'(x)| < 1$ для того, чтобы можно было применить метод простых итераций. $|\varphi'(x)| < 1$ – это достаточное условие сходимости метода простой итерации.

А что делать, если уравнение задано в виде $f(x) = 0$? В таком случае его можно привести к виду $x = \varphi(x)$ следующим образом. Умножим обе части уравнения $f(x) = 0$ на число λ . К обеим частям уравнения $\lambda f(x) = 0$ добавим число x , в результате получим $x = x + \lambda f(x)$. Это и есть уравнение вида $x = \varphi(x)$, где $\varphi(x) = x + \lambda f(x)$. Возникает вопрос, а зачем умножать на число λ и вводить его? Но ведь необходимо, чтобы $|\varphi'(x)|$ было меньше единицы на интервале $[a, b]$, следовательно, $|\varphi'(x)| = |1 + \lambda f'(x)|$ и $|1 + \lambda f'(x)| \leq 1$, а значит, с помощью подбора параметра λ можно добиться выполнения условия сходимости.

Блок-схема метода простой итерации приведена на рис. 4.7.

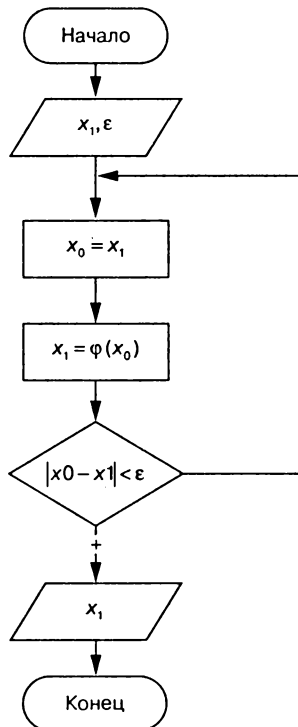


Рис. 4.7▼ Графическая интерпретация метода простой итерации

Существует еще множество методов решения уравнений: комбинированный (объединение метода хорд и касательных), метод секущих и т.д. Но и те, что рассматривались выше, позволяют получить решение большого класса уравнений с заданной точностью.

4.5. Упражнения по теме «Решение нелинейных и трансцендентных уравнений»

В табл. 4.1 приведены уравнения. Для каждого уравнения выполнить следующее:

1. Вычислить интервал изоляции одного из корней уравнения $f(x) = 0$.
2. Составить программы вычисления корней с погрешностью $\varepsilon = 0,0001$ в соответствии с указанным номером метода решения (1 – метод половинного деления, 2 – метод хорд, 3 – метод касательных, 4 – метод простой итерации). Вычислить количество итераций.

Таблица 4.1 ▼ Нелинейные и трансцендентные уравнения

№	Уравнение	Метод	№	Уравнение	Метод
1	$x - 0,2\sin(x + 0,5) = 0$	1	6	$x - \cos x = 0$	2
	$x^2 - \lg(x + 2) = 0$	3		$e^{-x} + x^2 = 0$	4
2	$x^2 - 20\sin x = 0$	2	7	$x^2 - \cos x^2 = 6$	1
	$\ln x + (x + 1)^3 = 0$	4		$(x - 1)^2 - 0,5e^x = 0$	3
3	$x^2 - \sin 5x = 0$	1	8	$\sqrt{x} - 2\cos x = 0$	2
	$e^x + x^2 = 2$	3		$2 - x = \ln x$	4
4	$1,8x^2 - \sin 10x = 0$	2	9	$3x - \cos x = 1$	1
	$2\ln x - \frac{1}{x} = 0$	4		$2\lg x - \frac{x}{2} = -1$	3
5	$\sqrt{x+1} - \frac{1}{x} = 0$	1	10	$\lg x - \frac{1}{x^2} = 0$	2
	$x \ln x - 100 = 0$	3		$2 - xe^x = 0$	4

Глава

Массивы. Алгоритмы обработки массивов

Все рассмотренные ранее алгоритмы позволяли работать со скалярными данными: числами, символами и т.д. Однако очень часто при решении задач необходимо использовать объекты, содержащие множество однотипных элементов. Для обработки таких объектов и служит массив.

Массив – структурированный тип данных, состоящий из фиксированного числа элементов одного типа: например, при обработке результатов многократных замеров температуры воздуха в течение года удобно рассматривать значения температур как массив вещественных чисел. Аналогом одномерного массива является вектор, двумерного – матрица. Рассмотрим работу с массивами на языке Турбо Паскаль.

5.1. Описание массивов

Для описания массива служат служебные слова `array of`. Сама же процедура может выполняться двумя способами:

- ввести новый тип данных, а потом описать переменные нового типа. В этом случае формат команды `type` следующий:

```
type  
<имя_типа> = array [<тип_индекса>] of <тип_компонентов>;
```

В качестве `типа_индекса` можно использовать перечислимый тип.

`Тип_компонентов` – это любой ранее определенный тип данных, например:

```
type  
massiv=array[0..12] of real;  
dabc=array[-3..6] of integer;
```

```
var
  x,y:massiv;
  z: dabc;
```

- можно не вводить новый тип, а просто описать переменную следующим образом:

```
var <переменная> : array [<тип_индекса>] of <тип_переменных>;
```

Например:

```
var
  z,x: array[1..25] of word;
  g:array[-2..7] of real;
```

Для описания массива можно использовать предварительно определенные константы:

```
const
  n=10;
  m=12;
var
  a: array[1..n] of real;
  b: array[0..m] of byte;
```

! Константы должны быть определены до использования, так как массив не может быть переменной длины!

Размер массива не может превышать 64 Кб. Это ограничение относится и к прочим структурам данных.

Двумерный массив можно описать, применив в качестве базового типа (типа компонентов) одномерный:

```
type
  massiv=array[1..20] of real;
  matrica=array[1..20] of massiv;
var
  a:matrica;
```

Такая же структура получается при использовании другой формы записи:

```
Type
  matrica = array [1..20,1..20] of real;
var
  a:matrica;
или
var
  a:array [1..20,1..20] of real;
```

Аналогично можно ввести трехмерный массив или массив большего числа измерений:

```
type
  abc=array [1..4,0..6,-7..8,3..11] of real;
var
  b:abc;
```

В качестве индекса могут выступать не только числа (целые), но и, например, перечислимые и интервальные значения. Имеет смысл следующее описание:

```
a: array ['a'..'t'] of longint;
```

5.2. Операции над массивами

Для работы с массивом как с единым целым надо использовать идентификатор массива без указания индекса в квадратных скобках. Доступ к каждому элементу массива осуществляется с помощью индекса, то есть порядкового номера элемента массива. Для обращения к элементу массива надо указать имя массива и порядковый номер элемента: $x[1]$, $y[5]$, $c[25]$, $A[8]$. Если указано только имя массива, то речь идет обо всем массиве (A , C и F). Обычно для обработки массива нужно организовать цикл и уже в нем работать с элементами.

В Турбо Паскале определена также операция присваивания для массивов, идентичных по структуре, то есть с одинаковыми типами индексов и компонентов. Например, если массивы C и D описаны как

```
var c,d: array [0..30] of real;
```

то можно записать оператор

```
c:=d;
```

Такая операция сразу всем элементам массива C присвоит значения элементов массива D , соответствующих им по номерам.

5.3. Ввод-вывод элементов массива

Паскаль не имеет специальных средств ввода-вывода всего массива, поэтому данную операцию следует организовывать поэлементно. При вводе массива необходимо последовательно вводить 1-й, 2-й, 3-й и т.д. элементы массива, аналогичным образом поступить и при выводе. Следовательно, для ввода-вывода необходимо организовать цикл, в котором практически все операции с массивами необходимо проводить поэлементно. Для обработки элементов массива удобно использовать цикл `for ... do`.

Блок-схемы алгоритмов ввода элементов массива изображены на рис. 5.1–5.2.

```
{ Ввод элементов массива X с помощью цикла while. }
var
  x: array [1..100] of real;
  i,n: integer;
begin
  writeln ('введите размер массива');
  readln(N);
  i:=1;
  while (i<=N) do
  begin
```

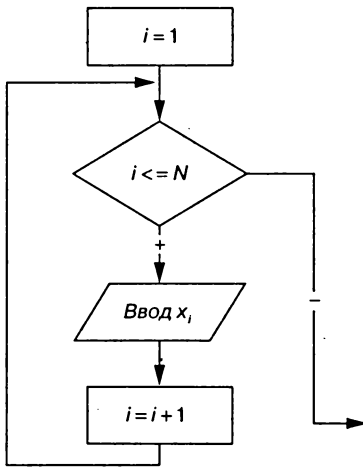


Рис. 5.1 ▼ Алгоритм ввода массива с использованием цикла с предусловием

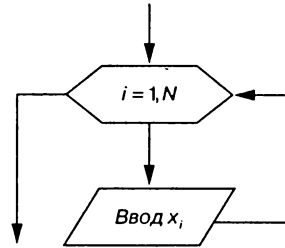


Рис. 5.2 ▼ Алгоритм ввода массива с использованием блока модификации

```

write('x(', i, ')= ');
readln(x[i]);
i:=i+1
end;
end.
{ Ввод элементов массива X с помощью цикла for. }
var
  x: array [1..100] of real;
  i, n: integer;
begin
  readln(N);
  for i:=1 to N do
    begin
      write('x(', i, ')= ');
      readln(x[i])
    end;
  end.
end.

```

Как видно, цикл `for ... do` удобно использовать для обработки всего массива, и в дальнейшем при выполнении таких операций мы будем применять именно его.

Вывод массива организуется аналогично вводу.

Предлагаем читателю рассмотреть несколько вариантов вывода массива и самостоятельно разобраться, чем они отличаются друг от друга и какой лучше.

```

{ Вариант 1. }
for i: = 1 to n do write (a[i]:6:2);
{ Вариант 2. }
for i: = 1 to n do write (a[i]:6:2, ' ');
{ Вариант 3. }
writeln ('массив A');

```

```

for i:=1 to n do writeln (a[i]:6:2);
{ Вариант 4. }
for i:=1 to n do write ('a[' ,i,']= ',a[i]:6:2, ' ')
{ Вариант 5. }
for i:=1 to n do writeln ('a[' ,i, '= ',a[i]:6:2);

```

Рассмотрим основные алгоритмы обработки массивов.

5.4. Алгоритм нахождения суммы элементов массива

Дан массив X , состоящий из n элементов. Найти сумму элементов этого массива. Блок-схема алгоритма расчета суммы приведена на рис. 5.3.

Соответствующий алгоритму фрагмент программы будет иметь вид:

```

s:=0;
for i:=1 to N do
s:=s+x[i];

```

5.5. Алгоритм нахождения произведения элементов массива

Дан массив X , состоящий из n элементов. Найти произведение элементов этого массива. Блок-схема алгоритма приведена на рис. 5.4.

Соответствующий фрагмент программы будет иметь вид:

```

p:=1;
for i:=1 to N do
p:=p*x[i];

```

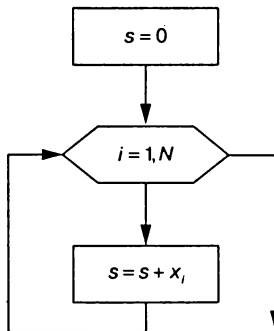


Рис. 5.3 ▼ Алгоритм расчета суммы элементов массива

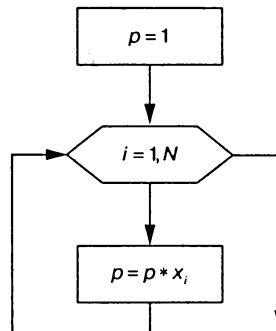


Рис. 5.4 ▼ Алгоритм расчета произведения элементов массива

5.6. Алгоритм поиска максимального элемента в массиве и его номера

Дан массив X , состоящий из n элементов. Найти максимальный элемент массива и номер, под которым он хранится в массиве. Блок-схема алгоритма приведена на рис. 5.5.

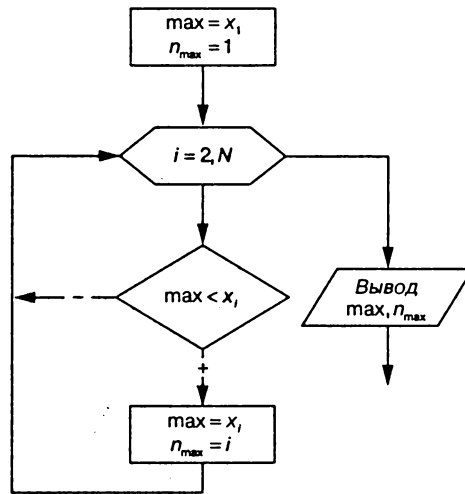


Рис. 5.5 ▼ Алгоритм поиска максимального элемента в массиве

Алгоритм решения задачи следующий. Пусть в переменной с именем max хранится значение максимального элемента массива, а в переменной с именем n_{max} – его номер. Предположим, что первый элемент массива является максимальным и запишем его в переменную max , а в n_{max} – его номер (то есть 1). Затем все элементы, начиная со второго, сравниваем в цикле с максимальным. Если текущий элемент массива оказывается больше максимального, то записываем его в переменную max , а в переменную n_{max} – текущее значение индекса i .

Соответствующий фрагмент программы будет иметь вид:

```

Max:=X[1];
Nmax:=1;
for i:=2 to N do
  if X[i]>Max then
    begin
      Max:=X[i];
      Nmax:=i;
    end;
writeln('Max=',Max:1:3, 'Nmax=', Nmax);

```



Алгоритм поиска минимального элемента в массиве будет отличаться от приведенного выше лишь тем, что в условном блоке `if`, соответственно, в конструкции `if` текста программы знак поменяется `<` на `>`.

5.7. Алгоритм упорядочивания элементов в массиве

Задан массив Y из n целых чисел. Расположить элементы массива в порядке возрастания их значений. Данную проблему можно решить одним из нижеприведенных способов.

5.7.1. Первый способ

Сравним первый элемент массива со вторым, если первый окажется больше второго, то поменяем их местами. Те же действия выполним для второго и третьего, третьего и четвертого, i -го и $(i+1)$ -го, $(n-1)$ -го и n -го элементов. В результате этих действий самый большой элемент станет на последнее (n -е) место. Теперь повторим данный алгоритм сначала, но последний (n -й) элемент, рассматривать не будем, так как он уже занял свое место. После проведения данной операции самый большой элемент оставшегося массива станет на $(n-1)$ -е место. Так повторяем до тех пор, пока не упорядочим весь массив.

В табл. 5.1 подробно расписан процесс упорядочивания элементов в массиве. Нетрудно заметить, что для преобразования массива, состоящего из n элементов, необходимо просмотреть его $n-1$ раз, каждый раз уменьшая диапазон просмотра на один элемент. Блок-схема описанного алгоритма приведена на рис. 5.6. Обратите внимание на то, что для перестановки элементов (блок 4) используется буферная переменная b , в которой временно хранится значение элемента, подлежащего замене.

Таблица 5.1 ▼ Процесс упорядочивания элементов в массиве по возрастанию

Номер элемента	1	2	3	4	5
Исходный массив	7	3	5	4	2
Первый просмотр	3	5	4	2	7
Второй просмотр	3	4	2	5	7
Третий просмотр	3	2	4	5	7
Четвертый просмотр	2	3	4	5	7

```
{ Упорядочивание элементов в массиве по возрастанию их значений. }
```

```
var
```

```
  i, n, j: integer;
```

```
  y: array [1..100] of word;
```

```
begin
```

```
  writeln ('введите размер массива');
```

```
  readln (n);
```

```
  for i:=1 to n do
```

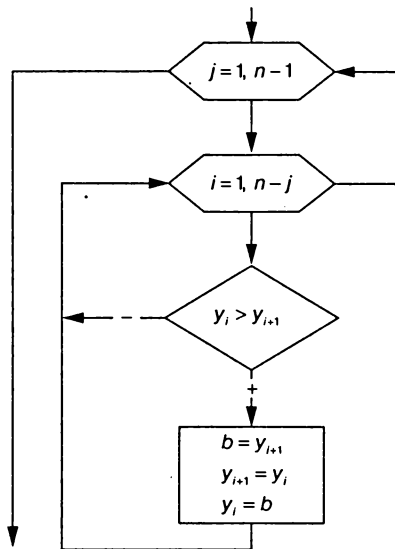


Рис. 5.6 ▼ Алгоритм упорядочивания элементов в массиве по возрастанию

```

begin
  write('y[' , i , ']=');
  readln (y[i]);
end;
writeln ('массив y');
for i:=1 to n do write (y [i], ' ');
writeln;
for j:=1 to n-1 do
  for i:=1 to n-j do
    if y[i] > y[i+1] then { Если текущий элемент больше следующего, то }
      begin { поменять их местами. }
        b:=y[i]; { Сохранить значение текущего элемента. }
        y[i]:=y[i+1]; { Заменить текущий элемент следующим. }
        y[i+1]:=b; { Заменить следующий элемент текущим. }
      end
    writeln('упорядоченный массив');
    for i:=1 to n do
      write (y[i], ' ');
    writeln;
  end.

```



Для перестановки элементов в массиве по убыванию их значений необходимо при сравнении элементов массива заменить знак $>$ на $<$.

5.7.2. Второй способ

Алгоритм приведен в виде блок-схемы на рис. 5.7. Найдем в массиве самый большой элемент (блоки 3–7) и поменяем его местами с последним элементом

(блок 8). Повторим алгоритм поиска максимального элемента, уменьшив количество просматриваемых элементов на единицу (блок 9), и поменяем его местами с предпоследним элементом (блоки 3–7). Описанную выше операцию поиска проводим до полного упорядочивания элементов в массиве. Так как в блоке 9 происходит изменение переменной n , то в начале алгоритма ее значение необходимо сохранить (блок 1).



Для упорядочивания массива по убыванию необходимо перемещать минимальный элемент.

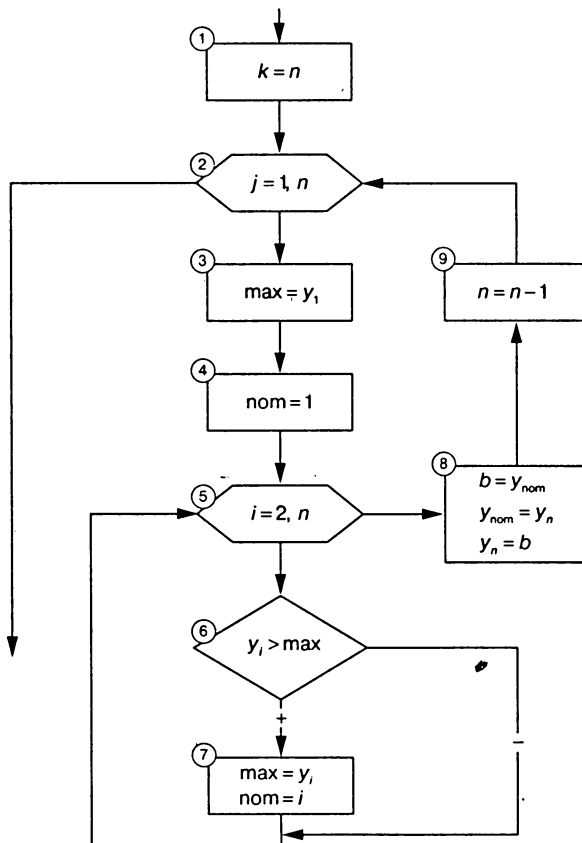


Рис. 5.7 ▼ Алгоритм упорядочивания элементов в массиве

5.8. Удаление элемента из массива

Необходимо удалить из массива X , состоящего из n элементов, m -й по номеру элемент. Для этого достаточно записать элемент $(m+1)$ на место элемента m ,

$(m+2)$ – на место $(m+1)$ и т.д., n – на место $(n-1)$ и при дальнейшей работе с этим массивом использовать $n-1$ элемент (рис. 5.8). Алгоритм удаления из массива X размером n элемента с номером m приведен на рис. 5.9.

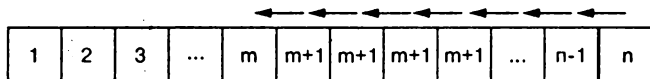


Рис. 5.8 ▼ Процесс удаления элемента из массива

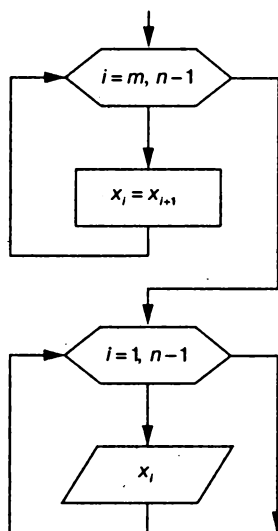


Рис. 5.9 ▼ Алгоритм удаления элемента из массива

5.9. Примеры программ

ПРИМЕР 5.1. Переменная x изменяется в интервале от x_a до x_b шагом dx . Сформировать массив y по формуле

$$y_i = \begin{cases} e^{\sin x} \cos x, & x < 0 \\ 0, & x = 0 \\ \lg x, & x > 0 \end{cases}$$

Вычислить сумму S элементов массива y и произведение p ненулевых элементов массива y , принадлежащих интервалу $(-3, 5; 4)$. Блок-схема представлена на рис. 5.10.

```
{ Пример 5.1. }
program mas_one;
type
```

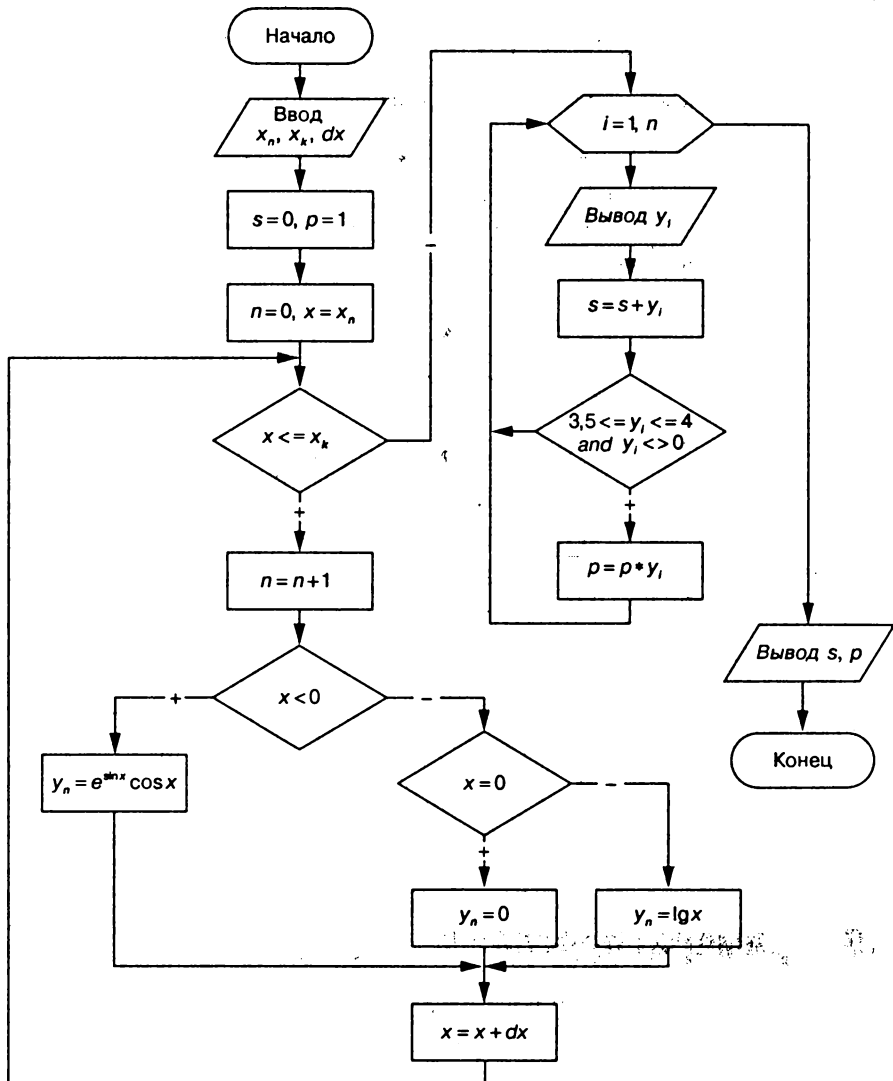


Рис. 5.10 ▽ Алгоритм примера 5.1

```

massiv=array [1..200] of real;
var
  y:massiv;
  x,xn,xk,dx:real;
  s,p:real;
  n,i:integer;
begin
  writeln('введите  xn,xk,dx');

```

```

readln (xn,xk,dx);
n:=0; x:=xn;
while x<=xk do
begin
  n:=n+1;
  if x<0 then y[n]:=exp(sin(x))*cos(x)
  else if x =0 then y[n]:=0
  else y[n]:=ln(x)/ln (10); { lg(x)=ln(x)/ln(10) }
  x:=x+dx;
end;
writeln ('сформирован массив Y:');
for i:=1 to n do
write(y[i]:1:2,' ');
s:=0;
p:=1;
for i:=1 to n do
begin
{ Накапливать сумму для всех элементов в массиве. }
  s:=s+y[i];
  { Накапливать произведение для ненулевых элементов массива, }
  { принадлежащих заданному интервалу. }
  if (y[i]<>0) and (y[i]>=-3.5) and (y[i]>=4) then
    p:=p*y[i];
end;
writeln ('сумма=',s:1:2);
writeln ('произведение=',p:1:2);
end.

```

ПРИМЕР 5.2. Задан массив y из n целых чисел. Сформировать массив z таким образом, чтобы в начале шли отрицательные элементы массива y , затем положительные и, наконец, нулевые. Блок-схема представлена на рис. 5.11.

```

{Пример 5.2.}
program mas_four;
var
  y,z:array [1..50] of integer;
  i,k,n: integer;
begin
  writeln ('введите n<=50');
  readln (n);
  for i:=1 to n do
  begin
    write('y[' ,i,']=');
    readln(y[i]);
  end;
  k:=0;
  for i:=1 to n do
  if y[i] < 0 then
  begin
    k:=k+1;
    z[k]:=y[i];
  end;
  for i:=1 to n do
  if y[i] > 0 then

```

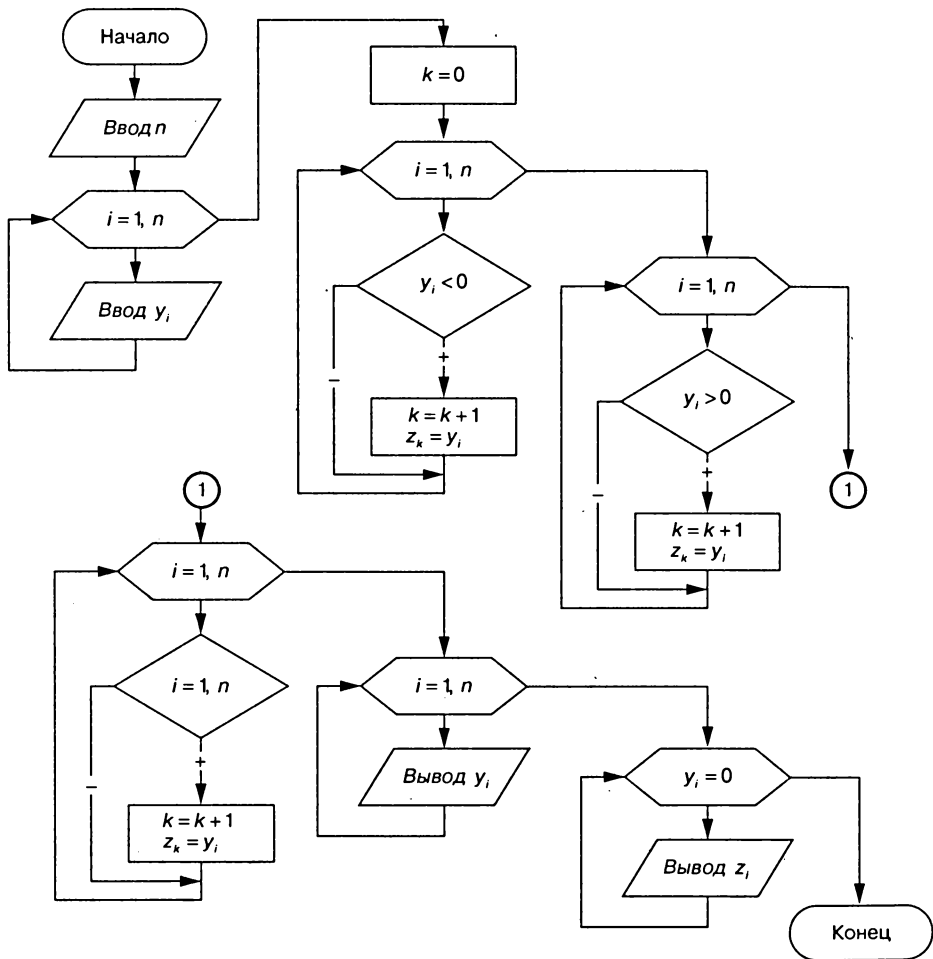


Рис. 5.11 ▼ Алгоритм примера 5.2

```

begin
    k:=k+1;
    z[k]:=y[i]
end;
for i:=1 to n do
    if y[i]=0 then
        begin
            k:=k+1;
            z[k]:= y[i];
        end;
    writeln ('Массив Y:');
    for i:=1 to n do

```

```

write(y[i], ' ');
writeln;
writeln ('Массив Z:');
for i:=1 to n do
  write (z[i], ' ');
  writeln;
end.

```

ПРИМЕР 5.3. Записать элементы массива x в обратном порядке. Вычислить сумму элементов массива x с четными индексами. Блок-схема представлена на рис. 5.12.

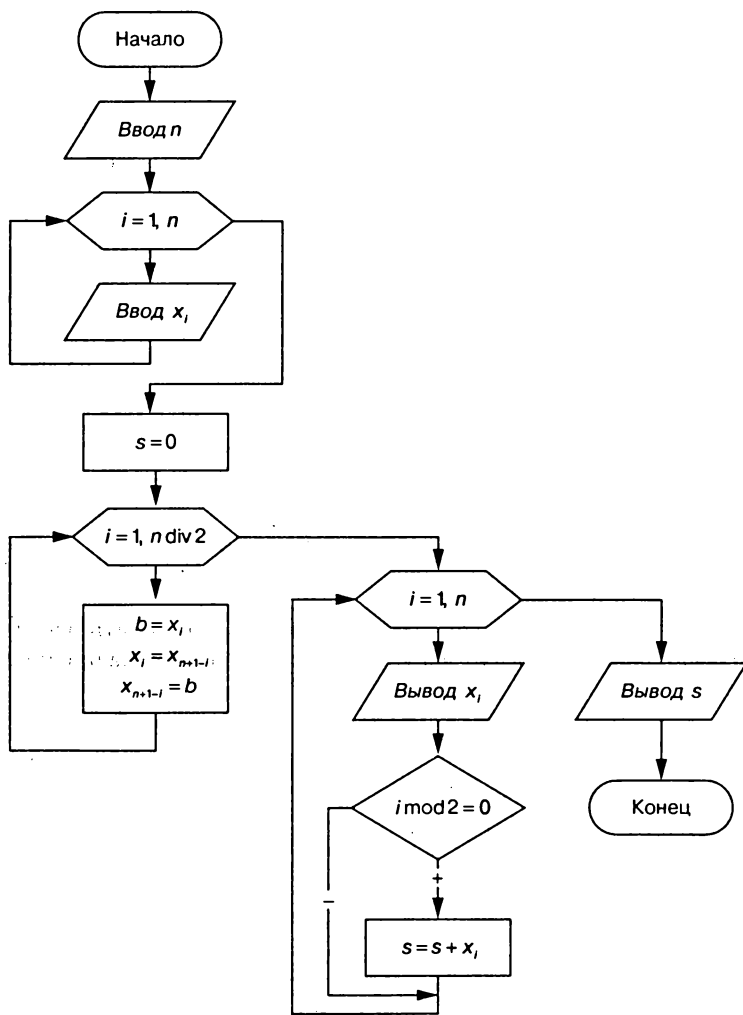


Рис. 5.12 ▼ Алгоритм примера 5.3

```
{Пример 5.3.}
program mas_five;
type
  massiv=array [1..100] of real;
var
  x:massiv;
  i,n:integer;
  b,s:real;
begin
  writeln ('введите размер массива');
  readln(n);
  for i:=1 to n do
    begin
      write ('x[' ,i,']=');
      readln(x[i]);
    end;
  s:=0;
  { Элементы массива меняются местами: 1-й с n-м, 2-й с (n-1) и т.д. }
  for i:=1 to n div 2 do
    begin
      b:=x[n+1-i];
      x[n+1-i]:=x[i];
      x[i]:=b;
    end;
  writeln('преобразованный массив');
  for i:=1 to n do
    begin
      write(x[i]:1:2, ' ');
      if i mod 2 = 0 then
        { Если номер элемента в массиве делится на 2 нацело, }
        s:=s+x[i] { то накапливать сумму. }
      end;
      writeln;
      writeln('s=',s:1:2);
    end.
end.
```

ПРИМЕР 5.4. Задан массив из n элементов. Сформировать массивы номеров положительных и отрицательных элементов. Блок-схема представлена на рис. 5.13.

```
{Пример 5.4.}
program mas_six;
type
  massiv=array [1..20] of real;
  massiv1=array[1..20] of integer;
var
  a,s:massiv1;
  y: massiv;
  i,k,l,n:integer;
begin
  write('n=');
  readln (n);
  for i:=1 to n do
    begin
```

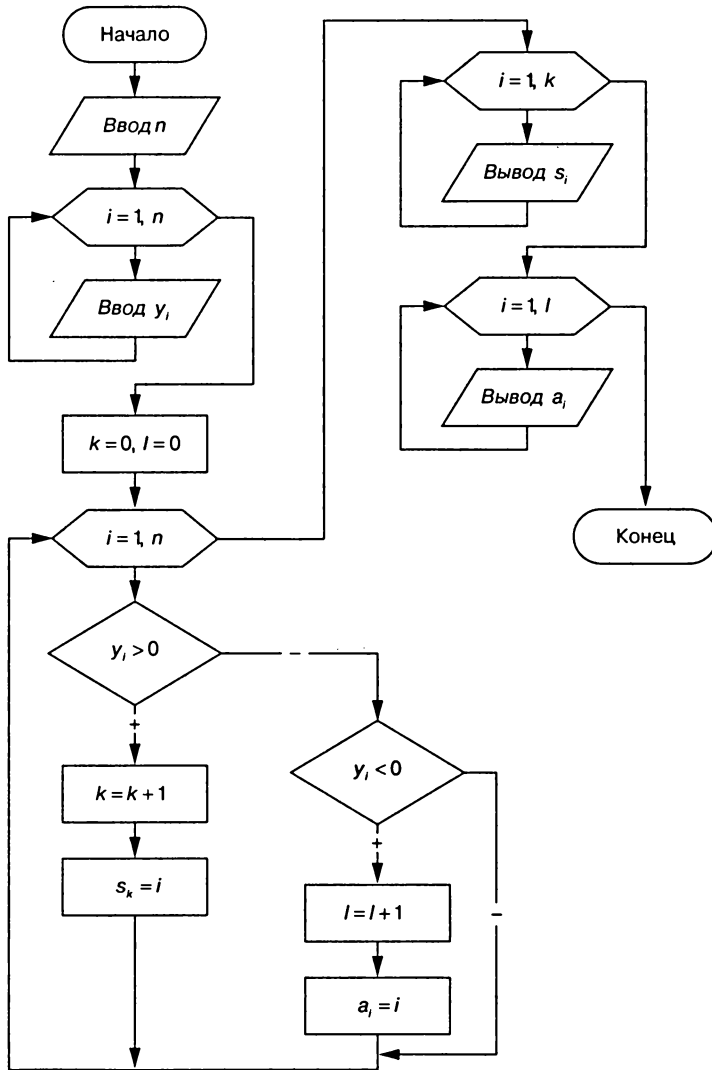


Рис. 5.13 ▼ Алгоритм примера 5.4

```

write('y[' , i, ']=');
readln (y[i]);
end;
k:=0; { Счетчик положительных элементов. }
l:=0; { Счетчик отрицательных элементов. }

```



```

for i:=1 to n do
begin
  if y[i] >0 then { Если элемент положительный, то }
  begin
    k:=k+1; { нарастить счетчик k на 1 и }
    s[k]:=i; { записать номер элемента в массив S. }
  end;
  if y[i] <0 then { Если элемент отрицательный, то }
  begin
    l:=l+1; { нарастить счетчик l на 1 и }
    a[l]:=i; { записать номер элемента в массив A. }
  end;
end;
writeln('массив индексов положительных элементов');
for i:=1 to n do
write (s[i], ' ');
writeln;
  writeln ('массив индексов отрицательных элементов');
  for i:=1 to n do write(a[i], ' ');
  writeln
end.

```

ПРИМЕР 5.5. Удалить из массива X , состоящего из n элементов, первые четыре нулевых элемента. Блок-схема представлена на рис. 5.14.

```

const n=20;
var X:array [1..n] of byte;
    k,i,j:integer;
begin
  for i:=1 to n do
    readln(X[i]);
  k:=0; { Количество нулевых элементов. }
  j:=1; { Номер элемента в массиве X. }
  while j<=n do { Пока не конец массива. }
  begin
    if x[j]=0 then { Если нашли нулевой элемент, то }
    begin
      k:=k+1; { посчитать его номер. }
      if k>4 then break { Если превышает 4, то выйти из цикла. }
      else { Иначе, удалить элемент из массива }
      for i:=j to n-k do { начиная с j-й позиции. }
      X[i]:=X[i+1];
    end
    { Возвращаемся на начало и просматриваем массив с j-й позиции, }
    { так как в ней оказался новый элемент. }
    else j:=j+1; {Если элемент ненулевой переходим к следующему.}
  end;
  {Вывод на печать измененного массива.}
  for i:=1 to n-k do
    write(X[i], ' ');
  end.

```

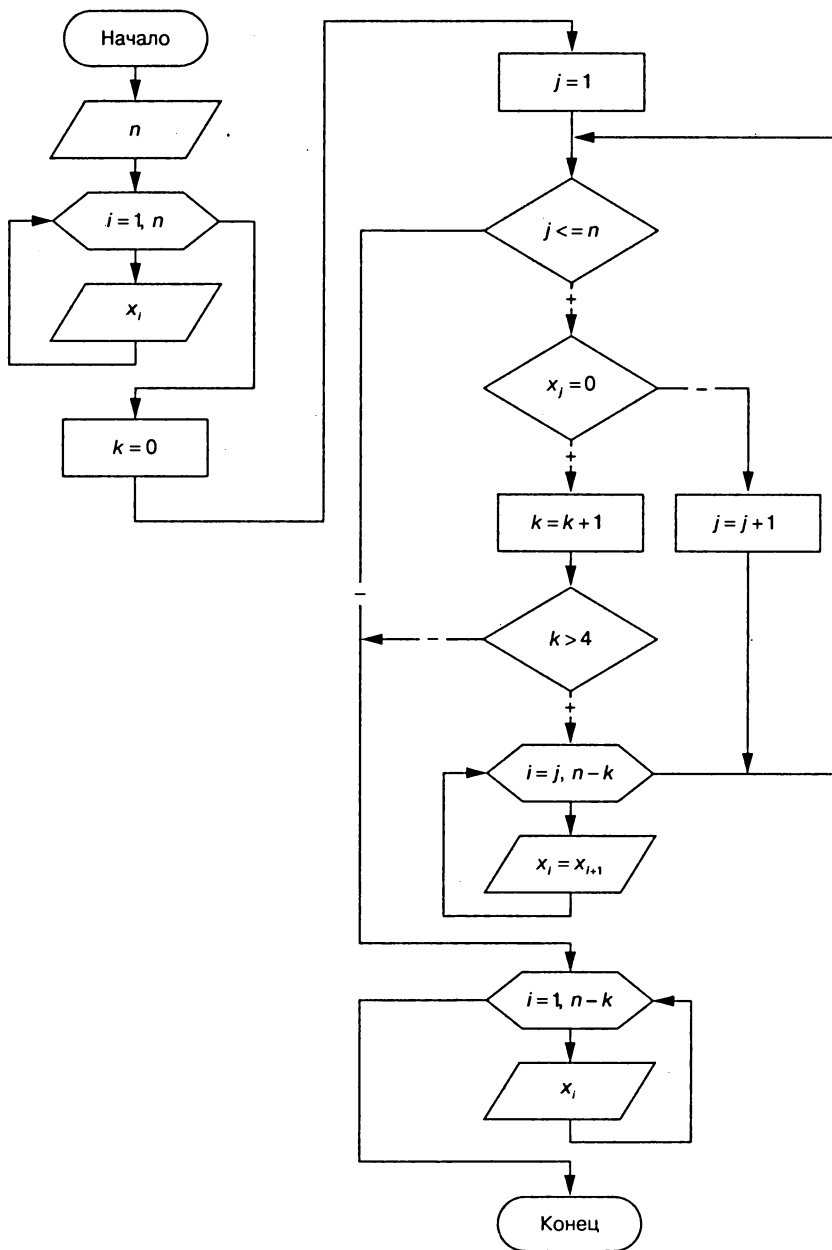


Рис. 5.14 ▼ Алгоритм примера 5.5

5.10. Упражнения по теме «Массивы»

1. Записать положительные элементы массива $X = (x_1, x_2, \dots, x_n)$ подряд в массив $Y = (y_1, y_2, \dots, y_k)$. Определить k – количество положительных элементов. Вычислить сумму элементов массива X и произведение элементов массива Y .
2. Сформировать массив $B = (b_1, b_2, \dots, b_k)$, записав в него элементы массива $A = (a_1, a_2, \dots, a_n)$ с четными индексами. Вычислить среднее арифметическое элементов массива B и удалить из него пятый элемент.
3. Дан массив $X = (x_1, \dots, x_n)$. Переписать пять первых положительных элементов массива подряд в массив $Y = (y_1, y_2, \dots, y_5)$. Найти максимальный элемент массива X .
4. Записать элементы массива $X = (x_1, x_2, \dots, x_n)$, удовлетворяющие условию $x_i \in [1, 2]$, подряд в массив $Y = (y_1, y_2, \dots, y_k)$. Определить минимальный элемент массива X .
5. Переписать элементы массива целых чисел $X = (x_1, x_2, \dots, x_n)$ в обратном порядке в массив $Y = (y_1, y_2, \dots, y_n)$. Вычислить количество четных, нечетных и нулевых элементов массива Y .
6. Определить максимальный и минимальный элементы среди положительных нечетных элементов целочисленного массива $X = (x_1, x_2, \dots, x_n)$. Удалить из массива все нулевые элементы.
7. Переписать элементы массива $X = (x_1, x_2, \dots, x_{12})$ в массив $Y = (y_1, y_2, \dots, y_{12})$, сдвинув элементы массива X вправо на три позиции. При этом три элемента с конца массива X перемещаются в начало: $(y_1, y_2, \dots, y_{12}) = (x_{10}, x_{11}, x_{12}, x_1, x_2, \dots, x_9)$. Определить номера максимального и минимального элементов в массивах X и Y .
8. Записать элементы массива $X = (x_1, x_2, \dots, x_{15})$ в массив $Y = (y_1, y_2, \dots, y_{15})$, сдвинув элементы массива X влево на четыре позиции. При этом четыре элемента, стоящие в начале массива X , перемещаются в конец: $(y_1, y_2, \dots, y_{15}) = (x_3, x_6, \dots, x_{15}, x_1, x_2, x_3, x_4)$. Поменять местами минимальный и максимальный элемент массива Y .
9. В массиве $X = (x_1, x_2, \dots, x_n)$ определить количество элементов меньших среднего арифметического значения. Не упорядочивая массив удалить из него элементы, расположенные между максимальным и минимальным.
10. Вычислить среднее арифметическое элементов массива $X = (x_1, x_2, \dots, x_n)$, расположенных между его минимальным и максимальным значениями. Если минимальный элемент размещается в массиве раньше максимального, то упорядочить массив на данном промежутке по возрастанию его элементов (возможна и обратная ситуация).

Глава

Обработка матриц в Турбо Паскале

Матрица – это двумерный массив, каждый элемент которого имеет два индекса: номер строки – i ; номер столбца – j . Поэтому для работы с элементами матрицы необходимо использовать два цикла. Если значениями параметра первого цикла будут номера строк матрицы, то значениями параметра второго – столбцы (или наоборот). Обработка матрицы заключается в том, что вначале поочередно рассматриваются элементы первой строки (столбца), затем второй и т.д. до последней. Рассмотрим основные операции, выполняемые над матрицами при решении задач.

6.1. Ввод-вывод матриц

Матрицы, как и массивы, нужно вводить (выводить) поэлементно. Блок-схема ввода элементов матрицы изображена на рис. 6.1.

Вывод можно осуществлять по строкам или по столбцам, но лучше, если элементы располагаются построчно, например:

```
6  -9  7  13
5   8  3   8
3   7 88 33
55 77 88 37
```

Алгоритм построчного вывода элементов матрицы приведен на рис. 6.2.

```
var
  a: array [1..20,1..20] of real;
  i,n,m: integer;
begin
  readln(N,M);
```

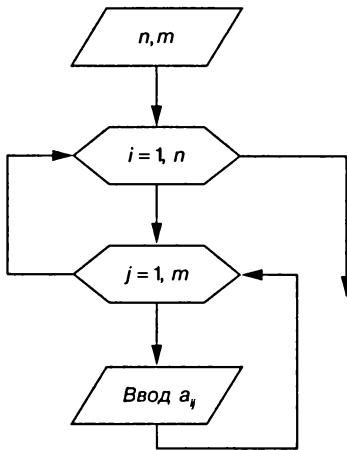


Рис. 6.1 ▼ Ввод элементов матрицы

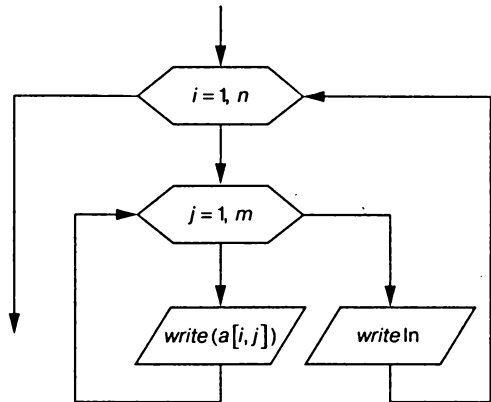


Рис. 6.2 ▼ Блок-схема вывода элементов матрицы по строкам

```

{ Ввод элементов матрицы. }
for i:=1 to n do
  for j:=1 to m do
    begin
      write('A(', i, ', ', j, ')=');
      readln(A[i, j])
    end;
  { Вывод элементов матрицы. }
  writeln ('матрица A ');
  for i:=1 to n do
    begin
      for j:=1 to m do
        write(a[i, j]:8:3, ' '); { Печатается строка. }
      writeln      { Переход на новую строку. }
    end;
  end.

```

Рассмотрим несколько задач обработки матриц, в которых приведены блок-схемы, реализующие только основную часть алгоритма (без ввода-вывода), а программы даны полностью.

6.2. Алгоритмы и программы работы с матрицами

ПРИМЕР 6.1. Найти сумму элементов матрицы, лежащих выше главной диагонали. Для решения этой задачи необходимо знать, что элементы матрицы обладают следующими свойствами:

- если номер строки элемента совпадает с номером столбца ($i = j$), это означает что элемент лежит на главной диагонали матрицы;

- если номер строки превышает номер столбца ($i > j$), то элемент находится ниже главной диагонали;
- если номер столбца больше номера строки ($i < j$), то элемент находится выше главной диагонали;
- элемент лежит на побочной диагонали, если его индексы удовлетворяют равенству $i + j - 1 = n$.

Блок-схемы алгоритмов решения данной задачи приведены на рис. 6.3–6.4. В программе, приведенной ниже, реализован алгоритм, изображенный на рис. 6.4.

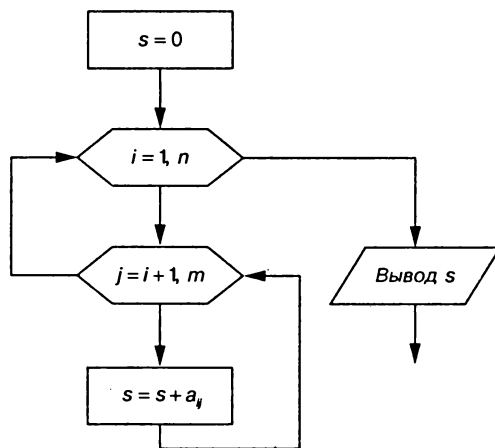


Рис. 6.3 ▼ Блок-схема примера 6.1 (алгоритм 1)

```

program one;
var
  a:array [1..15,1..10] of real;
  i,j,n,m: integer;
  s: real;
begin
  writeln('введите размеры матрицы');
  writeln('n - количество строк, m - количество столбцов');
  readln (n,m);
  for i:=1 to n do
    for j:=1 to m do
      begin
        write('a[' ,i ,',' ,j ,']=');
        readln(a[i,j]);
      end;
    s:=0;
    for i:=1 to n do
      for j:=1 to n do
        if j>i then { Если элемент лежит выше главной диагонали, то }
          s:=s+a[i,j];{ наращиваем сумму. }
        writeln('матрица A');
      end;
    end;
  end;
end;

```

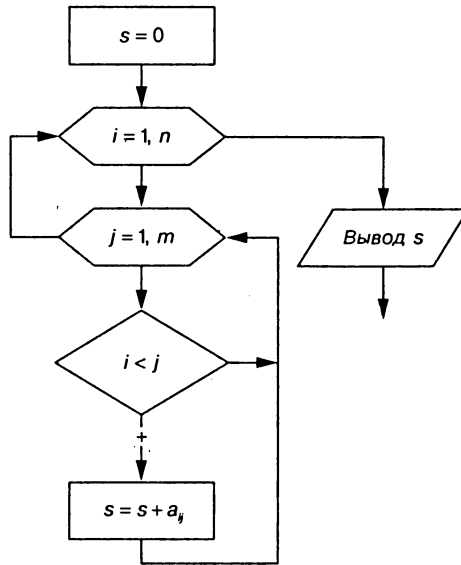


Рис. 6.4 ▼ Блок-схема примера 6.1 (алгоритм 2)

```

for i:=1 to n do
  begin
    for j:=1 to m do
      { Здесь формат важен, особенно общая ширина поля! }
      write(a[i,j]:8:3, ' ');
    writeln
  end;
  writeln('сумма элементов матрицы', s:8:3);
end.

```

Попробуйте самостоятельно написать программу по блок-схеме, изображенной на рис. 6.3.

ПРИМЕР 6.2. Преобразовать исходную матрицу так, чтобы первый элемент каждой строки был заменен средним арифметическим элементов этой строки. Для решения данной задачи необходимо найти в каждой строке сумму элементов, которую разделить на их количество. Полученный результат записать в первый элемент соответствующей строки. Блок-схема алгоритма решения приведена на рис. 6.5.

```

program two;
type
  matrica=array[1..15,1..15] of real;
var
  a:matrica;
  i,j,n,m: integer;
  s: real;
begin
  write('Введите n и m:');

```

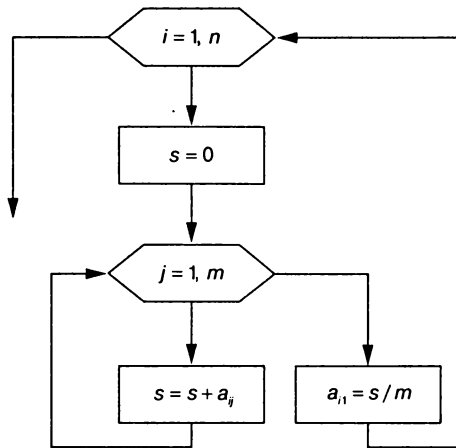


Рис. 6.5 ▼ Блок-схема алгоритма примера 6.2

```

readln(n,m);
for i:=1 to n do
  for j:=1 to m do
    begin
      write ('a[' ,i ,',' ,j ,'] = ');
      readln (a[i,j]);
    end;
  writeln ('Исходная матрица:');
  for i:=1 to n do
    begin
      for j:=1 to m do
        write (a[i,j]:8:3, ' ');
      writeln
    end;
  for i:=1 to n do
    begin
      s:=0;
      for j:=1 to m do
        s:=s+a[i,j]; { Вычисление суммы элементов строки. }
      a[i,1]:=s/m; { Запись среднего в первый элемент строки. }
    end;
  writeln ('Преобразованная матрица:');
  for i:=1 to n do
    begin
      for j:=1 to m do
        write (a[i,j]:8:3, ' ');
      writeln
    end;
  end.

```

ПРИМЕР 6.3. Задана матрица $A(n, m)$. Сформировать вектор $P(m)$, в который записать номера строк максимальных элементов каждого столбца. Алгоритм решения этой задачи следующий: для каждого столбца матрицы находим

максимальный элемент и его номер, номер максимального элемента j -го столбца матрицы записываем в j -й элемент массива P . Блок-схема алгоритма приведена на рис. 6.6.

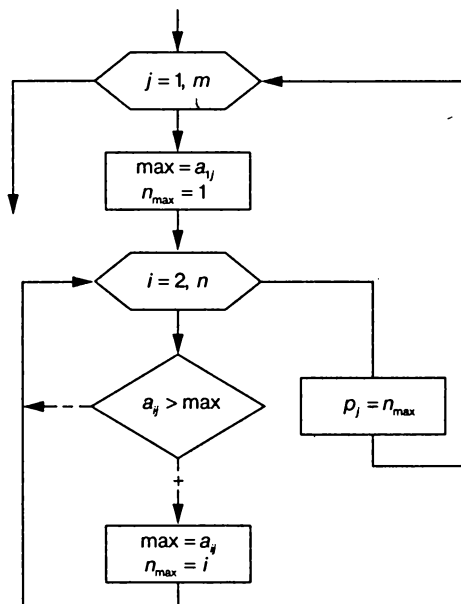


Рис. 6.6 ▼ Блок-схема алгоритма примера 6.3

```

program three;
type
  massiv = array [1..15,1..15] of real;
var
  i,j,n,nmax: byte;
  max: real;
  a: massiv;
  p: array [1..15] of byte;
begin
  write('n=');
  readln(n);
  write('m=');
  readln(m);
  for i:=1 to n do
    for j:=1 to m do
      begin
        write('a[' ,i ,', ' ,j ,']=');
        readln(a[i,j]);
      end;
  for j:=1 to m do { Цикл по столбцам. }
  begin
    max:= a[1,j];

```

```

nmax:=1;
for i:=2 to n do { Цикл по строкам. }
  if a[i,j] > max then
    begin
      max:=a[i,j];
      nmax:=i;
    end;
  p[j]:=nmax; { Запись номера максимального элемента в массив. }
end;
writeln('матрица A');
for i:=1 to n do
  begin
    for j:=1 to m do
      write(a[i,j]:8:3, ' ');
    writeln
  end;
writeln('вектор P');
for i:=1 to m do
  write (p[i]:2, ' ');
writeln;
end.

```

ПРИМЕР 6.4. Написать программу умножения двух матриц $A(n, m)$ и $B(m, l)$. Например, необходимо перемножить две матрицы

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix}.$$

Воспользовавшись правилом «строка на столбец», получим матрицу:

$$\begin{pmatrix} a_{11} \cdot b_{11} + a_{12} \cdot b_{21} + a_{13} \cdot b_{31} & a_{11} \cdot b_{12} + a_{12} \cdot b_{22} + a_{13} \cdot b_{32} \\ a_{21} \cdot b_{11} + a_{22} \cdot b_{21} + a_{23} \cdot b_{31} & a_{21} \cdot b_{12} + a_{22} \cdot b_{22} + a_{23} \cdot b_{32} \\ a_{31} \cdot b_{11} + a_{32} \cdot b_{21} + a_{33} \cdot b_{31} & a_{31} \cdot b_{12} + a_{32} \cdot b_{22} + a_{33} \cdot b_{32} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{pmatrix}.$$

В общем виде формула для нахождения элемента C_{ij} матрицы имеет вид:

$$C_{ij} = \sum_{k=1}^m A_{ik} B_{kj},$$

где $i = 1, N$ и $j = 1, L$. Обратите внимание, что проводить операцию умножения можно только в том случае, если количество строк левой матрицы совпадает с количеством столбцов правой. Кроме того, $A \times B \neq B \times A$. Блок-схема, изображенная на рис. 6.7, реализует расчет каждого элемента матрицы C в виде суммы по вышеприведенной формуле.

```

program four;
typ
  matrica=array [1..15,1..15] of real;
var
  a,b,c:matrica;
  i,j,m,n,l,k:byte;

```

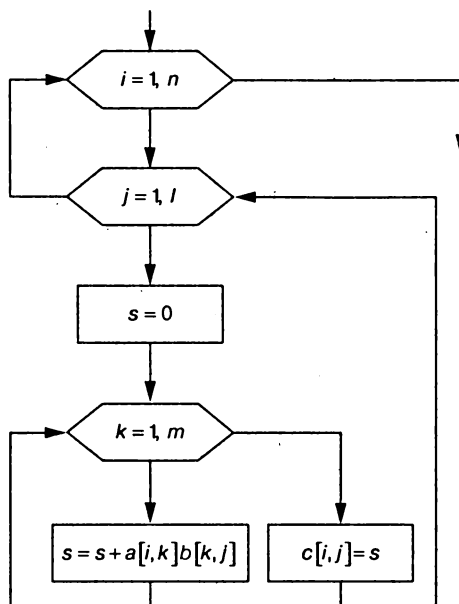


Рис. 6.7 ▼ Алгоритм умножения двух матриц

```

s:real;
begin
  writeln('введите n, m и l');
  readln(n, m, l);
  for i:=1 to n do
    for j:=1 to m do
      begin
        write('a[' , i , ' , ' , j , ' ] = ');
        readln(a[i, j]);
      end;
    for i:=1 to m do
      for j:=1 to l do
        begin
          write('b[' , i , ' , ' , j , ' ] = ');
          readln(b[i, j]);
        end;
      for i:=1 to n do
        for j:=1 to l do
          begin
            { В переменной S будет храниться результат скалярного }
            { произведения i-й строки на j-й столбец. }
            s:=0;
            for k:=1 to m do
              s:=s+a[i, k]*b[k, j];
            c[i, j]:=s;
          end;
        end;
      end;
    end;
  end;
end;

```

```

writeln('матрица a');
for i:=1 to n do
begin
  for j:=1 to m do
    write(a[i,j]:7:3, ' ');
  writeln;
end;
writeln('матрица b');
for i:=1 to m do
begin
  for j:=1 to l do
    write(b[i,j]:7:3, ' ');
  writeln;
end;
writeln('матрица c=a*b');
for i:=1 to n do
begin
  for j:=1 to l do
    write(c[i,j]:7:3, ' ');
  writeln;
end;
end.

```

ПРИМЕР 6.5. Преобразовать матрицу $A(m, n)$ таким образом, чтобы каждый столбец был упорядочен по убыванию. Алгоритм решения этой задачи сводится к тому, что уже известный нам по предыдущей главе алгоритм упорядочивания элементов в массиве выполняется для каждого столбца матрицы. Блок-схема приведена на рис. 6.8.

```

program five;
type
  matrica=array [1..15,1..15] of real;
var
  a:matrica;
  i,j,k,m,n:byte;
  b:real;
begin
  write ('m=');
  readln(m);
  write ('n=');
  readln(n);
  for i:=1 to m do
    for j:=1 to n do
      begin
        write ('a[' ,i, ', ',j, ']=');
        readln(a[i,j]);
      end;
  writeln ('матрица a');
  for i:=1 to m do
    begin
      for j:=1 to n do
        write (a[i,j]:7:3, ' ');
      writeln
    end;
end;

```

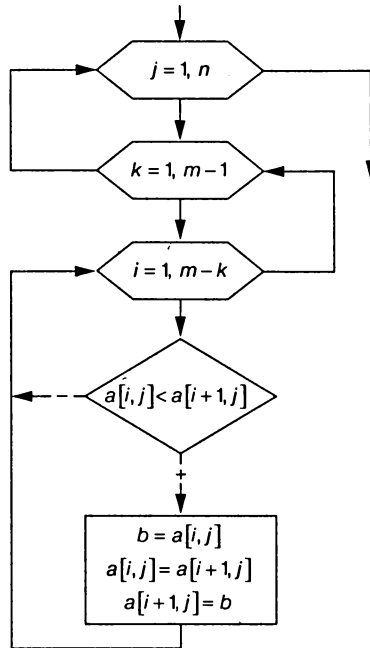


Рис 6.8 ▼ Блок-схема алгоритма примера 6.5

```

for j:=1 to n do      { j - номер столбца }
  for k:=1 to m-1 do  { k - номер просмотра }
    for i:=1 to m-k do { i - номер строки }
      if a[i,j] < a[i+1,j] then
        begin
          b:=a[i,j];
          a[i,j]:=a[i+1,j];
          a[i+1,j]:=b;
        end;
    writeln('преобразованная матрица a');
  for i:=1 to m do
    begin
      for j:=1 to n do
        write (a[i,j]:7:3, ' ');
      writeln;
    end;
  end.

```

ПРИМЕР 6.6. Преобразовать матрицу $A(m, n)$ так, чтобы строки с нечетными индексами были упорядочены по убыванию, с четными – по возрастанию.

```

program six;
var
  a:array [1..15,1..15] of real;
  j,i,k,m,n:byte;
  b:real;

```

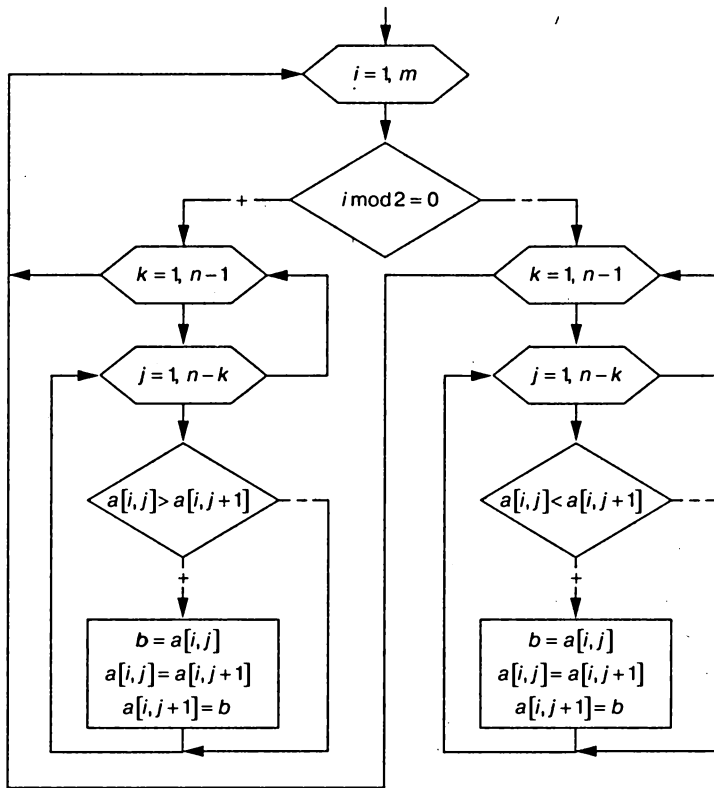


Рис. 6.9 ▼ Блок-схема алгоритма примера 6.6

```

begin
  writeln('введите m и n');
  readln(m,n);
  for i:=1 to m do
    for j:=1 to n do
      begin
        write('a[' ,i ,',' ,j ,']=');
        readln(a[i,j]);
      end;
    writeln ('матрица a');
    for i:=1 to m do
      begin
        for j:=1 to n do
          write(a[i,j]:7:3, ' ');
        writeln
      end;
    for i:=1 to m do

```

```

if (i mod 2)=0 then { Если номер строки четный, то }
begin { упорядочить ее элементы по возрастанию. }
  for k:=1 to n-1 do
    for j:=1 to n-k do
      if a[i,j] > a[i,j+1] then
        begin
          b:=a[i,j];
          a[i,j]:=a[i,j+1];
          a[i,j+1]:=b;
        end;
    end
end
else { Если номер строки нечетный, то }
  for k:=1 to n-1 do { упорядочить ее элементы по убыванию. }
    for j:=1 to n-k do
      if a[i] < a[i,j+1] then
        begin
          b:=a[i,j];
          a[i,j]:=a[i,j+1];
          a[i,j+1]:=b;
        end;
    end;
writeln('преобразованная матрица a');
for i:=1 to m do
begin
  for j:=1 to n do
    write (a[i,j]:7:3, ' ');
  writeln
end
end.

```

ПРИМЕР 6.7. Поменять местами n -й и l -й столбцы матрицы $A(k, m)$. Блок-схема приведена на рис. 6.10.

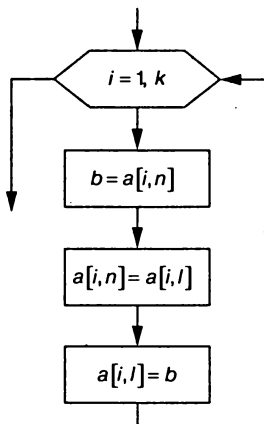


Рис. 6.10 ▼ Блок-схема алгоритма примера 6.7

```

program seven;
type
  matrica=array [1..15,1..15] of real;
var
  a:matrica;
  i,j,k,m,n,l:byte;
  b:real;
begin
  write ('k=');
  readln(k);
  write ('m=');
  readln(m);
  for i:=1 to k do
    for j:=1 to m do
      begin
        write ('a[' ,i ,',' ,j ,']=');
        readln (a[i,j])
      end;
  repeat
    write('n=');
    readln(n);
    write('l=');
    readln(l);
    { Ввод считается верным, если n и l меньше m и не равны друг другу. }
    until (n<=m) and (l<=m) and (n< >l);
    { Элементы столбца с номером l заменить элементами столбца с номером n. }
    for i:=1 to k do
      begin
        b:=a[i,n];
        a[i,n]:=a[i,l];
        a[i,l]:=b
      end;
    for i:=1 to k do
      begin
        for j:=1 to m do
          write(a[i,j]:7:3,' ');
        writeln;
      end;
  end.

```

6.3. Упражнения по теме «Работа с матрицами»

Разработать блок-схему и программу на Турбо Паскале:

1. Определить номера строки и столбца максимального элемента прямоугольной матрицы $A(n, m)$. Подсчитать количество нулевых элементов матрицы и напечатать их индексы.
2. Найти сумму элементов квадратной матрицы $X(n, m)$, находящихся по периметру этой матрицы и на ее диагоналях.

3. Сформировать вектор $D = (d_1, d_2, \dots, d_k)$, каждый элемент которого представляет собой среднее арифметическое значение элементов строк матрицы $C(k, m)$, и вектор $G = (g_1, g_2, \dots, g_m)$ – любой его компонент должен быть равен произведению значений элементов столбцов матрицы $C(k, p)$.
4. Задана матрица $A(n, m)$, в каждом столбце которой минимальный элемент необходимо заменить суммой положительных элементов этого же столбца.
5. Задана матрица $A(n, n)$. Определить максимальный элемент среди элементов матрицы, расположенных выше главной диагонали, и минимальный элемент среди тех, что находятся ниже главной диагонали. Отсортировать каждый столбец матрицы по возрастанию.
6. Заменить строку матрицы $P(n, m)$ с максимальной суммой элементов на первую строку поэлементно
7. Переместить максимальный элемент матрицы $F(k, p)$ в правый верхний угол, а минимальный элемент – в левый нижний.
8. Проверить, является ли матрица $A(n, n)$ диагональной (все элементы нули, кроме главной диагонали), единичной (все элементы нули, на главной диагонали только единицы) или нулевой (все элементы нули), и если нет, то сформировать из нее верхнетреугольную матрицу $B(n, n)$ (все элементы ниже главной диагонали нулевые) и нижнетреугольную матрицу $C(n, n)$ (все элементы выше главной диагонали нулевые).
9. Заданы матрицы $A(M, N)$ и $B(L, P)$. Найти, если возможно, матрицы $C = A \times B$ и $D = A + B$.
10. Выполнив действия над матрицами $A(n, n)$ и $B(n, n)$, вычислить матрицу $C(n, n)$ по формуле: $C = (A - B^T)(3A^T + B/2)$, где A^T и B^T – транспонированные матрицы, то есть те, что получают из исходных путем замены строк на столбцы.

7 Глава

Подпрограммы в языке Турбо Паскаль

В практике программирования часто складываются ситуации, когда одну и ту же группу операторов, реализующих определенную цель, требуется повторить без изменений в нескольких местах программы. Для избавления от столь нерациональной траты времени была предложена концепция подпрограммы.

Подпрограмма – именованная, логически законченная группа операторов языка, которую можно вызвать для выполнения любое количество раз из различных мест программы. В языке Паскаль существуют два вида подпрограмм: *процедуры* и *функции*.

Главное отличие процедур от функций заключается в том, что результатом последних является одно единственное значение. Все процедуры и функции языка Паскаль можно разделить на два класса:

- стандартные процедуры и функции языка Паскаль;
- процедуры и функции, определенные пользователем.

О стандартных функциях языка Паскаль говорилось в разделе 2.4. Рассмотрим некоторые стандартные арифметические процедуры.

$\text{Dec}(X, n)$ уменьшает значение целочисленной переменной X на n^1 . Если n отсутствует, то в процессе значение X уменьшается на 1. $\text{Inc}(X, n)$ увеличивает значение целочисленной переменной X на n .

7.1. Процедуры в языке Турбо Паскаль

Каждая новая *процедура* или *функция* должна быть предварительно описана в разделе описания процедур и функций. Для использования процедуры необходимо написать оператор вызова.

¹ Параметр n не является обязательным.

Описание процедуры состоит из заголовка процедуры и ее тела. Заголовок процедуры включает служебное слово `procedure`, имя процедуры и заключенный в круглые скобки список формальных параметров с указанием их типов (именно типов, а не описаний!):

```
procedure < имя > (<список формальных параметров>);
```

Например:

```
procedure a17 (a:real; b,c:real; var x1,x2:real; var k:integer);
```

Формальные параметры отделяются точкой с запятой (список однотипных параметров может быть перечислен через запятую). После заголовка идут разделы описаний (констант, типов, переменных, процедур и функций, используемых в процедуре) и операторы языка Паскаль, реализующие алгоритм процедуры.

Например:

```
{ Процедура f вычисляет значения факториала числа r }
{ и возвращает результат в переменной r1. }
{Для хранения значений факториалов (r1) использован тип longint. }
procedure f(r:integer; var r1:longint);
var
  i:integer;
begin
  { Если значение r отрицательно, то r1=0 и процедура завершается. }
  if r<0 then
    begin
      r1:=0;
      exit;
    end;
  { Ниже реализован алгоритм вычисления  $r!=1\cdot2\cdot3\cdot\ldots\cdot n$ . }
  r1:=1;
  for i:=2 to r do
    r1:=r1*i;
end.
```

Формальные параметры *нельзя* описывать в разделе описаний процедуры.


Для обращения к процедуре необходимо использовать оператор вызова процедуры. Он имеет следующий вид

```
<имя процедуры> (<список_фактических_параметров>);
```

Например:

```
f (n,fn);
```

Фактические параметры в списке отделяются друг от друга запятой. Механизм применения формальных – фактических параметров обеспечивает замену формальных параметров фактическими, что позволяет выполнять процедуру с различными данными. Между фактическими параметрами в операторе вызова процедуры и формальными параметрами в заголовке процедуры устанавливается взаимно однозначное соответствие.

 *Количество, типы и порядок следования формальных и фактических параметров должны совпадать.*

Пример:

```
program abc;
var
  r,r1,r2:real;
  procedure f1 (x:real; var y:real; var c:real);
begin
  y:=sin(x)/cos(x);
  c:=ln(x)/ln(10);
end;
begin
  read(r);
  f1(r,r1,r2);
  write (r1,r2);
end.
```


7.2. Формальные и фактические параметры

Можно выделить два основных класса формальных параметров:

- *параметры-значения;*
- *параметры-переменные.*

Параметры-значения используются в качестве входных данных подпрограммы. При обращении к подпрограмме фактические параметры передают свое значение формальным и больше не изменяются.

Параметры-переменные могут использоваться как в качестве входных данных, так и в качестве выходных. В заголовке процедуры перед параметрами-переменными необходимо указывать служебное слово `var`. При обращении к подпрограмме фактические параметры замещают формальные¹. В результате выполнения подпрограммы изменяются фактические параметры.

 *В качестве входных данных в подпрограмме следует использовать параметры-значения, в качестве выходных – параметры-переменные. Данные, которые являются и входными и выходными, следует описывать как параметры-переменные.*

ПРИМЕР 7.1. Написать программу решения группы квадратных уравнений $p_i x^2 + q_i x + r_i = 0$, где p, q, r – массивы вещественных чисел, состоящие из k элементов. Решение одного уравнения оформить в виде процедуры.

```
program urav;
{ Здесь начинается раздел описаний программы urav. }
var
  p,q,r:array [1..50] of real;
  x,y:real;
  L,i,k:integer;
```

¹ Это упрощенное объяснение механизма передачи данных между формальными и фактическими параметрами. Реально все несколько сложнее, происходит передача адреса параметра в процедуру.

```

{ Процедура korni решает квадратное уравнение. }
{ Коэффициенты уравнения a,b,c - входные параметры-значения. }
{ В качестве выходных выступают параметры-переменные x1, x2 и pr. }
{ В переменных x1 и x2 возвращаются корни квадратного уравнения. }
{ Переменная pr возвращает 1, если есть действительные корни уравнения. }
{ Значение 0 возвращается, если корней нет и дискриминант отрицателен. }
procedure korni(a,b,c:real;var x1,x2:real;var pr:integer);
var
  d:real;
begin
  d:=b*b-4*a*c;
  if d>=0 then
    begin
      pr:=1;
      x1:=(-b+sqrt(d))/2/a;
      x2:=(-b-sqrt(d))/2/a;
    end
  else pr:=0
end;
{ Начало основной программы. }
begin
  writeln ('введите количество уравнений');
  read (k);
  { Цикл предназначен для ввода коэффициентов уравнения, }
  { вызова процедуры решения квадратного уравнения korni }
  { и, если корни существуют, вывода их на экран. }
  for i:=1 to k do
    begin
      writeln ('введите коэффициенты',i,'-го уравнения');
      read(p[i],q[i],r[i]);
      {Обращение к процедуре korni.}
      korni(p[i],q[i],r[i],x,y,L);
      {Проверка существуют ли корни (L=1).}
      if L=1 then
        writeln('корни',i,'-го','уравнения',' ',x:8:5,' ',y:8:5)
      else
        writeln ('в ',i,'-м уравнении корней нет');
    end;
end.

```

ПРИМЕР 7.2. Найти максимальные элементы массивов a , b , c и их номера. Подпрограмму поиска оформить в виде процедуры.

```

program abc;
{ Тип данных vector, который будет определять тип формального параметра }
type
  vector=array [1..20] of real;
var
  a,b,c:vector;
  i,k:integer;
  max:real;
  nmax:integer;
  { Процедура max1 предназначена для поиска }
  { максимального элемента массива и его номера. }

```

```

{ В процедуре два входных параметра-значения: массив x и его размер L. }
{ При определении типа массива используется тип vector, }
{ выходные параметры-переменные: максимальный элемент max и его номер n. }
procedure max1(x:vector; L:integer; var rmax:real; var n:integer);
var
    j:integer;
begin
    rmax:=x[1]; n:=1;
    for j:=2 to L do
        if x[j]>rmax then
            begin
                rmax:=x[j]; n:=j;
            end
    end;
end;
begin
    writeln ('введите размер массивов'); read (k);
    for i:=1 to k do
        begin
            write ('a[' ,i ,']='); read (a[i]);
            end;
    { Обращение к процедуре для поиска максимального элемента }
    { и его номера в массиве a. }
    max1(a, k, max, nmax);
    writeln('max=',max:1:4,' nmax=',nmax);
    for i:=1 to k do
        begin
            write ('b[' ,i ,']='); read (b[i]);
            end;
    { Обращение к процедуре для поиска максимального элемента }
    { и его номера в массиве b. }
    max1(b, k, max, nmax);
    writeln (' max=',max:1:4,' nmax=',nmax);
    for i:=1 to k do
        begin
            write ('c[' ,i ,']='); read(c[i]);
            end;
    { Обращение к процедуре для поиска максимального элемента }
    { и его номера в массиве c. }
    max1(c, k, max, nmax);
    writeln (' max=',max:1:4,' nmax=',nmax);
end.

```

7.3. Функции в языке Турбо Паскаль

Описание функции состоит из заголовка функции и ее тела. Заголовок содержит служебное слово *function*, имя функции, список формальных параметров с указанием их типа и типа возвращаемого результата:

```
function <имя> (<список_формальных_ параметров>):<тип>.
```

Здесь <тип> – тип возвращаемого функцией значения. Функции могут возвращать *скалярные* значения целого, вещественного, логического, символьного или ссылочного типа.

Примеры описания функций:

```
function tan (x:real):real;  
function max(x,y:real):real;  
function al(x:real; y:real):real;
```

Обращение к функции осуществляется по имени с указанием списка фактических параметров. Количество, типы и порядок следования формальных и фактических параметров должны совпадать:

<имя_функции> (<список_фактических_параметров>);

В теле функции всегда должен быть *один* оператор, присваивающий значение имени функции.

Рассмотрим пример использования функции.

```
program abc;  
var  
  x,y:real;  
{ Определяем функцию tan, которая возвращает значения тангенса (tg). }  
function tan (c:real):real;  
begin  
  tan:=sin(c)/cos(c);  
end;  
begin  
  read (x,y);  
  { Вычисляются значения tg(x) и tg(y) путем обращения к функции tan. }  
  writeln ('tg(',x:1:3,')=',tan (x):1:4);  
  writeln ('tg(',y:1:3,')=',tan (y):1:4);  
end.
```

При использовании процедур и функций переменные объявляются несколько раз в основной программе и в подпрограммах.

Переменные и типы, указанные в основной программе до определения процедур и функций, называются *глобальными*, они доступны всем функциям и процедурам. Переменные, определенные в какой-либо подпрограмме или основной программе после раздела описаний процедур и функций, называются *локальными*.

Для правильного определения области действия идентификаторов (переменных) необходимо придерживаться следующих правил:

- каждая переменная должна быть описана перед тем, как она будет использована;
- областью действия переменной является та подпрограмма, в которой она описана;
- все переменные в подпрограммах должны быть уникальными;
- одна и та же переменная может быть по-разному определена в каждой из подпрограмм;
- если имя подпрограммы совпадает с названием стандартной подпрограммы, то последняя игнорируется, а выполняется подпрограмма пользователя;

- если внутри какой-либо процедуры встречается переменная с таким же именем, что и глобальная переменная, то внутри этой процедуры будет действовать *локальное* описание;
- каждая подпрограмма может изменить значение глобальной переменной.

7.4. Особенности работы с подпрограммами в Турбо Паскале версии 7.0

В Турбо Паскале версии 7.0 появились такие способы передачи данных в подпрограмму, как открытые массивы и параметры-константы.

7.4.1. Открытые массивы

В версии Турбо Паскаль 7.0 в качестве формальных параметров процедур могут использоваться *открытые массивы*, которые описываются с помощью служебного слова `array` и названия типа [9–10]. Тип элементов массива должен быть скалярным.

ПРИМЕР 7.3. Вычислить сумму и произведение элементов массивов `b`, `c`, `d`.

```
{ Процедура вычисления суммы и произведения массива. }
{ В процедуре три параметра. }
{ Входной параметр-значение - открытый массив a. }
{ Выходные параметры-переменные: сумма (sum) и произведение (proiz). }
procedure sum_proiz(a:array of real; var sum,proiz:real);
var
  i:integer;
begin
  sum:=0;
  proiz:=1;
  for i:=0 to high (a) do
  begin
    sum:=sum+a[i];
    proiz:=proiz*a[i]
  end
end;
var
  b:array [1..5] of real;
  c:array [6..14] of real;
  d:array [0..3] of real;
  s,p:real; k:integer;
begin
  for k:=1 to 5 do readln (b[k]);
  for k:=6 to 14 do readln (c[k]);
  for k:=0 to 3 do readln (d[k]);
  { Вызов sum_proiz для вычисления суммы и произведения массива b. }
  sum_proiz (b,s,p);
  writeln ('массив b');
  for k:=1 to 5 do write(b[k]:1:4,' ');
  writeln;
```



```

writeln ('s=',s:1:5,' p=',p:1:5);
{ Вызов sum_proiz для вычисления суммы и произведения массива c. }
sum_proiz (c,s,p);
writeln ('массив c');
for k:=6 to 14 do write(c[k]:1:4,' ');
writeln;
writeln ('s=',s:1:5,'p=',p:1:5);
{ Вызов sum_proiz для вычисления суммы и произведения массива d. }
sum_proiz (d,s,p);
writeln ('массив d');
for k:=0 to 3 do write(d[k]:1:4,' ');
writeln;
writeln ('s=',s:1:5,' p=',p:1:5);
end.

```

В процедуре `sum_proiz` формальный параметр `a` описан как открытый массив, у которого не указан размер, но указан тип его элементов. Это дает возможность обращаться к процедуре с фактическими параметрами различной длины. Функция `high`, введенная в Турбо Паскаль 7.0 [9–10], возвращает верхнее возможное значение индекса массива; тогда как нижнее значение всегда равно 0. Однако в вызывающей программе нижняя и верхняя граница индекса может быть любой. (Пересчет индексов выполняется транслятором без нашего участия.)



Механизм открытых массивов работает только для одномерных массивов.

7.4.2. Параметры константы

Параметр-константа указывается в заголовке подпрограммы подобно параметру-значению, но перед его именем записывается зарезервированное слово `const`, действие которого распространяется до ближайшей точки с запятой. Параметр-константа передается в подпрограмму как параметр-переменная, но компилятор блокирует любые присваивания параметру-константе нового значения в теле подпрограммы.

В Турбо Паскале можно использовать параметры-переменные и параметры-константы без указания типа, и тогда фактический параметр может быть переменной любого типа, а ответственность за правильность использования того или иного параметра возлагается на программиста.

7.5. Процедурные типы

Для передачи имен функций и процедур в качестве фактических параметров обращения к другим процедурам и функциям фирмой Borland были разработаны *процедурные типы*. Для объявления процедурного типа используется заголовок подпрограммы, в котором пропущено ее имя, например:

```

Type
Fun1=function (x,y:real):real;

```

```
Fun2=function (x:real):real;
Proc1=procedure (x,y:real; var c,z:real);
```

! У передаваемой в процедуру функции должен присутствовать описатель *far*, указывающий, что процедура будет компилироваться в расчете на дальнюю модель памяти [9].

В Турбо Паскале есть возможность вызвать функцию и не использовать значение, которое она возвращает, то есть обратиться к ней как к процедуре. Для этого необходимо включить *расширенный синтаксис* языка. Данный режим действует по умолчанию, для его включения в тексте программы надо указать директиву компилятора {\$X+} (знак «+» – включение расширенного синтаксиса, «-» – выключение). Кроме того, в оболочке Паскаля команда **Extended Syntax** диалогового окна **Options/Compiler** может включать/выключать расширенный синтаксис.

ПРИМЕР 7.4. Рассмотрим механизм передачи подпрограмм в качестве параметра на примере следующей задачи. Программа выводит на экран таблицу значений функций $f(x)$ и $g(x)$. Вычисление и вывод значений осуществляются с помощью функции *VivodFunc*. Ее параметры:


- интервал $[a, b]$, на котором будет проводиться расчет;
- количество интервалов (n), в узлах которых будет рассчитываться функция;
- имя функции.

```
program test;
{ Описание процедурного типа func. }
type
  func=function(x:real):real;
  { Функция vivod ничего не возвращает в качестве результата. }
  function VivodFunc (a,b:real;N:word;ff:func):integer;
var
  x,y,hx:real;
  f:text;
begin
  { Шаг изменения переменной x. }
  hx:=(b-a)/N;
  x:=a;
  while(x<=b) do
  begin
    y:=ff(x);
    writeln('x=',x:5:2,' y=',y:7:2);
    x:=x+hx;
  end;
end;
{ Определение функций f и g с описателем far. }
function f(x:real):real;far;
begin
  f:=exp(sin(x))*cos(x);
end;
function g(x:real):real;far;
begin
```

```

    g:=exp(cos(x))*sin(x);
end;
begin
    { Вызов VivodFunc с функцией f в качестве параметра. }
    VivodFunc(0,1,7,f);
    writeln;
    { Вызов VivodFunc с функцией g в качестве параметра. }
    VivodFunc(0,2,8,g);
end.

```

 *Нельзя использовать стандартные процедуры и функции из системных библиотек Турбо Паскаля в качестве фактических параметров процедурного типа.*

В Турбо Паскале есть возможность использования в качестве параметров формальных параметров *массивов функций*.

ПРИМЕР 7.5. Рассмотрим модификацию примера 7.4. Программа выводит на экран таблицу значений нескольких функций с помощью функции VivodFunc, которой в качестве параметров передают:

- интервал $[a, b]$ – на нем будет проводиться расчет;
- количество интервалов (n), в узлах которых будет рассчитываться функция;
- массив функций ff – в них необходимо вычислить значения;
- количество функций (m) в массиве ff .

В тексте программы прокомментированы моменты, отличающие ее от той, что рассматривается в примере 7.4.

```

program test;
type
    func=function(x:real):real;
    { В процедуру VivodFunc передается }
    { входной параметр ff - открытый массив функций. }
    function VivodFunc(a,b:real;N:word;ff:array of func; m:word):integer;
var
    x,y,hx:real;
    i:integer;
begin
    hx:=(b-a)/N;
    { Цикл по всем функциям. }
    for i:=0 to m-1 do
        begin
            x:=a;
            while(x<=b) do
                begin
                    { Вычисление значения i-й функции в точке x. }
                    y:=ff[i](x);
                    writeln('x=',x:5:2,' y=',y:7:2);
                    x:=x+hx;
                end;
            writeln;
        end;
    end;
end;

```

```

function f(x:real):real;far;
begin
    f:=exp(sin(x))*cos(x);
end;
function g(x:real):real;far;
begin
    g:=exp(cos(x))*sin(x);
end;
{ Описание массива восьми функций fff. }
var fff:array[1..8] of func;
begin
    clrscr;
    { Запись реальных функций в массив fff. }
    fff[1]:=f;
    fff[2]:=g;
    { Вызов процедуры. }
    VivodFunc(0,1,7,fff,2);
end.

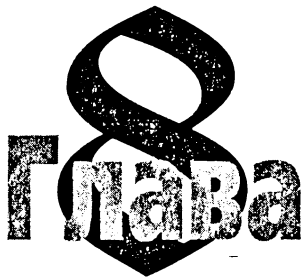
```

7.6. Упражнения по теме «Подпрограммы»

Во всех заданиях подпрограммы составить таким образом, чтобы ввод-вывод данных осуществлялся в основной программе:

1. Составить подпрограмму вычисления факториала. С помощью подпрограммы вычислить: $5!$, $12!$, $17!$.
2. Написать подпрограмму возведения числа в целую степень. Проверить ее работу на следующих данных: 2^{10} , 3^{-5} , 7^0 .
3. Составить подпрограмму, определяющую, является ли заданное число простым, то есть делящимся только на единицу и на само себя. С помощью подпрограммы определить, являются ли простыми числа 6, 11, 21, 31.
4. Составить подпрограмму, с помощью которой выяснить, являются ли совершенными заданные числа (6, 11, 28), то есть проверить, равны ли они сумме всех своих делителей, не превосходящих их самих.
5. Составить подпрограмму, которая формирует и выводит на печать единичную матрицу порядка N . Проверить ее работу на матрицах порядка 3, 5, 7, 12, 14.
6. Разработать подпрограмму вычисления расстояния между парой точек в p -мерном пространстве, задействовав которую, сформировать массив R (каждый его элемент должен быть равен расстоянию между соседними точками в двумерном пространстве). Координаты точек хранятся в массивах $x(10)$, $y(10)$ и вычисляются по формулам $x_i = 2i + 1$ и $y_j = j - 3$. Далее с помощью этой же подпрограммы сформировать массив Q , каждый элемент которого идентичен расстоянию между соседними точками в трехмерном пространстве. Координаты точек хранятся в массивах $x(9)$, $y(9)$, $z(9)$ и вычисляются по формулам: $x_i = 3i - 5$, $y_j = 2j + 1$ и $z_L = L^2$.

7. Составить подпрограмму решения квадратного уравнения $ax^2 + bx + c = 0$ и определить корни следующей совокупности биквадратных уравнений $a_i x^4 + b_i x^2 + c_i = 0$, где элементы векторов $a(8)$, $b(8)$, $c(8)$ вычисляются по формулам: $a_i = 3 \lg(I + 1) + 5i$; $b_i = j$; $c_i = (L - 3)^3$.
8. С помощью составленных заранее подпрограмм поиска минимального и максимального элементов массива Y найти разницу между минимумом и максимумом массивов $A(10)$, $B(17)$, $C(12)$. Для передачи имени массива в подпрограмму использовать понятие открытого массива. Элементы массивов вычисляются по формулам: $A_i = \sin(3,2i)$, $I = 1..10$; $B_k = \sin(\frac{\pi}{6}k) + k$, $k = 1..17$; $C_L = \arcsin(0,01j) + j \sin(5,3j)$, $L = 1..12$.
9. Составить подпрограмму умножения матриц $X(N,M)$, $Y(M,K)$ и подпрограмму их сравнения. Найти $F = A \cdot B$, $P = B \cdot A$, $G = C \cdot D$, $R = D \cdot C$. Сравнить F и P , G и R . Матрицы A , B , C и D вычисляются по формулам: $A_{ij} = 1,5i + 3j - 4$, $B_{ij} = 0,4j e^{\sin i}$, $C_{ij} = \cos(2,1j) e^{\sin(3i)}$, $D_{ij} = 2ij - 3i + 4j + 1$, $i = 1..5$; $j = 1..5$.
10. Разработать подпрограммы сортировки массива $Z(N)$ по возрастанию и убыванию. С помощью подпрограмм преобразовать матрицы $A(6,7)$, $B(7,7)$, $C(3,4)$ таким образом, чтобы нечетные столбцы были упорядочены по возрастанию, а четные – по убыванию. Элементы исходных матриц вычисляются по формулам: $A_{ij} = 1,5 \sin(3i) + 3 \cos(1,273j)$, $B_{ij} = (i^2 - 4j - 16 \cos(ij)) \sin(1,5i)$, $C_{ij} = i e^{\sin j}$.



Работа с файлами в языке Турбо Паскаль

В Паскале существует два класса файлов: типизированные и текстовые. *Текстовыми* называют файлы, состоящие из любых символов. Они организованы по строкам, каждая из которых заканчивается символом «конец строки». Конец самого файла обозначается символом «конец файла». При записи информации в текстовый файл, просмотреть который можно с помощью любого текстового редактора, все данные преобразуются к символьному типу и хранятся в символьном виде.

Файлы, состоящие из компонентов одного типа, число которых заранее не определено и может быть любым, называются *типизированными*. Они заканчиваются символом «конец файла», хранятся в двоичном виде и не просматриваются текстовыми редакторами.

8.1. Описание файловых переменных

Текстовый файл описывается с помощью служебного слова `text`.

```
var f:text;
```

Типизированные файлы могут описываться следующим образом:

```
var <имя>:file of <тип>;
```

или предварительно определяется новый тип данных:

```
type <имя>=file of <тип>;
```

Например:

```
type  
  massiv=array[1..25]of real;
```

```
type
  ff=file of real;
var
  a:text;
  b:ff;
  c:file of integer;
  d:file of massiv;
```

8.2. Обработка типизированных файлов

Ниже описаны процедуры и функции, которые используются для работы как с типизированными, так и с текстовыми файлами.

8.2.1. Процедура assign

Для начала работы с файлами необходимо связать файловую переменную в программе с файлом на диске. Для этого используется процедура `assign(f, s)`, где `f` — имя файловой переменной, `s` — полное имя файла на диске (файл должен находиться в текущем каталоге при условии, что к нему специально не указывается путь).

Например:

```
var
  f:file of real;
begin
  assign(f, 'd:\tp\tmp\abc.dat');
```

8.2.2. Процедуры reset, rewrite

После установления связи между файловой переменной и именем файла на диске нужно открыть файл, воспользовавшись процедурами `reset` или `rewrite`.

После выполнения процедуры `reset(f)`, где `f` — имя файловой переменной, файл будет открыт для чтения, и станет доступен его первый элемент. Далее можно выполнять чтение и запись информации из файла.

Файл можно открыть для записи и очистить при помощи процедуры, где `f` — имя файловой переменной. Процедура `rewrite(f)`, где `f` — имя файловой переменной, открывает и очищает файл (то есть удаляет из него информацию), после чего его можно использовать для записи.

8.2.3. Процедура close

Процедура `close(f)`, где `f` — имя файловой переменной, закрывает файл, который ранее был открыт процедурами `rewrite`, `reset`. Именно ее следует использовать при закрытии файла, в который была записана информация. Дело в том, что `write` не обращается непосредственно к диску — он пишет информацию в специальный участок памяти, называемый *буфером файла*. После того как буфер заполнится, вся информация из него вносится в файл. При

выполнении операции `close` сначала происходит запись буфера файла на диск, и только потом файл закрывается. Если файл не закрыть, то он автоматически закрывается при завершении работы программы, но при этом пропадает информация, хранящаяся в буфере файла. Поэтому после записи информации *файл необходимо закрывать*.

8.2.4. Процедура `rename`

Переименование файла, связанного с файловой переменной `f`, осуществляется в то время, когда он закрыт, при помощи процедуры `rename(f, s)`, где `f` – файловая переменная, `s` – новое имя файла (строковая переменная). Для переименования файл должен быть закрыт.

8.2.5. Процедура `erase`

Удаление файла, связанного с переменной `f`, выполняется посредством процедуры `erase(f)`, в которой `f` также является именем файловой переменной. Для корректного выполнения этой операции файл должен быть закрыт.

8.2.6. Функция `eof`

Функция `eof(f)` (end of file), где `f` – имя файловой переменной, принимает значение истина (`true`), если достигнут конец файла, иначе – ложь (`false`).

8.2.7. Процедуры `write`, `read`

Для чтения информации из файла, связанного с файловой переменной `f`, можно воспользоваться стандартными операторами чтения следующей структуры:

```
read(f, x1, x2, x3, ..., xn);  
read(f, x);
```

Операторы последовательно считывают компоненты из файла в указанные переменные, но процедура `read` не проверяет, достигнут ли конец файла, – за этим нужно следить с помощью функции `eof`.

Для записи в файл можно применять стандартные операторы записи следующей структуры:

```
write(f, x1, x2, ..., xn);  
write(f, x);
```

Операторы последовательно записывают в файл значения переменных.



Тип файла и тип переменных должны совпадать.

Для того чтобы *создать файл*, необходимо:

1. Описать файловую переменную.
2. Связать ее с физическим файлом (`assign`).
3. Открыть файл для записи (`rewrite`).
4. Внести необходимую информацию в файл (`write`).
5. Обязательно закрыть файл (`close`).

Для выполнения считывания информации из файла необходимо:

1. Описать файловую переменную.
2. Связать ее с физическим файлом.
3. Открыть файл для чтения.
4. Считать необходимую информацию (на этом этапе нужно проверять, достигнут ли конец файла).
5. Закрыть файл.

Рассмотрим несколько примеров.

ПРИМЕР 8.1. Записать n действительных чисел в файл.

```
program abc;
var
  f:file of real;
  a:real;
  i,n :integer;
begin
  { Связываем файловую переменную с файлом на диске. }
  assign (f,'d:\tp\abc.dat');
  { Открываем пустой файл для записи. }
  rewrite(f);
  { Определяем количество элементов в файле. }
  read(n);
  { В цикле вводим очередной элемент и записываем его в файл. }
  for i:=1 to n do
    begin
      write('a=');
      read(a);
      write(f,a)
    end;
  { Закрываем файл. Здесь это обязательно. }
  close (f);
end.
```

ПРИМЕР 8.2. На диске D в каталоге TP находится файл вещественных чисел, распечатать его содержимое и вычислить количество компонентов файла.

```
program bca;
var
  f1:file of real;
  a:real;
  n:integer;
begin
  { Связываем файловую переменную с файлом на диске. }
  assign(f1,'D:\TP\abc1.dat');
  { Открываем файл. }
  reset(f1);
  { В переменной n будет накапливаться количество элементов в файле. }
  n:=0;
  { Вход в цикл осуществляется, если не достигнут конец файла. }
  while not eof (f1) do
    begin
      { Считываем очередной элемент из файла. }
```

```
read(f1,a);
{ Увеличиваем количество элементов в файле на один. }
n:=n+1;
{ Выводим очередной элемент на экран. }
writeln(n, '-й элемент файла равен ', a:10:6)
end;
writeln;
writeln ('в файле ',n,' элементов');
{ Закрываем файл. Здесь это необязательно. }
close(f1);
end.
```

8.3. Последовательный и прямой доступ к файлам

Файл – последовательная структура данных. После открытия файла доступен первый компонент. Можно последовательно считывать или записывать один компонент файла за другим. Допустим, необходимо считать пятнадцатый, а затем первый элементы файла. С помощью последовательного доступа к файлу это можно сделать таким образом:

```
reset(f);
for i:=1 to 15 do
  read(f,b);
reset(f);
read(f,a);
```

Как видно, такое чтение компонента из файла, а затем повторное открытие файла – не самый удачный способ. Гораздо удобнее использовать встроенные процедуры и функции Турбо Паскаля для прямого доступа к компонентам файла, что означает возможность определять позицию интересующего нас компонента внутри файла и указывать непосредственно на него. При прямом доступе к файлу его компоненты нумеруются от 0 до n , где n – число компонентов в файле. Самый первый компонент имеет номер 0.

8.3.1. Функция filesize

Функция `filesize(f)`, где f – файловая переменная, возвращает значение типа `longint`, то есть число реальных компонентов в открытом файле f . Для пустого файла она вернет 0.

8.3.2. Функция filepos

Функция `filepos(f)` возвращает значение типа `longint` – текущую позицию в файле f , который должен быть открыт. Если файл только что открылся, то `filepos(f) = 0`. После прочтения последнего компонента из файла значение `filepos(f)` совпадает со значением `filesize(f)`, что указывает на достижение конца файла. Последнее можно проверить еще и так:

```
if filepos(f)=filesize(f) then
writeln('достигнут конца файла');
```

ПРИМЕР 8.3. Вычислить количество компонентов в файле вещественных чисел, вывести содержимое файла на экран.

```
var
  f:file of real;
  a:real;
  i:word;
begin
  assign(f,'abc.dat');
  reset(f);
  writeln('в файле',filesize(f), ' чисел');
  for i:=1 to filesize(f) do
    begin
      read(f,a);
      write(a, ' ')
    end;
  close(f)
end.
```

8.3.3. Процедура seek

Процедура `seek(f,n)` устанавливает указатель в открытом файле `f` на компонент с номером `n` (нумерация компонентов идет от 0). Затем значение компонента может быть считано.

ПРИМЕР 8.4. На диске **Е** в каталоге **ABC** есть файл целых чисел `a2.int`, поменять местами его максимальный и минимальный элементы.

Ниже приведены два варианта этой программы. В первом после считывания компонентов файла в массив происходит поиск минимального и максимального элементов массива и их индексов. Затем максимальное и минимальное значения перезаписываются в файл.

Вторая программа работает по другому алгоритму. Организован один цикл, в котором очередное значение считывается в переменную; в этом же цикле осуществляется поиск минимального и максимального элементов среди компонентов файла и их индексов. Затем происходит перезапись в файл максимального и минимального значений.

Вариант 1:

```
program var_1;
var
  f:file of integer;
  i,max,n max,min,n min:integer;
  a:array[0..200] of integer;
begin
  assign(f,'e:\abc\a2.int');
  reset(f);
  { Считываем компоненты файла в массив a. }
  for i:=0 to filesize(f)-1 do read(f,a[i]);
  { Начальное присваивание максимального }
  { и минимального элементов массива и их индексов. }
```

```

    max:=a[0]; nmax:=0;
    min:=a[0]; nmin:=0;
    { Основной цикл для поиска максимального и }
    { минимального элементов массива и их индексов. }
    for i:=1 to filesize(f)-1 do
        begin
            if a[i]>max then
                begin
                    max:=a[i];
                    nmax:=i
                end;
            if a[i]<min then
                begin
                    min:=a[i];
                    nmin:=i
                end;
        end;
    { Перезапись максимального и минимального значений в файл. }
    seek(f,nmax);
    write(f,min);
    seek(f,nmin);
    write(max)
    close(f)
end.

```

Вариант 2:

```

var
    f:file of integer;
    a,i,max,nmax,min,nmin:integer;
begin
    assign(f, 'e:\abc\al2.int');
    reset(f);
    for i:=0 to filesize(f)-1 do
        begin
            read(f,a);
            if i=0 then
                { Начальное присваивание максимального и }
                { минимального значений массива и их индексов. }
                begin
                    max:=a; nmax:=i;
                    min:=a; nmin:=i;
                end
            else
                begin
                    { Сравнение текущего значения с максимальным (минимальным). }
                    if max<a then
                        begin
                            max:=a;
                            nmax:=i
                        end;
                    if min>a then
                        begin
                            min:=a;
                            nmin:=i
                        end
                    end
                end
            end
        end
    end
end.

```

```

        end
    end
end;
{ Перезапись максимального и минимального значений в файл. }
seek(f,nmax);
write(f,min);
seek(f,nmin);
write(f,max);
end.

```

8.3.4. Процедура truncate

Процедура `truncate(f)`, где `f` – имя файловой переменной, отсекает часть открытого файла, начиная с текущего компонента, и подтягивает на его место конец файла.

ПРИМЕР 8.5. Задан файл вещественных чисел `abc.dat`. Удалить из него максимальный и минимальный элементы.

Приведем две программы, решающие эту задачу. Алгоритм первой состоит в следующем: считываем компоненты файла в массив, в котором находим максимальный и минимальный элементы и их номера. Открываем файл для записи и вносим в него все элементы, за исключением максимального и минимального.

```

program var3;
var
    f:file of real;
    max,min:real;
    j, i,nmax,nmin:integer;
    a:array [1..300] of real;
begin
    assign(f,'abc.dat');
    reset(f);
    { Запоминаем в переменной j количество компонентов в файле. }
    j:=filesize(f);
    { Считываем компоненты файла в массив a. }
    for i:=1 to filesize(f) do read(f,a[i]);
    close(f);
    { Открываем файл для записи. }
    rewrite(f);
    { Начальное присваивание максимального }
    { и минимального элементов массива и их индексов. }
    max:=a[1];min:=a[1];
    nmax:=1;nmin:=1;
    { Основной цикл для поиска максимального }
    { и минимального элементов массива и их индексов. }
    for i:=2 to filesize(f) do
    begin
        if a[i]>max then
        begin
            max:=a[i];
            nmax:=i
        end;
    end;

```

```

    if a[i]<min then
    begin
        min:=a[i];
        nmin:=i;
    end;
end;
{ Перезапись элементов массива в файл, }
{ за исключением элементов с номерами nmax и nmin. }
for i:=1 to filesize(f) do
    if (i<>nmax)and (i<>nmin) then
        write(f,a[i]);
close(f)
end.

```

Вторая программа работает следующим образом. Находим максимальный и минимальный элементы и их номера среди компонентов файла (описание программы var2). Если $n_{\min} > n_{\max}$, то меняем содержимое переменных n_{\min} и n_{\max} . Далее элементы, лежащие между минимальным и максимальным (формально между элементами с номерами n_{\min} и n_{\max}), сдвигаем на один влево. Тем самым мы убираем элемент с номером n_{\min} . После этого все компоненты, лежащие после элемента с номером n_{\max} , сдвинем на два влево. Этим мы сотрем максимальный элемент. Затем два последних компонента в файле необходимо удалить.

```

program var4;
var
    f:file of real;
    a:real;
    max,min:real;
    i,nmax,nmin:integer;
begin
    assign(f,'abc.dat'); reset (f);
    { Поиск максимального и минимального элементов в файле и их индексов. }
    for i:=0 to filesize(f)-1 do
    begin
        read(f,a);
        if i=0 then
        begin
            max:=a;
            nmax:=i;
            min:=a;
            nmin:=i
        end
    else
    begin
        if a>max then
        begin
            max:=a;
            nmax:=i;
        end;
        if a<min then
        begin
            min:=a;

```

```

        nmin:=i;
    end
end
end;
{ Сравниваем nmin и nmax. }
if nmax<nmin then
begin
    i:=nmax;
    nmax:=nmin;
    nmin:=i
end;
{ Сдвигаем элементы, лежащие между компонентами }
{ с номерами nmin и nmax, на один влево. }
for i:=nmin to nmax-2 do
begin
    seek(f,i+1);
    read(f,a);
    seek(f,i);
    write(f,a)
end;
{ Сдвигаем элементы, лежащие после компонента }
{ с номером nmax, на два влево. }
for i:=nmax to filesize(f)-3do
begin
    seek(f,i+1);
    read(f,a);
    seek(f,i-1);
    write(f,a);
end;
{ Отрезаем последние два компонента. }
truncate(f);
close(f);
end.

```

8.4. Обработка ошибок ввода-вывода

Компилятор Турбо Паскаля позволяет генерировать код программы в двух режимах: с проверкой корректности ввода-вывода и без нее. В оболочке этот режим включается при последовательном выполнении команд **Options/Compiler/T\O checking** (см. приложение).

В программу может быть включен ключ режима компиляции. Кроме того, предусмотрен перевод контроля ошибок ввода-вывода из рабочего состояния в обратное:

- {\$I+} – режим проверки ошибок ввода-вывода включен;
- {\$I-} – режим проверки ошибок ввода-вывода отключен.

По умолчанию, как правило, действует режим {\$I+}. Можно многократно включать и выключать режимы, создавая области с контролем ввода и без него. Все ключи компиляции описаны в приложении.

При включенном режиме проверки ошибка ввода-вывода будет фатальной, программа прервется, выдав номер ошибки [8, 9].

Если убрать режим проверки, то при возникновении ошибки ввода-вывода программа не будет останавливаться, а продолжит работу со следующего оператора. Результат операции ввода-вывода будет не определен.

Для опроса кода ошибки лучше пользоваться специальной функцией Турбо Паскаля `IOresult`, но необходимо помнить, что опросить ее можно только один раз после каждой операции ввода или вывода: она обнуляет свое значение при каждом вызове. `IOresult` возвращает целое число, соответствующее коду последней ошибки ввода-вывода. Если `IOresult = 0`, то при вводе-выводе ошибок не было, иначе `IOresult` возвращает код ошибки. Некоторые коды ошибок приведены в табл. 8.1.

Таблица 8.1 ▼ Описание кодов возврата ошибок ввода-вывода

Код ошибки	Описание
2	Файл не найден
3	Путь не найден
4	Слишком много открытых файлов
100	Ошибка чтения с диска
101	Ошибка записи на диск
102	Файл не связан
103	Файл не открыт
104	Файл не открыт для ввода
105	Файл не открыт для вывода
106	Неправильный числовой формат

Рассмотрим несколько практических примеров обработки ошибок ввода-вывода:

1. При открытии проверить, существует ли заданный файл и возможно ли чтение данных из него.

```
assign (f, 'abc.dat');
{$I-}
reset(f);
{$I+}
if IOresult<>0 then
  writeln ('файл не найден или не читается')
else
begin
  read(f,...);
  ...
  close(f);
end;
```

2. Проверить, является ли вводимое с клавиатуры число целым.


```
var i:integer;
begin
{$I-}  ~
```



```
repeat
  write('введите целое число i');
  readln(i);
until (IOresult=0);
{$I+}
{ Этот цикл повторяется до тех пор, пока не будет введено целое число. }
end.
```

8.5. Работа с текстовыми файлами

При работе с текстовыми файлами действие процедур `reset`, `rewrite`, `close`, `rename`, `erase` и функции `eof` аналогично их действию при работе с компируемыми (типизированными) файлами.

 *Процедуры `seek`, `truncate` и функции `filepos` относятся только к типизированным файлам.*

При работе с текстовыми файлами можно пользоваться процедурой `append`.

Процедура `append(f)`, где `f` – имя файловой переменной, служит для специального открытия файлов для записи. Она применима только к уже физически существующим файлам, открывает и готовит их для добавления информации в конец файла.

Запись и чтение в текстовый файл осуществляются с помощью операторов `write`, `writeln`, `read`, `readln` следующей структуры:

```
read(f, x1, x2, x3, ..., xn);
read(f, x);
readln(f, x1, x2, x3, ..., xn);
readln(f, x);
write(f, x1, x2, x3, ..., xn);
write(f, x);
writeln(f, x1, x2, x3, ..., xn);
writeln(f, x);
```

В этих операторах смысл переменных `f`, `x1`, `x2`, `x3`, ..., `xn` такой же, как и при работе с типизированными файлами.

Однако имеется ряд особенностей при работе операторов `write`, `writeln`, `read`, `readln` с текстовыми файлами. Итак, имена переменных могут быть целого, вещественного, символьного и строкового типа. Все переменные будут вноситься в текстовый файл в виде текстовой информации. Действие оператора `writeln` отличается тем, что записывает в текстовый файл символ конца строки.

Массивы могут вводиться поэлементно. При работе с текстовым файлом необходимо помнить специальные правила чтения значений переменных:

- когда вводятся числовые значения, два числа считаются разделенными, если между ними есть хотя бы один пробел, или символ табуляции, или символ конца строки;

- при вводе строк начало текущей строки идет сразу за последним, введенным до этого символом. Вводится количество символов, равное объявленной длине строки. Если при чтении встретился символ «конец строки» (.) то работа с этой строкой заканчивается. Сам символ конца строки является разделителем и в переменную никогда не считывается;
- процедура `readln` считывает значения текущей строки файла, курсор переводится в новую строку файла, и дальнейший ввод осуществляется с нее.

ПРИМЕР 8.6. Написать программу умножения двух матриц, результирующую матрицу вывести на экран дисплея и в текстовый файл.

```

program abc;
var
  f:text;
  a,b,c:array[1..5,1..5] of real;
  i,j,k:integer;
begin
  assign(f, 'abcd.txt');
  { Открываем пустой текстовый файл. }
  rewrite(f);
  writeln('Введите матрицу A');
  for i:=1 to 5 do
    for j:=1 to 5 do
      read(a[i,j]);
      writeln('Введите матрицу B');
  for i:=1 to 5 do
    for j:=1 to 5 do
      read(b[i,j]);
  for i:=1 to 5 do
    begin
      for j:=1 to 5 do
        begin
          c[i,j]:=0;
          for k:=1 to 5 do
            c[i,j]:=c[i,j]+a[i,k]*b[k,j];
          write (c[i,j]:8:3,' ');
          { Выводим очередной элемент в текстовый файл. }
          write (f,c[i,j]:8:3,' ');
        end;
      writeln;
      { Переводим курсор в файле на новую строчку. }
      writeln(f);
    end;
  close(f);
end.

```

В качестве текстовых файлов могут использоваться физические устройства: клавиатура, дисплей, последовательные и параллельные порты. Эти устройства имеют фиксированные имена и во многом схожи с файлами. В табл. 8.2 приведены имена основных устройств MS DOS.

Таблица 8.2 ▼ Имена основных устройств MS DOS

Имя	Название устройства	Назначение
Con	Клавиатура и экран	Ввод из con – это чтение с клавиатуры, вывод в con – запись на экран
Prn	Принтер	Вывод на принтер
Lpt1	Параллельные порты	Через эти имена файлов происходит вывод данных на принтер или другие устройства:
Lpt2	Параллельные порты	
Lpt3	Параллельные порты	
com1	Последовательные порты	Подключение к портам
com2	Последовательные порты	
Nul	Фиктивное устройство	Это бездонный файл, принимающий что угодно, но всегда пустой

Физические файлы-устройства организуются как текстовые файлы; для нормальной работы их надо связывать с текстовым логическим файлом и осуществлять операции ввода-вывода с этим файлом. Имена физических устройств надо записывать без точек, двоеточий, запятых и т.д., например: `assign(f, 'prn')`; `rewrite(f)`; но не так – `assign(f, 'prn.')`; `rewrite(f)`; В этом случае речь идет о файле с именем `prn`, а не о принтере.

Рассмотрим пример построчного вывода матрицы на принтер.

```

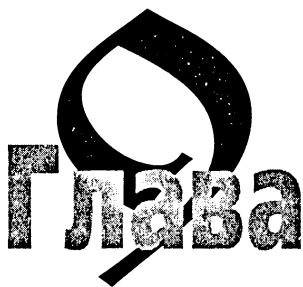
Var
  f:text;
  i,j:integer;
  A:array[1..20,1..20] of real;
Begin
  ...
  { Связываем файловую переменную с принтером. }
  assign(f,'prn');
  { Открываем вывод на принтер. }
  rewrite(f);
  writeln (f,'Матрица A');
  for i:=1 to 5 do
    begin
      for j:=1 to 5 do
        write(f,a[i,j]:10:5,' ');
        writeln(f);
      end;
    { Закрываем файл, с помощью которого осуществлялся вывод на принтер. }
    close(f);
  end.

```

8.6. Упражнения по теме «Работа с файлами в языке Турбо Паскаль»

Во всех заданиях составить две программы. Первая должна формировать типизированный файл исходных данных. Вторая – считывать данные из этого файла, выполнить соответствующие вычисления и записывать их результаты в текстовый файл.

1. Из компонентов исходного файла сформировать массивы четных и нечетных чисел. Определить наибольший четный компонент файла и наименьший нечетный.
2. На основе исходного файла создать массив удвоенных нечетных чисел. Упорядочить его по возрастанию элементов.
3. Сформировать массив положительных чисел, делящихся на пять без остатка, используя элементы исходного файла. Выстроить его по убыванию элементов.
4. Из компонентов исходного файла сформировать массивы положительных и отрицательных простых чисел. Вычислить количество нулевых компонентов файла.
5. Взяв за основу файл, создать массив, элементы которого не являются простыми числами и расположены до минимального элемента.
6. Из компонентов исходного файла сформировать массив, записав в него только ненулевые компоненты, находящиеся после максимального элемента.
7. Сделать массив из элементов исходного файла, внося в него числа, превосходящие среднее значение положительных компонентов файла.
8. Из компонентов исходного файла сформировать массив, записав в него числа, расположенные в файле до минимального элемента и после максимального.
9. Массив создать из компонентов исходного файла. Внести в него числа, расположенные в файле между минимальным и максимальным элементами.
10. Из компонентов исходного файла сформировать массив, в котором вначале расположить четные, а затем нечетные числа. Определить номера наибольшего нечетного и наименьшего четного компонентов.



Динамические переменные и указатели

Данный раздел посвящен описанию возможностей Турбо Паскаля при работе с динамическими переменными и указателями.

9.1. Динамическая память

Все переменные, объявленные в программе, размещаются в одной непрерывной области оперативной памяти, которая называется *сегментом данных*. Длина сегмента данных определяется архитектурой микропроцессора и составляет 65536 байт, что может вызвать известные затруднения при обработке больших массивов. Этим и обусловлено ограничение на размер массива 64 Кб.

С другой стороны, объем памяти даже компьютера XT (640 Кб) достаточен для решения задач с большой размерностью. Выходом из положения может служить так называемая *динамическая память*, то есть оперативная память ПК, выделяемая программе при ее работе за вычетом сегмента данных (64 Кб), стека (обычно 16 Кб) и собственно тела программы. Размер динамической памяти можно варьировать в широких пределах. По умолчанию этот размер определяется всей доступной памятью ПК и составляет не менее 200–300 Кб.

Динамическое размещение данных означает использование динамической памяти непосредственно при работе программы. Статическое размещение, в отличие от предыдущего, осуществляется компилятором в процессе работы программы. При динамическом размещении заранее не известно количество размещаемых данных. Кроме того, к ним нельзя обращаться по именам, как к статическим переменным.

9.2. Адреса и указатели

Оперативная память ПК представляет собой совокупность элементарных ячеек для хранения информации – байтов, каждый из которых имеет собственный номер. Эти номера называются *адресами*, они позволяют обращаться к любому байту памяти.

Турбо Паскаль имеет гибкое средство управления памятью – указатели.

Указатель – переменная, которая в качестве своего значения содержит адрес байта памяти. Один указатель позволяет адресовать 64 Кб. Указатель занимает 2 байта.

9.3. Объявление указателей

Как правило, в Турбо Паскале указатель связывается с некоторым типом данных. В таком случае он называется *типизированным*. Для его объявления используется знак ^, который помещается перед соответствующим типом, например:

```
type
  massiv=array [1..2500] of real;
var
  a:^integer; b,c:^real; d:^massiv;
```

В Турбо Паскале можно объявлять указатель, не связывая его с конкретным типом данных. Для этого служит стандартный тип *pointer*, например:

```
var
  p,c,h: pointer;
```

Указатели такого рода будем называть *нетипизированными*. Поскольку нетипизированные указатели не связаны с конкретным типом, с их помощью удобно динамически размещать данные, структура и тип которых меняется в ходе работы программы.

Значениями указателей являются адреса переменных памяти, поэтому следовало ожидать, что значение одного из них можно передавать другому. На самом деле все не совсем так. В Турбо Паскале эта операция проводится только среди указателей, связанных с одними и теми же типами данных.

Например:

```
Var
  p1, p2:^integer; p3:^real; pp:pointer;
```

В этом случае присваивание $p1:=p2$; допустимо, в то время как $p1:=p3$; запрещено, поскольку $p1$ и $p3$ указывают на разные типы данных. Это ограничение не распространяется на нетипизированные указатели, поэтому можно записать $pp:=p3$; $p1:=pp$; и достичь необходимого результата.

9.4. Выделение и освобождение динамической памяти

Вся динамическая память в Турбо Паскале представляет собой сплошной массив байтов, называемый *кучей*. Физически куча располагается за областью памяти, которую занимает тело программы.

Начало кучи хранится в стандартной переменной `heaporg`, конец – в переменной `heapend`. Текущая граница незанятой динамической памяти хранится в указателе `heapprt`.

Память под любую динамическую переменную выделяется процедурой `new`, параметром обращения к которой является типизированный указатель. В результате обращения последний принимает значение, соответствующее динамическому адресу, начиная с которого можно разместить данные, например:

```
var
  i,j:^integer;
  r:^real;
begin
  new(i);
  new(R);
  new(j)
```

В результате выполнения первого оператора указатель `i` принимает значение, которое перед этим имел указатель кучи `heapprt`. Сам `heapprt` увеличивает свое значение на два, так как длина внутреннего представления типа `integer`, связанного с указателем `i`, составляет 2 байта. Оператор `new(r)` вызывает еще одно смещение указателя `heapprt`, но уже на 6 байт, потому что такова длина внутреннего представления типа `real`. Аналогичная процедура применяется и для переменной любого другого типа. После того как указатель стал определять конкретный физический байт памяти, по этому адресу можно разместить любое значение соответствующего типа, для чего сразу за указателем без каких-либо пробелов ставится значок `^`, например:

```
i^:=4+3;
j^:=17;
r^:=2 * pi.
```

Таким образом, значение, которое определяет указатель, то есть собственно данные, размещенные в куче, обозначаются значком `^`, который ставится сразу за указателем. Если за последним этот значок отсутствует, то имеется в виду адрес, по которому размещаются данные. Динамически размещенные данные (но не их адрес!) можно использовать для констант и переменных соответствующего типа в любом месте, где это допустимо, например:

```
r^:=sqr(r^)+sin(r^+i^)-2.3.
```

Невозможен оператор

```
r:=sqr(r^)+i^;
```

так как указателю `r` нельзя присвоить значение вещественного типа.

Точно также недопустим оператор

```
r^:=sqr(r);
```

поскольку значением указателя `r` является адрес, и его (в отличие от того значения, которое размещено по данному адресу) нельзя возводить в квадрат. Ошибочным будет и присваивание `r^:=I;`, так как вещественным данным, на которое указывает `r^`, нельзя давать значение указателя (адрес). Динамическую память можно не только забирать из кучи, но и возвращать обратно. Для этого используется процедура `dispose(p)`, где `p` – указатель, который не изменяет значение указателя, а лишь возвращает в кучу память, ранее связанную с указателем.

При работе с указателями и динамической памятью необходимо самостоятельно следить за правильностью использования процедур `new`, `dispose` и работы с адресами и динамическими переменными, так как транслятор такие ошибки не контролирует. Ошибки этого класса могут не только привести к зависанию компьютера, но и иметь более серьезные последствия.

Другая возможность состоит в освобождении целого фрагмента кучи. С этой целью перед началом выделения динамической памяти текущее значение указателя `heapptr` запоминается в переменной-указателе с помощью процедуры `mark`. Теперь можно в любой момент освободить фрагмент кучи, начиная с того адреса, который запомнила процедура `mark`, и до конца динамической памяти. Для этого используется процедура `release`.

Процедура `mark` запоминает текущее указание кучи `heapptr` (обращение `mark(ptr)`, где `ptr` – указатель любого типа, в котором будет возвращено текущее значение `heapptr`). Процедура `release(ptr)`, где `ptr` – указатель любого типа, освобождает участок кучи от адреса, хранящегося в указателе до конца кучи.

9.5. Процедуры `freemem`, `getmem`.

Использование динамических массивов

Для работы с указателями любого типа используются процедуры `getmem`, `freemem`. Процедура `getmem(p,size)`, где `p` – указатель, `size` – размер в байтах выделяемого фрагмента динамической памяти (`size` типа `word`), резервирует за указателем фрагмент динамической памяти требуемого размера.

Процедура `freemem(p,size)`, где `p` – указатель, `size` – размер в байтах освобождаемого фрагмента динамической памяти (`size` типа `word`), возвращает в кучу фрагмент динамической памяти, который был зарезервирован за указателем. При применении процедуры к уже освобожденному участку памяти возникает ошибка.

После рассмотрения основных принципов и процедур работы с указателями возникает вопрос: а зачем это нужно? В основном для того, чтобы работать с так называемыми динамическими массивами. Последние представляют собой массивы переменной длины, память под которые может выделяться (и изменяться) в процессе выполнения программы, как при каждом новом запуске программы, так и в разных ее частях. Обращение к i -му элементу динамического массива x имеет вид $x[i]$.

ПРИМЕР 10.1. Рассмотрим процесс функционирования динамических массивов на примере решения следующей задачи: найти максимальный и минимальный элементы массива $x(n)$.

Рассмотрим процесс решения задачи традиционным способом.

```
Program din_mas1;
Var
  x:array [1..150] of real;
  i,n:integer; max,min:real;
begin
  writeln('введите размер массива');
  readln (n);
  for i:=1 to N do
    begin
      write('x[' ,i,']='); readln(x[i]);
    end,
  max:=x[1]; min:=x[1];
  for i:=2 to N do
    begin
      if x[i] > max then max:=x[i];
      if x[i] < min then min:=x[i];
    end,
  writeln ('максимум=',max:=1:4);
  writeln ('минимум=',min:=1:4);
end.
```

Теперь рассмотрим процесс решения задачи с использованием указателей. Распределение памяти проводим с помощью процедур `new-dispose` или `getmem-freemem`.

```
Program din_mas2;
type massiw= array[1..150]of real;
var x:^massiw;
    i,n:integer;max,min:real;
begin
  { Выделяем память под динамический массив из 150 вещественных чисел. }
  new(x);
  writeln('введите размер массива'); readln(n);
  for i:=1 to N do
    begin
      write('x(' ,i,')='); readln(x[i]);
    end,
  max:=x[1];min:=x[1];
  for i:=2 to N do
```

```

begin
    if x^[i] > max then max:=x^[i];
    if x^[i] < min then min:=x^[i];
end;
writeln('максимум=',max:1:4, ' минимум=',min:1:4);
{ Освобождаем память. }
dispose(x);
end.

Program din_mas3;
type
    massiw=array[1..150]of real;
var x:^massiw;
    i,n:integer;max,min:real;
begin
    writeln('введите размер массива'); readln(n);
    { Выделяем память под n элементов массива. }
    getmem(x,n*sizeof(real));
    for i:=1 to N do
        begin
            write('x(',i,')='); readln(x^[i]);
        end,
    max:=x^[1];min:=x^[1];
    for i:=2 to N do
        begin
            if x^[i] > max then max:=x^[i];
            if x^[i] < min then min:=x^[i];
        end,
    writeln('максимум=',max:1:4, ' минимум=',min:1:4);
    { Освобождаем память. }
    freemem(x,n*sizeof(real));
end.

```

При работе с динамическими переменными необходимо соблюдать следующий порядок работы:

1. Описать указатели.
2. Распределить память.
3. Обработать динамический массив.
4. Освободить память.

9.6. Массивы больше 64 Кб в Турбо Паскале

В куче нельзя выделить память больше 64 Кб [1]; сегмент данных не может превышать этого размера. Дело в том, что адресуемое пространство памяти в операционной системе MS DOS организовано сегментами – последовательными блоками памяти по 64 Кб каждый. Однако в Турбо С можно выделить блок памяти более 64 Кб (функции `farcalloc` и `farmalloc`), хотя и тот, и другой язык являются разработкой фирмы Borland. Возникает вопрос: почему нельзя выделить блок памяти более 64 Кб средствами Турбо Паскаля? А если можно, то как обращаться к элементам массива, лежащими за пределами 64 Кб?

Синтаксис языка Паскаль не позволяет выделить более 65521 байт. Как кажется авторам, это ограничение можно обойти. Заметим, что адрес с точки зрения MS DOS состоит из сегмента и смещения:

```
адрес = сегмент * 16 + смещение;  
сегмент = адрес div 16;  
смещение = адрес mod 16;
```

Один сегмент может адресовать 64 Кб.

В Турбо Паскале существуют две функции типа word, которые возвращают сегментную часть адреса (`seg(p)`) и смещение (`ofs(p)`).

Аргумент `p` в обращении может быть любого типа. Существует функция `ptr(seg, ofs: word)`, которая возвращает значения указателя (тип `pointer`), а фактически по значению сегмента и смещения вычисляет значение адреса.

Есть функция MS DOS с номером 48H, которая позволяет выделять память параграфами (параграф равен 16 байт).

Входные данные функции 48H: в регистре `BX` – количество выделяемой памяти в параграфах.

Выходные данные функции 48H: в регистре `AX` – адрес сегмента, начиная с которого выделено указанное количество памяти.

Если невозможно выделить указанное число параграфов, то в регистре `AX` возвращается число 8. Когда при выделении памяти происходит сбой, в регистре `AX` возвращается число 7.

На базе этой функции была написана процедура `far_getmem` (`var p:pointer; size:longint`), которая позволяет выделить всю доступную свободную оперативную память.

```
procedure far_getmem(var p:pointer;size:longint);  
var  
  r:registers; newsize,dosseg:word;  
begin  
  newsize:=(size+15) div 16;  
  r.bx:=newsize;  
  r.ah:=$48;  
  msdos(r);  
  if r.ax=$07 then  
  begin  
    writeln('Сбойные блоки управления памятью');  
    halt(1);  
  end  
  else  
  if r.ax = $08 then  
  begin  
    writeln('нельзя выделить',size,' байт');  
    halt(1);  
  end  
  else  
  p:=ptr(r.ax,0);  
end;
```

Необходимо помнить, что данная процедура выделяет память средствами MS DOS за пределами Паскаль-программы. Если пользоваться установками

Турбо Паскаля по умолчанию, то ваша программа захватит под свою кучу всю доступную память и процедуре `far_getmem` ничего не останется.

Если использовать эту идею на практике, то необходимо в директиве `{$M. .}` явно указать максимум кучи для вашей программы.

При использовании только статических переменных Турбо Паскаля можно максимум кучи установить равными нулю. Однако некоторые стандартные процедуры и функции языка используют память кучи: так, при работе с графикой процедура `initgraph` в динамической памяти размещает графический драйвер. Поэтому максимум кучи в директиве `{$M. .}` необходимо установить, исходя из требований вашей задачи. В среде Турбо Паскаль 7.0 для процедуры `far_getmem` доступно около 300 Кб памяти. При запуске из командной строки процедура `far_getmem` позволяет выделить память размером около 600 Кб.

Функция MS DOS 49H дает возможность освободить память, выделенную с помощью функции 48H.

Входные данные функции 49H: в регистре ES – адрес сегмента, начиная с которого освобождается память.

Выходные данные функции 49H: в регистре AX – возвращает код ошибки;

AX = 7 – сбойные блоки управления памятью;

AX = 8 – память не была выделена процедурой 48H.

На базе этой функции была написана процедура `far_freemem`, которая освобождает память, выделенную процедурой `far_getmem`.

```
procedure far_freemem(var p:pointer;size:longint);
var
  r:registers;
  segment,dosseg:word;
begin
  if ofs(p^)<0 then
    begin
      p:=nil;
      system.freemem(p,1);
    end;
  segment:=seg(p^);
  r.es:=segment;
  r.ah:=$49;
  msdos(r);
  if r.ax=$07 then
    begin
      writeln('Сбойные блоки управления памятью ');
      halt(1);
    end
  else
    if r.ax = $08 then
      begin
        writeln('Память не была выделена процедурой getmem');
        halt(1);
      end;
    end;
end.
```

Процедура `far_maxavail` позволяет вычислить максимальный объем памяти, который может быть выделен `far_getmem`.

```
Function far_MaxAvail:LongInt;
var
  NewSize :Word;
begin
  asm
    mov ah, 048h
    mov bx, 0FFFFh
    int 21h
    mov NewSize, bx
  end;
  far_MaxAvail:=LongInt(NewSize)*16;
end.
```

Рассмотрим механизм выделения и освобождения памяти с помощью рассмотренных ранее процедур на примере:

```
{ $M 16384,0,16384 }
uses
  dos_mem;
type
  massiv = array [1..100] of real;
var
  a: ^massiv;
  i,n:longint;
begin
  far_getmem(pointer(a), 20000 * Sizeof(real));
  { Выделяем память для массива из 20000 вещественных чисел, но обращение }
  { вида a[i] правильно адресует элементы массива, находящиеся в первых }
  { 64 Кб памяти. }
  far_freemem(pointer(a), 20000 * Sizeof(real))
end.
```

Однако нужно не только выделить большую область памяти, но и получить возможность обратиться к определенному ее участку (элементу динамического массива). Стандартными средствами Турбо Паскаля можно адресовать 64 Кб, но с помощью конструкции `a[i]` нельзя будет обратиться к элементу массива, лежащему за пределами 64 Кб.

Возникла проблема, как обратиться к элементу массива (конкретному участку памяти). Решить ее можно с помощью функция `far_adres`, которая вычисляет адрес *i*-го элемента массива.

```
Function far_ADRES(P:pointer;n:longint;size:byte):pointer;
var
  longintadr:longint; newseg,newofs:word;
Begin
  longintadr:=longint(seg(p)) shl 4 + (n-1)*size;
  newseg:=longintadr div 16;
  newofs:=longintadr mod 16;
  far_adres:=ptr(newseg,newofs)
end;
```

Узнав адрес, требуется только преобразовать значение, хранящееся там, к нужному типу данных: `тип(far_adres(p,i,Sizeof(тип))^)`.

Задан файл вещественных чисел объемом более 64 Кб, считать элементы в один динамический массив. Структура программы выглядит так:

```
{ $M,16384,0,16384 }
{ Здесь должны быть тексты подпрограммы работы с массивами более 64 Кб. }
var
  a:pointer;
  i,n:integer;
  f:file of real;
begin
  assign(f,'abc.dat');
  reset(f);
  if (filesize(f)*6)<=far_maxavail then
  begin
    { Выделение памяти под большой массив с помощью процедуры far_getmem. }
    far_getmem(f,filesize(f)*Sizeof(real);
    { обращение к i-му элементу массива real(far_adres(a,i,Sizeof(real))^) }
    for i:=0 to filesize(f)-1 do
      read(f,real(far_adres(a,i,Sizeof(real))^));
    { Считаны элементы в массив. }
    { Далее - обработка массива. }
    { Освобождение памяти. }
    far_freemem(f,filesize(f) * 6)
  end
end.
```

Процедуры, описанные в этой главе, делают доступной всю основную память ПК. Если выделять под кучу всю память, доступную под управление DOS, то размер массива, определяемый с помощью кучи, может достигать 600 Кб, что полностью снимает ограничение 64 Кб в Турбо Паскале.

Глава 10

Модули в Турбо Паскале

Возможности Турбо Паскаля расширяются благодаря использованию модулей, которые представляет собой набор констант, типов данных, переменных, процедур и функций. Язык располагает стандартными (встроенными) модулями SYSTEM, DOS, OVERLAY, GRAPH, CRT, PRINTER, TURBO3, GRAPH3. Два последних предназначены для поддержки совместимости программ, написанных в версии 3.0. Обычно модуль имеет расширение tpu (turbo pascal unit). Модули CRT, SYSTEM, DOS, OVERLAY, PRINTER объединены в библиотеку и хранятся в файле turbo.tpl (turbo pascal library). Модуль GRAPH находится в файле graph.tpu, модули TURBO3, GRAPH3 – в файлах graph3.tpu, turbo3.tpu соответственно.

Модуль SYSTEM поддерживает все стандартные процедуры и функции, обеспечивает ввод-вывод данных, обработку строк, динамическое распределение оперативной памяти и ряд других возможностей Турбо Паскаля. Он подключается к любой программе автоматически.

Модуль DOS содержит многочисленные процедуры и функции, многие из которых по своему действию эквивалентны командам DOS.

Поддержку системы оверлеев обеспечивает модуль OVERLAY.

Вывод информации на принтер позволяет организовать модуль PRINTER.

Модуль CRT содержит процедуры и функции, которые обеспечивают работу с клавиатурой, экраном дисплея в текстовом режиме и управление звуком.

Модуль GRAPH обеспечивает работу с экраном дисплея в графическом режиме.

Для того чтобы использовать модули в программах, их следует указывать в операторе uses, который должен размещаться до раздела описаний. Например:

```
uses crt, graph, printer;
```

После того как модуль будет подключен с помощью `uses`, все его константы, типы, переменные, процедуры и функции окажутся доступными в программе.

Перед тем как познакомиться с модулем `CRT`, рассмотрим еще две функции:

- `chr(x)`, x – переменная, константа или выражение типа `byte`. Эта функция возвращает символ (тип `char`), соответствующий коду x .
- `ord(x)`, x – символ (`char`). Эта функция возвращает значение типа `byte`, являющееся кодом символа x .

10.1. Использование модуля `CRT`

Процедуры и функции модуля `CRT` могут управлять работой клавиатуры, текстовым выводом на экран и звуком.

В разделе 3.2 уже говорилось об операторах `write` и `writeln`, которые предназначены для вывода информации на экран дисплея. Теперь более подробно рассмотрим текстовые режимы работы дисплея и управление цветом фона и выводимых на экран символов.

10.1.1. Основные процедуры и функции модуля `CRT`

Процедура `textmode` предназначена для задания одного из возможных текстовых режимов работы адаптера:

```
procedure textmode (Mode:word);
```

`Mode` – код текстового режима. В качестве его значения могут использоваться следующие константы, определенные в модуле `CRT`:

- `bw40=0`; {черно-белый, 40 символов в строке, 25 строк};
- `co40=1`; {цветной, 40 символов в строке, 25 строк};
- `bw80=2`; {черно-белый, 80 символов в строке, 25 строк};
- `co80=3`; {цветной, 80 символов в строке, 25 строк};
- `Mono=7`; {для MDA адаптеров};
- `Font8x8=256`; {цветной, 80 символов в строке, 43 строки для адаптеров EGA, 50 строк для адаптеров VGA}.

Код режима, установленного с помощью процедуры `textmode`, запоминается в глобальной переменной `lastmode` модуля `CRT` и может использоваться для начального состояния экрана.

В модуле `CRT` есть глобальная переменная `textattr` (типа `byte`). В ней хранятся текущие цветовые атрибуты мерцания и цвета символов, а также цвета фона. Структура этого байта следующая:

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

- 7-й бит – бит мерцания (1 – да, 0 – нет);
- 4–6 бит – цвет фона;
- 0–3 бит – цвет символов.

Исходя из этого, для фона предусмотрено 8 цветов, для символов – 16.

Переменную `textattr` можно использовать для управления цветовым режимом вывода символов на экран с помощью формулы:

$$\text{textattr} = \text{«цвет символов»} + 16 * \text{«цвет фона»} [+128].$$

Добавление 128 приводит к мерцанию символов. Для задания цвета фона и цвета символа можно использовать mnemonic константы, определенные в модуле CRT:

```
black=0;
blue=1;
green=2;
cyan=3;
red=4;
magenta=5;
brown=6;
lightgray=7;
darkgray=8;
lightblue=9;
lightgreen=10;
lightcyan=11;
lightred=12;
lightmagenta=13;
yellow=14;
white=15;
blink=128{мерцание}
```

Кроме того, для установления цвета фона и цвета выводимых символов в модуле CRT есть две процедуры: `textcolor` и `textbackground`.

```
procedure textcolor (color:byte);
```

определяет цвет выводимых символов.

```
procedure textbackground(color:byte);
```

определяет цвет фона.

В качестве переменной `color` могут использоваться рассмотренные ранее цветовые константы модуля CRT.

Процедура `window` определяет текстовое окно, которое в дальнейшем будет рассматриваться процедурами вывода как весь экран.

```
procedure window (x1, y1, x2, y2: byte);
```

`x1, y1; x2, y2` – координаты левого верхнего и правого нижнего углов окна. Границы текущего окна запоминаются в двух глобальных переменных модуля CRT: `WindMin` типа `word` хранит `x1` и `y1` (`x1` – в младшем байте), а переменная того же типа `WindMax` – `x2` и `y2` (`x2` – в младшем байте).

Турбо Паскаль предоставляет функции для работы над отдельными байтами машинных слов:

- `Hi(x)` – старший байт `x`, `Lo(x)` – младший байт `x`, `x` – переменная типа `word` или `integer`;

- `Xmin:=Lo(WindMin);`
- `Xmax:=Lo(WindMax);`
- `Ymin:=Hi(WindMin);`
- `Ymax:=Hi(WindMax);`

Каждое новое использование функции `window` отменяет действие предыдущей процедуры `window` (по умолчанию окном является весь экран).

Очистка экрана (или текущего окна) осуществляется с помощью процедуры `clrscr`. После обращения к ней окно (экран) заполняется цветом фона и курсор перемещается в левый верхний угол окна.

Процедура `gotoxy(x, y: byte)` переводит курсор к позиции экрана с координатами (x, y) . Координаты (x, y) задаются относительно границ экрана (окна).

Функции `wherex` и `wherey` возвращают горизонтальную (`wherex`) и вертикальную (`wherey`) координаты курсора (типа `byte`).

Процедура `clreoln` стирает часть строки от текущего положения курсора до правой границы экрана (окна).

Процедура `delline` уничтожает всю текущую строку.

Процедура `insline` вставляет строку, после чего строка с курсором и все, что ниже ее, сдвигаются вниз на одну позицию, а последняя строка теряется. Положение курсора не изменяется.

Процедуры `lowvideo`, `normvideo`, `highvideo` устанавливают соответственно пониженную, нормальную или повышенную яркость символов.

Управление клавиатурой реализовано в языке Турбо Паскаль всего двумя функциями `keypressed` и `readkey`. Несмотря на простоту этих двух функций, они предоставляют собой довольно мощные средства для работы с клавиатурой.

`Keypressed` – функция логического типа. Она описывает состояние буфера клавиатуры и принимает значение `true`, если в буфере есть хотя бы один символ, и `false`, если он пуст.

В MS DOS реализуется так называемый асинхронный ввод с клавиатуры. По мере нажатия клавиш соответствующие коды помещаются в буфер клавиатуры¹, откуда затем могут быть считаны процедурами `read` (`readln`) или функцией `readkey` [8].

Следовательно, процедуры `read`, `readln` и функция `readkey` работают непосредственно с клавиатурой, а с ее буфером.

При работе с подпрограммами `read`, `readln`, `readkey` и считывании с их помощью текстовых и символьных переменных необходимо помнить, что реально данные считываются из буфера. Коды нажатых клавиш будут помещаться в буфер. В таком случае возможно следующее:

- символ, возвращаемый из буфера, не совпадает с нажатой клавишей;
- переменная будет принимать значения из буфера, и при этом нет необходимости ввода данных.

¹ Буфер клавиатуры – участок оперативной памяти, в котором может храниться до 127 символов, вводимых с клавиатуры.

Чтобы избавиться от этого эффекта, нужно очистить буфер клавиатуры перед вводом с экрана дисплея строковых (символьных) переменных. Как это сделать, рассмотрим после знакомства с функцией `readkey`, которая возвращает значение типа `char`. При обращении к этой функции анализируется буфер клавиатуры: если в нем есть хотя бы один символ, то первый символ буфера и возвращается в качестве результата. В противном случае функция ожидает нажатия на любую клавишу.

Возникает вопрос, что вернет `readkey` в случае нажатия комбинаций клавиш **Alt+символ**, **Shift+символ**, **Ctrl+символ**. Алфавитно-цифровые клавиши, нажатые одновременно с клавишей **Alt**, функциональные клавиши в любом регистре, клавиши **↑**, **→**, **↓**, **←**, **Ins**, **Home**, **End**, **Delete**, **Page Up**, **Page Down** возвращают сразу два символа: первый – с кодом 0, второй – с цифровым кодом, характеризующим нажатую клавишу. Эти символы называются расширенным кодом. Часть кодов [8] возвращаемых функцией `readkey` значений приведена в табл. 10.1.

Таблица 10.1 ▼ Таблица кодов функциональных клавиш

Клавиша	Нормальное нажатие	+Shift	+Ctrl	+Alt
A-Z	65–90	97–123	1–26	
↑	0 72	56		8
→	0 77	54	0 116	6
↓	0 80	50		2
←	0 75	52	0 115	4
Ins	0 82	48		
Home	0 71	55	0 119	
End	0 79	49	0 117	1
Delete	0 83	46		
Page Up	0 73	57	0 132	9
Page Down	0 81	51	0 118	3
F1–F10	0 59–0 68	0 84–0 93	0 94–0 103	0 104–0 113
F11	0 133	0 135	0 137	0 139
F12	0 134	0 136	0 138	0 140

Рассмотрим ряд примеров использования модуля CRT.

ПРИМЕР 10.1. Считать символ с клавиатуры; если это клавиша **↑**, то переместить курсор на строку вверх, если **↓**, то, соответственно, вниз, если **→**, то переместить курсор на символ вправо, если **←**, то, следовательно, влево.

```
uses crt;
var
  c:char;
begin
  gotoxy(2,2);
  c:=readkey;
  { Проверяем, нажата ли клавиша с кодом 0. }
  if c = #0 then
  { Если нажата, то проверяем, какой второй символ. }
    case ord(readkey) of
```

```

{ Если это символ с кодом 72, то двигаем на одну строку вверх. }
72: gotoxy (wherex, wherey-1);
{ Если это символ с кодом 80, то двигаем на одну строку вниз. }
80: gotoxy (wherex, wherey+1);
{ Если это символ с кодом 75, то двигаем на один символ влево. }
75: gotoxy (wherex-1, wherey);
{ Если это символ с кодом 77, то двигаем на один символ вправо. }
77: gotoxy (wherex+1, wherey);
end
else writeln(c)
readln
end.

```

Приведем процедуру очистки буфера клавиатуры [8].

```

procedure clrbuf;
var
  c:char;
begin
{ Пока буфер занят, считывать его первый символ в переменную ch. }
while keypressed do
  c:=readkey
end;

```

Рассмотрим еще одну часто встречающуюся операцию:

```

{ Процедура wait - ожидание нажатия клавиши. }
procedure wait;
begin
repeat until keypressed
end;

```

В модуле CRT есть три процедуры, с помощью которых можно запрограммировать любую последовательность звуков.

Процедура `sound (f: word)`, где `f` – выражение типа `word`, определяющее частоту звука в герцах. Эта процедура заставляет звучать динамик с заданной частотой. После обращения к ней включается динамик, управление возвращается в основную программу. Звук будет включен впредь до вызова процедуры `nosound`, которая завершает работу динамика.

Процедура `delay (t: word)` обеспечивает задержку (приостановку) выполнения программы на `t` миллисекунд. При генерации любой мелодии эти процедуры вызываются по схеме `sound-delay-nosound`.

В приведенной ниже программе [8] вычисляется частота по заданной ноте и октаве:

```

var
  hz: word; okt; integer; nota: byte;
begin
  readln(okt,nota);
  hz:=round(440*exp(ln(2)*okt-(10-nota)/12));
end.

```

`okt` – номер октавы (-3, -2, ...4), (-3 – низкая, 4 – высокая), `nota` – номер ноты в октаве («до» – 1, «до-диез» – 2, ..., «си» – 12).

Рассмотрим пример восходящей и нисходящей хроматической гаммы.

ПРИМЕР 10.2.

```

program soundone;
uses crt;
var
  i, k: integer;
  hz: word;
begin
  { Восходящая хроматическая гамма. }
  for i:=-3 to 4 do
    for k:=1 to 12 do
      begin
        hz:=round(440*exp(ln(2)*i-(10-k)/12));
        sound(hz);
        delay(5000);
        nosound
      end;
    delay(10000);
  { Нисходящая хроматическая гамма. }
  for i:=4 downto - 3 do
    for k:=1 to 12 do
      begin
        hz:=round(440*exp(ln(2)*(i-(10-k)/12)));
        sound(hz);
        delay(5000);
        nosound
      end
    end;
  end.

```

10.1.2. Вывод псевдографики и спецсимволов

Оператор write (writeln) позволяет выводить любой символ (даже если его нет на клавиатуре) на экран. Для этого в операторе write необходимо набрать символ # и код требуемого символа, например writeln(#194#191). Для рисования таблиц могут понадобиться символы псевдографики, в табл. 10.2 приведены их коды.

Таблица 10.2 ▼ Коды символов псевдографики

Символ	Код
Г	218
┐	191
┌	217
└	192
┤	195
├	180
┘	194
└	193

Таблица 10.2 ▼ Коды символов псевдографики (окончание)

Символ	Код
—	196
	179



Для того чтобы отобразить символ на экране дисплея, можно поступить следующим образом: нажать клавишу **ALT** и, не отпуская ее, набрать код символа, затем отпустить клавишу **ALT**, после чего на экране дисплея появится необходимый символ.

Ниже приведена программа, которая выводит массивы *x* и *y* в виде следующей таблицы

x	y
12	34
45	176
23	78
...	...
2	909

```
uses crt;
var
  x, y: array [1..10] of integer;
  i: integer;
begin
  for i:=1 to 10 do
  begin
    write('x(',i,')='); readln(x[i])
  end;
  for i:=1 to 10 do
  begin
    write('y(',i,')='); readln(y[i])
  end;
  clrscr;
  write(#218);
  for i:=1 to 5 do
    write(#196);
  write(#194);
  for i:=1 to 5 do
    write(#196);
  writeln(#191); write(#179);
  write(' x ',#179,' y ');
  writeln(#179); write(#195);
  for i:=1 to 5 do
    write(#196);
  write(#197);
  for i:=1 to 5 do
    write(#196);
  writeln(#180);
  for i:=1 to 10 do
```

```
begin
  write(#179); write(' ',x[i]:3,' ');
  write(#179); write(' ',y[i]:3,' ');
  writeln(#179)
end;
write(#192);
for i:=1 to 5 do
  write(#196);
write(#193); '
for i:=1 to 5 do
  write(#196);
writeln(#217);
writeln('нажмите любую клавишу для окончания');
repeat until keypressed;
end.
```

10.2. Использование модуля PRINTER

После подключения модуля PRINTER в предложении uses можно выводить информацию не только на экран, но и на принтер. Для этого первым параметром в операторе write (writeln) необходимо указать lst, например:

```
writeln ('x=', x:1:3, ' y=', y:1:3);
writeln (lst, 'x=', x:1:3, ' y=',y:1:3);
```

Глава 11

Строки и записи в языке Турбо Паскаль

В данной главе кратко описана работа со строками и записями. Приведены примеры работы с файлами записей.

11.1. Строки в языке Турбо Паскаль

Строка – это последовательность символов, которая при использовании в выражениях заключается в апострофы. Количество символов в строке может изменяться динамически от 0 до 255. Для определения данных строкового типа используется идентификатор `string`, за которым следует заключенная в квадратные скобки длина строки. Если последняя не указывается специально, то по умолчанию она равна 255.

Например:

```
Var  
s:string[80];  
s2,s1:string[14];  
ps:string;
```

или

```
type  
stroka=string[85];  
var  
s1,s2:stroka;
```

Фактически строка представляет собой массив из $n+1$ символов.

```
string[n]=array [0..n] of char;
```

Нулевой символ предназначен для указания используемого количества символов строки: там хранится символ, код которого равен длине строки.

Для строк определены операции *конкатенации* (+) и *сравнения*.
Например:

```
Var
s1,s2,s3,s:string[80];
begin
s1:='turbo';
s2:='pascal';
s3:='7.0';
{ В переменной s хранится строка символов 'turbo pascal 7.0'. }
s:=s1+' '+s2+' '+s3;
end;
```

Сравнение строк производится слева направо до первого несовпадающего символа: длиннее считается та строка, в которой первый несовпадающий символ имеет больший номер в таблице кодов (буквы в таблице кодов расположены по возрастанию):

```
'abd'>'abc'
```

Если строки имеют различную длину, но в общей части символы совпадают, считается, что более короткая строка меньше:

```
'abcd'<'abcdef'
```

Обращение к элементу строки аналогично обращению к элементу массива: `s[6]`, `s[8]`, `s[10]`.

При вводе строковых переменных оператором `read` необходимо ввести количество символов, соответствующее описанной длине строки. При этом удобнее пользоваться оператором `readln`, который осуществляет ввод переменной `s` до конца строки. Обратите внимание на то, что при вводе текста пробел – это такой же символ, как и любой другой.

Над строками в Паскале определены следующие процедуры и функции.

Процедура `delete(st,pos,n)` удаляет из строки `st` (тип `string`) `n` (тип `integer`) символов, начиная с позиции `pos` (тип `integer`), например:

```
st:='turbo pascal 7.0';
delete(st,6,11);
```

Процедура `insert(s,subs,n)` вставляет в строку `s` (тип `string`) подстроку `subs` (тип `string`), начиная с позиции `n` (тип `integer`), например:

```
s1:='pascal';
s:='turbo 7.0';
insert(s,s1,7);
```

Функция `length(s)` возвращает текущую длину (тип `integer`) строки `s` (тип `string`).

Функция `copy(s,n,l)` возвращает подстроку длиной `l` (тип `integer`), начиная с позиции `n` (тип `integer`) строки `s` (тип `string`).

Процедура `str(x[:f] [:n],s)` преобразует числовое значение `x` (тип `real`) в строку `s` (возможно задание формата, в этом случае `f` – число позиций в числе, `n` – количество позиций после точки).

Процедура `val(s, x, err)` превращает строковое значение `s` (тип `string`) в числовую переменную `x` (тип `real`), `err` (тип `integer`) возвращает номер позиции, в которой произошла ошибка преобразования, или 0, если ошибки не было.

11.2. Работа с записями

Запись – это структурированный тип данных, состоящий из фиксированного числа компонентов. Эти компоненты называются *полями записи*. Определение типа записи начинается идентификатором `record`, затем идет список полей с указанием их типов, а заканчивается описание служебным словом `end`. Например:

```
Type
abc=record
a,b:string[15];
c:real;
d:real;
ef:array [1..20] of byte;
end;
var
skl: abc;
```

Для того чтобы обратиться к полю записи, необходимо указать имя переменной и через точку – имя поля.

Например:

```
skl.c:=skl.d+17;
```

Обращение к полям имеет громоздкий вид, и, если работать с полями записи, постоянно указывать имя переменной и поля нерационально. Для решения этой проблемы предназначен оператор

```
with s do <операторы>;
```

где `s` – переменная типа «запись». Внутри оператора с полями записи можно работать как с обычными переменными (то есть без указания составного имени). Например:

```
with skl do
c:=d+17;
with skl do
begin
c:=c+1;
d:=d+0.1;
end;
```

В Паскале существует возможность задавать тип записи, содержащий произвольное число вариантов структуры. Такие записи называются записями с вариантами. Например:

```
Type
complex=record
```

```
re,im:real;
end;
urav=record
a,b,c:real;
case pr: byte of
0:(x1,x2:real);
1:(x:real);
2:(y1,y2:complex);
end;
var
abc:urav;
```

Обработка отдельных полей определенных типов осуществляется так же, как и переменных это типа.

Ранее было сказано, что записи часто используют для обработки таблиц. Но запись – это одна строка таблицы. Для того чтобы обработать всю таблицу, необходимо использовать массивы записей, например:

```
Type
ved=record
fio:string[35];
date:integer;
zarhlata:real;
end;
var a:array [1..30] of ved;
```

Для обращения к какому-то полю *i*-го элемента таблицы необходимо обратиться к этому полю *i*-го элемента массива `a[i].fio`.

Рассмотрим несколько примеров задач обработки таблиц.

ПРИМЕР 11.1.

О каждом студенте известна следующая информация:

- фамилия, инициалы;
- год рождения;
- группа;
- оценка по математике;
- оценка по истории;
- оценка по ВТ;
- оценка по статистике.

Сформируйте таблицу, записав в нее всю известную информацию о каждом студенте и его средний балл. Подсчитайте средний балл по каждому предмету, выведите таблицу на экран дисплея в алфавитном порядке.

```
{ Подключение модуля CRT. }
uses CRT;
{ Описание записи о каждом студенте. }
type
tablica=record
name:string[15];
group:string[8];
god:integer;
```

```

vt,history,stat,math:byte;
sr_bal:real;
end;
var
  i,j,n:integer; a:tablica;
  { Таблица - массив записей. }
mas:array [1..30] of tablica;
  { Переменные для хранения средних значений по предметам. }
s_vt, s_history, s_stat, s_math:real;
begin
  { Ввод количества n записей. }
write('n='); readln(n);
  { Ввод элементов в массив записей. }
for i:=1 to n do
  with mas[i] do
    begin
      writeln('i=',i:4);
      writeln('FIO');
      readln(name);
      write('Group');
      readln(group);
      write('Year');
      readln(god);
      write('Otsenki');
      readln(vt,history,stat,math);
      sr_bal:=(vt+history+stat+math)/4;
    end;
  { Вычисление средних значений по каждому предмету. }
s_vt:=0; s_history:=0; s_stat:=0; s_math:=0;
for i:=1 to n do
begin
  s_vt:=s_vt+mas[i].vt;
  s_history:=s_history+mas[i].history;
  s_stat:=s_stat+mas[i].stat;
  s_math:=s_math+mas[i].math;
end;
s_vt:=s_vt/n;
s_history:=s_history/n;
s_stat:=s_stat/n;
s_math:=s_math/n;
  { Упорядочение массива записей в алфавитном порядке фамилий. }
for i:=1 to n-1 do
  for j:=1 to n-i do
    if mas[j].name>mas[j+1].name then
      begin
        a:=mas[j];
        mas[j]:=mas[j+1];
        mas[j+1]:=a;
      end;
  { Вывод результатов. }
clrscr;
write(' ':4,'FIO  ',' ':4);
write(' ':2,'GROUP ',' ':2);

```

```

write(' ':5,'OTSENKI',' ':4);
writeln('Sr. Bal');
for i:=1 to n do
  with mas[i] do
    begin
      write(name:15);
      write(' ',group:8);
      write(' ',god:4);
      writeln(' ',vt:3,' ',history:3,' ',stat:3,' ',math:3,
        ' ',sr_bal:5:2);
    end;
  writeln(' ', 'Sr. Bal:', ' ':17, s_vt:3:1, ' ', s_history:3:1,
    s_stat:3:1, ' ', s_math:3:1);
end.

```

ПРИМЕР 11.2.

В файле записей steel.dat хранится информация о химическом составе различных сталей, представленная в табл. 11.1.

Таблица 11.1 ▼ Химический состав различных сталей

Марка стали	Содержание элементов								
	C	Cr	Ni	Mn	Si	P	S	Ti	N
X17T	1	17		1	1	0,02	0,02		1
30XTCA	0,3	1		1	1	0,01	0,01		1
...

Вывести на экран содержимое файла. Компоненты файла расположить в порядке возрастания концентрации Cr. На их основе сформировать два файла, в один из которых поместить информацию о сталях, включающих хром (хромированные стали), а во второй – данные о сталях, содержащих титан и меньше 0,3% углерода. Информацию о сталях, которые вошли в оба созданных файла, удалить из файла steel.dat. Вывести на экран данные о сталях, в состав которых входит марганец. Попробуйте самостоятельно разобраться с приведенным текстом программы и дополните его выходными данными в виде таблиц с использованием символов псевдографики.

```

{ Тип steel - запись для хранения информации о типах стали. }
type steel=record
  name:string[10];
  c,cr,ni,mn,si,p,s,ti,n:real;
end;
var
  b:steel;
  { f - исходный файл; f1,f2 - выходные файлы }
  f,f1,f2:file of steel;
  a:array [1..50] of steel;
  i,j,k,n:integer;
  fl:boolean;
  s: array [1..50] of string;
begin
  assign(f,'steel.dat'); reset(f);

```

```

assign(f1,'cr.dat');
assign(f,'ti.dat');
i:=1;
{ Считывание содержимого файла в массив a. }
while not eof(f) do
begin
    read(f,a[i]);
    i:=i+1;
end;
close(f);
rewrite(f1);
rewrite(f2);
fl:=false;
k:=0;
for n:=1 to i-1 do
begin
    if a[n].cr>0 then
    begin
        write(f1,a[n]);
        fl:=true;
    end;
    if fl then
    if (a[n].ti>0) and (a[n].c<0.3) then
    begin
        fl:=false;
        write(f2,a[n]);
    { В массиве s список сталей, попавших в оба файла. }
        s[k+1]:=a[n].name;
        k:=k+1;
    end;
    if not fl then
    if (a[n].ti>0) and (a[n].c<0.3) then
        write(f2,a[n]);
    end;
close(f1); close(f2);
{ Упорядочение таблицы сталей в порядке возрастания концентраций хрома. }
for n:=1 to i-2 do
    for j:=1 to i-2-n do
        if a[j].cr>a[j+1].cr then
        begin
            b:=a[j];
            a[j]:=a[j+1];
            a[j+1]:=b;
        end;
{ Вывод упорядоченного массива сталей. }
for n:=1 to i-1 do
    with a[n] do
    begin
        write(name:10);
        write(c:6:2);
        write(cr:6:2);
        write(ni:6:2);
        write(mn:6:2);
    end;
end;

```

```
        write(si:6:2);
        write(p:6:2);
        write(ti:6:2);
        write(n:6:2);
    end;
    rewrite(f);
    for n:=1 to i-1 do
        with a[n] do
            begin
                fl:=false;
                { Цикл по j проверяет, встретилась ли текущая сталь в массиве s. }
                for j:=1 to k do
                    if name=s[j] then fl:=true;
                { Если текущей стали нет в массиве s, то записать ее в исходный файл f. }
                if not fl then write(f,a[n]);
            end;
        close(f);
    { Печать списка сталей, в которых есть марганец. }
    for n:=1 to i-1 do
        with a[n] do
            begin
                if mn>0 then
                    begin
                        write(name:10);
                        write(c:6:2);
                        write(cr:6:2);
                        write(ni:6:2);
                        write(mn:6:2);
                        write(si:6:2);
                        write(p:6:2);
                        write(ti:6:2);
                        write(n:6:2);
                    end;
            end;
    end.
```

11.3. Упражнения по теме «Строки в языке Турбо Паскаль»

Дана строка до точки, группа символов в которой между пробелами считается словом:

1. Подсчитать количество слов в строке.
2. Дано слово. Удалить его из строки.
3. Приведено некоторое число. Вставить его после четвертого слова.
4. Найти сумму цифр, встречающихся в строке.
5. Удалить из строки все числа.
6. Удалить из строки третье слово после первой запятой.
7. Определить, сколько слов заканчивается буквой «о».

8. Вывести на печать подстроку, заключенную между двумя запятыми.
9. Подсчитать, сколько слов длиной не более пяти символов содержит данная строка.
10. Найти самое короткое слово.

11.4. Упражнения по теме «Обработка записей»

При выполнении заданий предусмотреть:

- создание файла данных;
- вывод на экран и печать файла данных;
- выполнение задания, совпадающего с номером варианта;
- вывод на экран и печать результирующего файла.

Создать файл записей `steel.dat` с информацией о химическом составе различных сталей (см. табл. 11.1). Вывести на экран содержимое файла.

Создать новый файл `steel_1.dat`, куда поочередно поместить компоненты исходного файла:

- выстроенные по возрастанию концентрации углерода;
- упорядоченные по убыванию концентрации серы;
- с информацией о сталях, в состав которых входит хром, но не входит никель;
- с данными о сталях, содержащих азот и углерод не более 0,3%.
- с информацией о сталях, содержание фосфора в которых максимально.

Удалить из исходного файла следующие компоненты:

- с информацией о сталях, содержание фосфора в которых минимально.
- с данными о сталях, в состав которых не входит сера и марганец.
- с информацией о сталях, в которых одинаковое количество никеля, титана и фосфора;
- с информацией о сталях, содержащих все указанные в таблице компоненты.
- с данными о сталях, в которых содержится не более трех указанных в таблице компонентов.

12 Глава

Графические средства Турбо Паскаля

Начиная с версии 4.0 в состав Турбо Паскаля включен графический модуль GRAPH, который содержит в общей сложности более 50 процедур и функций, предоставляющих программисту различные возможности управления графическим экраном.

Стандартный режим работы дисплея ПК под управлением DOS – текстовый, поэтому любая программа, использующая графические средства, должна переключить экран в графический режим. После завершения работы графической части программы можно вернуть дисплей ПК в текстовый режим.

12.1. Краткая характеристика графических режимов

Настройка графических процедур на работу с конкретным адаптером достигается за счет подключения нужного графического драйвера. *Драйвер* – специальная программа, осуществляющая управление теми или иными техническими средствами ПК. Графический драйвер управляет дисплейным адаптером в графическом режиме. Графические драйверы разработаны фирмой Borland практически для всех типов адаптеров. Они располагаются на диске в виде файлов с расширением bgi (от английского Borland Graphics Interface), например cga.bgi – драйвер для CGA-адаптера (дисплея), egavga.bgi – драйвер для адаптеров EGA и VGA. Графические возможности конкретного адаптера определяются расширением экрана, то есть общим количеством точек, а также количеством цветов, которым может светиться любая из них. Кроме того, многие адаптеры могут работать с несколькими графическими страницами – областями оперативной памяти, используемыми для создания «карты» экрана, то есть

содержащими информацию о светимости (цвете) каждой точки. Рассмотрим графические режимы под управлением DOS.

Адаптер CGA (Color Graphics Adapter – цветной графический адаптер) имеет пять графических режимов. Четыре из них соответствуют низкой разрешающей способности экрана (320 точек по горизонтали и 200 точек по вертикали, то есть 320×200) и отличаются только набором допустимых цветов – палитрой. Каждая палитра состоит из трех цветов, а с учетом несветящегося цвета, – из четырех. Пятый графический режим соответствует разрешающей способности экрана 640×200, палитра в этом случае состоит из единственного цвета, а с учетом несветящегося символа в этом режиме доступны два цвета. Таким образом, пять графических режимов для адаптера CGA представляют собой следующее:

- режим 1 – разрешающая способность 320×200, палитра 0 (светло-зеленый, розовый, желтый);
- режим 2 – разрешающая способность 320×200, палитра 1 (светло-бирюзовый, малиновый, белый);
- режим 3 – разрешающая способность 320×200, палитра 2 (зеленый, красный, коричневый);
- режим 4 – разрешающая способность 320×200, палитра 3 (бирюзовый, фиолетовый, светло-серый);
- режим 5 соответствует высокому разрешению 640×200, но каждая точка в этом случае может светиться либо каким-то заранее определенным и одинаковым для всех точек цветом, либо не светиться вовсе, то есть палитра данного режима содержит два цвета.

В графическом режиме адаптер CGA использует одну страницу.

Адаптер EGA (Enhanced Graphics Adapter – усиленный графический адаптер) поддерживает все режимы адаптера CGA. Кроме того, в нем возможны режимы низкого разрешения (640×200, 16 цветов, 4 страницы) и высокого разрешения (640×350, 16 цветов, 2 страницы).

Адаптер MCGA (Multi-Color Graphics Adapter – многоцветный графический адаптер) совместим с CGA и имеет еще один режим (640×480, 2 цвета, 1 страница).

Адаптер VGA (Video Graphics Array – графический видеомассив) поддерживает режимы адаптеров CGA и EGA и дополняет их режимом высокого разрешения (640×480, 16 цветов, 1 страница).

Адаптер SVGA (SuperVGA) с разрешением 1024×768 использует 256 цветов.

Адаптеры фирмы Hercules (адаптер HGC) имеет разрешение 720×348 и два цвета.

12.2. Управление графическими режимами

Процедура `InitGraph` инициализирует графический режим работы адаптера (переводит экран в графический режим). Заголовок процедуры:

InitGraph(var grdr, grmd:integer, path:string);

Параметр `grdr` определяет тип графического адаптера. В качестве него могут использоваться константы, определенные модуле GRAPH:

```
DETECT=0; – режим автоматического определения
CGA=1;
MCGA=2;
EGA=3;
EGA64=4;
EGAMONO=5;
IBM8514=6;
HERCMONO=7;
ATT400=8;
VGA=9;
PC3270=10.
```

Если параметру `grdr` присвоить значение `detect`, то система переходит в режим самоопределения. При возможном переключении в графический режим система переходит в него с максимальным разрешением.

Параметр `grmd` – номер (значение) режима, допустимого при данном адаптере. Рассмотрим допустимые режимы для адаптеров CGA, EGA и VGA.

Адаптер CGA допускает пять графических режимов:

- CGAC0=0 320×200, 4 цвета, палитра 0;
- CGAC1=1 320×200, 4 цвета, палитра 1;
- CGAC2=2 320×200, 4 цвета, палитра 2;
- CGAC3=3 320×200, 4 цвета, палитра 3;
- CGAHi=4 640×200, 2 цвета.

Адаптер EGA поддерживает два режима:

- EGALo=0 640×200, 16 цветов, 4 страницы;
- EGAHi=1 640×350, 16 цветов, 2 страницы.

Адаптер VGA поддерживает три режима:

- VGALo=0 640×200, 16 цветов, 4 страницы;
- VGAMed=1 640×350, 16 цветов, 2 страницы;
- VGAHi=2 640×480, 16 цветов, 1 страница.

Третий параметр `pathstr` – строковая константа или переменная, которая указывает путь в каталог, где находится драйвер (файл с расширением `bgi`).

Если все параметры указаны верно (и, кроме того, подключен модуль GRAPH), то экран дисплея переключится в графический режим.

Процедура `InitGraph` возвращает два значения `grdr`, `grmd`. Если значение `grdr` определено как `detect`, то `InitGraph` вернет конкретные значения `grdr` и `grmd`. Только после того, как система перейдет в графический режим, можно пользоваться всеми функциями и процедурами модуля GRAPH (рисование линий, окружностей и т.д.). И еще одно замечание по поводу параметра `pathstr`.

Система сначала ищет графический драйвер в текущем каталоге, а затем в том, что указан в `pathstr`. Следовательно, если поместить драйвер в текущий каталог, то проблема его поиска отпадает.

Процедура без параметров `CloseGraph` завершает работу в графическом режиме и переводит компьютер в текстовый режим.

В случае неудачного завершения инициализации процедура `InitGraph` возвращает ошибку в параметре `grdr`. В случае ошибки параметр `grdr` отрицательный и может принимать следующие значения (см. табл. 12.1).

Таблица 12.1. Значение ошибок в параметре `grdr`

Значение	Причина ошибки
-2	Нет графического адаптера
-3	Не найден файл драйвера
-4	Ошибка в файле (его коде)
-5	Не хватает памяти для загрузки драйвера
-10	Невозможный режим для выбранного драйвера
-15	Нет такого драйвера

В модуле `GRAPH` есть функция `GraphResult`, которая тоже возвращает код результата (тип `integer`) последнего вызова одной из графических функций. Если `GraphResult` меньше 0, то произошла ошибка (`GraphResult` возвращает 0, если ошибки не было). Для быстрой выдачи простого сообщения о типе ошибки графической системы используется функция, преобразующая результат вызова `GraphResult` в сообщение, которое можно вывести на экран процедурой `Write` или `OutTextXY`. Эта функция объявлена так:

```
GraphErrorMsg(ErrorCode):String
```

Константы ошибок выглядят следующим образом:

- `grok = 0` – нет ошибок;
- `grInitGraph = -1` – не инициирован графический режим;
- `grNotDetected = -2` – не определен тип драйвера;
- `grFileNotFind = -3` – не найден графический драйвер;
- `grInvalidDriver = -4` – неправильный тип драйвера;
- `grNoLoadMem = -5` – нет памяти для размещения драйвера;
- `grNoScanMem = -6` – нет памяти для просмотра областей;
- `grNoFloodMem = -7` – нет памяти для закраски областей;
- `grFontNotFound = -8` – не найден файл со шрифтом;
- `grNoFontMem = -9` – нет памяти для размещения шрифта;
- `grInvalidMode = -10` – неправильный графический режим;
- `grError = -11` – общая ошибка;
- `grIOError = -12` – ошибка ввода-вывода;
- `grInvalidFont = -13` – неправильный формат шрифта;
- `grInvalidFontNum = -14` – неправильный номер шрифта.

Процедура `RestoreCRTMode` служит для кратковременного возврата в текстовый режим. В отличие от процедуры `CloseGraph`, в этом случае параметры

графического режима не сбрасываются и память, выделенная для размещения графического драйвера, не освобождается.

Функция `GetGraphMode` возвращает значение типа `integer`, в котором содержится код установленного режима работы графического адаптера.

Процедура `SetGraphMode` устанавливает новый графический режим работы адаптера. Обращение к процедуре:

```
SetGraphMode (Mode) ;
```

`Mode` – переменная типа `integer`, в которой содержится код устанавливаемого режима.

Представляет интерес процедура определения типа графического адаптера и его режима, доступная до переключения экрана в графический режим. Обращение к процедуре `DetectGraph` имеет вид:

```
Detectgraph (Drv, Md) ;
```

где `Drv` – переменная типа `integer`, возвращающая тип адаптера; `Md` – переменная типа `integer`, возвращающая максимально возможный режим для данного адаптера.

Процедура `GetModeRange` возвращает диапазон возможных режимов работы заданного графического адаптера. Обращение к процедуре имеет вид:

```
GetModeRange (Drv, Min, Max) ;
```

`Drv` – переменная типа `integer`, в которой передается в процедуру тип графического адаптера. `Min`, `Max` – переменные типа `integer`, в которые возвращаются нижнее и верхнее из возможных значений режима. Если задано неправильное значение параметра `Drv`, то процедура в переменных `Min` и `Max` вернет значение `-1`.

ПРИМЕР 12.1. Рассмотрим следующую программу, демонстрирующую возможности переключения между текстовым и графическим режимами, а также анализ корректности инициализации последнего.

```
Uses Crt, Graph;
var GrDr, GrMd, Error: integer;
Begin
  GrDr := Detect;
  { Переход в графический режим. }
  InitGraph(GrDr, GrMd, 'd:\tp7');
  { В переменную Error записывается код ошибки после оператора InitGraph }
  { (ошибка инициализации графики). }
  Error := GraphResult;
  if Error <> grOk then
  Begin
    { Если была ошибка, выводится стандартное сообщение о ней }
    Writeln(GraphErrorMsg(error));
    { и прекращается выполнение программы. }
    Halt;
  End;
  { Если ошибки инициализации не было, то оператор будет выполняться. }
  OutText ('this is graph mode');
```

```

{ Вывод текста в графическом режиме. }
Delay(2000); { Задержка на две секунды. }
RestoreCRTMode; { Переключение в текстовый режим. }
Writeln('а это текстовый режим');
Delay(3000);
SetGraphMode(GetGraphMode); { Возврат в графический режим. }
OutText ('This is graph mode');
repeat until keypressed; { Ожидание нажатия клавиши. }
CloseGraph;
End.

```

12.3. Некоторые графические процедуры и функции

После перехода дисплея в графический режим система координат определена следующим образом (см. рис. 12.1):

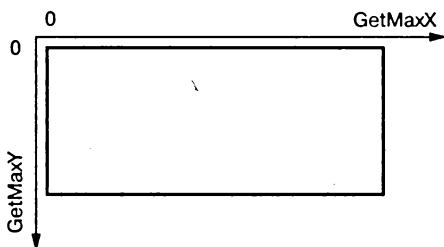


Рис. 12.1▼ Система координат в графическом режиме

В отличие от текстового режима, единицей информации в графическом режиме является точка. Система координат в графическом режиме расположена следующим образом: начало координат (точка с координатами 0,0) находится в левом верхнем углу, ось ОХ направлена слева направо (от 0 до GetMaxX), ось ОУ направлена сверху вниз (от 0 до GetMaxY). Функции GetMaxX и GetMaxY возвращают значения типа word, содержащие максимальные координаты экрана в текущем режиме работы, по горизонтали и по вертикали соответственно.

Функции GetX и GetY возвращают значения типа integer, содержащие текущие координаты графического курсора, по горизонтали и вертикали соответственно.

Процедура MoveTo устанавливает новое текущее значение графического курсора. Обращение к ней имеет вид:

```
MoveTo(X,Y),
```

где x, y – переменные типа integer с новыми координатами графического курсора по горизонтали и вертикали.

Процедура `MoveRel` устанавливает новое положение графического курсора в относительных координатах. Обращение к ней имеет вид:

```
MoveRel(dx,dy),
```

где `dx`, `dy` – переменные типа `integer`, которые содержат приращение координат графического курсора по горизонтали и вертикали относительно его текущего положения.

Процедура `SetColor` устанавливает текущий цвет выводимых линий и символов. Обращение к ней имеет вид:

```
SetColor(color),
```

где `color` – переменная типа `word`, в которой задается цвет. Константы задания цвета в модуле `GRAPH` определены точно так же, как и в модуле `CRT`.

Процедура `SetBkColor` устанавливает цвет фона. Обращение к ней выглядит так:

```
SetBkColor(color),
```

где `color` – переменная типа `word`, определяющая цвет фона.

В отличие от цвета фона в текстовом режиме, в графическом режиме он может быть любым. Установка нового цвета фона немедленно изменяет цвет графического экрана. Нельзя создать изображение двух участков, которые имеют разный цвет фона. На работе с графикой в режиме адаптера `CGA` есть сложности со сменой цвета фона: в частности, данная процедура не рекомендуется по той причине, что это изменяет и цвет активных точек.

Процедура `PutPixel` выводит точку с заданным цветом. Обращение к ней таково:

```
PutPixel(x,y,color),
```

где `x`, `y` – константы или переменные типа `integer`, задающие координаты точки.

Процедура `Line` рисует линию с указанными координатами начала и конца. Обращение к процедуре имеет вид:

```
line(x1, y1, x2, y2),
```

где `x1`, `y1`, `x2`, `y2` – переменные или константы типа `integer`, (`x1`, `y1`) – координаты начала линии, (`x2`, `y2`) – координаты конца линии. Линия рисуется текущим цветом.

Процедура `SetViewport` устанавливает прямоугольное окно на графическом экране. Обращение к ней выглядит так:

```
SetViewport(x1,y1,x2,y2,clip),
```

где `x1`, `y1`, `x2`, `y2` – переменные типа `integer`, содержащие координаты левого верхнего (`x1`, `y1`) и правого нижнего (`x2`, `y2`) углов окна. `Clip` – переменная или выражение типа `boolean`, определяющая удаление не вмещающихся элементов изображения. Если `Clip=True`, то последние отсекаются; в противном случае границы окна игнорируются.

Процедура `ClearDevice` очищает графический экран цветом фона, графический курсор устанавливается в начало координат (левый верхний угол экрана).

Процедура `ClearViewport` очищает графическое окно, а если оно было не определено – весь экран.

Для изображения окружности используется процедура `Circle`. Обращение к ней имеет вид:

```
Circle(x, y, r),
```

где x, y – выражения, константы или переменные типа `integer`, определяющие координаты центра окружности, а r – выражение типа `word` – радиус окружности.

Более точно параметр r определяет количество точек в горизонтальном направлении, в вертикальном же эта величина находится с учетом соотношения сторон экрана. Окружность всегда рисуется правильная, если не пользоваться процедурой `SetAspectRatio`, которая устанавливает масштабный коэффициент соотношения сторон графического экрана. Обращение к ней имеет вид:

```
SetAspectRatio(x,y),
```

где x, y – выражения типа `word`, которые устанавливают масштабные соотношения сторон.

Процедура `GetAspectRatio` возвращает два числа, позволяющие оценить соотношение сторон экрана. Обращение выглядит так:

```
GetAspectRatio(x, y),
```

где x, y – выражения типа `word`. Значения, возвращаемые в этих переменных, позволяют вычислить соотношения сторон графического экрана в точках. Коэффициенты x/y или y/x позволяют строить правильные геометрические фигуры (квадраты, окружности).

Процедура `Rectangle` предназначена для рисования прямоугольника. Обращение к ней таково:

```
Rectangle(x1, y1, x2, y2),
```

где $x1, y1, x2, y2$ – выражения типа `integer`, определяющие координаты верхнего левого и правого нижнего углов экрана.

Рассмотрим теперь несколько примеров, в которых используются описанные ранее процедуры.

ПРИМЕР 12.2. Пример работы с процедурой `SetViewport`.

```
program gr_one;
uses Crt, Graph;
Var x, y, l, x11, x12, y11, y12, x21, x22, r, k: integer;
begin
x:=Detect;
InitGraph(x, y, 'C:\bp\Bgi');
SetBkColor(White);
SetColor(Blue);
```



```

l:=GraphResult;
if l<>grOk then
  writeln(GraphErrorMsg(l))
else
  begin
    x11:=GetMaxX div 60;
    x12:=GetMaxX div 3;
    y11:=GetMaxY div 4;
    y12:=2*y11;
    r:=(x12-x11)div 4;
    x21:=x12*2;
    x22:=x21+x12-x11;
    Rectangle(x11,y11,x12,y12);
    Rectangle(x21,y11,x22,y12);
    SetViewport(x11,y11,x12,y12,ClipOn);
    for k:=1 to 4 do Circle(0,y11,r*k);
    SetViewport(x21,y11,x22,y12,ClipOff);
    for k:=1 to 4 do Circle(0,y11,r*k);
  repeat until keypressed
end;
end.

```

До нажатия клавиши на экране появится картинка, изображенная на рис. 12.2.

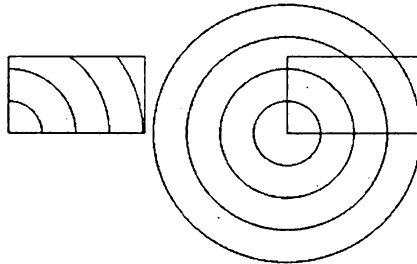


Рис. 12.2 ▼ Результат программы 12.2

ПРИМЕР 12.3. Пример использования SetAspectRatio [8,9].

```

program gr_three;
uses Graph,Crt;
const r=50;
    dx=1000;
var l1,xa,ya:word;
d,l,m,k:integer;
begin
d:=Detect;
InitGraph(d,m,'c:\bp\bgi');
l:=GraphResult;
if l<>GrOk then
begin
writeln(GraphErrorMsg(l));

```

```

exit;
end;
SetBkColor(White);
SetColor(Blue);
GetAspectRatio(xa,ya);
for k:=1 to 26 do
begin
  l1:=xa+k*dx;
  SetAspectRatio(l1,ya);
  circle(GetMaxX div 2, GetMaxY div 2,r);
end;
repeat until keypressed;
CloseGraph;
end.

```

Результат работы программы представлен на рис. 12.3.

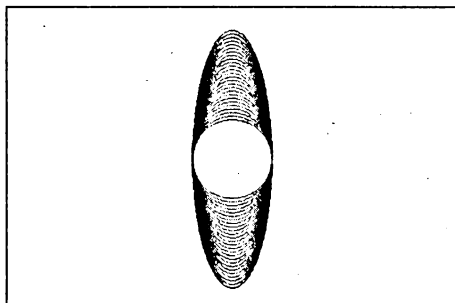


Рис. 12.3 ▼ Результат примера 12.3

Рассмотрим еще несколько процедур и функций для работы с точками и линиями.

Функция `GetPixel` возвращает значение типа `word`, содержащее цвет точки с указанными координатами. Ее заголовок имеет вид:

```
Function GetPixel(x,y:integer):word,
```

где x, y – координаты точки.

Процедура `LineTo` вычерчивает линию от текущего положения графического курсора до положения, заданного его новыми координатами. Обращение к процедуре выглядит так:

```
LineTo(x, y),
```

где x, y – выражения типа `integer`, указывающие координаты конца линии.

Процедура `LineRel` вычерчивает линию от текущего местонахождения графического курсора до положения, заданного приращениями его координат. Обращение к процедуре имеет вид:

```
LineRel(dx, dy),
```

где dx , dy – выражения типа `integer`, задающие приращение координат для нового положения указателя. В процедурах рисования линий они изображаются текущими цветом и стилем.

Процедура `SetLineStyle` устанавливает новый стиль вычерчиваемых линий. Вызвать ее можно следующим образом:

```
SetLineStyle(type,pattern,thick),
```

где `type`, `pattern`, `thick` – выражения, переменные или константы типа `word`, определяющие тип, образец и толщину линии.

Тип линии задан с помощью одной из следующих констант модуля `GRAPH`:

- `Solidln = 0` – сплошная линия;
- `Dottedln = 1` – точечная линия;
- `Centernln = 2` – штрих-пунктирная линия;
- `Dashedln = 3` – пунктирная линия;
- `Userbitln = 4` – линия, определенная пользователем.

Параметр `Pattern` учитывается только для линий, определенных пользователем. Он представляет собой 16 бит. Если бит = 1, то соответствующая точка у образца светящаяся, если 0 – несветящаяся.

Параметр `Thick` может принимать значение одной из двух констант модуля `GRAPH`:

- `Normwidth = 1` – тонкая линия (1 точка);
- `Thickwidth = 3` – толстая линия (3 точки).

ПРИМЕР 12.4. Программа рисования звездного неба [8].

```
program gr_four;uses crt,Graph;
const n=15000;
var d,e,r,k:integer;
    l,x1,y1,x2,y2:integer;
    x,y:array [1..n] of integer;
begin
  d:=detect;
  InitGraph(d,r,'c:\bp\bgi');
  l:=GraphResult;
  if l<>GrOk then
  begin
    writeln(GraphErrorMsg(l));
    exit;
  end;
  x1:=0;
  y1:=0;
  x2:=GetmaxX-2;
  y2:=GetMaxY-2;
  Rectangle(x1,y1,x2,y2);
  SetViewport(x1+1,y1+1,x2-1,y2-1,ClipOn);
  for k:=1 to n do
  begin
```

```

x[k]:=random(x2-x1);
y[k]:=random(y2-y1);
end;
for k:=1 to n do
  PutPixel(x[k],y[k],random(7));
repeat until keypressed;
CloseGraphend.

```

Результат работы программы представлен на рис. 12.4.

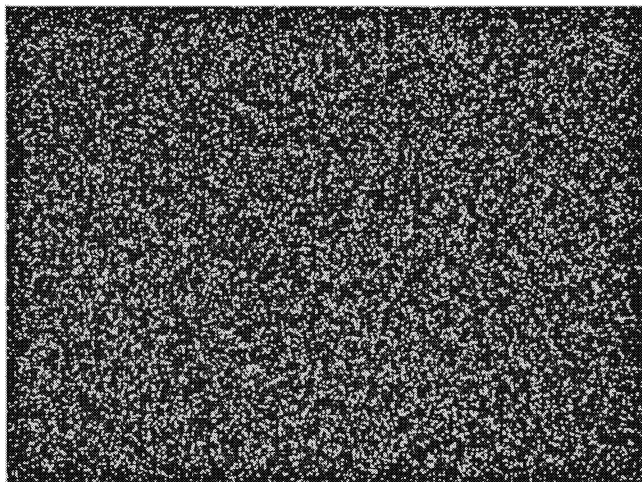


Рис. 12.4 ▼ Результат работы программы рисования звездного неба

Процедура `Arc` чертит дугу окружности. Обращение к ней имеет вид:

```
Arc(x,y,begA,endA,r),
```

где `x, y` – переменные или выражения типа `integer`, определяющие координаты центра. `BegA, EndA, r` – переменные или выражения типа `word`, определяющие начальный и конечный углы дуги и ее радиус. Углы отсчитываются против часовой стрелки и в градусах от 0 до 359.

Процедура `Ellipse` вычерчивает эллиптическую дугу. Обращение к процедуре имеет вид:

```
Ellipse(x, y, BegA, EndA, rx, ry),
```

где `x, y` – переменные или выражения типа `integer`, содержащие координаты центра, `BegA, EndA` – значения типа `word`, определяющие начальный и конечный углы дуги, `rx, ry` – переменные типа `word`, в которых записаны данные горизонтального и вертикального радиуса эллипса в точках.

Процедура `SetFillStyle` устанавливает тип и цвет заполнения, с помощью которого можно покрывать какие-либо фрагменты изображения периодически повторяющимся узором. Обращение к процедуре имеет вид:

```
SetFillStyle(fill,color),
```

где *fill*, *color* – переменные типа *word*, определяющие тип и цвет заполнения, в качестве типа которого иногда используется одна из констант модуля **GRAPH**:

- *Emptyfill* = 0 – заполнение фоном;
- *Solidfill* = 1 – сплошное заполнение;
- *Linefill* = 2 – заполнение ———;
- *Ltslashfill* = 3 – заполнение ///\\\\\\\\;
- *Slashfill* = 4 – заполнение утолщенными \\\\\\\\\\\\\;
- *Bkslashfill* = 5 – заполнение утолщенными \\\\\\\\\\\\\;
- *Ltbkslash* = 6 – заполнение \\\\\\\\\\\\\;
- *Hatchfill* = 7 – заполнение ++++++;
- *Xhatchfill* = 8 – заполнение XXXXXXXX;
- *Centerleavefill* = 9 – заполнение в прямоугольную клетку;
- *Widedotfill* = 10 – заполнение редкими точками;
- *Closedotfill* = 11 – заполнение частыми точками;
- *Userfill* = 12 – узор заполнения определяется пользователем.

В качестве *Color* можно взять одну из доступных констант. Если установлен тип заполнения *Userfill*, то для его определения необходимо использовать процедуру *SetFillPattern* [7–9].

Процедура *Floodfill* заполняет произвольную замкнутую фигуру, используя текущий стиль заполнения (узор и цвет). Обращение к процедуре имеет вид:

```
FloodFill (x, y, border),
```

где *x*, *y* – переменные типа *integer*, в которых хранятся координаты любой точки внутри замкнутой фигуры, *border* – цвет пограничной линии.

Процедура *Bar* заполняет прямоугольную область экрана. Указание на нее следующее:

```
Bar(x1, y1, x2, y2),
```

где *x1*, *y1*, *x2*, *y2* – переменные или выражения типа *integer*, определяющие координаты закрашивающей области. Процедура закрашивает, но не обводит прямоугольники текущими образцами узора и цвета, которые задаются процедурой *SetFillStyle*.

Процедура *Bar3D* вычерчивает трехмерное изображение параллелепипеда и закрашивает его переднюю грань. Обращение к процедуре имеет вид:

```
Bar3D(x1, y1, x2, y2, Depth, Top),
```

где *x1*, *y1*, *x2*, *y2* – выражения или переменные типа *integer*, определяющие координаты левого верхнего и правого нижнего углов передней грани, *Top* – выражение, переменная или константа логического типа. Если *Top* = *True*, то вычерчивается верхняя грань параллелепипеда, а в противном случае – нет.

Depth – третье измерение («глубина») трехмерного изображения. Для задания этого параметра в модуле GRAPH определены следующие константы:

```
TopOn = True;  
TopOff = False.
```

Процедура FillEllipse обводит линией и заполняет эллипс. Обращение к процедуре имеет вид:

```
FillEllipse(x, y, rx, ry),
```

где x, y – переменные типа integer, содержащие координаты центра, rx, ry – переменные типа word, определяющие горизонтальный и вертикальный радиусы эллипса в точках.

Эллипс обводится линией, заданной процедурами SetLineStyle и SetColor, и заполняется с использованием параметров, установленных процедурой SetFillStyle.

Процедура Sector вычерчивает и заполняет эллипсный сектор. Обращение к процедуре имеет вид:

```
Sector(x, y, BegA, EndA, rx, ry).
```

Здесь BegA, EndA – переменные типа word, определяющие соответственно начальный и конечный углы эллипсного сектора. Остальные параметры обращения аналогичны параметрам процедуры FillEllipse.

Процедура PieSlise вычерчивает и заполняет сектор окружности. Обращение к процедуре имеет вид:

```
PieSlise(x, y, BegA, EndA, r).
```

В отличие от процедуры Sector, указывается лишь один радиус r , остальные параметры аналогичны параметрам процедуры Sector.

12.4. Вывод текста в графическом режиме

Специально для графического режима разработаны процедуры, обеспечивающие вывод сообщений различными шрифтами в горизонтальном или вертикальном направлении с изменением размеров в стандартных шрифтах. В шрифтах, разработанных фирмой Borland, отсутствует кириллица, что исключает вывод русскоязычных сообщений. Однако пользователями разработаны шрифты, позволяющие выводить русскоязычные сообщения. Рассмотрим более подробно все стандартные средства модуля GRAPH для вывода текста.

Процедура OutText выводит текстовую строку, начиная с текущего положения графического курсора. Обращение к процедуре имеет вид:

```
OutText (Txt),
```

где Txt – строковая переменная или константа. Строка выводится в соответствии с установленным стилем и выравниванием. Если текст выходит за границы экрана, то в случае стандартного шрифта он не выводится, а при использовании штриховых шрифтов отсекается.

Процедура `OutTextXY` выводит строку, начиная с заданного места. Обращение к процедуре имеет вид:

```
OutTextXY(x, y, txt);
```

где `x`, `y` – переменные или константы типа `integer` (координаты точки вывода), `txt` – выводимая строка (переменная типа `string`).

А что делать, если необходимо вывести в графическом режиме переменную целого или вещественного типа (то есть численную переменную)? Следует предварительно перенести ее в строку символов с помощью функции `Str`. Обращение к функции `Str` имеет вид:

```
Str(x[:N[:M]], s),
```

где `x` – численная переменная, `s` – строковая переменная, `n`, `m` – формат представления переменной `x` (`n`, `m` – целые). Функция `Str` преобразует численное значение `x` в соответствии с форматом в строковое представление `s`.

Процедура `SetTextStyle` устанавливает стиль текстового вывода на экран. Обращение к процедуре имеет вид:

```
SetTextStyle(Font, Direct, Size),
```

где `Font`, `Direct`, `Size` – переменные или выражения типа `word`, определяющие код (номер) шрифта, его направления и размера. Для указания кода шрифта можно использовать следующие предварительно определенные константы:

```
DefaultFont = 0 - точечный шрифт 8x8.
```

Этот шрифт входит в модуль `GRAPH` и доступен в любой момент. Он используется по умолчанию. Для того чтобы корректно работать с русскими символами при выводе текста в графическом режиме, драйвер кириллицы должен поддерживать графический режим.

Все остальные шрифты векторные. Они отличаются богатыми изобразительными возможностями, главная же их особенность заключается в легкости изменения размеров без существенного ухудшения качества изображения. Каждый шрифт хранится в отдельном файле с расширением `.chr`. Чтобы шрифт был доступен, соответствующий файл должен находиться в текущем каталоге. Если в файле с расширением `.chr` есть символы кириллицы (файл русифицирован), то тогда процедура `OutText` будет выводить сообщение на русском языке (это вообще отдельная, довольно сложная проблема):

- `TriplexFont = 1` – утроенный шрифт `Trip.chr`;
- `SmallFont = 2` – уменьшенный шрифт `litt.chr`;
- `SansSerifFont = 3` – прямой шрифт;
- `GothicFont = 4` – готический шрифт.

Эти константы определяют все шрифты для версий 4.0–6.0. В версии 7.0 набор шрифтов значительно расширен, однако для новых шрифтов не предусмотрены соответствующие мнемонические константы – надо просто указать номер шрифта. При обращении к `SetTextStyle` можно использовать следующие константы:

- 5 – scri.chr – рукописный шрифт;
- 6 – simp.chr – одноштриховый шрифт типа Courier;
- 7 – tscr.chr – наклонный шрифт типа Times Italic;
- 8 – lcom.chr – шрифт типа Times Roman;
- 9 – euro.chr – шрифт типа 6 увеличенный;
- 10 – bold.chr – крупный двухштриховой шрифт.

В качестве направления Direct выдачи текста можно использовать следующие константы модуля GRAPH:

- HorizDir = 0 – слева направо;
- VertDir = 1 – снизу вверх.

Размер выводимых символов (параметр Size) принимает значение от 1 до 10 (точечный шрифт от 1 до 32). Если значение параметра равно 0, то устанавливается размер 1, если больше 10 – 10.

ПРИМЕР 12.5. Рассмотрим программу использования различных шрифтов [8, 9].

```
Uses Crt, graph;
Var i, d, m: integer;
Begin
  d:= detect;
  InitGraph(d, m, 'c:\bp\bgi');
  For i:=1 to 10 do
    begin
      SetTextStyle(i, 0, 0);
      OutTextXY(20, 20+(i-1)*18, 'Example');
      delay(2000);
    end;
  Repeat until KeyPressed;
  CloseGraph;
End.
```

12.5. Сохранение и выдача изображений

Для работы с изображениями в Паскале предусмотрены следующие процедуры и функции.

Функция ImageSize возвращает размер памяти в байтах, необходимый для размещения фрагмента изображения. Заголовок процедуры:

```
Function ImageSize(X1, Y1, X2, Y2: Integer): word.
```

Здесь X1, Y1, X2, Y2 – координаты верхнего левого и правого нижнего углов фрагмента изображения.

Процедура GetImage помещает в память копию прямоугольного фрагмента изображения. Заголовок процедуры имеет вид:

```
GetImage (X1, Y1, X2, Y2: Integer; var Buf).
```


Здесь $X1, Y1, X2, Y2$ – координаты верхнего левого и правого нижнего углов фрагмента изображения, Buf – переменная или участок кучи (см. главу 9), куда будет помещена копия видеопамати с фрагментом изображения. Но следует помнить, что Buf не может превышать 64 Кб (см. главу 9), однако для хранения всего графического экрана может потребоваться участок памяти большего размера. Ведь для хранения одной точки в режиме 256 цветов требуется 1 байт. Как обойти ограничение? Можно разделить изображение на несколько участков и хранить каждый из них, но это не совсем удобно. Нами был разработан модуль с подпрограммами `FAR_IMAGESIZE`, `FAR_GETIMAGE`, `FAR_PUTIMAGE`, позволяющими обойти это ограничение.

Процедура `PutImage` выводит в заданное место копию фрагмента изображения, ранее помещенную в память процедурой `GetImage`. Заголовок процедуры имеет вид:

```
Procedure PutImage(X, Y: Integer; var Buf; Mode: Word);
```

где X, Y – координаты левого верхнего угла того участка на экране, куда будет скопирован фрагмент изображения, хранящийся в переменной Buf ; $Mode$ – способ копирования изображения. Параметр $Mode$ наиболее часто может принимать следующие значения:

- `NormalPut = 0`; в этом случае будет стираться часть экрана, а на ее место помещаться изображение, хранящееся в переменной Buf ;
- `XorPut = 1`; эта операция, примененная к тому же месту экрана, откуда была получена копия, сотрет эту же часть экрана. Если данную операцию использовать дважды на одном и том же участке экрана, то в целом вид не изменится. Этот эффект был положен в основу алгоритма создания движущихся объектов. Для создания эффекта движения объект сохраняем в памяти с помощью процедуры `getImage`, а затем с помощью процедуры `putImage` перемещаем его по экрану;
- `NotPut = 4`; операция `NotPut` стирает часть экрана и на это место помещает содержимое переменной Buf в инверсном виде. Инверсия применяется к цифровому коду цвета точки. Для цвета `Red` (код 0100) получим цвет `LightCyan` (код 1011) и т.д.

12.6. Построение графика функции на экране дисплея

Алгоритм построения графика непрерывной функции $y = f(x)$ на отрезке $[a, b]$ состоит в следующем: необходимо построить точки $(x_i, f(x_i))$ ¹ в декартовой системе координат и соединить их прямыми линиями. Чем больше точек будет изображено, тем более плавным будет построенный график.

¹ $hx = (b-a)/N$, $x_i = a + (i-1)hx$, $y_i = f(x_i)$; $i = 1, N$; N – количество точек на отрезке $[a, b]$.

При переносе этого алгоритма на экран дисплея следует учесть размеры экрана (ось X направлена слева направо и ее координаты лежат в пределах от 0 до $GetMaxX$, ось OY направлена вниз, и ее координаты находятся в пределах от 0 до $GetMaxY$). Необходимо помнить, что значения координат X и Y должны быть целыми.

Необходимо все точки из «бумажной» системы координат (X изменяется в пределах от a до b , Y изменяется от минимального (\min) до максимального (\max)) пересчитать в «экранную» (в этой системе координат ось абсцисс обозначим буквой U , которая изменяется от 0 до $GetMaxX$, а ось ординат – буквой V ($V \in [0, GetMaxY]$)).

Для преобразования координаты X в координату U построим линейную функцию $cX + d$, которая переведет точки из интервала (a, b) в точки интервала $(X0, GetMaxX - XK)^1$. В связи с тем, что точка a «бумажной» системы координат перейдет в точку $X0$ «экранной», а точка b – в точку $GetMaxX - XK$, система линейных уравнений для нахождения коэффициентов c и d имеет вид:

$$\begin{cases} c \cdot a + d = X0 \\ c \cdot b + d = GetMaxX - XK \end{cases} \text{ . Решив ее, найдем коэффициенты } c, d$$

$$\begin{cases} c = \frac{GetMaxX - XK - X0}{b - a} \\ d = X0 - c \cdot a \end{cases}$$

Для преобразования координаты Y в координату V построим линейную функцию $V = gY + h$. Точка \min «бумажной» системы координат перейдет в точку $GetMaxY - Y0$ «экранной», а точка \max – в точку $Y0$. Для нахождения коэффициентов g и h необходимо решить систему линейных алгебраических уравнений

$$\begin{cases} g \cdot \min + h = GetMaxY - YK \\ g \cdot \max + h = Y0 \end{cases}$$

После этого можно найти коэффициенты g и h :

$$\begin{cases} g = \frac{GetMaxY - YK - Y0}{\min - \max} \\ h = Y0 - g \cdot \max \end{cases} \text{ .}$$

Алгоритм построения графика на экране дисплея можно разделить на следующие этапы:

1. Определить число точек N , шаг изменения переменной X ($hx = (b - a) / N$).
2. Сформировать массивы X , Y ; вычислить максимальное (\max) и минимальное (\min) значение Y .
3. Перевести экран в графический режим.
4. Найти коэффициенты c , d , g и h .
5. Создать массивы $U_i = cX_i + d$ и $V_i = gY_i + h$.

¹ $X0$, XK , $Y0$, YK – отступы от левой, правой, нижней, верхней границы экрана.

6. Соединить соседние точки $(U_i, V_i, U_{i+1}, V_{i+1})$ прямыми линиями с помощью функции line.
7. Изобразить систему координат, линий сетки и подписей.

При построении графиков k нескольких непрерывных функций вместо массивов U и V рациональнее использовать матрицы размером $k \times n$, элементы каждой строки которых являются ординатами соответствующего графика в «бумажной» (U) и «экранной» (V) системе координат.

Блок-схема алгоритма изображения графиков k непрерывных функций на экране дисплея представлена на рис 12.5. Блоки 2–3 реализуют первый этап алгоритма. В блоках 4–8 формируется массив X и матрица U . Блоки 9–15 предназначены для определения максимального и минимального значения, отображаемого на графике. Блок 16 реализует перевод экрана в графический режим. В блоке 17 рассчитываются коэффициенты c, d, g и h перевода из «бумажной» системы координат в «экранную». В блоках 18–21 формируются массив значений U и матрица значений V в «экранной» системе координат. Блоки 22–24 предназначены для вывода графиков на экран дисплея. В блоке 25 представлен седьмой этап алгоритма, блок-схема этого этапа дана на рис. 12.6.

Особенностью рисования линий сетки является то, что изображать линии сетки надо в «экранной» системе координат, а выводить реальные значения – из «бумажной» системы координат. Можно выделить следующие этапы алгоритма рисования линий сетки:

1. Определить количество линий сетки по оси OX (k_{vox}) и по оси OY (k_{voy}) (блок 2).
2. Определить расстояние между линиями сетки в «экранной» системе координат (h_u и h_v) (блок 3).
3. Изобразить линии сетки, параллельные оси OX (блоки 6, 7) и OY (блоки 4–5).
4. Вычислить значения, которые будут находиться под осью OX ($a + (i - 1) h_x$), и вывести их на экран (блоки 9–10). Найти данные, которые разместятся левее оси OY ($\max - (i - 1) h_y$), и вывести их на экран (блоки 11–12).
5. Изобразить ось координат (если они попадают в интервал, отображаемый на графике) (блоки 13–16), для чего необходимо провести прямую в точке $\text{round}(d)$ (в «бумажной» системе координат это $X = 0$), параллельную оси OY , и прямую в точке $\text{round}(g)$ ($Y = 0$ в «бумажной» системе координат), параллельную оси OX в экранной системе координат.

Реализация рассмотренного алгоритма представлена в виде программы изображения четырех графиков функций $y = \sin(\alpha, x) \cos(x)^1$ на интервале $[a, b]$. Приведем текст программы:

```
uses crt, graph;
type matr_real=array[1..10,1..200] of real;
matr_int=array[1..10,1..200] of integer;
```

¹ Заменяв функцию f в программе, можно заставить рисовать ее графики любых непрерывных функций.

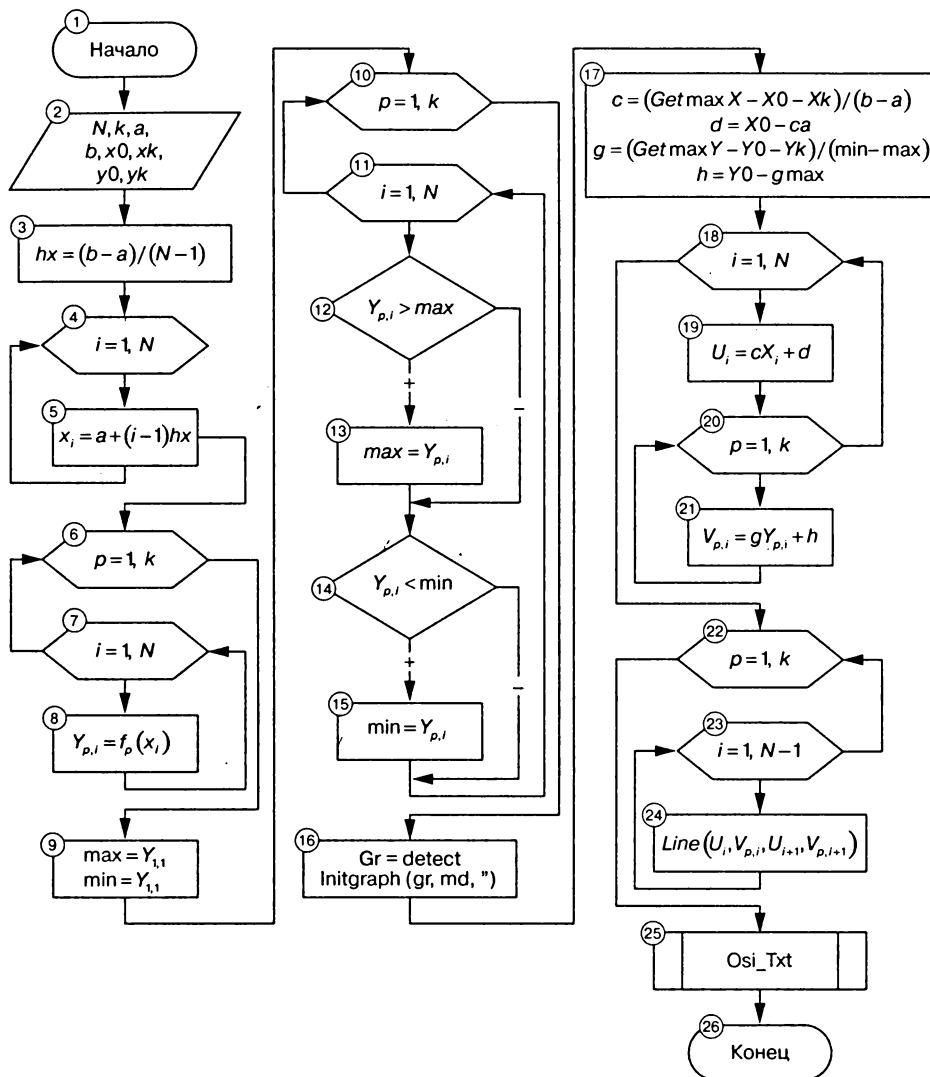


Рис. 12.5 ▼ Блок-схема алгоритма построения графиков нескольких функций

```

mas_real=array[1..200] of real;
mas_int=array[1..200] of integer;
function f(a,x:real):real;
begin
f:=sin(a*x)*cos(x)
end;
var
alf,x:mas_real;u:mas_int;

```

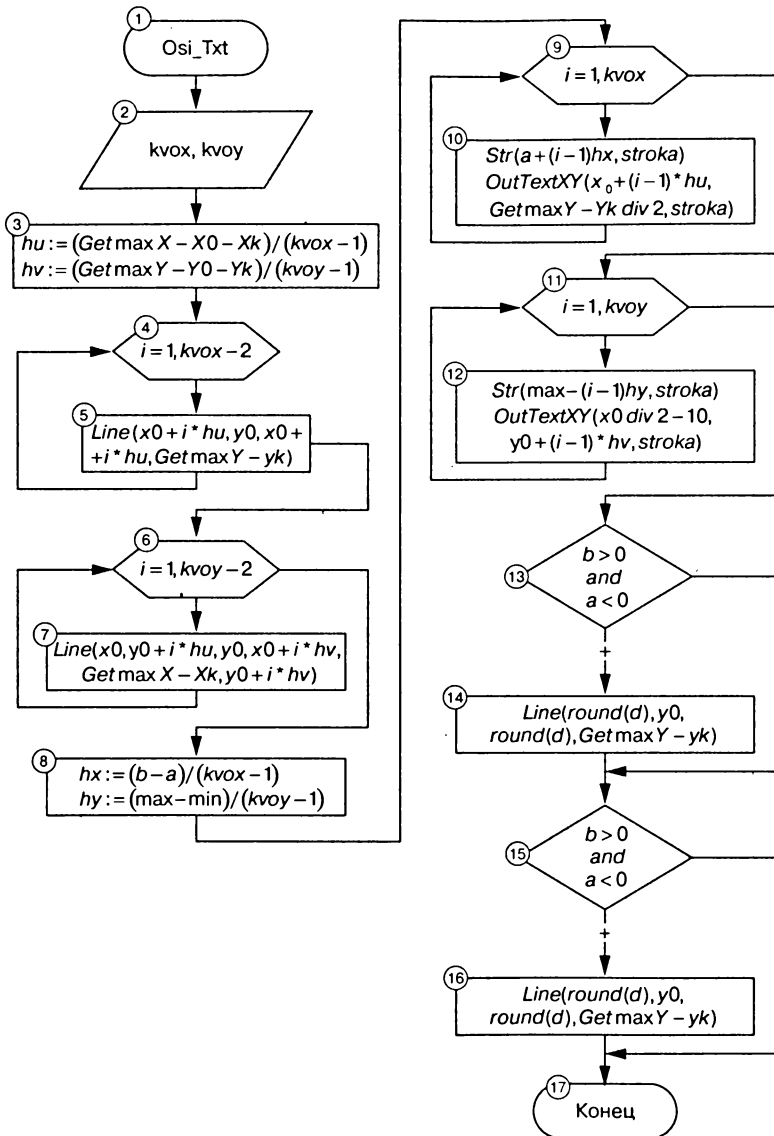


Рис. 12.6 ▼ Блок-схема алгоритма изображения линий сетки и осей координат

```

y:matr_real;v:matr_int;
hu,hv,kvoy,kvox,grdr,grmd,i,p,n,x0,y0,xk,yk:integer;
max,min,a,b,hx,hy:real;
c,d,g,h:real;
Pattern:word;
    
```

```

f1:text;
stroka:string;
begin
assign(f1,'graphik.txt');
{ В файле graphik.txt исходные данные программы. }
{ N - количество точек для изображения графиков. }
{ Интервал [a,b], на котором строится график. }
{ Alf(i) - коэффициенты функции. }
{ X0,xk,y0,yk - отступы от левой, правой, нижней, верхней границ экрана. }
{ Kvox, kvoу - количество линий сетки. }
reset(f1);
readln(f1,n);readln(f1,a,b);
for i:=1 to 4 do
  read(f1,alf[i]);
readln(f1,x0,xk,y0,yk);
readln(f1,kvox,kvoу);
close(f1);
{ Формирование массива X и матрицы Y, блоки 4-8 (см. рис. 12.6). }
x[1]:=a;
hx:=(b-a)/(n-1);
for i:=1 to n do
  x[i]:=a+(i-1)*hx;
for p:=1 to 4 do
  for i:=1 to n do
    y[p,i]:=f(alf[p],x[i]);
{ Поиск максимального и минимального значения Y, блоки 9-15
(см. рис. 12.6). }
max:=y[1,1];
min:=y[1,1];
for p:=1 to 4 do
  for i:=1 to n do
    begin
      if y[p,i]>max then max:=y[p,i];
      if y[p,i]<min then min:=y[p,i];
    end;
{ Перевод экрана в графический режим. }
grdr:=detect;
InitGraph(grdr,grmd,'c:\bp\bgi');
{ Цвет фона в графическом режиме белый. }
SetBkColor(15);
{ Цвет пера синий. }
SetColor(1);
{ Изображение рамки вокруг графика. }
Rectangle(x0,y0,GetmaxX-xk,GetmaxY-yk);
{ Вычисление коэффициентов пересчета. }
c:=(GetMaxX-x0-xk)/(b-a);
d:=x0-c*a;
g:=(GetMaxY-y0-yk)/(min-max);
h:=y0-g*max;
{ Формирование значений в «экранной» системе координат. }
for i:=1 to n do
  u[i]:=trunc(c*x[i]+d);
for p:=1 to 4 do

```

```

    for i:=1 to n do
        v[p,i]:=trunc(g*y[p,i]+h);
    for p:=1 to 4 do
    begin
        { Выбор цвета изображения очередного графика. }
        setcolor(p+2);
        { Выбор стиля линии для очередного графика. }
        SetLineStyle(p-1,pattern,1);
        { Изображение графика. }
        for i:=1 to n-1 do
            line(u[i],v[p,i],u[i+1],v[p,i+1]);
        end;
        { Определение тонких пунктирных линий для изображения линий сетки. }
        SetLineStyle(DashedLn,pattern,1);
        { Определение цвета для изображения линий сетки. }
        setcolor(1);
        { Определение расстояния между линиями сетки по оси OX }
        { в «экранной» системе координат. }
        hu:=trunc((GetmaxX-x0-xk)/(kvox-1));
        { Определение расстояния между линиями сетки по оси OY }
        { в «экранной» системе координат. }
        hv:=trunc((GetmaxY-y0-yk)/(kvoy-1));
        { Изображение линий сетки, параллельных оси OY. }
        for i:=1 to kvox-2 do
            line(x0+i*hu,y0,x0+i*hu,GetMaxY-yk);
        { Изображение линий сетки, параллельных оси OX. }
        for i:=1 to kvoy-2 do
            line(x0,y0+i*hv,GetMaxX-Xk,y0+i*hv);
        { Определение расстояния между линиями сетки по оси OX и оси OY }
        { в «бумажной» системе координат. }
        hx:=(b-a)/(kvox-1);
        hy:=(max-min)/(kvoy-1);
        for i:=1 to kvox do
        begin
            { Вычисление очередного значения, выводимого под осью OX, и перевод его }
            { в строковое значение. }
            Str(a+(i-1)*hx:1:1,stroka);
            { Вывод строкового значения на экран под осью OX. }
            OutTextXY(x0+(i-1)*hu-15,GetMaxY-yk div 2 -15,stroka);
        end;
        for i:=1 to kvoy do
        begin
            { Вычисление очередного значения, выводимого левее оси OY, и перевод его }
            { в строковое значение. }
            Str(max-(i-1)*hy:1:1,stroka);
            { Вывод строкового значения на экран левее оси OX. }
            OutTextXY(x0 div 2 - 10,y0+(i-1)*hv,stroka);
        end;
        OutTextXY(GetMaxX-Xk+50,y0 div 2 -5,'A');
        for p:=1 to 4 do
        begin
            { Определение цвета и стиля выводимой легенды. }
            setcolor(p+2);

```

```

SetLineStyle(p-1,pattern,1);
{ Вывод шаблона легенды. }
Line(GetMaxX-Xk+10,(p+1)*Y0 div 2,GetMaxX-Xk+40,(p+1)*Y0 div 2);
{ Вывод коэффициента alf в качестве легенды. }
Str(alf[p]:1:1,stroka);
OutTextXY(GetMaxX-Xk+50,(p+1)*Y0 div 2 -5,stroka);
end;
{ Определение цвета и стиля (сплошная линия) осей координат. }
SetLineStyle(SolidLn,pattern,1);
setcolor(1);
{ Изображение линий сетки в случае необходимости. }
if (b>0) and (a<0) then line(round(d),y0,round(d),getmaxY-yk);
if (max>0) and (min<0) then line(x0,round(h),GetMaxX-Xk,round(h));
{ Подпись над графиком. }
OutTextXY(2* GetMaxX div 5, y0 div 2,'y=sin(Ax)cos(x)');
repeat until keypressed;
end.

```

После запуска программы на выполнение на экране появится следующее графическое окно (см. рис. 12.7).

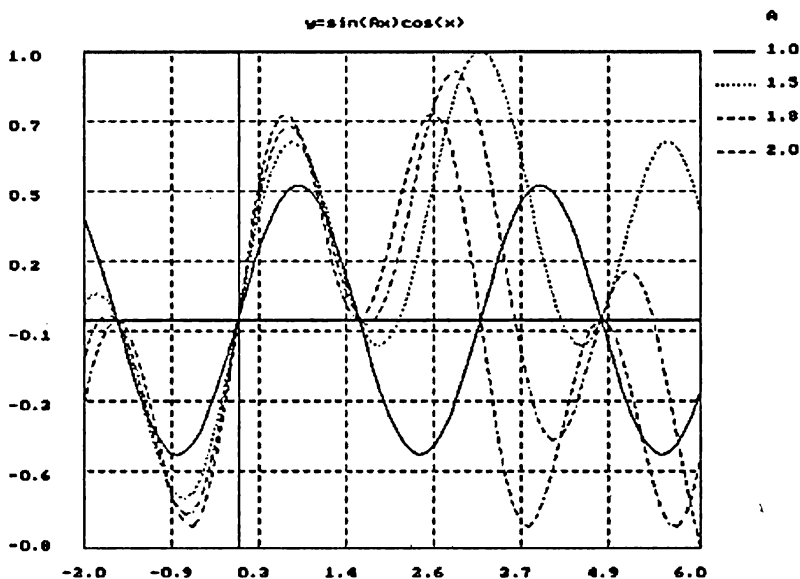


Рис. 12.7 ▼ График функций $y = \sin(\alpha, x) \cos(x)$

ПРИМЕР 12.6. Изобразить на экране дисплея графики разрывных функций $y = \frac{C}{x^2}$ при различных значениях C . Ниже приведен текст программы, где прокомментированы участки, относящиеся к особенностям изображения графика разрывной функции.


```

uses crt,graph;
type matr_real=array[1..10,1..200] of real;
matr_int=array[1..10,1..200] of integer;
mas_real=array[1..200] of real;
mas_int=array[1..200] of integer;
function f(a,x:real):real;
begin
f:=A/sqr(x)
end;
var
c,x:mas_real;u:mas_int;
y:matr_real;v:matr_int;
hu,hv,kvoy,kvox,grdr,grmd,i,j,n,x0,y0,xk,yk:integer;
max,min,a,b,hx:real;
razr1,razr2,a1,b1,c1,d1:real;
Pattern:word;
fl:text;
stroka:string;
begin
assign(fl,'gr.txt');
{ В файле gr.txt исходные данные программы. }
{ N - количество точек для изображения графиков. }
{ Интервал [a,b], на котором строится график. }
{ Интервал разрыва [razr1,razr2], исключаемый из графика. }
{ График строится на двух интервалах [a,razr1] и [razr2,b]. }
{ C(i) - коэффициенты функции. }
{ X0,xk,y0,yk -отступы от левой, правой, нижней, верхней границ экрана. }
{ Kvox, kvoy - количество линий сетки. }
reset(fl);
readln(fl,n);readln(fl,a,b,razr1,razr2);
for i:=1 to 4 do
  read(fl,c[i]);
readln(fl,x0,xk,y0,yk);
readln(fl,kvox,kvoy);
close(fl);
{ В данном случае интервал массива x формируется из двух участков }
{ [a,razr1] и [razr2,b]. }
x[1]:=a;
{ Шаг изменения X на интервале [a,razr1]. }
hx:=(razr1-a)/(n div 2 -1);
for i:=2 to n div 2 do
  x[i]:=x[i-1]+hx;
{ Шаг изменения X на интервале [razr2,b]. }
hx:=(b-razr2)/(n div 2 -1);
x[n div 2+1]:=razr2;
for i:=n div 2 + 2 to n do
  x[i]:=x[i-1]+hx;
for j:=1 to 4 do
  for i:=1 to n do
    y[j,i]:=f(c[j],x[i]);
max:=y[1,1];
min:=y[1,1];
for j:=1 to 4 do

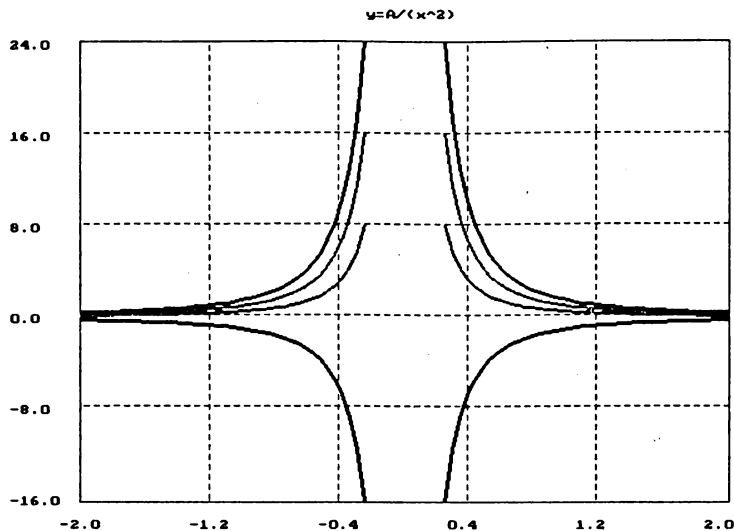
```

```

for i:=1 to n do
begin
  if y[j,i]>max then max:=y[j,i];
  if y[j,i]<min then min:=y[j,i];
end;
grdr:=detect;
InitGraph(grdr,grmd,'c:\bp\bgi');
Rectangle(x0,y0,GetmaxX-xk,GetmaxY-yk);
a1:=(GetMaxX-x0-xk)/(b-a);
b1:=x0-a1*a;
c1:=(GetMaxY-y0-yk)/(min-max);
d1:=y0-c1*max;
for i:=1 to n do
  u[i]:=trunc(a1*x[i]+b1);
for j:=1 to 4 do
  for i:=1 to n do
    v[j,i]:=trunc(c1*y[j,i]+d1);
SetLineStyle(SolidLn,pattern,3);
for j:=1 to 4 do
begin
  SetColor(j);
  for i:=1 to n-1 do
  { При построении графика исключаем соединительную линию
  двух подинтервалов [a,razr1] и [razr2,b]. }
    if i<> n div 2 then line(u[i],v[j,i],u[i+1],v[j,i+1]);
end;
SetLineStyle(DashedLn,pattern,1);
SetColor(White);
hu:=trunc((GetmaxX-x0-xk)/kvox);
hv:=trunc((GetmaxY-y0-yk)/kvoy);
for i:=1 to kvox-1 do
  line(x0+i*hu,y0,x0+i*hu,GetMaxY-yk);
hx:=(b-a)/kvox;
for i:=1 to kvox+1 do
begin
  Str(a+(i-1)*hx:1:1,stroka);
  OutTextXY(x0+(i-1)*hu-15,GetMaxY-Yk div 2 -15,stroka);
end;
for i:=1 to kvoy-1 do
  line(x0,y0+i*hv,GetMaxX-Xk,y0+i*hv);
hx:=(max-min)/kvoy;
for i:=1 to kvoy+1 do
begin
  Str(max-(i-1)*hx:1:1,stroka);
  OutTextXY(x0 div 2 -10,y0+(i-1)*hv,stroka);
end;
OutTextXY(GetMaxX div 2, y0 div 2,'y=A/(x^2)');
repeat until keypressed;
end.

```

Результат работы программы представлен на рис. 12.8. Если функция имеет несколько точек разрыва (например, $\text{tg}(x)$, $\text{ctg}(x)$), следует вводить массивы

Рис. 12.8 ▽ Графики функций $y = \frac{C}{x^2}$

интервалов разрыва [razr1i, razr2i] – точки которые не следует соединять между собой.

12.7. Упражнения по теме «Графические средства Турбо Паскаля»

Изобразить на экране дисплея графики указанных ниже функций (A принимает следующие значения: -1; 0,5; 1; 1,5). Построить координатные оси и выполнить соответствующие надписи на них:

- $Y = A \sin(\cos(x))$; $x \in [-3,2; 3,2]$;
- $Y = A 2^x$; $x \in [-2, 5]$;
- $Y = A e^{\sin(x)}$; $x \in [-4, 4]$;
- $Y = A e^{-x^2}$; $x \in [-3, 3]$;
- $Y = A(3x^3 - 4x^2 + 5x) \cdot \sin(x)$ $x \in [-10, 10]$;
- $Y = A(2x^4 - 7x^3 + 13x^2 - 8x) \cdot \cos(x)$; $x \in [-2, 5]$;
- $Y = A(\sin(x) + x^2)$; $x \in [-3, 3]$;
- $Y = A(3x^3 + 7x) \cdot e^{\cos 5x}$; $x \in [-7, 7]$;
- $Y = A \cos(x) / (x^2)$; $x \in [-4, 4]$;
- $Y = A \cdot e^x \cdot \cos(x)$; $x \in [0, 6]$.

Глава 13

Создание личных модулей

Модуль – это автономная программная единица, включающая в себя различные компоненты: константы, переменные, типы, процедуры и функции. До этого момента мы рассматривали, как пользоваться уже существующими программными модулями. В данном разделе кратко рассказано, как создавать свои личные модули, и приведен готовый графический модуль. Подпрограммы создания графиков функций разработаны авторами, те же, что используются при изображении поверхностей двумерных функций, являются переработанным алгоритмом [2].

13.1. Структура модуля

Модуль имеет следующую структуру:

```
UNIT <имя модуля>;  
INTERFACE  
<интерфейсная часть>  
IMPLEMENTATION  
<исполняемая часть>  
BEGIN  
<иницилирующая часть>  
END.
```

Заголовок модуля состоит из служебного слова UNIT и следующего за ним имени. Причем имя модуля должно совпадать с именем файла, в котором он хранится. Модуль EDIT должен храниться в файле edit.pas.

Интерфейсная часть начинается служебным словом INTERFACE, за которым находятся объявления всех глобальных объектов модуля: типов, констант,

переменных и подпрограмм. Эти объекты будут доступны всем модулям и программам, вызывающим данный модуль.

Исполняемая часть начинается служебным словом IMPLEMENTATION и содержит описания подпрограмм, объявленных в интерфейсной части. Здесь же могут объявляться локальные объекты, которые используются только в интерфейсной части и остаются не доступными программам и модулям, вызывающим данный модуль.

В иницирующей части размещаются исполняемые операторы, содержащие некоторый фрагмент программы. Эти программы исполняются до передачи управления основной программе и обычно используются для подготовки ее работы. Иницирующая часть может отсутствовать вместе с начинающим ее словом begin.

После написания модуля его следует откомпилировать, предварительно установив Destination Disk (пункт меню **Compile**) (см. описание оболочки Турбо Паскаля в приложении). В результате компиляции будет создан файл с расширением `.tpr`. Затем этот модуль можно использовать так же, как любой стандартный.

13.2. Модуль изображения графиков и поверхностей непрерывных функций

Ниже приведен модуль рисования графиков функций $y = f(x)$ ¹ и поверхностей двумерных функций $z = f(x, y)$ [2]. Он позволяет изображать график одной или нескольких функций в любой области экрана дисплея. В связи с тем, что программа рисует графики различных непрерывных функций на любом участке экрана, вывод текстовой информации в подпрограммах сведен к минимуму. Читатель может самостоятельно расширить возможности подпрограмм.

```
{ Это большой модуль рисования графиков и поверхностей. }
unit GRAPHIC;
interface
uses graph,crt;
type massiv=array [1..10000] of real;
pmas=^massiv;
dinmas=array [1..8] of pmas;
func=function(x:real):real;
func1=function(x,y:real):real;
PROCEDURE GRAFIK (a, b :real; n: integer; fun:ARRAY OF FUNC;
x1, y1, x2, y2: word; Colors: array of WORD; var Err:shortint);
{ Процедура рисования графиков восьми функций в одной системе координат. }
{ a,b - интервал, на котором рисуются графики, n - количество графиков
(n<=8).
fun1,...,fun8 - имена изображаемых функций. }
```

¹ В подпрограмме изображения графиков функций использован алгоритм, описанный в предыдущей главе.

```

{ x1, y1, x2, y2 - прямоугольник на экране,
  внутри которого будут изображаться графики. }
{ Colors - массив цветов. }
{ Colors[0] - цвет фона. }
{ Colors[1] - цвет первого графика. }
{ Colors[2] - цвет второго графика. }

...
{ Colors[8] - цвет восьмого графика. }
{ Colors[9] - цвет рамки и подписей. }
{ Err - код ошибки. }
{ 0 - нормальное завершение программы. }
{ -1 - n>8 or n<1. }
{ -2 - x1,y1<0 or x2,y2>getmaxX,getmaxY. }
PROCEDURE GRAFIK1(n:integer;m:integer; x:pmas;y:dinmas;
x1,y1,x2,y2:word;Colors: array of WORD; var Err:shortint);
{ Процедура рисования восьми графиков функций по точкам в одной системе
  координат. }
{ a,b - интервал, на котором рисуются графики. }
{ m - количество точек в массивах x, y. }
{ n - количество графиков (n <= 8). }
{ x- динамический массив x (указатель). }
{ y- массив указателей y. }
{ x1, y1, x2, y2 - прямоугольник на экране, внутри которого будут
  изображаться графики. }
{ Colors - массив цветов. }
{ Colors[0] - цвет фона. }
{ Colors[1] - цвет первого графика. }
{ Colors[2] - цвет второго графика. }

...
{ Colors[8] - цвет восьмого графика. }
{ Colors[9] - цвет рамки и подписей. }
{ Err - код ошибки. }
{ 0 - нормальное завершение программы. }
{ -1 - n>8 or n<1. }
{ -2 - x1,y1<0 or x2,y2>getmaxX,getmaxY. }
Procedure surfase(xMin,xMax,yMin, yMax, Rad, Theta, Phi, D:real;
fun:func1);
{ Процедура рисования графика двумерной функции Z=F(X,Y). }
{ fun - функция f(x,y). }
{ xmin,xmax,ymin,ymax - границы изменения переменных x и y. }
{ d - расстояние до кривой. }
{ rad, theta, phi - три угла в радианах. }
implementation
PROCEDURE GRAFIK(a,b:real; n:integer; fun:ARRAY OF FUNC;
x1,y1,x2,y2:word; Colors:array of WORD; var Err:shortint);
var
x:pmas;
y:dinmas;
rx,ry,ymax,ymin,a1,b1,c1,d1:real;
kol,y1n,y2n,x1n,x2n,k,dx,dy,i,j,nmax,nmin:integer;
s:string[10];
begin
if (n<1) or (n>8) then

```

```

begin
  Err:=-1;
exit
end;
if (x1<0) or (y1<0) or (x2>getmaxX) or (y2>GetmaxY) then
begin
  Err:=-2;
  exit
end;
{ Устанавливается цвет фона. }
SetBkColor(Colors[0]);
{ Устанавливается цвет линий и текста. }
SetColor(Colors[9]);
dx:=x2-x1;
dy:=y2-y1;
x1n:=x1+trunc(dx/10);
x2n:=x2-trunc(dx/20);
y1n:=y1+trunc(dy/10);
y2n:=y2-trunc(dy/10);
kol:=x2n-x1n;
{ Выделение памяти под динамический массив X. }
getmem(x,(kol+1)*6);
{ Формирование массива X. }
x^[1]:=a;
for i:=2 to kol+1 do
  x^[i]:=x^[i-1]+(b-a)/kol;
{ Выделение памяти под динамический массив Y. }
for i:=1 to n do
  GetMem(y[i],(kol+1)*6);
{ Запись в массив Y значений функций. }
for i:=1 to kol+1 do
begin
  y[1]^i:=fun[0](x^[i]);
  if n>=2 then y[2]^i:=fun[1](x^[i]);
  if n>=3 then y[3]^i:=fun[2](x^[i]);
  if n>=4 then y[4]^i:=fun[3](x^[i]);
  if n>=5 then y[5]^i:=fun[4](x^[i]);
  if n>=6 then y[6]^i:=fun[5](x^[i]);
  if n>=7 then y[7]^i:=fun[6](x^[i]);
  if n>=8 then y[8]^i:=fun[7](x^[i]);
end;
{ Поиск максимума и минимума в массиве Y. }
ymin:=y[1]^1;
ymax:=y[1]^1;
nmin:=1;
nmax:=1;
for j:=1 to n do
  for i:=1 to kol+1 do
  begin
    if y[j]^i>ymax then
    begin
      ymax:=y[j]^i;
      nmax:=i;
    end;
  end;

```

```

    if y[j]^i<ymin then
    begin
        ymin:=y[j]^i;
        nmin:=i;
    end
end;
{ Формирование коэффициентов пересчета в «экранную» систему координат. }
a1:=(x2n-x1n)/(b-a);
b1:=x1n-a1*a;
c1:=(y1n-y2n)/(ymax-ymin);
d1:=y2n-c1*ymin;
{ Рисование осей и подписи. }
rx:=(x2n-x1n-1)/5;
ry:=(y2n-y1n+1)/5;
line(x1n-1,y2n+1,x2n+trunc(0.0375*dx),y2n+1);
str(x^[1]:1:2,s);
settextjustify(0,1);
outtextxy(x1n-25,y2n+1+trunc(dy/20),s);
str(x^[kol+1]:1:2,s);
settextjustify(2,1);
outtextxy(x1n+5*trunc(rx)+25,y2n+1+trunc(dy/20),s);
for i:=1 to 5 do
begin
    line(x1n+1+i*trunc(rx), y2n+1-trunc(dy/50), x1n+1+i*trunc(rx), y2n+1);
    line(x1n+1+i*trunc(rx), y2n+1, x1n+1+i*trunc(rx), y1n);
end;
line(x1n-1,y2n+1,x1n-1,y1n-trunc(0.075*dy));
str(ymin:1:2, s);
settextjustify(2,1);
outtextxy(x1n-trunc(dx/200),y2n+1-trunc(dy/50)+10,s);
str(ymax:1:2, s);
settextjustify(2,1);
outtextxy(x1n-trunc(dx/200),y1n+trunc(dy/50)-10,s);
for i:=1 to 5 do
begin
    line(x1n+1+trunc(dx/100),y2n-1-i*trunc(ry),x1n,y2n-1-i*trunc(ry));
    line(x2n,y2n-1-i*trunc(ry),x1n,y2n-1-i*trunc(ry));
end;
line(x2n+trunc(0.0375*dx),y2n+1,
x2n+trunc(0.0375*dx)-trunc(dx/50),trunc(y2n+1+dy/200));
line(x2n+trunc(0.0375*dx),y2n+1,
x2n+trunc(0.0375*dx)-trunc(dx/50),trunc(y2n+1-dy/200));
line(x1n-1,y1n-trunc(0.075*dy),x1n-1+trunc(dx/120),
y1n-trunc(0.075*dy)+trunc(dy/50));
line(x1n-1,y1n-trunc(0.075*dy),x1n-1-trunc(dx/120),
y1n-trunc(0.075*dy)+trunc(dy/50));
{ Рисование графиков непрерывных функций. }
for j:=1 to n do
begin
    SetColor(Colors[j]);
    for i:=1 to kol do
        line(trunc(a1*x^[i]+b1),trunc(c1*y[j]^i+d1),
            trunc(a1*x^[i+1]+b1),trunc(c1*y[j]^[i+1]+d1));
    end;

```



```
{ Освобождение памяти. }
FreeMem(x, (kol+1)*6);
for i:=1 to n do
  FreeMem(y[i], (kol+1)*6)
end;
PROCEDURE GRAFIK1(n:integer;m:integer; x:pmas;y:dinmas;
x1,y1,x2,y2:word;Colors: array of WORD; var Err:shortint);
var
a,b:real;
rx,ry,ymax,ymin,a1,b1,c1,d1:real;
kol,y1n,y2n,x1n,x2n,k,dx,dy,i,j,nmax,nmin:integer;
s:string[10];
begin
a:=x^[1];
b:=x^[m];
if (n<1) or (n>8) then
begin
  Err:=-1;
  exit
end;
if (x1<0) or (y1<0) or (x2>getmaxX) or (y2>GetmaxY) then
begin
  Err:=-2;
  exit
end;
SetColor(Colors[9]);
dx:=x2-x1;
dy:=y2-y1;
x1n:=x1+trunc(dx/10);
x2n:=x2-trunc(dx/20);
y1n:=y1+trunc(dy/10);
y2n:=y2-trunc(dy/10);
kol:=m;
ymin:=y[1]^1;
ymax:=y[1]^1;
nmin:=1;
nmax:=1;
for j:=1 to n do
  for i:=1 to kol do
    begin
      if y[j]^i>ymax then
      begin
        ymax:=y[j]^i;
        nmax:=i;
      end;
      if y[j]^i<ymin then
      begin
        ymin:=y[j]^i;
        nmin:=i;
      end
    end;
  end;
a1:=(x2n-x1n)/(b-a);
b1:=x1n-a1*a;
c1:=(y1n-y2n)/(ymax-ymin);
```

```

d1:=y2n-cl*ymin;
rx:=(x2n-x1n-1)/5;
ry:=(y2n-y1n+1)/5;
line(x1n-1,y2n+1,x2n+trunc(0.0375*dx),y2n+1);
str(x^[1]:1:2,s);
settextjustify(0,1);
outtextxy(x1n-25,y2n+1+trunc(dy/20),s);
str(x^[kol]:1:2,s);
settextjustify(2,1);
outtextxy(x1n+5*trunc(rx)+25,y2n+1+trunc(dy/20),s);
for i:=1 to 5 do
  line(x1n+1+i*trunc(rx),y2n-1,x1n+1+i*trunc(rx),y1n);
line(x1n-1,y2n-1,x1n-1,y1n-trunc(0.075*dy));
str(ymin:1:2,s);
settextjustify(2,1);
outtextxy(x1n-trunc(dx/200),y2n+1-trunc(dy/50),s);
str(ymax:1:2,s);
settextjustify(2,1);
outtextxy(x1n-trunc(dx/200),y1n+trunc(dy/50),s);
for i:=1 to 5 do
  line(x2n,y2n-1-i*trunc(ry),x1n,y2n-1-i*trunc(ry));
line(x2n+trunc(0.0375*dx),y2n+1,
x2n+trunc(0.0375*dx)-trunc(dx/50),trunc(y2n+1+dy/200));
line(x2n+trunc(0.0375*dx),y2n+1,
x2n+trunc(0.0375*dx)-trunc(dx/50),trunc(y2n+1-dy/200));
line(x1n-1,y1n-trunc(0.075*dy),x1n-1+trunc(dx/120),
y1n-trunc(0.075*dy)+trunc(dy/50));
line(x1n-1,y1n-trunc(0.075*dy),x1n-1-trunc(dx/120),
y1n-trunc(0.075*dy)+trunc(dy/50));
for j:=1 to n do
begin
  SetColor(Colors[j]);
  for i:=1 to kol-1 do
    line(trunc(a1*x^[i]+b1),trunc(c1*y[j]^i+d1),
trunc(a1*x^[i+1]+b1),trunc(c1*y[j]^[i+1]+d1))
end;
end;
Procedure surfase (xMin,xMax,yMin,yMax,Rad,Theta,Phi,D:real; fun:func1);
var
x,y,dx,dy,Ax,Ay,Bx,By:real;
dxMax,dxMin,dyMax,dyMin:real;
xStep,yStep:real;
i,j,xCount,yCount:integer;
xNew,yNew,xOld,yOld:integer;
Show:boolean;
const
Big=9.999999E+10;
Margin=0.1;
procedure FindEyeCoordinates(x,y: real; var dx,dy:real;
Theta,Phi,Rad,D:real);
var
z,xx,yy,zz:real;
begin
z:=Fun(x,y);

```

```

xx:=-x*sin(Theta)+y*cos(Theta);
yy:=-x*cos(Theta)*cos(Phi)-y*sin(Theta)*cos(Phi)+z*sin(Phi);
zz:=-x*cos(Theta)*sin(Phi)-y*sin(Theta)*sin(Phi)-z*cos(Phi)+Rad;
dx:=D*xx/zz;
dy:=D*yy/zz
end;
procedure FindScreenCoordinates(Ax,Ay,Bx,By,dx,dy:real;
var xNew,yNew:integer);
begin
xNew:=trunc(Ax+Bx*dx);
yNew:=GetMaxY-trunc(Ay+By*dy)
end;
procedure FindLimits(dx,dy:real;var dxMax,dxMin,dyMax,dyMin:real);
begin
if dx>dxMax then dxMax:=dx;
if dx<dxMin then dxMin:=dx;
if dy>dyMax then dyMax:=dy;
if dy<dyMin then dyMin:=dy;
end;
procedure FindWindow (dxMax,dxMin,dyMax,dyMin :real; var
Ax,Ay,Bx,By:real);
var
xSize,ySize:real;
begin
xSize:=dxMax-dxMin;
ySize:=dyMax-dyMin;
dxMin:=dxMin-Margin*xSize;
dyMin:=dyMin-Margin*ySize;
dxMax:=dxMax+Margin*xSize;
dyMax:=dyMax+Margin*ySize;
Bx:=GetMaxX/(dxMax-dxMin);
By:=GetMaxY/(dyMax-dyMin);
Ax:=-dxMin*Bx;
Ay:=-dyMin*By
end;
begin
xCount:=20;
yCount:=20;
line(0,0,GetMaxX,0);
line(GetMaxX,GetMaxY,0,GetMaxY);
line(0,0,0,GetMaxY);
line(GetMaxX,0,GetMaxX,GetMaxY);
xStep:=(xMax-xMin)/xCount;
yStep:=(yMax-yMin)/yCount;
dxMin:=Big;
dxMax:=-Big;
dyMin:=Big;
dyMax:=-Big;
for Show:=false to true do
begin
for i:=0 to xCount do
begin
x:=xMin+i*xStep;
y:=yMin;

```

```

FindEyeCoordinates(x,y,dx,dy,Theta,Phi,Rad,D);
if Show then
begin
    FindScreenCoordinates(Ax,Ay,Bx,By,dx,dy,xNew,yNew);
    xOld:=xNew;
    yOld:=yNew;
    MoveTo(xOld,yOld)
end
else FindLimits(dx,dy,dxMax,dxMin,dyMax,dyMin);
for j:=0 to yCount do
begin
    y:=yMin+j*yStep;
    FindEyeCoordinates(x,y,dx,dy,Theta,Phi,Rad,D);
    if Show then
    begin
        FindScreenCoordinates(Ax,Ay,Bx,By,dx,dy,xNew,yNew);
        lineTo (xNew,yNew);
        xOld:=xNew;
        yOld:=yNew
    end
    else FindLimits(dx,dy,dxMax,dxMin,dyMax,dyMin)
end;
end;
if not Show then FindWindow(dxMax, dxMin, dyMax, dyMin, Ax, Ay, Bx, By)
end;
for i:=0 to yCount do
Begin
    y:=yMin+i*yStep;
    x:=xMin;
    FindEyeCoordinates(x,y,dx,dy,Theta,Phi,Rad,D);
    if show then
    begin
        FindScreenCoordinates(Ax,Ay,Bx,By,dx,dy,xNew,yNew);
        xOld:=xNew;
        yOld:=yNew;
        moveTo(xOld,yOld);
    end
    else
        FindLimits(dx,dx,dxMax,dxMin,dyMax,dyMin);
    for j:=0 to xCount do
    begin
        x:=xMin+j*xStep;
        FindEyeCoordinates(x,y,dx,dy,Theta,Phi,Rad,D);
        if Show then
        begin
            FindScreenCoordinates(Ax,Ay,Bx,By,dx,dy,xNew,yNew);
            LineTo(xNew,yNew);
            xOld:=xNew;
            yOld:=yNew
        end
        else
            FindLimits(dx,dy,dxMax,dxMin,dyMax,dyMin)
        end;
    end;
end;
end;

```

```
repeat until KeyPressed;  
end;  
end.
```

Ниже приведен пример использования модуля GRAPHIC.

```
uses graphic,graph,crt;  
function g1(x:real):real;far;  
begin  
g1:=sin(x);  
end;  
function g2(x:real):real;FAR;  
begin  
g2:=cos(X)  
end;  
function g3(x:real):real;FAR;  
begin  
g3:=cos(X)+0.6;  
end;  
function g4(x:real):real;FAR;  
begin  
g4:=cos(sin(X))  
end;  
function g(x,y:real):real;FAR;  
begin  
g:=sin(x)/(x*x+y*y+0.3)  
end;  
var color:array[1..10] of word;  
err:shortint;  
grdr,grmd:integer;  
i:integer;  
f1:array [1..8] of func;  
begin  
grdr:=detect;  
initgraph(grdr,grmd,'c:\bp\bgi');  
f1[1]:=g1;  
f1[2]:=g2;  
color[1]:=white;  
color[2]:=lightgray;  
color[3]:=darkgray;  
color[10]:=blue;  
grafik(-pi,pi,2,f1,40,20,getmaxx div 2-30,getmaxy,color,err);  
color[2]:=red;  
color[3]:=green;  
f1[1]:=g3; f1[2]:=g4;  
grafik(-2*pi,2*pi,2,f1,getmaxx div 2,20,getmaxx-20,getmaxy,color,err);  
delay(8000);  
readln;  
cleardevice;  
setbkcolor(white);  
setcolor(brown);  
surfase(-1,1,-1,1,-10,0.8,-0.5,5,g);  
readln;  
end.
```

В результате выполнения этой программы на экране дисплея будут нарисованы следующие два изображения (см. рис. 13.1, 13.2).

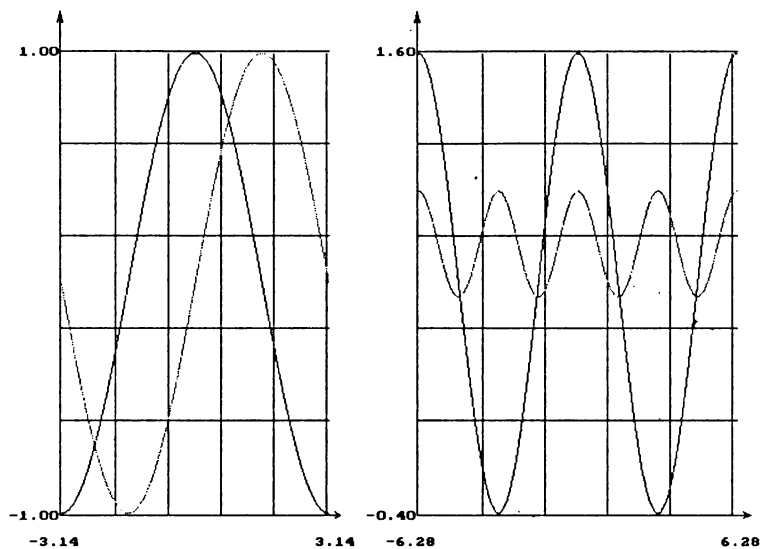


Рис. 13.1 ▼ График функций

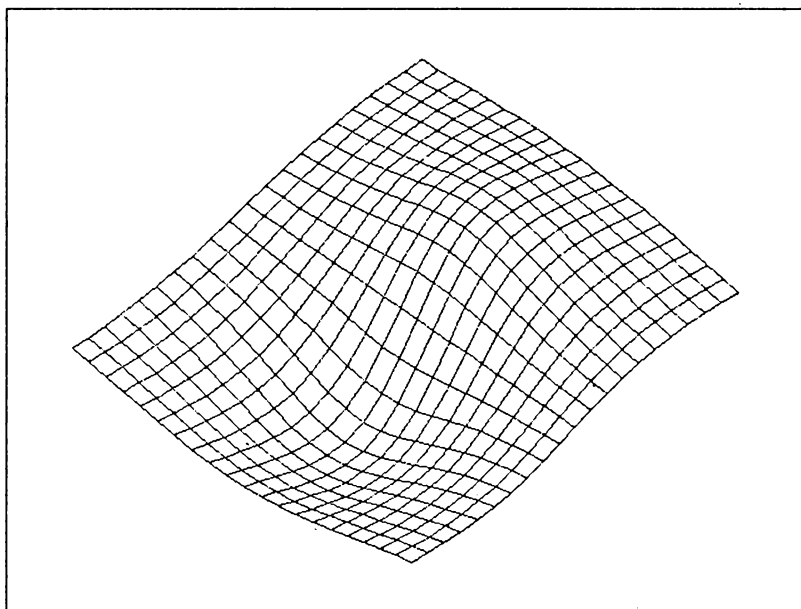


Рис 13.2 ▼ График поверхности

14 Глава

Решение задач линейной алгебры

В данной главе рассмотрены алгоритмы решения систем линейных алгебраических уравнений, вычисления определителя и обратной матрицы методом Гаусса [4, 6]. Приведено подробное математическое описание алгоритмов и даны блок-схемы.

14.1. Решение систем линейных алгебраических уравнений методом Гаусса

Пусть дана система линейных алгебраических уравнений (СЛАУ) с n неизвестными

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n \end{cases} \quad (14.1)$$

Обозначим через

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

матрицу коэффициентов системы (1), через

$$b = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{pmatrix} - \text{столбец ее свободных членов, и через}$$

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} - \text{столбец из неизвестных (искомый вектор). Тогда система (14.1)}$$

кратко может быть записана в виде матричного уравнения $Ax = b$.

Наиболее распространенным приемом решения систем линейных уравнений является алгоритм последовательного исключения неизвестных – *метод Гаусса*.

При решении систем линейных алгебраических уравнений этим методом всевозможные преобразования производят не над уравнениями системы (14.1), а над так называемой *расширенной матрицей* системы, которая получается путем добавления к основной матрице A столбца свободных членов b .

Первый этап решения системы уравнений, называемый *прямым ходом метода Гаусса*, заключается в приведении расширенной матрицы (14.2) к *треугольному виду*. Это означает, что все элементы матрицы (14.2) ниже главной диагонали должны быть равны нулю.

$$A' = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{pmatrix} \quad (14.2)$$

Для формирования первого столбца матрицы (14.3) необходимо из каждой строки (начиная со второй) вычесть первую, умноженную на некоторое число M .

$$A' = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} & b_1 \\ 0 & a_{22} & a_{23} & \dots & a_{2n} & b_2 \\ 0 & 0 & a_{33} & \dots & a_{3n} & b_3 \\ 0 & 0 & 0 & \dots & a_{4n} & b_4 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & a_{nn} & b_n \end{pmatrix} \quad (14.3)$$

В общем виде этот процесс можно записать так:

$$2\text{-я строка} = 2\text{-я строка} - M \times 1\text{-я строка}$$

$$3\text{-я строка} = 3\text{-я строка} - M \times 1\text{-я строка}$$

...

$$i\text{-я строка} = i\text{-я строка} - M \times 1\text{-я строка}$$

...

$$n\text{-я строка} = n\text{-я строка} - M \times 1\text{-я строка}$$

Понятно, что преобразование элементов второй строки будет происходить по формулам:

$$a_{21} = a_{21} - Ma_{11} \quad a_{22} = a_{22} - Ma_{12} \quad \dots \quad a_{2i} = a_{2i} - Ma_{1i} \quad \dots \quad a_{2n} = a_{2n} - Ma_{1n} \quad b_2 = b_2 - Mb_1.$$

Так как целью данных преобразований является обнуление первого элемента строки, то M выбирается из условия:

$$a_{21} - Ma_{11} = 0.$$

Следовательно,

$$M = \frac{a_{21}}{a_{11}}.$$

Элементы третьей строки и коэффициент M можно рассчитать аналогично:

$$a_{31} = a_{31} - Ma_{11} \quad a_{32} = a_{32} - Ma_{12} \quad \dots \quad a_{3i} = a_{3i} - Ma_{1i} \quad \dots \quad a_{3n} = a_{3n} - Ma_{1n} \quad b_3 = b_3 - Mb_1,$$

$$a_{31} - Ma_{11} = 0 \Rightarrow M = \frac{a_{31}}{a_{11}}.$$

Таким образом, преобразование элементов i -й строки будет происходить следующим образом:

$$a_{i1} = a_{i1} - Ma_{11} \quad a_{i2} = a_{i2} - Ma_{12} \quad \dots \quad a_{ii} = a_{ii} - Ma_{1i} \quad \dots \quad a_{in} = a_{in} - Ma_{1n} \quad b_i = b_i - Mb_1.$$

Коэффициент M для i -й строки выбирается из условия

$$a_{i1} - Ma_{11} = 0$$

и равен

$$M = \frac{a_{i1}}{a_{11}}.$$

После проведения подобных преобразований для всех строк матрица (14.2) примет вид

$$A' = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ 0 & a_{22} & \dots & a_{2n} & b_2 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & a_{n2} & \dots & a_{nn} & b_n \end{pmatrix}.$$

```

graph TD
    1{{1  
K = 1, n}} --> 2{{2  
i = k + 1, n}}
    2 --> 3[M = a_k / a_kk]
    3 --> 4{{4  
j = k, n}}
    4 --> 5[a_ij = a_ij - M a_kj]
    5 --> 6[b_i = b_i - M b_k]
    6 --> 1
    4 --> 2
    2 --> Exit(( ))
    1 --> Exit
    
```

Заметим, что если в матрице (14.2) на главной диагонали встретится элемент a_{ii} , равный нулю, то расчет коэффициента

$$M = \frac{a_{ik}}{a_{kk}}$$

В результате выполнения прямого хода метода Гаусса матрица (14.2) преобразуется в матрицу (14.3), а система уравнений (14.1) будет иметь следующий вид:

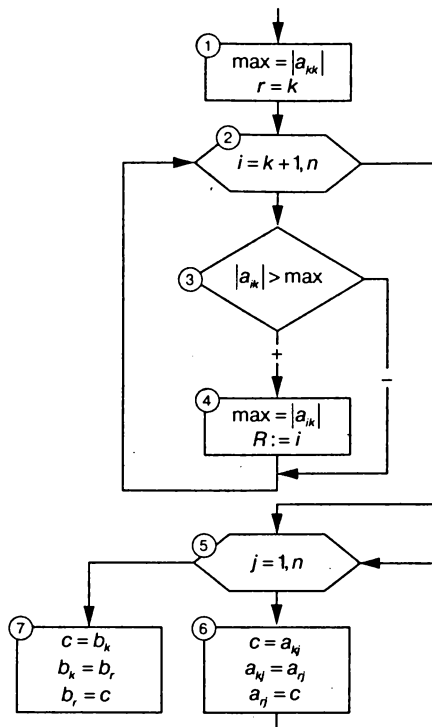


Рис. 14.2 ▼ Блок-схема алгоритма перестановки строк расширенной матрицы

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \dots + a_{1n}x_n = b_1 \\ a_{22}x_2 + a_{13}x_3 + \dots + a_{2n}x_n = b_2 \\ a_{13}x_3 + \dots + a_{2n}x_n = b_3 \\ \\ a_{nn}x_n = b_n \end{cases} \quad (14.4)$$

Решение системы (14.4) называют *обратным ходом метода Гаусса*.
Последнее n -е уравнение системы (14.4) имеет вид:

$$a_{nn}x_n = b_n.$$

Тогда, если

$$a_{nn} \neq 0,$$

то

$$x_n = \frac{b_n}{a_{nn}}.$$

В случае, если

$$a_{nn} = 0 \text{ и } b_n = 0,$$

то система (14.4), а следовательно, и система (14.1) имеют бесконечное множество решений.

При

$$a_{nn} = 0 \text{ и } b_n \neq 0$$

система (14.4), а значит, и система (14.1) решения не имеет.

Предпоследнее $(n-1)$ -е уравнение системы (14.4) имеет вид

$$a_{n-1,n-1}x_{n-1} + a_{n-1,n}x_n = b_{n-1}.$$

Значит,

$$x_{n-1} = \frac{b_{n-1} - a_{n-1,n}x_n}{a_{n-1,n-1}}.$$

Следующее $(n-2)$ -е уравнение системы (14.4) будет выглядеть так:

$$a_{n-2,n-2}x_{n-2} + a_{n-2,n-1}x_{n-1} + a_{n-2,n}x_n = b_{n-2}.$$

Отсюда имеем

$$x_{n-2} = \frac{b_{n-2} - a_{n-2,n-1}x_{n-1} - a_{n-2,n}x_n}{a_{n-2,n-2}}$$

или

$$x_{n-2} = \frac{b_{n-2} - (a_{n-2,n-1}x_{n-1} + a_{n-2,n}x_n)}{a_{n-2,n-2}} = \frac{b_{n-2} - \sum_{j=n-1}^n a_{n-2,j}x_j}{a_{n-2,n-2}}.$$

Таким образом, формула для вычисления i -го значения x будет иметь вид:

$$x_i = \frac{b_i - \sum_{j=i+1}^n a_{ij}x_j}{a_{ii}}.$$

Алгоритм, реализующий обратный ход метода Гаусса, представлен в виде блок-схемы на рис. 14.3.

Объединив блок-схемы, изображенные на рис. 14.1, 14.2 и 14.3, получим общую блок-схему метода Гаусса (рис. 14.4). Блоки 2–6 содержат последовательный ввод данных, где n – это размерность системы линейных алгебраических уравнений, а сама система задается в виде матрицы коэффициентов при неизвестных A и вектора свободных коэффициентов b . Блоки 7–19 предусматривают

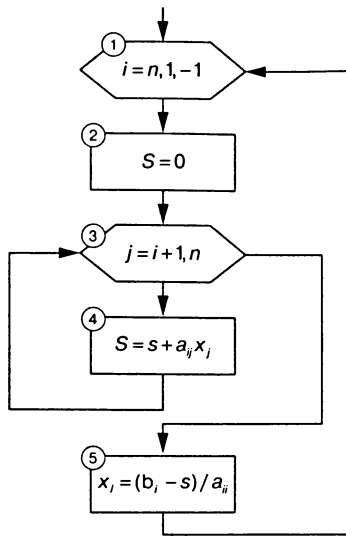


Рис. 14.3 ▼ Блок-схема алгоритма обратного хода метода Гаусса

прямой ход метода Гаусса, а блоки 20–29 – обратный. Для вывода результатов предусмотрено несколько блоков вывода. Если результат проверки условий 20 и 21 положительный, то выдается сообщение о том, что система имеет бесконечное множество решений (блок 22). Если условие 20 выполняется, а 21 – нет, то появляется сообщение о том, что система не имеет решений (блок 23). Сами же решения системы уравнений, представленные вектором x , вычисляются (блоки 24–28) и выводятся на печать (блок 29) только в случае невыполнения условия 20.

14.2. Вычисление обратной матрицы методом Гаусса

Один из методов вычисления *обратной матрицы* основан на решении систем линейных алгебраических уравнений. Пусть задана некоторая матрица A :

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \quad (14.5)$$

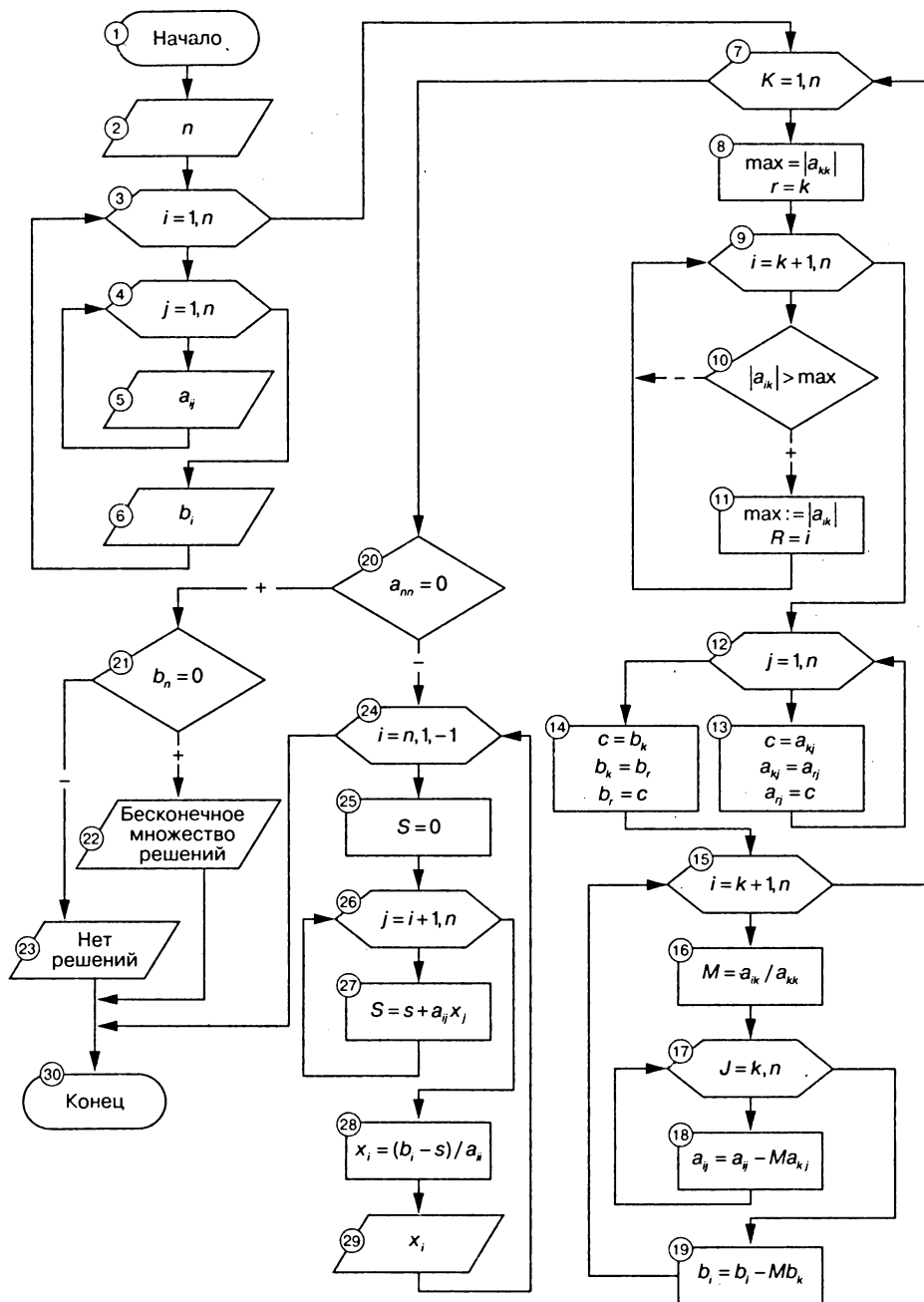


Рис. 14.4 ▼ Блок-схема алгоритма решения СЛАУ методом Гаусса

Необходимо найти матрицу A^{-1} , которая является обратной к матрице A :

$$A^{-1} = \begin{pmatrix} y_{11} & y_{12} & \dots & y_{1n} \\ y_{21} & y_{22} & \dots & y_{2n} \\ \dots & \dots & \dots & \dots \\ y_{n1} & y_{n2} & \dots & y_{nn} \end{pmatrix} \quad (14.6)$$

Матрица (14.6) будет обратной к матрице (14.5), если выполняется соотношение

$$A \cdot A^{-1} = E,$$

где E – это единичная матрица, или более подробно:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \times \begin{pmatrix} y_{11}y_{12}\dots y_{1n} \\ y_{21}y_{22}\dots y_{2n} \\ y_{31}y_{32}\dots y_{3n} \\ \dots & \dots & \dots & \dots \\ y_{n1}y_{n2}\dots y_{nn} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (14.7)$$

Результат перемножения матриц из соотношения (14.7) можно представить поэлементно в виде n -го числа систем линейных уравнений. Умножение матрицы (14.5) на первый столбец матрицы (14.6) даст первый столбец единичной матрицы:

$$\left. \begin{aligned} a_{11}y_{11} + a_{12}y_{21} + \dots + a_{1n}y_{n1} &= 1 \\ a_{21}y_{11} + a_{22}y_{21} + \dots + a_{2n}y_{n1} &= 0 \\ \dots & \dots \\ a_{n1}y_{11} + a_{n2}y_{21} + \dots + a_{nn}y_{n1} &= 0 \end{aligned} \right\}.$$

Система, полученная в результате умножения матрицы (14.5) на i -й столбец матрицы (14.6), будет выглядеть следующим образом:

$$\left. \begin{aligned} a_{11}y_{1i} + a_{12}y_{2i} + \dots + a_{1n}y_{ni} &= 0 \\ a_{21}y_{1i} + a_{22}y_{2i} + \dots + a_{2n}y_{ni} &= 0 \\ \dots & \dots \\ a_{i1}y_{1i} + a_{i2}y_{2i} + \dots + a_{in}y_{ni} &= 1 \\ \dots & \dots \\ a_{n1}y_{1i} + a_{n2}y_{2i} + \dots + a_{nn}y_{ni} &= 0 \end{aligned} \right\}.$$

Понятно, что n -я система будет иметь вид:

[illegible]

Решением каждой из приведенных выше систем будет i -й столбец обратной матрицы. Количество систем равно размерности обратной матрицы. Для решения систем линейных алгебраических уравнений можно воспользоваться методом Гаусса.

Описанный алгоритм представлен в виде блок-схемы на рис. 14.5. Блоки 2–5 отражают формирование столбца единичной матрицы. Если условие 3 выполняется и элемент находится на главной диагонали, то он равен единице, все остальные элементы нулевые. В блоке 6 происходит вызов подпрограммы для решения системы уравнений методом Гаусса. В качестве параметров в эту подпрограмму передается исходная матрица A , сформированный в пунктах 2–5 вектор свободных коэффициентов B , размерность системы n . Вектор X будет решением i -ой системы уравнений и, следовательно, i -ым столбцом искомой матрицы Y .

14.3. Вычисление определителя методом Гаусса

Пусть задана матрица (14.2), необходимо вычислить ее *определитель*. Для этого матрицу необходимо преобразовать к треугольному виду (14.3), а затем воспользоваться свойством, известным из курса линейной алгебры, которое гласит, что определитель треугольной матрицы равен произведению ее диагональных элементов:

$$\det A = \prod_{i=1}^n a_{ii}.$$

Преобразование матрицы (14.2) к виду (14.3) можно осуществить с помощью прямого хода метода Гаусса. Алгоритм вычисления определителя матрицы, изображенный в виде блок-схемы на рис. 14.6, представляет собой алгоритм прямого хода метода Гаусса, в процессе выполнения которого проводится перестановка строк матрицы. Эта операция приводит к смене знака определителя. В блок-схеме момент смены знака отражен в блоках 8–9. В блоке 8 определяется, будут ли строки меняться местами, и если ответ утвердительный, то в блоке 9 происходит смена знака определителя. В блоках 17–18 выполняется непосредственное вычисление определителя путем перемножения диагональных элементов преобразованной матрицы.

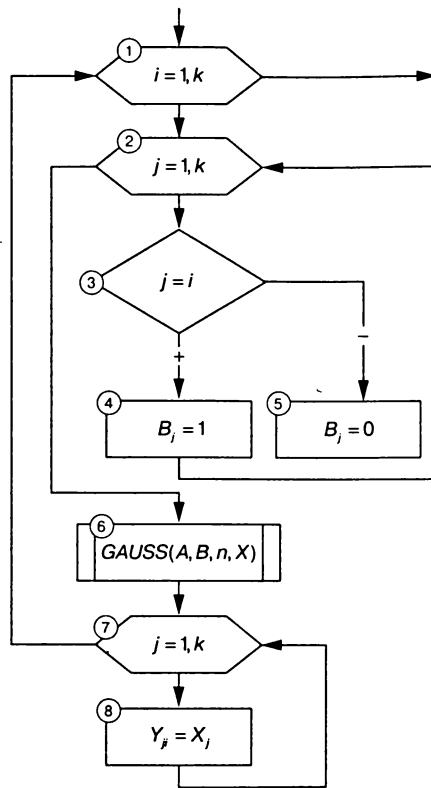


Рис. 14.5 ▼ Блок-схема алгоритма вычисления обратной матрицы

ПРИМЕР 14.1. Решить матричное уравнение $H \times X = F$, где $X = H^{-1} \times F$.

$$H = \begin{pmatrix} 1,8 & -3,8 & 0,7 & -3,7 \\ 0,7 & 2,1 & -2,6 & -2,8 \\ 7,3 & 8,1 & 1,7 & -4,9 \\ 1,9 & -4,3 & -4,9 & -4,7 \end{pmatrix}, F = \begin{pmatrix} 7,4 & 2,2 & -3,1 & 0,7 \\ 1,6 & 4,8 & -8,5 & 4,5 \\ 4,7 & 7,0 & -6,0 & 6,6 \\ 5,9 & 2,7 & 4,9 & -5,3 \end{pmatrix}.$$

```

type
  matrica=array [1..4,1..4] of real;
  massiv=array [1..4] of real;
const { исходные данные }
  H:matrica=((1.8,-3.8,0.7,-3.7),
    0.7,2.1,-2.6,-2.8),
    (7.3,8.1,1.7,-4.9),
    (1.9,-4.3,-4.9,-4.7));
  F:matrica=((7.4,2.2,-3.1,0.7),
  
```

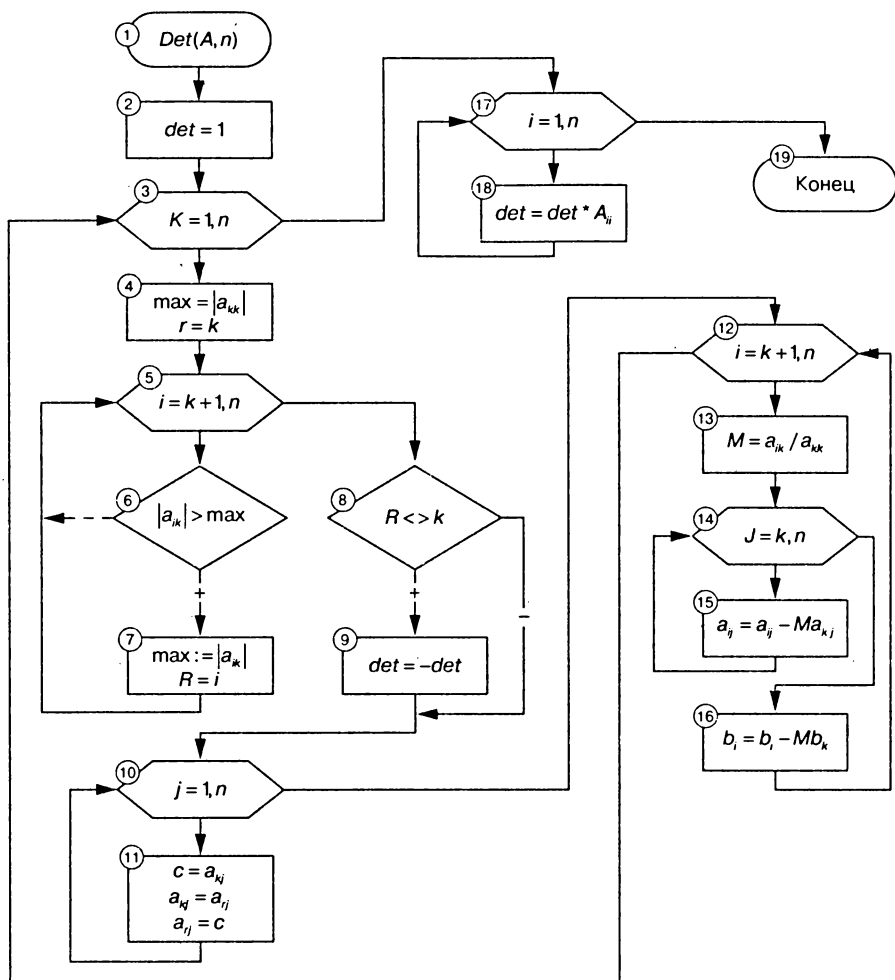


Рис. 14.6 ▼ Блок-схема алгоритма вычисления определителя

(1.6, 4.8, -8.5, 4.5),
 (4.7, 7.0, -6.0, 6.6),
 (5.9, 2.7, 4.9, -5.3));

var H_obr, { Обратная матрица. }

HF:matrica; { Решение матричного уравнения. }

b,y:massiv;

l,p,m:integer;

Pr:byte;S:real;

{ Решение СЛАУ методом Гаусса. }

{ Входные данные: a - матрица коэффициентов при неизвестных, }

{ b - массив свободных коэффициентов, }

```

{ n - размерность СЛАУ. }
{ Выходные данные: x - массив решений СЛАУ, Pr - признак существования }
{ решения. }
procedure gauss(a:matrica;b:massiv;n:integer;var x:massiv;var Pr:byte);
var i,j,k,r:integer; max,m,c,s:real;
begin
  { Прямой ход метода Гаусса. }
  for k:=1 to n do
  begin
    max:=abs(a[k,k]); { Пусть диагональный элемент - максимальный. }
    r:=k; { Номер строки, в которой он находится. }
    for i:=k+1 to n do
      if abs(a[i,k])>max then
      begin { Если в столбце найдется элемент, превышающий }
        max:=abs(a[i,k]); { максимум, сохранить его, a }
        r:=i; { также номер строки, в которой он находится. }
      end;
    for j:=1 to n do { Текущая строка меняется местами со }
      begin { строкой, содержащей максимальный элемент. }
        c:=a[k,j];
        a[k,j]:=a[r,j];
        a[r,j]:=c;
      end;
    c:=b[k];
    b[k]:=b[r];
    b[r]:=c;
    { Приведение расширенной матрицы к треугольному виду. }
    for i:=k+1 to n do
      begin
        m:=a[i,k]/a[k,k];
        for j:=k to n do
          a[i,j]:=a[i,j]-m*a[k,j];
        b[i]:=b[i]-m*b[k];
      end;
    end;
    { Обратный ход метода Гаусса. }
    if a[n,n]=0 then { Если последний диагональный элемент равен нулю }
    if b[n]=0 then { и если последний свободный коэфф. равен нулю, то }
    Pr:=1 { система не имеет решений, иначе, если последний }
    else Pr:=2 { свободный коэффициент не нулевой, то система имеет }
    else { бесконечное множество решений. }
    Begin { Если последний диагональный элемент не нулевой, то }
      Pr:=0; { переходим к определению решений СЛАУ. }
      x[n]:=b[n]/a[n,n];
      for i:=n downto 1 do
      begin
        s:=0;
        for j:=i+1 to n do
          s:=s+a[i,j]*x[j];
        x[i]:=(b[i]-s)/a[i,i];
      end;
    end;
  end;
end;

```

```

{ Вывод матрицы на печать в виде таблицы. }
procedure input_matr(A:matrica;n:integer);
var i,j:integer;
begin
  for i:=1 to n do
  begin
    for j:=1 to n do
      write(A[i,j]:1:3,' ');
    writeln;
  end;
end;
begin
  writeln('Матрица H'); input_matr(H,4);
  writeln('Матрица F'); input_matr(F,4);
  { Вычисление обратной матрицы. }
  for l:=1 to 4 do
  begin
    for p:=1 to 4 do
      if l=p then
        b[p]:=1
      else b[p]:=0;
    gauss(H,b,4,y,Pr); { Решение СЛАУ методом Гаусса. }
    if Pr<>0 then
      begin
        writeln('Обратная матрица не существует'); exit;
      end
    else
      for p:=1 to 4 do { Формирование обратной матрицы. }
        H_obr[p,L]:=y[p];
      end;
    writeln('Матрица обратная H'); input_matr(H_obr,4);
    { Умножение матриц. }
    for l:=1 to 4 do
      for p:=1 to 4 do
      begin
        S:=0;
        for m:=1 to 4 do
          S:=S+H_obr[l,m]*F[m,p];
        HF[l,p]:=S;
      end;
    writeln('Решение матричного уравнения HX=F');
    input_matr(HF,4);
  end.

```

14.4. Упражнения по теме

«Решение задач линейной алгебры»

Разработать блок-схему и написать программу на Турбо Паскале для решения следующих задач.

1. Решить матричное уравнение $XA = B$, где $X = BA^{-1}$

$$A = \begin{bmatrix} 2 & 1 & 3 \\ 1 & -2 & 0 \\ 4 & -3 & 0 \end{bmatrix}, B = \begin{bmatrix} 22 & -14 & 3 \\ 6 & 7 & 0 \\ 11 & 3 & 15 \end{bmatrix}.$$

2. Определить, имеет ли система уравнений решение и, если имеет, сколько.

$$\begin{cases} 0,34x_1 + 0,71x_2 + 0,63x_3 = 2,08; \\ 0,71x_1 - 0,65x_2 - 0,18x_3 = 0,17; \\ 1,17x_1 - 2,35x_2 + 0,75x_3 = 1,28. \end{cases}$$

3. Проверить, образуют ли базис векторы

$$f_1 = \begin{bmatrix} 1 \\ -2 \\ 1 \\ 1 \end{bmatrix}, f_2 = \begin{bmatrix} 2 \\ -1 \\ 1 \\ -1 \end{bmatrix}, f_3 = \begin{bmatrix} 5 \\ -2 \\ -3 \\ 1 \end{bmatrix}, f_4 = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}.$$

Для решения задачи необходимо найти определитель матрицы F , состоящей из столбцов f_1, f_2, f_3, f_4 . Если определитель отличен от нуля, то заданные векторы образуют базис.

4. Решить СЛАУ:

$$\begin{pmatrix} 0,42 & 0,26 & 0,33 & -0,22 \\ 0,74 & -0,55 & 0,28 & -0,65 \\ 0,88 & 0,42 & -0,33 & 0,75 \\ 0,92 & 0,82 & -0,62 & 0,75 \end{pmatrix} \cdot X = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

5. Решить СЛАУ $A^2X = Y^T$, где

$$A = \begin{bmatrix} 2 & 1 & 5 & 2 \\ 5 & 2 & 2 & 6 \\ 2 & 2 & 1 & 2 \\ 1 & 3 & 3 & 1 \end{bmatrix}, Y = [3 \ 1 \ 2 \ 1].$$

6. Дана квадратная матрица A . Матрица B получена из матрицы A по формуле

$$B_{ij} = \begin{cases} A_{ij}^2, & i = j \\ 2A_{ij}, & i \neq j \end{cases}.$$

Вычислить матрицу $C = 2(A + B^1) - A^T B$.

7. Задан вектор C , состоящий из n элементов. Сформировать матрицу A как произведение векторов C и C^T и матрицу B , элементы которой вычислить по формуле

$$B_{ij} = \frac{A_{ij}}{\max[A]}.$$

Решить матричное уравнение $XA = 3B - E$, где E – единичная матрица.

8. Проверить для матрицы H свойство ортогональности $H^T = H^{-1}$, если матрица H задана следующим образом:

$$H = E - \frac{vv^T}{|v|^2}, \quad v = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix},$$

E – единичная матрица.

9. Ортогональная матрица обладает следующими свойствами: модуль определителя ортогональной матрицы равен 1; сумма квадратов элементов любого столбца ортогональной матрицы равна 1; сумма произведений элементов любого столбца ортогональной матрицы на соответствующие элементы другого столбца равна 0. Проверить эти свойства для матриц

$$\begin{pmatrix} -2,00 & 3,01 & 0,12 & -0,11 \\ 2,92 & -0,17 & 0,11 & 0,22 \\ 0,66 & 0,52 & 3,17 & 2,11 \\ 3,01 & 0,42 & -0,27 & -0,15 \end{pmatrix} \text{ и } \begin{pmatrix} -2,00 & 2,92 & 0,66 & 3,01 \\ 2,92 & -2,00 & 0,11 & 0,22 \\ 0,66 & 0,11 & -2,00 & 2,11 \\ 3,01 & 0,22 & 2,11 & -2,00 \end{pmatrix}.$$

10. Даны векторы:

$$f_1 = \begin{bmatrix} 0,25 \\ 0,333 \\ 0,2 \\ 0,1 \end{bmatrix}, f_2 = \begin{bmatrix} 0,33 \\ 0,25 \\ 0,167 \\ 0,143 \end{bmatrix}, f_3 = \begin{bmatrix} 1,25 \\ -0,667 \\ 2,2 \\ 3,1 \end{bmatrix}, f_4 = \begin{bmatrix} -0,667 \\ 1,333 \\ 1,25 \\ -0,75 \end{bmatrix}.$$

Найти координаты вектора $z = [1 \ 1 \ 1 \ 1]^T$ в базисе заданных векторов, если это возможно. Для решения задачи необходимо решить СЛАУ $Fx = z$, где F – матрица, построенная на векторах f_1, f_2, f_3, f_4 .

15 Глава

Метод наименьших квадратов

Метод наименьших квадратов используется при обработке реальных количественных данных, полученных в результате всевозможных научных опытов, технических испытаний, астрономических, геодезических и т.п. наблюдений.

15.1. Постановка задачи

Пусть в результате эксперимента были получены некоторые данные, представленные в виде таблицы (табл. 15.1).

Таблица 15.1 ▼ Экспериментальные данные

x_1	x_1	x_2	x_3	x_4	x_5	x_6	x_7		x_n
y_1	y_1	y_2	y_3	y_4	y_5	y_6	y_7	...	y_n

Необходимо построить аналитическую зависимость, наиболее близко описывающую результаты эксперимента.

Идея метода наименьших квадратов [4] заключается в том, что функцию

$$Y = f(x, a_0, a_1, \dots, a_k)$$

необходимо подобрать таким образом, чтобы сумма квадратов отклонений измеренных значений y_i от расчетных Y_i была наименьшей (см. рис. 15.1):

$$S = \sum_{i=1}^n [y_i - f(x_i, a_0, a_1, \dots, a_k)]^2 \rightarrow \min \quad (15.1)$$

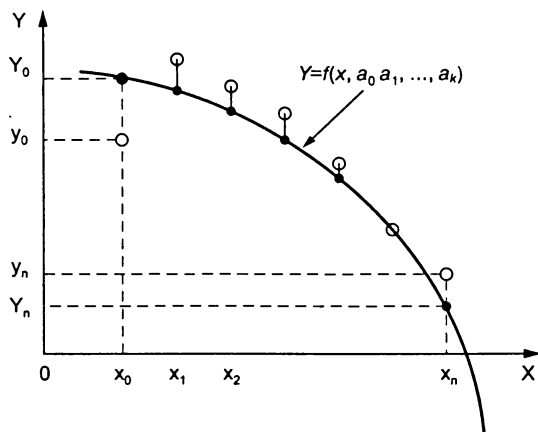


Рис. 15.1 ▼ Геометрическая интерпретация метода наименьших квадратов

Задача сводится к определению коэффициентов a_i из условия (15.1). Для ее решения необходимо составить систему уравнений

$$\left. \begin{aligned} \frac{\partial S}{\partial a_0} &= 0, \\ \frac{\partial S}{\partial a_1} &= 0, \\ \frac{\partial S}{\partial a_k} &= 0. \end{aligned} \right\}.$$

Если параметры a_i входят в зависимость $Y = f(x, a_0, a_1, \dots, a_k)$ линейно, то получим систему из $k + 1$ -линейного уравнения с $k + 1$ неизвестным:

$$\left. \begin{aligned} \sum_{i=1}^n (y_i - f(x_i, a_0, a_1, \dots, a_k)) \frac{\partial f}{\partial a_0} &= 0 \\ \sum_{i=1}^n (y_i - f(x_i, a_0, a_1, \dots, a_k)) \frac{\partial f}{\partial a_1} &= 0 \\ \sum_{i=1}^n (y_i - f(x_i, a_0, a_1, \dots, a_k)) \frac{\partial f}{\partial a_k} &= 0 \end{aligned} \right\} \quad (15.2)$$

Зная коэффициенты a_i , являющиеся решением системы (15.2), можно составить искомую функцию $Y = f(x, a_0, a_1, \dots, a_k)$.

15.2. Подбор параметров функций

Прежде чем говорить о реализации данной задачи при помощи программирования, выясним, как с точки зрения математики происходит подбор параметров наиболее часто встречающихся функций.

15.2.1. Линейная функция

Необходимо определить параметры функции $Y = a_0 + a_1 x$. Составим многочлен (15.1) для заданной функции:

$$S = \sum_{i=1}^n [y_i - a_0 - a_1 x_i]^2.$$

Сформируем систему линейных уравнений (15.2):

$$\left. \begin{aligned} 2 \sum_{i=1}^n (y_i - a_0 - a_1 x_i) \cdot (-1) &= 0 \\ 2 \sum_{i=1}^n (y_i - a_0 - a_1 x_i) \cdot (-x_i) &= 0 \end{aligned} \right\} \Rightarrow \left. \begin{aligned} a_0 \cdot n + a_1 \sum_{i=1}^n x_i &= \sum_{i=1}^n y_i \\ a_0 \sum_{i=1}^n x_i + a_1 \sum_{i=1}^n x_i^2 &= \sum_{i=1}^n y_i x_i \end{aligned} \right\},$$

решив которую, определим коэффициенты функции $Y = a_0 + a_1 x$.

$$a_0 = \frac{\sum_{i=1}^n y_i}{n} - a_1 \frac{\sum_{i=1}^n x_i}{n}, \quad a_1 = \frac{n \sum_{i=1}^n y_i x_i - \sum_{i=1}^n y_i \sum_{i=1}^n x_i}{n \sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i \right)^2} \quad (15.3)$$

15.2.2. Квадратичная функция

Необходимо определить параметры функции $Y = a_0 + a_1 x + a_2 x^2$. Составим функцию (15.1):

$$S = \sum_{i=1}^n [y_i - a_0 - a_1 x_i - a_2 x_i^2]^2.$$

После дифференцирования система уравнений (15.2) примет вид:

$$\left. \begin{aligned} n a_0 + a_1 \sum_{i=1}^n x_i + a_2 \sum_{i=1}^n x_i^2 &= \sum_{i=1}^n y_i \\ a_0 \sum_{i=1}^n x_i + a_1 \sum_{i=1}^n x_i^2 + a_2 \sum_{i=1}^n x_i^3 &= \sum_{i=1}^n y_i x_i \\ a_0 \sum_{i=1}^n x_i^2 + a_1 \sum_{i=1}^n x_i^3 + a_2 \sum_{i=1}^n x_i^4 &= \sum_{i=1}^n y_i x_i^2 \end{aligned} \right\} \quad (15.4)$$

Решив систему (15.4), найдем значение параметров a_0, a_1, a_2 .

15.2.3. Кубическая функция

Необходимо определить параметры многочлена третьей степени:

$$Y = a_0 + a_1 x + a_2 x^2 + a_3 x^3.$$

Составим функцию S:

$$S = \sum_{i=1}^n [y_i - a_0 - a_1 x_i - a_2 x_i^2 - a_3 x_i^3]^2.$$

Система уравнений для вычисления параметров a_0, a_1, a_2, a_3 примет вид:

$$\left. \begin{aligned} a_0 n + a_1 \sum_{i=1}^n x_i + a_2 \sum_{i=1}^n x_i^2 + a_3 \sum_{i=1}^n x_i^3 &= \sum_{i=1}^n y_i \\ a_0 \sum_{i=1}^n x_i + a_1 \sum_{i=1}^n x_i^2 + a_2 \sum_{i=1}^n x_i^3 + a_3 \sum_{i=1}^n x_i^4 &= \sum_{i=1}^n y_i x_i \\ a_0 \sum_{i=1}^n x_i^2 + a_1 \sum_{i=1}^n x_i^3 + a_2 \sum_{i=1}^n x_i^4 + a_3 \sum_{i=1}^n x_i^5 &= \sum_{i=1}^n y_i x_i^2 \\ a_0 \sum_{i=1}^n x_i^3 + a_1 \sum_{i=1}^n x_i^4 + a_2 \sum_{i=1}^n x_i^5 + a_3 \sum_{i=1}^n x_i^6 &= \sum_{i=1}^n y_i x_i^3 \end{aligned} \right\} \quad (15.5)$$

15.2.4. Полином k -й степени

Необходимо определить параметры многочлена k -й степени:

$$Y = a_0 + a_1 x + a_2 x^2 + \dots + a_k x^k.$$

Тогда система уравнений для определения параметров a принимает вид:

$$\left. \begin{aligned} a_0 n + a_1 \sum_{i=1}^n x_i + a_2 \sum_{i=1}^n x_i^2 + \dots + a_k \sum_{i=1}^n x_i^k &= \sum_{i=1}^n y_i \\ a_0 \sum_{i=1}^n x_i + a_1 \sum_{i=1}^n x_i^2 + a_2 \sum_{i=1}^n x_i^3 + \dots + a_k \sum_{i=1}^n x_i^{k+1} &= \sum_{i=1}^n x_i y_i \\ a_0 \sum_{i=1}^n x_i^k + a_1 \sum_{i=1}^n x_i^{k+1} + \dots + a_k \sum_{i=1}^n x_i^{2k} &= \sum_{i=1}^n x_i^k y_i \end{aligned} \right\} \quad (15.6)$$

15.2.5. Функции, приводимые к линейной

Для вычисления параметров функции $Y = ax^b$ необходимо выполнить некоторые арифметические преобразования:

$$\ln(y) = \ln(ax^b) = \ln(a) + b \ln(x)$$

и сделать замену

$$Y = \ln(y), X = \ln(x), A = \ln(a),$$

которая приведет заданную функцию к линейному виду $Y = A + bX$, где коэффициенты A и b вычисляются по формулам (15.3) и, соответственно, $a = e^A$.

Аналогично можно подобрать параметры функции вида $Y = ae^{bx}$. Прологарифмируем заданную функцию:

$$\ln(y) = \ln(a) + bx \ln(e) \Rightarrow \ln(y) = \ln(a) + bx.$$

Проведем замену $Y = \ln(y)$, $A = \ln(a)$ и получим линейную зависимость $Y = bx + A$. По формулам (15.3) найдем A и b , а затем вычислим $a = e^A$.

Ниже, в табл. 15.2, приведены примеры еще нескольких функций, которые сводятся к линейной функции элементарными заменами.

Таблица 15.2 ▽ Преобразование функций $Y = f(x, a, b)$ к виду $Y = Ax + b$

Функция	Замена
$y = \frac{1}{ax + b}$	$Y = \frac{1}{y}$
$y = \frac{x}{ax + b}$	$Y = \frac{1}{y}; X = \frac{1}{x}$
$y = \frac{1}{ae^{-x} + b}$	$Y = \frac{1}{y}, X = e^{-x}$

15.2.6. Подбор параметров функции $y = ax^b e^{cx}$

Прологарифмируем выражение $Y = ax^b e^{cx}$:

$$\ln(y) = \ln(a) + b \ln(x) + cx \ln(e).$$

Сделаем замену $Y = \ln(y)$, $A = \ln(a)$:

$$Y = A + b \ln(x) + cx.$$

Составим функцию по формуле (15.1):

$$S(A, b, C) = \sum_{i=1}^n (y_i - A - b \ln(x_i) - cx_i)^2.$$

После дифференцирования получим систему трех линейных алгебраических уравнений для определения коэффициентов A , b и c .

$$\left. \begin{aligned} nA + b \sum_{i=1}^n \ln(x_i) + c \sum_{i=1}^n x_i &= \sum_{i=1}^n y_i \\ A \sum_{i=1}^n \ln(x_i) + b \sum_{i=1}^n (\ln(x_i))^2 + c \sum_{i=1}^n x_i \ln(x_i) &= \sum_{i=1}^n y_i \ln(x_i) \\ A \sum_{i=1}^n x_i + b \sum_{i=1}^n x_i \ln(x_i) + c \sum_{i=1}^n x_i^2 &= \sum_{i=1}^n y_i x_i \end{aligned} \right\} \quad (15.7)$$

Значение коэффициента a вычислим по формуле $a = e^A$.

15.2.7. Уравнение регрессии и коэффициент корреляции

Линия, описываемая уравнением вида $y = a_0 + a_1 x$, называется *линией регрессии* y на x , параметры a_0 и a_1 называются *коэффициентами регрессии* и определяются формулами (15.3).

Чем меньше величина

$$S = \sum_{i=1}^n (y_i - a_1 x_i - a_0)^2,$$

тем более обоснованно предположение, что экспериментальные данные описываются линейной функцией. Существует показатель, характеризующий тесноту линейной связи между x и y , называемый *коэффициентом корреляции* и рассчитываемый по формуле:

$$r = \frac{\sum_{i=1}^n (x_i - M_x)(y_i - M_y)}{\sqrt{\sum_{i=1}^n (x_i - M_x)^2 \sum_{i=1}^n (y_i - M_y)^2}}, \quad Mx = \frac{\sum_{i=1}^n x_i}{n}, \quad My = \frac{\sum_{i=1}^n y_i}{n} \quad (15.8)$$

Значение коэффициента корреляции удовлетворяет соотношению:

$$-1 \leq r \leq 1.$$

Чем меньше отличается абсолютная величина r от единицы, тем ближе к линии регрессии располагаются экспериментальные точки. Если коэффициент корреляции равен нулю, то это означает, что между x и y не существует линейной связи, но между ними может существовать зависимость, отличная от линейной.

Для того чтобы проверить, значимо ли отличается от нуля коэффициент корреляции, можно использовать *критерий Стьюдента*. Вычисленное значение критерия определяется по формуле:

$$t = r \sqrt{\frac{n-2}{1-r^2}}.$$

Значение t сравнивается со значением, взятым из *таблицы распределения Стьюдента* (табл.15.3) в соответствии с уровнем значимости p и числом степеней свободы $k = n - 2$. Если t больше табличного, то коэффициент корреляции значимо отличен от нуля.

Таблица 15.3 ▼ Распределение Стьюдента

Число степеней свободы, k	Уровень значимости, p						
	0,99	0,98	0,95	0,90	0,80	0,70	0,60
1	63,657	31,821	12,706	6,314	3,078	1,963	1,376
2	9,925	6,965	4,303	2,920	1,886	1,386	1,061
3	5,841	4,541	3,182	2,353	1,638	1,250	0,978
4	4,604	3,747	2,776	2,132	1,533	1,190	0,941
5	4,032	3,365	2,571	2,05	1,476	1,156	0,920
6	3,707	3,141	2,447	1,943	1,440	1,134	0,906
7	3,499	2,998	2,365	1,895	1,415	1,119	0,896
8	3,355	2,896	2,306	1,860	1,387	1,108	0,889
9	3,250	2,821	2,261	1,833	1,383	1,100	0,883
10	3,169	2,764	2,228	1,812	1,372	1,093	0,879
11	3,106	2,718	2,201	1,796	1,363	1,088	0,876
12	3,055	2,681	2,179	1,782	1,356	1,083	0,873
13	3,012	2,650	2,160	1,771	1,350	1,079	0,870
14	2,977	2,624	2,145	1,761	1,345	1,076	0,868
15	2,947	2,602	2,131	1,753	1,341	1,074	0,866
16	2,921	2,583	2,120	1,746	1,337	1,071	0,865
17	2,898	2,567	2,110	1,740	1,333	1,069	0,863
18	2,878	2,552	2,101	1,734	1,330	1,067	0,862
19	2,861	2,539	2,093	1,729	1,328	1,066	0,861
20	2,845	2,528	2,086	1,725	1,325	1,064	0,860
21	2,831	2,518	2,080	1,721	1,323	1,063	0,859
22	2,819	2,508	2,074	1,717	1,321	1,061	0,858
23	2,807	2,500	2,069	1,714	1,319	1,060	0,858
24	2,797	2,492	2,064	1,711	1,318	1,059	0,857
25	2,779	2,485	2,060	1,708	1,316	1,058	0,856
26	2,771	2,479	2,056	1,706	1,315	1,058	0,856
27	2,763	2,473	2,052	1,703	1,314	1,057	0,855
28	2,756	2,467	2,048	1,701	1,313	1,056	0,855
29	2,750	2,462	2,045	1,699	1,311	1,055	0,854
30	2,704	2,457	2,042	1,697	1,310	1,055	0,854
40	2,660	2,423	2,021	1,684	1,303	1,050	0,851
60	2,612	2,390	2,000	1,671	1,296	1,046	0,848
120	2,617	2,358	1,980	1,658	1,289	1,041	0,845
∞	2,576	2,326	1,960	1,645	1,282	1,036	0,842

Кроме того, таблицу распределений Стьюдента можно использовать для построения доверительного интервала, накрывающего изучаемую величину. Оценка в этом случае выполняется по формуле:

$$\delta = \sqrt{\sum_{i=1}^n (y_i - M_y)^2 \frac{1-r^2}{n-2} \left(1 + \frac{(\tau - M_x)^2}{\sum_{i=1}^n (x_i - M_x)^2} \right)},$$

где t – величина, для которой необходимо найти значение, используя метод наименьших квадратов. Тогда доверительным интервалом для t будет интервал:

$$[\tau - \gamma\delta; \tau + \gamma\delta],$$

где g – значение, определенное по таблице Стьюдента при надежности p и с числом степеней свободы $k = n - 2$.

15.2.8. Криволинейная корреляция

Обычно коэффициент корреляции r применяется только в тех случаях, когда между данными существует прямолинейная связь. Если же связь криволинейная, то пользуются *индексом корреляции*, который рассчитывается по формуле:

$$R = \sqrt{1 - \frac{\sum_{i=1}^n (y_i - Y_i)^2}{\sum_{i=1}^n (y_i - M_y)^2}}, \quad (15.9)$$

где y – экспериментальные значения, Y – теоретические значения, M_y – среднее значение y .

Индекс корреляции по своему абсолютному значению колеблется в пределах от 0 до 1. При функциональной зависимости индекс корреляции равен 1. При отсутствии связи $R = 0$. Если коэффициент корреляции r является мерой тесноты связи только для линейной формы связи, то индекс корреляции R – и для линейной, и для криволинейной. При прямолинейной связи коэффициент корреляции по своей абсолютной величине равен индексу корреляции: $|r| = R$.

15.3. Примеры

Для того чтобы понять, как на практике использовать метод наименьших квадратов, разберем несколько конкретных примеров.

ПРИМЕР 15.1. В «Основах химии» Д. И. Менделеева [7] приводятся данные о растворимости азотнокислого натрия NaNO_3 в зависимости от температуры воды. В 100 частях воды (табл. 15.4) растворяется следующее число условных частей NaNO_3 при соответствующих температурах:

Таблица 15.4 ▾ Данные о растворимости NaNO_3 в зависимости от температуры воды

0 °C	4 °C	10 °C	15 °C	21 °C	29 °C	36 °C	51 °C	68 °C
66,7	71,0	76,3	80,6	85,7	92,9	99,4	113,6	125,1

Температура $t = 32\text{ °C}$ не входит в наблюдавшиеся значения. Необходимо определить, какова будет растворимость азотнокислого натрия при этой температуре.

Предположим, что количественная сторона этого явления довольно точно описывается линейной зависимостью

$$Y(x) = a + bx,$$

где x – температура в градусах, а Y – растворимость в условных частях на 100 частей воды. Для того чтобы определить, чему будет равна растворимость азотнокислого натрия при температуре $t = 32\text{ °C}$, требуется определить коэффициенты линейной функции по формулам (15.3), а затем вычислить $Y(t)$.

Алгоритм решения задачи представляется достаточно простым, поэтому сразу приведем текст программы, реализующий его..

```
const
  n=9; { Количество экспериментальных данных. }
  t=32; { Заданное значение температуры. }
  Gamma=1.895; { Коэффициент из таблицы Стьюдента, p=0.9, k=7. }
type mass=array [1..n] of real;
const
  { Экспериментальные данные. }
  X:mass=(0,4,10,15,21,29,36,51,68);
  Y:mass=(66.7,71.0,76.3,80.6,85.7,92.2,99.4,113.6,125.1);
var
  i:integer; { Параметр цикла. }
  Sxy,Sx,Sy,Sxx,mx,my:real;
  a,b:real; { Коэффициенты линейной зависимости. }
  r:real; { Коэффициент корреляции. }
  Sigma:real; { Оценка погрешности вычислений. }
begin
  Sy:=0; Sx:=0; Sxy:=0; Sxx:=0;
  for i:=1 to n do
    begin
      Sx:=Sx+X[i]; { Сумма всех значений x. }
      Sy:=Sy+Y[i]; { Сумма всех значений y. }
      Sxy:=Sxy+X[i]*Y[i]; { Сумма произведений x на y. }
      Sxx:=Sxx+X[i]*X[i]; { Сумма квадратов x. }
    end;
  mx:=Sx/n; { Среднее значение x. }
  my:=Sy/n; { Среднее значение y. }
  b:=(n*Sxy-Sx*Sy)/(n*Sxx-Sx*Sx); { Коэффициент при x. }
  a:=my-b*mx; { Свободный коэффициент. }
  Sxy:=0;
  Sy:=0; Sx:=0;
  for i:=1 to n do
    begin
```

```

Sxy:=Sxy+(X[i]-mx)*(Y[i]-my);
Sx:=Sx+sqr(X[i]-mx);
Sy:=Sy+sqr(Y[i]-my);
end;
r:=Sxy/sqrt(Sx*Sy);      { Коэффициент корреляции. }
Sigma:=sqrt(Sy*(1-r*r))/(n-2)*(1+sqr(t-mx)/sqr(Sx)); { Погрешность. }
writeln('Зависимость, подобранная методом наименьших квадратов:');
writeln('Y=',a:1:2,' + ',b:1:2,'*X');
writeln('Коэффициент корреляции r=',r:1:4);
write('При температуре', t)
writeln('градусов растворимость составляет', a+b*t:1:3);
writeln('Доверительный интервал');
writeln(['',a+b*t-Gamma*Sigma:1:3,',', a+b*t+Gamma*Sigma:1:3,']);
end.

```

В результате работы программы стало известно, что функция, описывающая заданную зависимость растворимости азотнокислого натрия NaNO_3 от температуры воды, имеет вид:

$$Y(x) = 67,44 + 0,87x.$$

Коэффициент корреляции достаточно близок к единице, $r = 0,9989$. При температуре $t = 32^\circ\text{C}$ растворимость азотнокислого натрия составит 95,287 условных частей. Доверительным интервалом для вычисленного значения $Y = 95,287$ будет интервал $[94,6; 96,0]$.

Недостатком данной программы является то, что она предназначена для расчета параметров конкретной зависимости. Для того чтобы эта программа стала более универсальной, ввод данных лучше организовать из текстового файла. В отдельный файл можно поместить и таблицу распределений Стьюдента, выбирая значения в зависимости от числа степеней свободы и надежности.

ПРИМЕР 15.2. Производится наблюдение над двумя переменными – процентным содержанием протеина x и крахмала y в зернах пшеницы. Обе переменные характеризуют качество пшеницы, но определение x требует сложного химического анализа, а определение y может быть сделано гораздо проще и без приборов. В табл. 15.5 [7] приведены результаты 20 наблюдений.

Таблица 15.5 ▼ Содержание протеина и крахмала в зернах пшеницы

Номера наблюдений	Содержание (в %)	
	протеин	крахмал
1	10,3	6
2	12,2	75
3	14,5	87
4	11,1	55
5	10,9	34
6	18,1	98
7	14,0	91
8	10,8	45
9	11,4	51
10	11,0	17

Таблица 15.5 ▼ Содержание протеина и крахмала в зернах пшеницы (окончание)

Номера наблюдений	Содержание (в %)	
	протеин	крахмал
11	10,2	36
12	17,0	97
13	13,8	74
14	10,1	24
15	14,4	85
16	15,8	96
17	15,6	92
18	15,0	94
19	13,3	84
20	19,0	99

Необходимо произвести выравнивание этих наблюдений по квадратичной и по кубической параболам, а затем сравнить результаты полученных вычислений.

Расчет коэффициентов квадратичной функции приведет к решению системы из трех уравнений с тремя неизвестными (15.4), а коэффициенты кубической функции можно рассчитать, решив систему из четырех уравнений с четырьмя неизвестными (15.5). Сравнивая системы (15.4) и (15.5), можно сделать вывод, что при программировании данной задачи удобно будет воспользоваться подпрограммой, которая выполняла бы расчет коэффициентов для полинома k -й степени. Используя формулу (15.6), составим блок-схему алгоритма этой подпрограммы (рис. 15.2). Входными данными для нее будут значения x и y , их количество n и k – степень искомого полинома. Результатом работы подпрограммы будет массив A , состоящий из значений коэффициентов полинома. В блоках 1–5 происходит формирование матрицы коэффициентов системы линейных уравнений. Формирование вектора свободных коэффициентов выполняется в блоках 6–9. Затем в блоке 10 происходит вызов подпрограммы решения системы линейных алгебраических уравнений методом Гаусса. Завершается подпрограмма выводом на печать массива коэффициентов (блоки 11–12).

Получив коэффициенты полинома, необходимо произвести выравнивание значений экспериментальной зависимости, то есть выполнить расчет теоретических значений по полученной формуле. Блок-схема, изображенная на рис. 15.3, отражает алгоритм функции `Polinom`, вычисляющей для некоторого заданного числа t значение полинома k -й степени с коэффициентами A .

Отметим также, что по ходу вычислений довольно часто приходится отыскивать различные степени. Поэтому целесообразным будет написать функцию для возведения вещественного числа z в целую степень m .

Кроме того, как отмечалось в предыдущем примере, данные удобнее хранить и передавать в программу из текстового файла. Поэтому создадим файл `dan.txt` следующей структуры: первая строка содержит количество экспериментальных данных n , вторая – значения протеина (y), третья – значения крахмала (x).

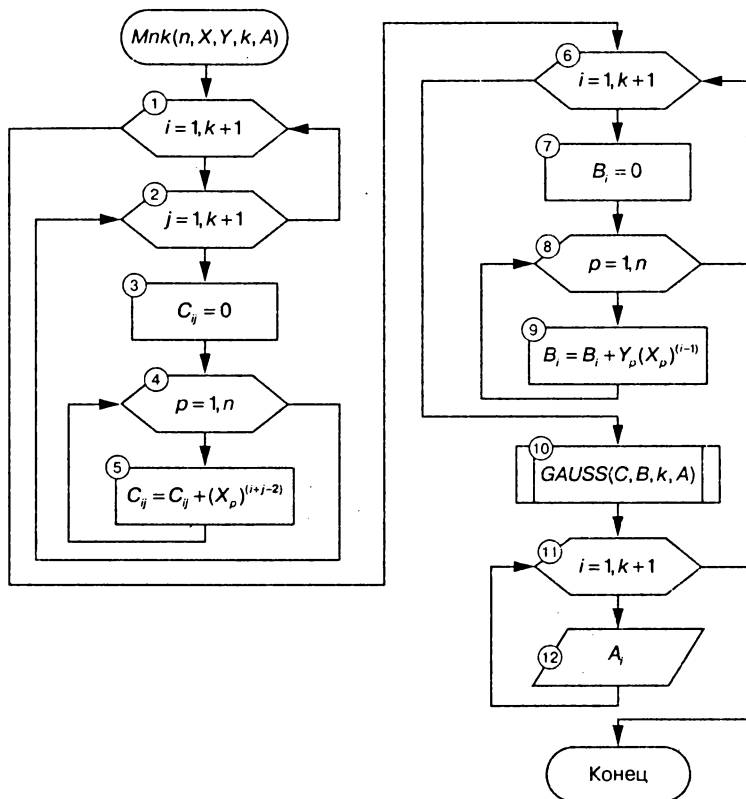


Рис. 15.2 ▼ Блок-схема алгоритма вычисления коэффициентов полинома k -й степени

Сравнительный анализ полученных зависимостей проведем, вычислив среднеквадратичные отклонения и индексы криволинейной корреляции.

```

uses crt;
type massiv=array [1..20] of real;
matr=array [1..20,1..20] of real;
var
  n_exp:byte;      { Количество экспериментальных данных. }
  x_exp,y_exp,    { Экспериментальные значения. }
  Yt_2,           { Теоретические значения для квадратичной параболы. }
  Yt_3:massiv;    { Теоретические значения для кубической параболы. }
  koff_2,         { Массив коэффициентов квадратичной параболы. }
  koff_3:massiv;  { Массив коэффициентов кубической параболы. }
  i:byte;
  f:text;         { Файловая переменная. }
  Sr_2,Sr_3,      { Суммарные ошибки. }
  Kor_2,Kor_3:real; { Коэффициенты корреляции. }
{ Функция возведения вещественного числа в целую степень. }
function step(z:real;m:byte):real;

```

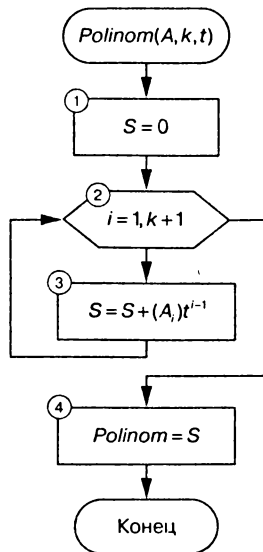


Рис. 15.3 ▼ Блок-схема алгоритма вычисления значений полинома k -й степени

```

var i:byte;S:real;
begin
  s:=1;
  for i:=1 to m do
    s:=s*z;
    step:=s;
  end;
  { Процедура решения системы линейных уравнений методом Гаусса. }
  procedure gauss(a:matr;b:massiv;n:integer;var x:massiv;var Pr:byte);
  var i,j,k,r:integer;
  max,m,c,s:real;
  begin
    for k:=1 to n do
      begin
        max:=abs(a[k,k]);
        r:=k;
        for i:=k+1 to n do
          if abs(a[i,k])>max then
            begin
              max:=abs(a[i,k]);
              r:=i;
            end;
        end;
        for j:=1 to n do
          begin
            c:=a[k,j];
            a[k,j]:=a[r,j];
            a[r,j]:=c;
          end;
        end;
      end;
    end;
  end;

```

```

c:=b[k];
b[k]:=b[r];
b[r]:=c;
for i:=k+1 to n do
begin
  m:=a[i,k]/a[k,k];
  for j:=k to n do
    a[i,j]:=a[i,j]-m*a[k,j];
    b[i]:=b[i]-m*b[k];
  end;
end;
if a[n,n]=0 then
if b[n]=0 then
  Pr:=1
else Pr:=2
else
begin
  Pr:=0;
  x[n]:=b[n]/a[n,n];
  for i:=n downto 1 do
begin
  s:=0;
  for j:=i+1 to n do
    s:=s+a[i,j]*x[j];
  x[i]:=(b[i]-s)/a[i,i];
end;
end;
end;
{ Процедура определения коэффициентов полинома }
{ методом наименьших квадратов. }
{ Входные параметры: x_ex,y_ex - массивы экспериментальных данных, }
{ n_ex - количество экспериментальных данных, k - степень полинома. }
{ Выходные параметры: X - массив коэффициентов полинома. }
procedure MNK(x_ex,y_ex:massiv;n_ex:byte; k:byte;var X:massiv);
var i,j,p:integer;
A:matr;
B:massiv;
pr:byte;
begin
  for i:=1 to k+1 do
  for j:=1 to k+1 do
begin
    A[i,j]:=0; { Формирование матрицы коэффициентов СЛАУ. }
    for p:=1 to n_ex do
      A[i,j]:=A[i,j]+step(x_ex[p], i+j-2);
    end;
  for i:=1 to k+1 do
begin ~
    B[i]:=0; { Формирование вектора свободных коэффициентов СЛАУ. }
    for p:=1 to n_ex do
      B[i]:=B[i]+step(x_ex[p], i-1)*y_ex[p];
    end;
  Gauss(A,B,k+1,X,pr); { Решение СЛАУ методом Гаусса. }

```

```

if pr=0 then { Если система имеет решение, то }
    { осуществляется вывод коэффициентов полинома. }
    for i:=1 to k+1 do
        writeln('X[' ,i-1,']= ',X[i]:1:7, ' ')
    else writeln('ERROR'); { Иначе - сообщение об ошибке. }
end;
{ Функция, определяющая значение полинома в заданной точке. }
{ На входе koeff - массив коэффициентов полинома, }
{ k - степень полинома, t - аргумент. }
{ На выходе - значение полинома в точке t. }
function polinom(koeff:massiv;k:byte;t:real):real;
var S:real;j:byte;
begin
    S:=0;
    for j:=1 to k+1 do
        S:=S+koeff[j]*step(t,j-1);
    polinom:=S;
end;
{ Функция расчета суммарной ошибки для экспериментального массива y_ex }
{ и массива теоретических значений Y_teor размерностью n_ex. }
function Sr_otkl(y_ex,Y_teor:massiv;n_ex:byte):real;
var S:real;j:byte;
begin
    S:=0;
    for j:=1 to n_ex do
        S:=S+(sqr(y_ex[j])-sqr(Y_teor[j]));
    Sr_otkl:=sqrt(S);
end;
{ Функция определения индекса криволинейной корреляции. }
function Korr_kr(y_ex,Y_teor:massiv;n_ex:byte):real;
var Sr,S1,S2:real;j:byte;
begin
    Sr:=0;
    for j:=1 to n_ex do
        Sr:=Sr+y_ex[j];
    Sr:=Sr/n_ex;
    S1:=0;
    for j:=1 to n_ex do
        S1:=S1+sqr(y_ex[j]-Y_teor[j]);
    S2:=0;
    for j:=1 to n_ex do
        S2:=S2+sqr(y_ex[j]-Sr);
    Korr_kr:=sqrt(1-S1/S2);
end;
begin
    clrscr;
    assign(f,'dan.txt');
    reset(f);
    { Чтение данных из файла. }
    readln(f,n_exp);
    writeln('n=',n_exp);
    for i:=1 to n_exp do
        read(f,y_exp[i]);

```

```

for i:=1 to n_exp do
read(f,x_exp[i]);
{ Расчет коэффициентов квадратичной параболы. }
MNK(x_exp,y_exp,n_exp,2,koff_2);
writeln;
{ Расчет коэффициентов кубической параболы. }
MNK(x_exp,y_exp,n_exp,3,koff_3);
for i:=1 to n_exp do { Расчет теоретических значений. }
begin
  Yt_2[i]:=polinom(koff_2,2,x_exp[i]); { для квадратного полинома }
  Yt_3[i]:=polinom(koff_3,3,x_exp[i]); { для кубического полинома }
end;
SR_2:=Sr_otkl(y_exp,Yt_2,n_exp); { Суммарные ошибки. }
SR_3:=Sr_otkl(y_exp,Yt_3,n_exp);
Kor_2:=Korr_kr(y_exp,Yt_2,n_exp); { Коэффициенты корреляции. }
Kor_3:=Korr_kr(y_exp,Yt_3,n_exp);
writeln('Таблица экспериментальных и теоретических значений');
writeln('_____');
writeln('| N | Y | X | Yt_2 | Yt_3 |');
writeln('_____');
for i:=1 to n_exp do
begin
  write('| ',i:2,' | ',x_exp[i]:6:2,' | ',y_exp[i]:6:2,' | ');
  writeln(Yt_2[i]:6:2,' | ',Yt_3[i]:6:2,' | ');
end;
writeln('_____');
writeln('Sr_2=',Sr_2:1:3);
writeln('Sr_3=',Sr_3:1:3);
writeln('Kor_2=',Kor_2:1:3);
writeln('Kor_3=',Kor_3:1:3);
end.

```

Итак, искомые полиномы имеют вид:

$$y = 11,5595 - 0,0879x + 0,00144x^2,$$

$$y = 9,721 + 0,0899x - 0,0025x^2 + 0,000249x^3.$$

Индексы криволинейной корреляции составили 0,946 и 0,963, а суммарная ошибка – 3,754 и 3,218. Оценивая полученные результаты, можно сделать вывод о том, что кубический полином лучше описывает экспериментальные данные, так как для него суммарная ошибка меньше, а индекс корреляции больше, чем для квадратичного полинома.

Программу можно усовершенствовать, добавив процедуру построения в одной графической области графиков экспериментальных и теоретических значений.

Глава 16

Интерполяция функций

Простейшая задача *интерполирования* [6] заключается в следующем. На отрезке $[a, b]$ заданы $n + 1$ точки $x_0, x_1, x_2, \dots, x_n$, которые называют *узлами интерполяции*, и значения некоторой функции $f(x)$ в этих точках

$$f(x_0) = y_0, f(x_1) = y_1, f(x_2) = y_2, \dots, f(x_n) = y_n \quad (16.1)$$

Требуется построить *интерполирующую функцию* $F(x)$, принадлежащую известному классу и принимающую в узлах интерполяции те же значения, что и $f(x)$:

$$F(x_0) = y_0, F(x_1) = y_1, F(x_2) = y_2, \dots, F(x_n) = y_n \quad (16.2)$$

Геометрически (рис. 16.1) это означает, что нужно найти кривую $y = F(x)$ определенного типа, проходящую через заданную систему точек $M_i(x_i, y_i)$ ($i = 0, 1, 2, \dots, n$).

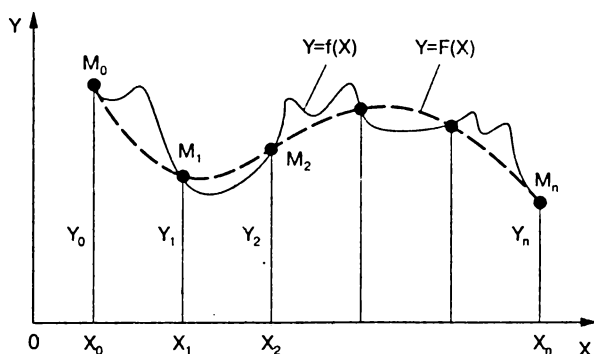


Рис. 16. 1 ▼ Геометрическая интерпретация интерполирования

В такой общей постановке задача может иметь бесчисленное множество решений или совсем не иметь решений. Однако эта задача становится однозначной, если вместо произвольной функции $F(x)$ искать полином $P_n(x)$ степени n , удовлетворяющий условиям (16.2), то есть такой, что

$$P_n(x_0) = y_0, P_n(x_1) = y_1, P_n(x_2) = y_2, \dots P_n(x_n) = y_n.$$

Полученную интерполяционную формулу $y = F(x)$ обычно используют для приближенного вычисления значений данной функции $f(x)$ для значений аргумента x , отличных от узлов интерполирования. Такая операция называется *интерполированием функции $f(x)$* . При этом различают *интерполирование в узком смысле*, когда $X \in [x_0, x_n]$, и *экстраполирование*, когда $X \in [x_0, x_n]$.

Рассмотрим некоторые наиболее часто используемые интерполяционные полиномы.

16.1. Канонический полином

Будем искать интерполирующую функцию $F(x)$ в виде канонического полинома степени n :

$$F(x) = P_n(x) = a_0 x^n + a_1 x^{n-1} + a_2 x^{n-2} + \dots + a_{n-1} x + a_n \quad (16.3)$$

Выбор многочлена степени n основан на том факте, что через $n + 1$ точку проходит единственная кривая степени n . Подставив (16.1) в (16.3), получим систему линейных алгебраических уравнений.

$$\begin{cases} a_0 x_0^n + a_1 x_0^{n-1} + a_2 x_0^{n-2} + \dots + a_{n-1} x_0 + a_n = y_0 \\ a_0 x_1^n + a_1 x_1^{n-1} + a_2 x_0^{n-2} + \dots + a_{n-1} x_1 + a_n = y_1 \\ \\ a_0 x_n^n + a_1 x_n^{n-1} + a_2 x_0^{n-2} + \dots + a_{n-1} x_n + a_n = y_n \end{cases} \quad (16.4)$$

Решая данную систему относительно переменных a_0, a_1, \dots, a_n , найдем аналитическое выражение интерполяционного полинома (16.3).

Итак, пусть функция $f(x)$ задана на отрезке $[x_0, x_n]$ точками $x_0, x_1, x_2, \dots, x_n$, в которых она принимает значения $y_0, y_1, y_2, \dots, y_n$. Необходимо вычислить значение этой функции в некоторой точке $T \in [x_0, x_n]$, используя интерполяционный канонический полином. Алгоритм решения этой задачи приведен в виде блок-схемы на рис. 16.2. Обратите внимание, что элементы массивов X и Y пронумерованы с 1 до $n + 1$. Это сделано для того, чтобы не менять индексы при вызове процедуры метода Гаусса (блок 7).

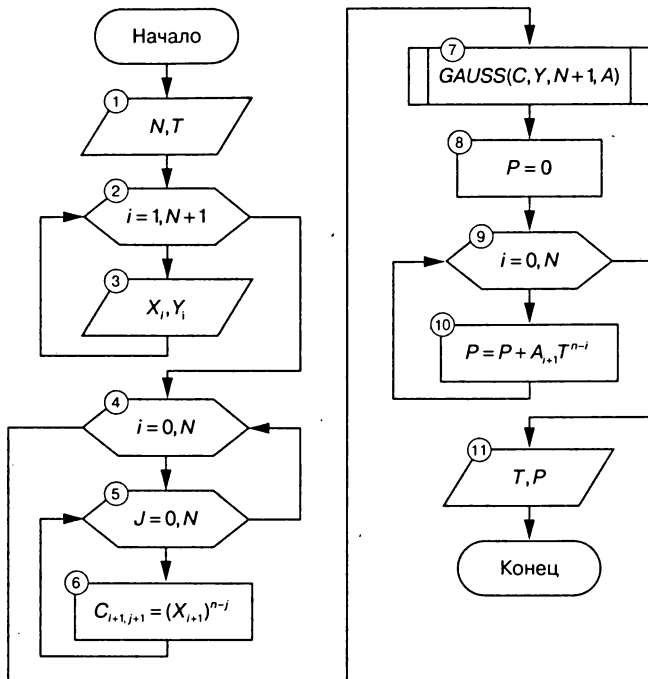


Рис. 16. 2 ▼ Алгоритм интерполирования с помощью канонического полинома

16.2. Полином Ньютона

И. Ньютон предложил интерполирующую функцию записать в виде следующего полинома n -й степени.

$$F(x) = A_0 + A_1 (x - x_0) + A_2 (x - x_0)(x - x_1) + \dots + A_n (x - x_0)(x - x_1) \dots (x - x_{n-1}) \quad (16.5)$$

Подставим $F(x_0) = y_0$ в (16.5), вычислив тем самым значение коэффициента A_0 :

$$A_0 = y_0.$$

Для вычисления A_1 воспользуемся следующим соотношением:

$$F(x_1) = A_0 + A_1 (x_1 - x_0) = y_1.$$

Отсюда коэффициент A_1 рассчитывается по формуле:

$$A_1 = \frac{y_0 - y_1}{x_0 - x_1} = y_{01},$$

где y_{01} – разделенная разность первого порядка – стремится к первой производной функции при $x_1 \rightarrow x_0$. Аналогичным образом вводятся разности первого порядка:

$$y_{02} = \frac{y_0 - y_2}{x_0 - x_2}, y_{03} = \frac{y_0 - y_3}{x_0 - x_3}, \dots, y_{0n} = \frac{y_0 - y_n}{x_0 - x_n}.$$

Подставим $F(x_2) = y_2$ в (16.5):

$$A_0 + A_1(x_2 - x_0) + A_2(x_2 - x_0)(x_2 - x_1) = y_2;$$

$$y_0 + y_{01}(x_2 - x_0) + A_2(x_2 - x_0)(x_2 - x_1) = y_2;$$

$$A_2(x_2 - x_0)(x_2 - x_1) = y_2 - y_0 - y_{01}(x_2 - x_0);$$

$$A_2 = \frac{y_2 - y_0}{(x_2 - x_0)(x_2 - x_1)} - \frac{y_{01}(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)};$$

$$A_2 = \frac{y_{02}}{(x_2 - x_1)} - \frac{y_{01}}{(x_2 - x_1)} = \frac{y_{01} - y_{02}}{(x_1 - x_2)} = y_{012},$$

где y_{012} – разделенная разность второго порядка – стремится к второй производной. Аналогично вводятся $y_{013}, y_{014}, \dots, y_{01n}$:

$$y_{013} = \frac{y_{01} - y_{03}}{(x_1 - x_3)}, y_{014} = \frac{y_{01} - y_{04}}{(x_1 - x_4)}, \dots, y_{01n} = \frac{y_{01} - y_{0n}}{(x_1 - x_n)}.$$

Подставив $F(x_3) = y_3$ в (16.5), получим:

$$A_3 = y_{0123} = \frac{y_{012} - y_{013}}{x_2 - x_3}.$$

Аналогично можно ввести коэффициенты:

$$y_{0124} = \frac{y_{012} - y_{014}}{(x_2 - x_4)}.$$

Этот процесс будем продолжать до тех пор, пока не вычислим

$$A_n = y_{012\dots n} = \frac{y_{012\dots n-1} - y_{012\dots n-2n}}{x_{n-1} - x_n}.$$

Полученные результаты запишем в табл. 16.1.

Таблица 16.1 ▼ Таблица разделенных разностей полинома Ньютона

x	$f(x)$	1	2	3	4	...	n
x_0	y_0						
x_1	y_1	y_{01}					
x_2	y_2	y_{02}	y_{012}				
x_3	y_3	y_{03}	y_{013}	y_{0123}			
x_4	y_4	y_{04}	y_{014}	y_{0124}	y_{01234}		
...
x_n	y_n	y_{0n}	y_{01n}	y_{012n}	y_{0123n}	...	$y_{012...n}$

В вычислении по формуле (16.5) будут участвовать только диагональные элементы таблицы (то есть коэффициенты A_j), а все остальные элементы таблицы являются промежуточными и нужны для формирования диагональных элементов. Исходя из вышеизложенного, составим алгоритм (рис. 16.3) решения следующей задачи: вычислить значение функции, заданной в виде таблицы, в некоторой точке $T \in [x_0, x_n]$, используя интерполяционный полином Ньютона.

16.3. Полином Лагранжа

Полином Лагранжа будем искать в виде:

$$F(x) = \sum_{i=1}^n y_i L_i(x),$$

где $L_i(x)$ – функция, удовлетворяющая в узлах x_k следующему свойству:

$$L_i(x_k) = \begin{cases} 1, i = k \\ 0, i \neq k \end{cases} \quad \text{или} \quad L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^n \frac{x - x_j}{x_i - x_j}.$$

Таким образом, полином Лагранжа выражается следующей формулой:

$$F(x) = \sum_{i=0}^n y_i \prod_{\substack{j=0 \\ j \neq i}}^n \frac{x - x_j}{x_i - x_j} \quad (16.6)$$

Алгоритм вычисления значения функции в заданной точке (рис. 16.4) достаточно прост и не нуждается в пояснении.

16.4. Сплайн-интерполяция

Полиномиальная интерполяция не всегда дает удовлетворительные результаты при аппроксимации зависимостей. Так, например, при представлении полиномами резонансных кривых колебательных систем большая погрешность возникает на концах этих кривых. Несмотря на выполнение условий в узлах,

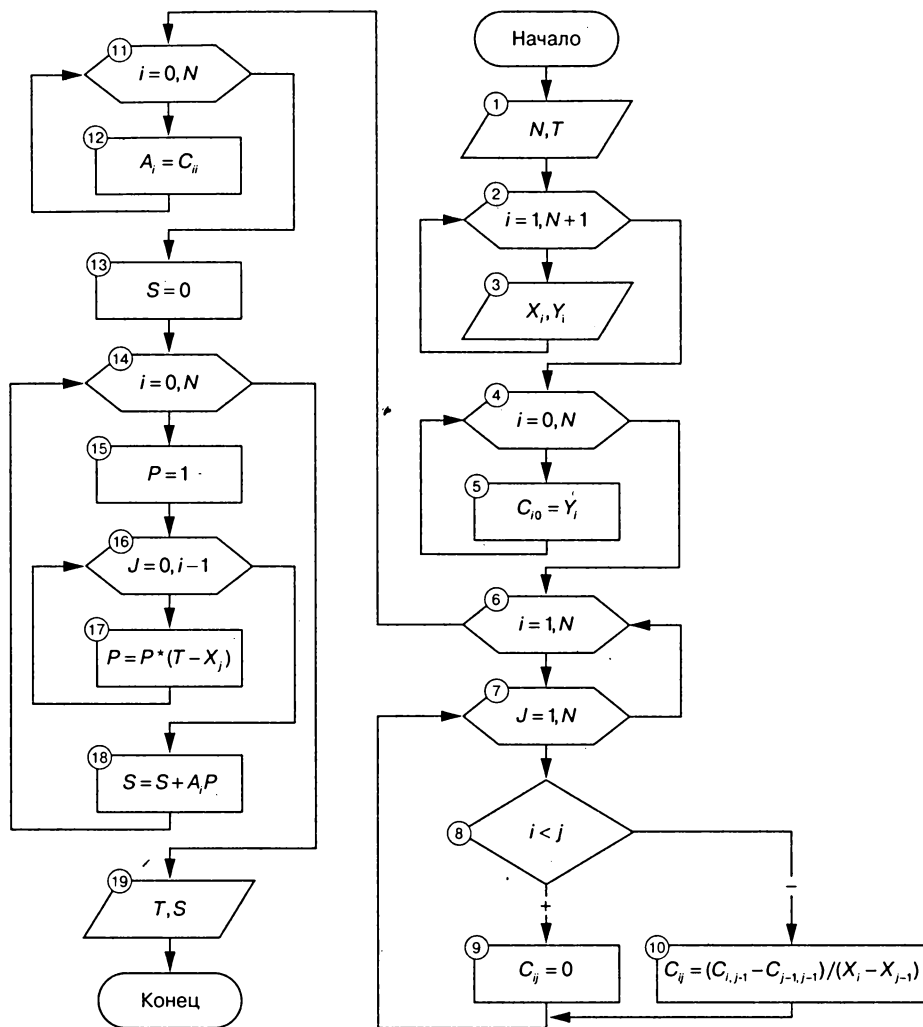


Рис. 16.3 ▼ Алгоритм интерполирования с помощью полинома Ньютона

интерполяционная функция может иметь значительное отклонение от аппроксимируемой кривой между узлами. При этом повышение степени интерполяционного полинома приводит не к уменьшению, а к увеличению погрешности. Решение этой проблемы предложено теорией сплайн-интерполяции (от английского слова spline – «рейка», «линейка»).

Для понимания алгоритма сплайн-интерполяции не обойтись без изложения хотя бы основ математической теории сплайн-интерполяции. Авторы

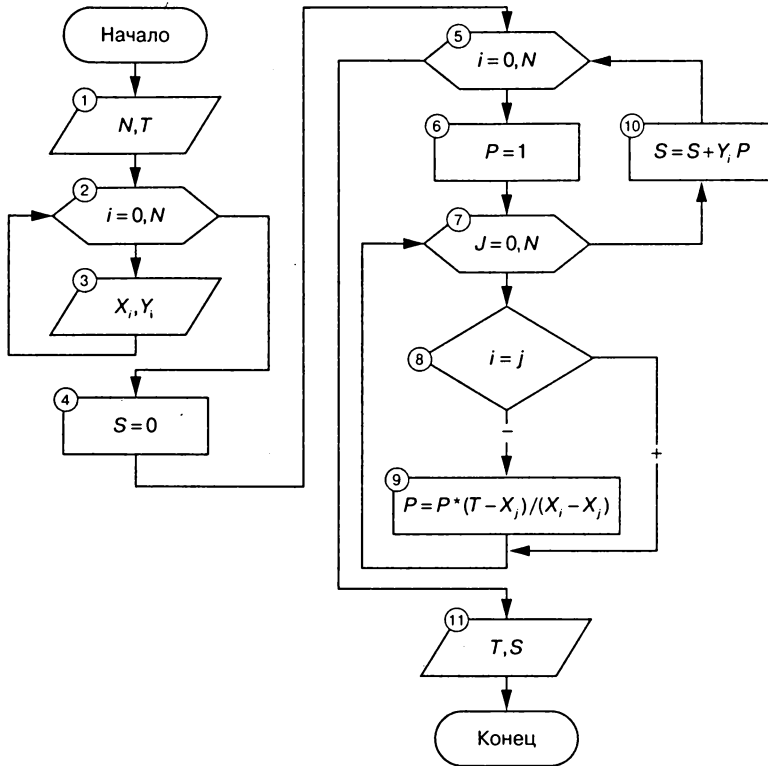


Рис. 16.4 ▼ Алгоритм интерполирования с помощью полинома Лагранжа

постарались не загружать читателя сложными математическими выкладками и для более глубокого изучения сплайн-интерполяции рекомендуют ознакомиться с прекрасной книгой В. М. Вержбицкого [4].

Рассмотрим один из наиболее распространенных вариантов интерполяции кубическими сплайнами [4]. Используя законы упругости, можно установить, что недеформируемая линейка между соседними углами проходит по линии, удовлетворяющей уравнению:

$$\varphi^{IV}(x) = 0 \quad (16.7)$$

Функцию $\varphi(x)$ будем использовать для интерполяции зависимости $y(x)$, заданной на интервале (a, b) в узлах $a = x_0, x_1, \dots, b = x_n$ значениями y_0, y_1, \dots, y_n .

Кубическим сплайном, интерполирующим на отрезке $[a, b]$ данную функцию $y(x)$, называется функция [4]:

$$g_k(s) = a_k + b_k(s - x_k) + c_k(s - x_k)^2 + d_k(s - x_k)^3, \quad s \in [x_{k-1}, x_k], \quad k = 1, \dots, n \quad (16.8)$$

Она удовлетворяет следующим условиям:

- $g_k(x_k) = y_k; g_k(x_{k-1}) = y_{k-1}$ (условие интерполяции в узлах сплайна);
- функция $g(x)$ дважды непрерывно дифференцируема на интервале $[a, b]$;
- на концах интервала функция g должна удовлетворять следующим соотношениям: $g_1''(a) = g_n''(b)$.

Для построения интерполяционного сплайна необходимо найти $4n$ коэффициентов a_k, b_k, c_k, d_k ($k = 1, 2, \dots, n$).

Из определения сплайна получаем $n + 1$ соотношение (16.9) [4]

$$g_1(x_0) = y_0; g_k(x_k) = y_k; k = 1, 2, \dots, n \quad (16.9)$$

Из условий гладкой стыковки звеньев сплайна (во внутренних узловых точках совпадают значения двух соседних звеньев сплайна¹, их первые и вторые производные) получаем еще ряд соотношений (16.10) – (16.11) [4]:

$$\left. \begin{aligned} g_{k-1}(x_{k-1}) &= g_k(x_{k-1}) \\ g_{k-1}'(x_{k-1}) &= g_k'(x_{k-1}) \\ g_{k-1}''(x_{k-1}) &= g_k''(x_{k-1}) \end{aligned} \right\}, k = 2, 3, \dots, n \quad (16.10)$$

$$g_1''(x_0) = 0, g_n''(x_n) = 0 \quad (16.11)$$

Соотношения (16.9) – (16.11) образуют $4n$ соотношений для нахождения коэффициентов сплайна. Подставляя выражения функций (16.8) и их производных (16.12):

$$\begin{aligned} g_k'(s) &= b_k + 2c_k(s - x_k) + 3d_k(s - x_k)^2 \\ g_k''(s) &= 2c_k + 6d_k(s - x_k) \end{aligned} \quad (16.12)$$

в соотношения (16.9) – (16.11) и принимая во внимание соотношение:

$$h_k = x_k - x_{k-1}, k = 1, 2, \dots, n \quad (16.13)$$

получим следующую систему уравнений (16.14) – (16.20) [4]:

$$a_1 - b_1 h_1 + c_1 h_1^2 - d_1 h_1^3 = y_0 \quad (16.14)$$

$$a_k = y_k, k = 1, 2, \dots, n \quad (16.15)$$

$$a_{k-1} = a_k - b_k h_k + c_k h_k^2 - d_k h_k^3; k = 2, 3, \dots, n \quad (16.16)$$

$$b_{k-1} = b_k - 2c_k h_k + 3d_k h_k^2; k = 2, 3, \dots, n \quad (16.17)$$

¹ Звеном сплайна называется функция $g_k(x)$ на интервале $[x_{k-1}, x_k]$.

$$c_{k-1} = c_k - 3d_k h_k; \quad k = 2, 3, \dots, n \quad (16.18)$$

$$c_1 - 3d_1 h_1 = 0 \quad (16.19)$$

$$c_n = 0 \quad (16.20)$$

Задача интерполяции свелась к решению системы (16.14)–(16.20). Из соотношения (16.15) следует, что все коэффициенты $a_k = y_k$, $k = 1, 2, \dots, n$. Подставив соотношения (16.14), (16.15) в (16.16) и используя фиктивный коэффициент $c_0 = 0$, получим соотношение между b_k , c_k и d_k [4]:

$$b_k h_k - c_k h_k^2 + d_k h_k^3 = y_k - y_{k-1}.$$

Отсюда коэффициенты b_k вычисляются при $k = 1, 2, \dots, n$ по формуле [4]

$$b_k = \frac{y_k - y_{k-1}}{h_k} + c_k h_k - d_k h_k^2 \quad (16.21)$$

Из (16.18) и (16.19) выразим d_k через c_k (с учетом коэффициента $c_0 = 0$) [4]:

$$d_k = \frac{c_k - c_{k-1}}{3h_k} \quad (k = 1, 2, \dots, n) \quad (16.22)$$

Подставим (16.22) в (16.21)

$$b_k = \frac{y_k - y_{k-1}}{h_k} + \frac{2}{3} c_k h_k + \frac{1}{3} h_k c_{k-1} \quad (k = 1, 2, \dots, n) \quad (16.23)$$

Введем обозначение:

$$l_k = \frac{y_k - y_{k-1}}{h_k} \quad (k = 1, 2, \dots, n) \quad (16.24)$$

после чего соотношение (16.23) примет вид:

$$b_k = l_k + \frac{2}{3} c_k h_k + \frac{1}{3} h_k c_{k-1} \quad (k = 1, 2, \dots, n) \quad (16.25)$$

Подставим (16.25) и (16.22) в соотношение (16.17), получим систему относительно c_k :

$$h_{k-1} c_{k-2} + 2(h_{k-1} + h_k) c_{k-1} + h_k c_k = 3l_k - 3l_{k-1} \quad (k = 2, 3, \dots, n) \quad (16.26)$$

$$c_0 = 0, \quad c_n = 0 \quad (16.27)$$

Систему (16.26) можно решить, используя метод прогонки [4], который сводится к нахождению прогоночных коэффициентов по формулам прямой прогонки [4]:

$$\delta_1 = -\frac{h_2}{2(h_1 + h_2)}, \quad \lambda_1 = \frac{3(l_2 - l_1)}{2(h_1 + h_2)} \quad (16.28)$$

$$\delta_{k-1} = -\frac{h_k}{2h_{k-1} + 2h_k + h_{k-1}\delta_{k-2}},$$

$$\lambda_{k-1} = \frac{3l_k - 3l_{k-1} - h_{k-1}\lambda_{k-2}}{2h_{k-1} + 2h_k + h_{k-1}\delta_{k-2}} \quad (k = 3, 4, \dots, n) \quad (16.29)$$

Далее следует найти коэффициенты c_k по формулам обратной прогонки:

$$c_{k-1} = \delta_{k-1}c_k + \lambda_{k-1} \quad (k = n, n-1, \dots, 2) \quad (16.30)$$

После нахождения коэффициентов c по формуле (16.30) находим b и d по формулам (16.22), (16.25).

Таким образом, алгоритм расчета коэффициентов интерполяционного сплайна состоит из следующих этапов:

1. Ввод значений табличной зависимости $y(x)$, массивов x и y .
2. Расчет элементов массивов h и l по формулам (16.13) и (16.24).
3. Вычисление массивов прогоночных коэффициентов δ и λ по формулам (16.28), (16.29).
4. Расчет массивов коэффициентов c по формуле (16.30).
5. Нахождение массивов коэффициентов b по формуле (16.25).
6. Расчет массивов коэффициентов d по формуле (16.22).

После этого в формулу (16.8) можно подставлять любую точку z и вычислять ожидаемое значение.

Алгоритм расчета коэффициентов интерполяционного сплайна рассмотрен на примере решения следующей задачи.

ПРИМЕР 16.2. В результате опыта холостого хода определена зависимость потребляемой из сети мощности (P_0 , Вт) от входного напряжения (U_1 , В) для асинхронного двигателя МТН111-6 (см. табл. 16.2).

Таблица 16.2 ▼ Зависимость $P_0(U_1)$

Напряжение, В	132	140	150	162	170	180	190	200	211	220	232	240	251
P_0 , Вт	330	350	385	425	450	485	540	600	660	730	920	1020	1350

Определить ожидаемое значение мощности при напряжениях U 195, 205, 215, 235 В. Построить график полученной с помощью сплайн-интерполяции зависимости $P_0(U_1)$, на котором изобразить заданные экспериментальные точки и рассчитанные значения.

Исходные данные: массивы вещественных чисел P_0 , U_1 (экспериментальные точки), U .

Выходные данные: массив рассчитанных значений мощностей P при напряжениях U .

Блок-схема алгоритма представлена на рис. 16.5.

Блоки 2–8 реализуют ввод исходных данных (N , P_0 и U_1 , k^1 , U). Блок 10 содержит обращение к процедуре расчета коэффициентов сплайна b , c и d (формулы 16.22, 16.25, 16.26–16.29). Ее блок-схема представлена на рис. 16.6.

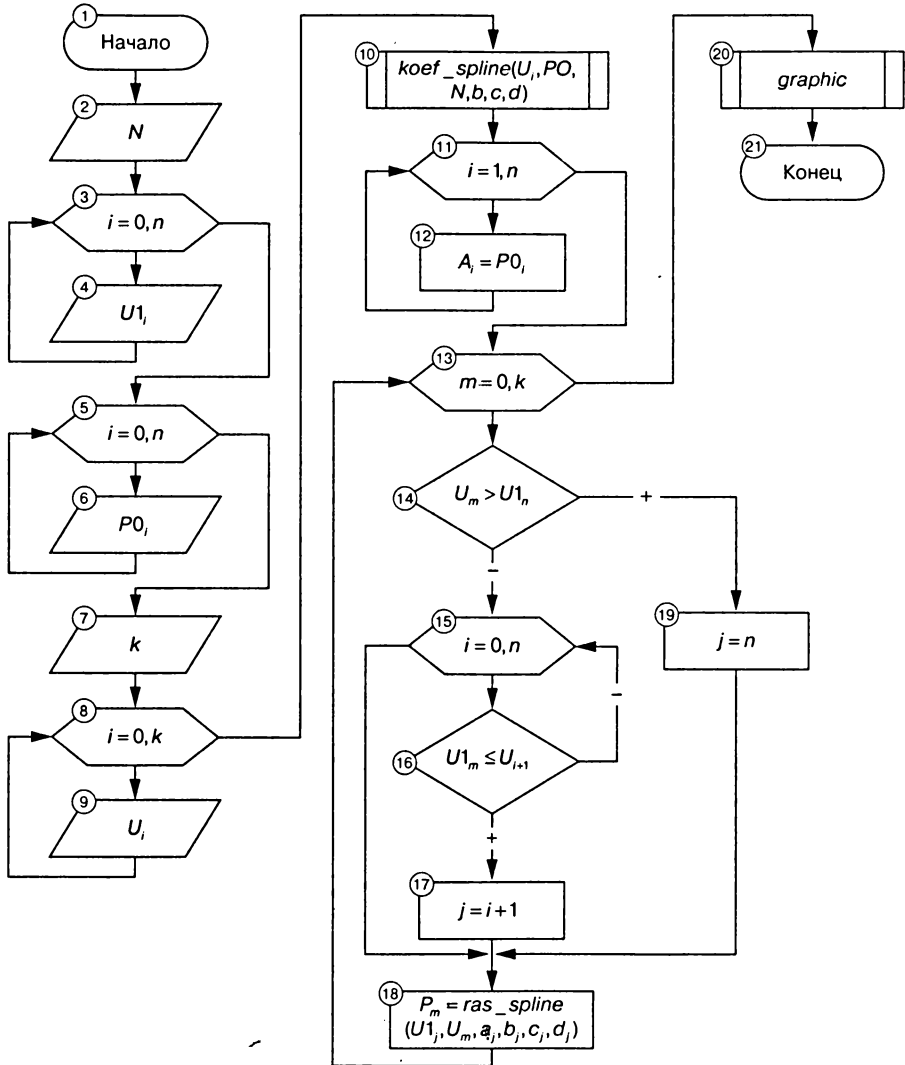


Рис. 16.5 ▼ Блок-схема алгоритма сплайн-интерполяции

¹ $k+1$ – количество точек в массиве U , элементы в массивах P_0 , U , U_1 нумеруются с нуля.

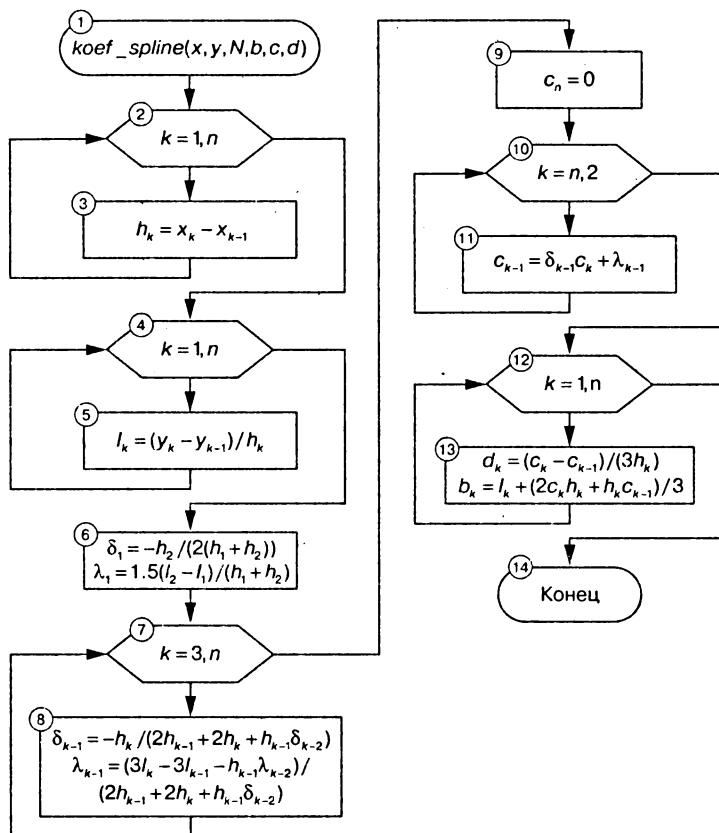


Рис. 16.6 ▼ Блок-схема расчета коэффициентов сплайна

В блоках 11–12 реализован расчет коэффициентов сплайна $a[k]$ (см. формулу (16.15)). Блоки 13–19 содержат расчет значений мощностей P при заданных значениях U . Блок 13 организует цикл по всем точкам. Блоки 14–17, 19 определяют номер интервала (переменная j), на котором находится заданная точка $U[m]$. В блоке 18 реализовано обращение к функции, реализующей расчет по формуле (16.8). Блок 20 обращается к процедуре рисования графика на экране дисплея.

Ниже приведен полный текст программы с необходимыми комментариями. Процедура рисования графика функции основана на алгоритме, описанном в главе 12 (блок-схемы 12.15 и 12.16).

```

uses crt, graph;
type
  mas0=array[0..300] of real;
  mas1=array[1..300] of real;

```

```

{ Процедура расчетов коэффициентов сплайна, x, y - массивы }
{ экспериментальных значений, N - размер этих массивов. }
{ Коэффициенты сплайна хранятся в массивах b, c, d. }
procedure koef_spline(x,y:mas0;N:word; var b,c,d:mas1);
var
  k:word;
  h,l,lyam,delt:mas1;
begin
  { Следующий цикл реализует формулу (16.13). }
  for k:=1 to n do
    h[k]:=x[k]-x[k-1];
    { Следующий цикл реализует формулу (16.24). }
    for k:=1 to n do
      l[k]:=(y[k]-y[k-1])/h[k];
      { Расчет прогоночных коэффициентов Delt и lyam по формулам (16.28),
(16.29). }
      delt[1]:=-h[2]/(2*(h[1]+h[2]));
      lyam[1]:=1.5*(l[2]-l[1])/(h[1]+h[2]);
      for k:=3 to n do
        begin
          delt[k-1]:=-h[k]/(2*h[k-1]+2*h[k]+h[k-1]*delt[k-2]);
          lyam[k-1]:=(3*l[k]-3*l[k-1]-h[k-1]*lyam[k-2])/
            (2*h[k-1]+2*h[k]+h[k-1]*delt[k-2]);
        end;
      { Расчет коэффициентов сплайна c по формулам (16.27), (16.30). }
      c[n]:=0;
      for k:=n downto 2 do
        c[k-1]:=delt[k-1]*c[k]+lyam[k-1];
      { Расчет коэффициентов сплайна d и b по формулам (16.22), (16.23). }
      for k:=1 to n do
        begin
          d[k]:=(c[k]-c[k-1])/(3*h[k]);
          b[k]:=l[k]+2/3*c[k]*h[k]+h[k]*c[k-1]/3;
        end;
      end;
  { Функция ras_spline реализует расчет по формуле (16.8). }
  function ras_spline(x,s,a,b,c,d:real):real;
  begin
    ras_spline:=a+b*(s-x)+c*sqr(s-x)+d*(s-x)*sqr(s-x)
  end;
var
  P,U,P0,U1:mas0;
  a,b,c,d:mas1;
  N,k,i,j,m:word;
  f,f2:text;
  s:real;
  procedure graphic;
var
  kvo,x0,xk,y0,yk,kvox,kvoy:word;
  UU,PP:mas1;
  cl,d1,g1,h1,hx,hy,max,min:real;
  hu,hv,grdr,grmd:integer;
  pl,pattern:word;
  stroka:string;

```

```

begin
  kvo:=200;
  { Отступы на экране слева(x0), справа (xk), снизу(yk) и сверху(y0). }
  x0:=100; xk:=120;
  y0:=50; yk:=50;
  { Уплотненный массив (UU) значений U1, количество точек в нем kvo+1. }
  hx:=(U1[N]-U1[0])/kvo;
  for i:=1 to kvo+1 do
    UU[i]:=U1[0]+(i-1)*hx;
  { Расчет уплотненного массива(PP) значений P0 с помощью }
  { сплайн-интерполяции. }
  for m:=1 to kvo+1 do
    begin
      for i:=1 to n do
        if UU[m]<=U1[i+1] then
          begin
            j:=i+1;
            break;
          end;
      PP[m]:=ras_spline(U1[j],UU[m],a[j],b[j],c[j],d[j]);
    end;
  { Определение максимального и минимального значений в массиве PP. }
  { Если значения U будут находиться за пределами области }
  { интерполирования,то надо будет искать общий максимум и минимум }
  { в массивах PP и P. }
  max:=PP[1];
  for i:=2 to kvo+1 do
    if PP[i]>max then max:=PP[i];
    min:=PP[1];
  for i:=2 to kvo+1 do
    if PP[i]<min then min:=PP[i];
  { Перевод экрана в графический режим. }
  grdr:=detect;
  InitGraph(grdr,grmd,'c:\bp\bgi');
  SetBkColor(15);
  SetColor(1);
  Rectangle(x0,y0,GetmaxX-xk,GetmaxY-yk);
  { Вычисление коэффициентов пересчета в «экранную» систему координат. }
  c1:=(GetMaxX-x0-xk)/(U1[N]-U1[0]);
  d1:=x0-c1*U1[0];
  g1:=(GetMaxY-y0-yk)/(min-max);
  h1:=y0-g1*max;
  { Изображение рассчитанного интерполяционного сплайна. }
  for i:=1 to kvo do
    line(trunc(c1*uu[i]+d1),trunc(g1*pp[i]+h1),
      trunc(c1*uu[i+1]+d1),trunc(g1*pp[i+1]+h1));
    { Изображение расчетных точек. }
  for i:=0 to k do
    circle(trunc(c1*u[i]+d1),trunc(g1*p[i]+h1),3);
    setfillstyle(1,blue);
    { Изображение заданных экспериментальных точек. }
  for i:=0 to n do
    FillEllipse(trunc(c1*u1[i]+d1),trunc(g1*p0[i]+h1),3,3);

```

```

    { Изображение линий сетки и подписей. }
    SetLineStyle(DashedLn,pattern,1);
    setcolor(1);
    kvox:=5;
    kvoy:=5;
    hu:=trunc((GetmaxX-x0-xk)/(kvox-1));
    hv:=trunc((GetmaxY-y0-yk)/(kvoy-1));
    for i:=1 to kvox-2 do
        line(x0+i*hu,y0,x0+i*hu,GetMaxY-yk);
    for i:=1 to kvoy-2 do
        line(x0,y0+i*hv,GetMaxX-Xk,y0+i*hv);
        hx:=(U1[N]-U1[0])/(kvox-1);
        hy:=(max-min)/(kvoy-1);
    for i:=1 to kvox do
    begin
        Str(U1[0]+(i-1)*hx:1:1,stroka);
        OutTextXY(x0+(i-1)*hu-15,GetMaxY-Yk div 2 -15,stroka);
    end;
    for i:=1 to kvoy do
    begin
        Str(max-(i-1)*hy:1:1,stroka);
        OutTextXY(x0 div 2 - 10,y0+(i-1)*hv,stroka);
    end;
    { Вывод легенд. }
    SetLineStyle(SolidLn,pattern,1);
    { Переменная P1 определяет выводимый график и может принимать значения: }
    { 1 - кривая, построенная методом сплайн интерполяции; }
    { 2 - рассчитанные значения; }
    { 3 - экспериментальные точки. }
    for p1:=1 to 3 do
    begin
        if p1=1 then .
        { Вывод непрерывной линии для легенды интерполяционной кривой. }
            Line(GetMaxX-Xk+10,(p1+1)*Y0 div 2,GetMaxX-Xk+20,(p1+1)*Y0 div 2);
        if p1=2 then
        { Изображение окружности для легенды рассчитанных значений. }
            circle(GetMaxX-Xk+10,(p1+1)*Y0 div 2,3);
        if p1=3 then
        { Изображение эллипса для легенды экспериментальных значений. }
            FillEllipse(GetMaxX-Xk+10,(p1+1)*Y0 div 2,3,3);
        { Формирование текста для легенды. }
        if p1=1 then stroka:='spline';
        if p1=2 then stroka:='calculation';
        if p1=3 then stroka:='experiment';
        OutTextXY(GetMaxX-Xk+25,(p1+1)*Y0 div 2 -5,stroka);
    end;
    SetLineStyle(SolidLn,pattern,1);
    setcolor(1);
    { Вывод осей координат. }
    if (U1[N]>0) and (U1[0]<0) then line(round(d1),y0,round(d1),getmaxY-yk);
    if (max>0) and (min<0) then line(x0,round(h1),GetMaxX-Xk,round(h1));
    OutTextXY(GetMaxX div 3,Y0 div 2,'P0(U1) IM MTH111-6');
    readln;

```

```

end;
begin
  clrscr;
  assign(f, 'spline.txt');
  { Ввод исходных данных из тестового файла. }
  reset(f);
  readln(f, N);
  for i:=0 to N do
    read(f, U1[i]);
  for i:=0 to N do
    read(f, P0[i]);
  readln(f, k);
  for i:=0 to k do
    read(f, U[i]);
  close(f);
  { Подготовка файла результатов. }
  assign(f2, 'rez.txt');
  rewrite(f2);
  { Расчет коэффициентов b, c, d сплайна. }
  koef_spline(U1, P0, N, b, c, d);
  { Формирование коэффициентов a сплайна. }
  for i:=1 to n do
    a[i]:=P0[i];
  { Формирование значений P по заданным значениям U с помощью кубического }
  { сплайна. }
  for m:=0 to k do
    begin
      { Определение номера интервала (j), которому принадлежит точка. }
      if U[m]>U1[n] then j:=i
      else
        for i:=0 to n-1 do
          if U[m]<=U1[i+1] then
            begin
              j:=i+1;
              break;
            end;
      P[m]:=ras_spline(U1[j], U[m], a[j], b[j], c[j], d[j]);
      { Вывод рассчитанных значений в результирующий текстовый файл. }
      writeln(f2, 'U=', U[m]:1:0, ' ', 'P=', P[m]:1:0);
    end;
  close(f2);
  { Обращение к процедуре рисования графика. }
  graphic;
end.

```

После запуска программы на выполнение будет сформирован следующий текстовый файл с результатами. В нем будут храниться вычисленные значения мощностей P при заданных напряжениях U .

```

U=195 P=570
U=205 P=628
U=215 P=683
U=235 P=953

```

Кроме того, программа выводит на экран график с экспериментальными точками и рассчитанными значениями (см. рис. 16.7), полученный с помощью сплайн-интерполяции зависимости $P_0(U_1)$.

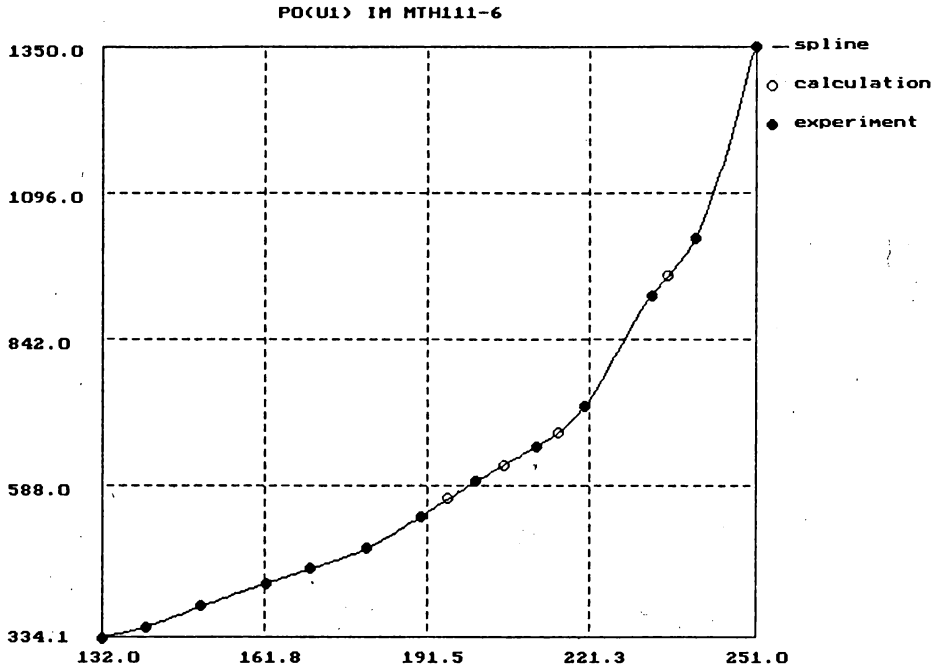


Рис. 16.7 ▽ График интерполяционной зависимости

На практике часто используемым методом интерполирования является линейная интерполяция (линейный сплайн). В этом случае в качестве сплайна выступает линейная функция (16.31):

$$f_k(s) = a_k + b_k s, \quad s \in [x_{k-1}, x_k], \quad k = 1, \dots, n \quad (16.31)$$

удовлетворяющая условию интерполяции в узлах сплайна $g_k(x_k) = y_k$; $g_k(x_{k-1}) = y_{k-1}$. Коэффициенты a и b в этом случае рассчитываются по формулам (16.32), которые получаются из уравнения прямой, проходящей через две точки.

$$\begin{aligned} a_k &= y_{k-1} - \frac{y_k - y_{k-1}}{x_k - x_{k-1}} x_{k-1}, \\ b_k &= \frac{y_k - y_{k-1}}{x_k - x_{k-1}} \end{aligned} \quad (16.32)$$

Найдя коэффициенты линейного сплайна, можно рассчитать значения в любой точке интервала $[x_0, x_n]$. Программа подбора зависимости с помощью линейного сплайна будет отличаться от программы подбора зависимости кубической интерполяции более простой подпрограммой расчета коэффициентов линейного сплайна a_k и b_k (см. формулу (16.32)). Линейная интерполяция дает хорошие результаты при практическом счете внутри интервала $[x_0, x_n]$, когда от получаемой функции не требуют дополнительных свойств (дифференцируемости и т.д.).

16.5. Упражнения по теме

«Обработка результатов эксперимента»

Подобрать функциональную зависимость заданного вида с помощью метода наименьших квадратов. Определить суммарную ошибку. Изобразить график зависимости и экспериментальные точки на экране дисплея.

1. В результате эксперимента была определена некоторая табличная зависимость (табл. 16.3). Подобрать функциональную зависимость вида $P(s) = As^3 + Bs^2 + D$.

Таблица 16.3 ▼ Экспериментальные данные к упражнению 1

S	0	1	1,5	2	2,5	3	3,5	4	4,5	5
P	12	10,1	11,58	17,4	30,68	53,6	87,78	136,9	202,5	287

2. В результате эксперимента были получены некоторые данные, представленные в виде таблицы (табл. 16.4). Необходимо построить аналитическую зависимость вида $G(s) = As^b$.

Таблица 16.4 ▼ Экспериментальные данные к упражнению 2

S	0,5	1,5	2	2,5	3	3,5	4	4,5	5
G	3,99	5,65	6,41	6,71	7,215	7,611	7,83	8,19	8,3

3. Данные, представленные в виде таблицы (табл. 16.5), были получены в результате эксперимента. Определить коэффициенты функции $K(s) = Ae^{bs}$.

Таблица 16.5 ▼ Экспериментальные данные к упражнению 3

S	0	0,5	1	1,5	2	2,5	3,5	3,5	4
K	2,31	2,899	3,534	4,412	5,578	6,92	8,699	10,69	13,39

4. Определить параметры функции $V(s) = As^b e^{Cs}$ для экспериментальных данных, представленных в виде таблицы (табл. 16.6).

Таблица 16.6 ▼ Экспериментальные данные к упражнению 4

s	0,2	0,7	1,2	1,7	2,2	2,7	3,2
v	2,3198	2,8569	3,5999	4,4357	5,5781	6,9459	8,6621

5. Подобрать аналитическую зависимость вида $\dot{Y} = X/(Ax - B)$, используя данные, полученные в результате эксперимента и сведенные в таблицу (табл. 16.7).

Таблица 16.7 ▼ Экспериментальные данные к упражнению 5

x	3	3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9
y	0,61	0,6	0,592	0,58	0,585	0,583	0,582	0,57	0,572	0,571

6. Найти приближенное значение функции при заданном значении аргумента с помощью сплайн-интерполяции в точках $x_1 = 3,75$, $x_2 = 4,75$, $x_3 = 5,25$. Функция задана таблично (табл. 16.8). Изобразить график построенной интерполяционной зависимости, экспериментальные и рассчитанные точки на экране дисплея.

Таблица 16.8 ▼ Исходные данные к упражнению 6

u	2	2,5	3	3,5	4	4,5	5	5,5	6
v	5,197	7,78	11,14	15,09	19,245	23,11	26,25	28,6	30,3

7. Найти приближенное значение функции при заданном значении аргумента с помощью линейной интерполяции в точках $x_1 = 1$, $x_2 = 1,5$, $x_3 = 2$. Функция задана таблично (табл. 16.9). Изобразить график построенной интерполяционной зависимости, экспериментальные и рассчитанные точки на экране дисплея.

Таблица 16.9 ▼ Исходные данные к упражнению 7

t	0,66	0,9	1,17	1,47	1,7	1,74	2,08	2,63	3,12
z	38,9	68,8	64,4	66,5	64,95	59,36	82,6	90,63	113,5

8. Найти приближенное значение функции при заданном значении аргумента с помощью интерполяционного полинома Лагранжа в точках $x_1 = 0,702$, $x_2 = 0,512$, $x_3 = 0,608$. Функция задана таблично (табл. 16.10).

Таблица 16.10 ▼ Исходные данные к упражнению 8

x	0,43	0,48	0,55	0,62	0,7	0,75
y	1,63597	1,73234	1,87686	2,03345	2,22846	2,35973

9. Найти приближенное значение функции при заданном значении аргумента с помощью интерполяционного полинома Ньютона в точках $x_1 = 0,308$, $x_2 = 0,325$, $x_3 = 0,312$. Функция задана таблично (табл. 16.11).

Таблица 16.11 ▼ Исходные данные к упражнению 9

x	0,298	0,303	0,310	0,317	0,323	0,330
y	3,25578	3,17639	3,12180	3,04819	2,98755	2,91950

10. Найти приближенное значение функции при заданном значении аргумента с помощью канонического полинома в точках $x_1 = 0,608$, $x_2 = 0,594$, $x_3 = 0,631$. Функция задана таблично (табл. 16.12).

Таблица 16.12 ▼ Исходные данные к упражнению 10

x	0,593	0,598	0,605	0,613	0,619	0,627
y	0,53205	0,53562	0,54059	0,54623	0,55043	0,55598

17 Глава

Численное интегрирование функций

На практике часто встречаются интегралы, которые не выражаются через элементарные функции или выражаются очень сложно. Нередко подынтегральная функция задается таблицей или графиком. В этом случае интегралы находят приближенными методами, основу которых составляет геометрический смысл определенного интеграла

$$y = \int_b^a f(x) dx,$$

где y – это площадь фигуры, ограниченной подынтегральной кривой, осью абсцисс и ординатами $f(a)$ и $f(b)$.

Известные численные методы приближенного вычисления интегралов (метод прямоугольников, трапеций, парабол) заменяют последние суммой площадей конечного числа элементарных участков.

17.1. Интегрирование по методу прямоугольников

Интегрирование по *методу прямоугольников* [6] заключается в том, что интервал интегрирования $[a, b]$ делится точками x_0, x_1, \dots, x_n на n равных частей (рис. 17.1), причем

$$x_0 = a, x_n = b,$$

длина каждой части составляет

$$h = (b-a)/n,$$

и тогда

$$x_i = x_0 + ih, \quad i = 0, \dots, n.$$

Из каждой точки x проведем перпендикуляр до пересечения с кривой $f(x)$, а затем заменим кривую подынтегральной функции ломаной линией, отрезки которой параллельны оси абсцисс.

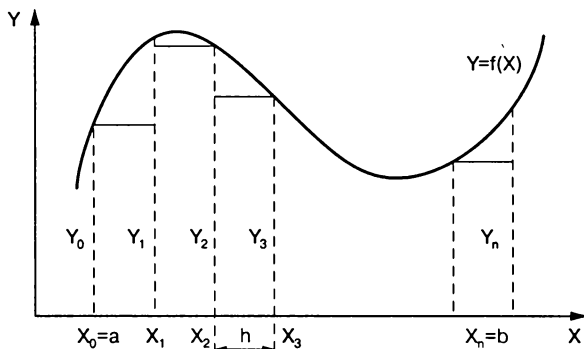


Рис. 17.1 ▼ Графическая интерпретация интегрирования по методу прямоугольников

Площадь полученной ступенчатой фигуры можно найти как сумму площадей прямоугольников, стороны которых равны h и y_i . Следовательно, площадь отдельного прямоугольника составит

$$S_i = y_i \cdot h,$$

тогда

$$\begin{aligned} S &= \int_a^b f(x) dx \approx y_0 h + y_1 h + y_2 h + \dots + y_{n-1} h = \\ &= f(x_0)h + f(x_1)h + f(x_2)h + \dots + f(x_{n-1})h = h \sum_{i=0}^{n-1} f(x_i). \end{aligned}$$

Следовательно, формула вычисления определенного интеграла по методу прямоугольников имеет вид:

$$I = \int_a^b f(x) dx = h \sum_{i=0}^{n-1} f(x_i) = h \sum_{i=1}^n f(x_i) \quad (17.1)$$

Запрограммировать эту формулу несложно. Ниже приведен фрагмент программы, реализующий данный алгоритм в виде функции, где f – это подынтегральная функция, определенная в тексте программы ранее. Ее входными параметрами являются пределы интегрирования (a , b) и число разбиений отрезка (n).

```

function integ(a,b:real;n:byte):real;
var S,h:real; j:integer;
begin
  h:=(b-a)/n; { Шаг интегрирования. }
  S:=0;
  for j:=0 to n-1 do
    S:=S+f(a+h*j); { Сумма площадей прямоугольников. }
    integ:=h*S;    { Значение интеграла. }
  end;

```

17.2. Интегрирование по методу трапеций

При использовании *метода трапеций* [6] участок интегрирования также разбивается на n равных частей. Если провести ординаты во всех точках деления и заменить каждую из полученных криволинейных трапеций прямолинейной (рис. 17.2), то приближенное значение интеграла будет равно сумме площадей прямолинейных трапеций.

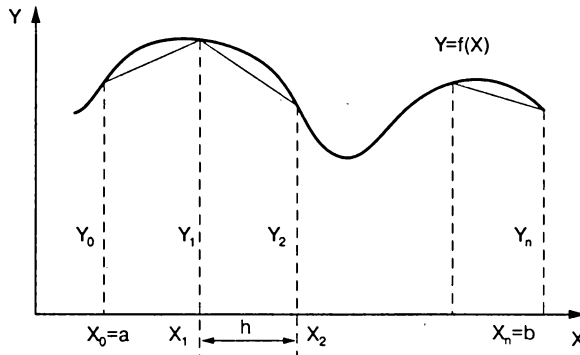


Рис. 17.2 ▼ Геометрическая интерпретация интегрирования по методу трапеций

Площадь отдельной трапеции составляет:

$$S = \frac{y_{i-1} + y_i}{2} h,$$

тогда площадь искомой фигуры будем искать по формуле:

$$S = \int_a^b f(x) dx \approx \sum_{i=1}^n S_i = \frac{h}{2} \sum_{i=1}^n (y_{i-1} + y_i) = h \cdot \left(\frac{y_0 + y_n}{2} + \sum_{i=1}^{n-1} y_i \right).$$

Следовательно, формула трапеций для численного интегрирования имеет вид:

$$I = \int_a^b f(x) dx = h \cdot \left(\frac{f(a) + f(b)}{2} + \sum_{i=1}^{n-1} f(x_i) \right) \quad (17.2)$$

Фрагмент текста программы, представленный далее, реализует численное интегрирование при помощи формулы трапеций.

```
function integ(a,b:real;n:byte):real;
var S,h:real; j:integer;
begin
  h:=(b-a)/n; { Шаг интегрирования. }
  S:=0;
  for j:=1 to n-1 do
    S:=S+f(a+h*j); { Сумма площадей трапеций. }
  integ:=h*((f(a)+f(b))/2+S); { Значение интеграла. }
end;end.
```

17.3. Интегрирование по методу Симпсона

Пусть $n = 2m$ – четное число, а $y_i = f(x_i)$ ($i = 0, n$) – значения функции $y = f(x)$ для равноотстоящих точек $a = x_0, x_1, x_2, \dots, x_n = b$ с шагом $h = (b - a)/n = (b - a)/2m$. На паре соседних участков (рис. 17.3) кривая $y = f(x)$ заменяется параболой $y = L(x)$, коэффициенты которой подобраны так, что она проходит через точки y_0, y_1, y_2 .

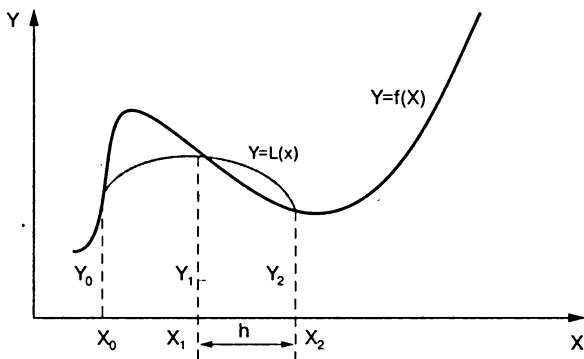


Рис. 17.3 ▼ Геометрическая интерпретация интегрирования по методу Симпсона

Площадь криволинейной трапеции [6], ограниченной сверху параболой, составит:

$$S_i = \frac{h}{3}(y_{i-1} + 4y_i + y_{i+1}).$$

Суммируя площади всех криволинейных трапеций, получим:

$$S = \int_a^b f(x)dx \approx \frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + \dots + 2y_{2m-2} + 4y_{2m-1} + y_{2m}) = \frac{h}{3} \left(y_0 + y_{2m} + \sum_{i=1}^{2m-1} p y_i \right),$$

где $p = 6 - p$, $p = 4$. Следовательно, формула Симпсона для численного интегрирования имеет вид:

$$I = \int_a^b f(x)dx = \frac{h}{3} \left(f(a) + f(b) + \sum_{i=1}^{2m-1} p y_i \right) \quad (17.3)$$

Фрагмент программы, реализующий интегрирование по формуле Симпсона:

```
function integ(a,b:real;n:byte):real;
var S,h:real; j,p:integer;
begin
  h:=(b-a)/n; { Шаг интегрирования. }
  S:=0;p:=4;
  for j:=1 to n-1 do
    begin
      S:=S+p*f(a+h*j); { Сумма площадей криволинейных трапеций. }
      p:=6-p;
    end;
  integ:=h/3*(f(a)+f(b)+S); { Значение интеграла. }
end;
```

17.4. Вычисление интеграла с заданной точностью

При одном и том же числе точек формула Симпсона в большинстве случаев намного точнее, чем формулы прямоугольников и трапеций [4].

Точность интегрирования ε по методу прямоугольников с постоянным шагом h связана с величиной шага следующим образом:

$$\varepsilon \approx h.$$

При интегрировании методом трапеций это соотношение имеет вид:

$$\varepsilon \approx h^2.$$

Метод Симпсона обеспечивает точность вычислений

$$\varepsilon \approx h^2 \div h^4.$$

Ошибка в выборе величины шага интегрирования либо не обеспечит нужной точности, либо приведет к необоснованным затратам машинного времени. Заданную точность при минимальных сроках обеспечивают алгоритмы интегрирования с автоматическим выбором величины шага. Блок-схема вычисления интеграла с заданной точностью приведена на рис. 17.4. Здесь $\text{Integ}(a, b, n)$ – функция, вычисляющая значение интеграла на участке от a до b любым из приведенных выше методов, с заданной точностью ε , n – число разбиений подынтегральной функции на заданном участке интегрирования.

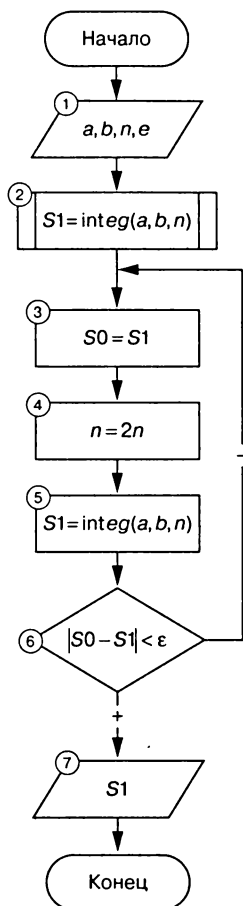


Рис. 17.4 ▼ Численное интегрирование с автоматическим выбором шага

17.5. Упражнения для самостоятельной работы

1. Вычислить определенный интеграл, используя метод прямоугольников, $\varepsilon = 0,001$:

$$I = \int_{1,2}^{1,6} \frac{1}{\sqrt{1+2x^3}} dx.$$

2. Найти заданный интеграл, применив метод трапеций, $\varepsilon = 0,001$:

$$I = \int_{0,3}^{1,9} \sqrt{7+2x^3} dx.$$

С помощью метода Симпсона вычислить интеграл $\varepsilon = 0,0001$:

$$I = \int_{0,5}^{1,5} \frac{\sin(2x)}{x^2 + 1} dx.$$

4. Вычислить определенный интеграл, используя методы прямоугольников и трапеций. Подсчитать количество выполненных итераций в каждом из методов; точность вычислений $\varepsilon = 0,001$:

$$I = \int_{0,4}^{0,6} \sqrt{x} \cdot \cos(x^2) dx.$$

5. Провести сравнительный анализ вычислений определенного интеграла методами прямоугольников и Симпсона; точность вычислений $\varepsilon = 0,001$:

$$I = \int_{0,1}^{0,5} \frac{1}{1 + \sin x + x} dx.$$

6. Сравнить данные, полученные при определении интеграла методами трапеций и Симпсона; точность вычислений $\varepsilon = 0,0001$:

$$I = \int_{0,2}^{1,8} \sqrt{0,2x^2 + 1} dx,$$

7. Вычислить определенный интеграл, используя метод прямоугольников с автоматическим выбором шага, определить количество выполненных итераций; точность вычислений $\varepsilon = 0,001$:

$$I = \int_{0,1}^{0,9} \frac{\sqrt{x+1}}{\cos^2 x} dx.$$

8. Найти заданный интеграл, используя метод трапеций. Повторить расчет тем же методом, но с автоматическим выбором шага. Сравнить количество выполненных итераций, если точность вычислений $\varepsilon = 0,001$:

$$I = \int_{1,1}^{2,1} \frac{\sqrt{x}}{e^{x^2}} dx.$$

9. Вычислить определенный интеграл, используя метод Симпсона с автоматическим выбором шага, подсчитать количество выполненных итераций, если точность вычислений $\varepsilon = 0,00001$:

$$I = \int_{0.35}^{0.75} \frac{\lg(x)}{\sqrt{x+3}} dx.$$

10. Провести сравнительный анализ вычислений определенного интеграла методами прямоугольников, трапеций и Симпсона; точность вычислений $\varepsilon = 0,0001$:

$$I = \int_{0.1}^{0.5} \frac{e^x}{1 + \sin 3x} dx.$$

Приложение

Основные приемы работы в среде Турбо Паскаля

Вызов Турбо Паскаля

Формат команды MS DOS, осуществляющей вызов Турбо Паскаля, имеет вид:

```
[PATH] turbo [Options] [FileName]
```

Здесь PATH – путь к каталогу, в котором находится файл turbo.exe; Options – список опций; FileName – имя PAS-файла.

В квадратных скобках отмечены необязательные параметры команды.

Параметр **FileName**, если он указан, задает имя текстового файла, который будет автоматически загружен средой в активное окно редактора. Если имя файла не содержит расширения, то автоматически добавляется расширение pas.

С помощью параметров **Options** среде можно передать указание изменить нужным образом свою настройку. Каждый параметр задается в формате /Z, где Z – буква, определяющая настройку среды – знаки «+» (включить настройку), или «-» (отключить настройку); несколько параметров отделяются друг от друга пробелами. В списке **Options** можно использовать следующие управляющие параметры:

- /C<имя> – загрузить нужный файл конфигурации (эквивалентно опции OPTIONS/OPEN среды); <имя> – имя конфигурационного файла, например /Cay.tp;

- /D+ – использовать дополнительный монитор (эквивалентно опции OPTIONS/ENVIRONMENT/STARTUP/DUAL MONITOR SUPPORT);
- /E<размер> – установить нужный размер памяти для экранного буфера (эквивалентно опции OPTIONS/ENVIRONMENT/STARTUP/EDITOR HEAP SIZE); <размер> – устанавливаемый размер (килобайт); например, /E15 – установить размер экранного буфера в 15 Кб;
- /G+ – сохранять копию графического экрана (эквивалентно опции OPTIONS/ENVIRONMENT/STARTUP/GRAPHICS SCREEN SAVE);
- /J+ – ПК оснащен жидкокристаллическим дисплеем (эквивалентно опции OPTIONS/ENVIRONMENT/STARTUP/LCD COLOR SET);
- /N+ – ПК оснащен адаптером типа CGA (эквивалентно опции OPTIONS/ENVIRONMENT/STARTUP/CGA SNOW CHECKING);
- /O<размер> – установить нужный размер памяти для хранения оверлейных модулей системы Турбо Паскаль (эквивалентно опции OPTIONS/ENVIRONMENT/STARTUP/OVERLAY HEAP SIZE);
- /P+ – сохранять цветовую палитру экрана (эквивалентно опции OPTIONS/ENVIRONMENT/STARTUP/EGA/VGA PALETTE SAVE);
- /S<путь> – определяет путь к «быстрому» диску (эквивалентно опции OPTIONS/ENVIRONMENT/STARTUP/SWAP FILE DIRECTORY);
- /T+ – загружать в память библиотеку SYSTEM.TPU из файла turbo.tpl (эквивалентно опции OPTIONS/ENVIRONMENT/STARTUP/ LOAD turbo.tpl);
- /W<размер> – установить нужный размер памяти для хранения окон Турбо Паскаля (эквивалентно опции OPTIONS/ENVIRONMENT/STARTUP/ WINDOW HEAP SIZE);
- /X+ – использовать EMS-память (эквивалентно опции OPTIONS/ENVIRONMENT/ STARTUP/USE EXPANDED MEMORY).
- Для того чтобы осуществить вызов Турбо Паскаля, необходимо запустить файл turbo.exe (рис. П.1).

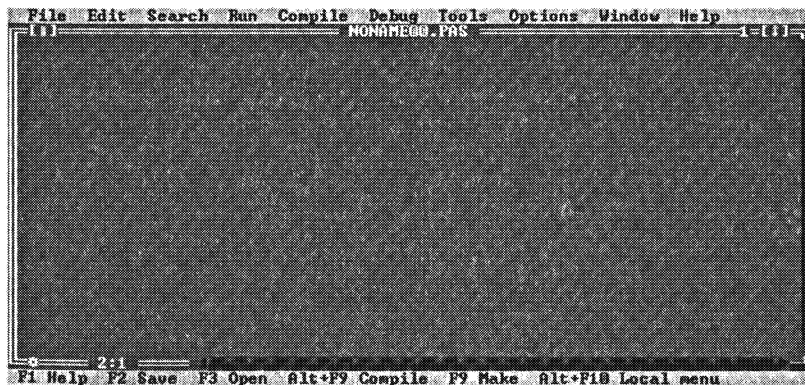


Рис. П.1 Внешний вид экрана среды Турбо Паскаль

Текстовый редактор

Редактор среды Турбо Паскаля предоставляет пользователю удобные средства создания и редактирования текстов программ. Признаком того, что среда находится в состоянии редактирования, является наличие в окне редактора курсора небольшого мигающего прямоугольника. Режим редактирования автоматически устанавливается после загрузки Турбо Паскаля. Из него можно перейти к любому другому режиму работы Турбо Паскаля с помощью функциональных клавиш или выбора нужного режима из главного меню. Если среда находится в состоянии выбора режима в меню, курсор исчезает, а в строке меню появляется цветной указатель – прямоугольник, выделяющий одно из кодовых слов. Для перехода из состояния выбора режима в главном меню в состояние редактирования нужно нажать клавишу **Esc**, а для перехода к выбору из главного меню – **F10**.

Рассмотрим основные приемы работы с текстовым редактором.

Окно редактора имитирует длинный и достаточно широкий лист бумаги, фрагмент которого виден в окне. Если курсор достиг нижнего края, осуществляется прокрутка окна редактора: его содержимое смещается вверх на одну строку и снизу появляется новая строка листа. Когда курсор достигает правой границы экрана, окно начинает по мере ввода смещаться вправо, показывая правый край листа. Компилятор Турбо Паскаля воспринимает строки программы длиной не более 126 символов.

Для создания текста программы нужно ввести символы с помощью клавиатуры ПК подобно тому, как это делается при печатании на пишущей машинке. После заполнения очередной строки следует нажать на клавишу **Enter**, тогда в конце ее появится специальный символ разделитель, а курсор окажется на следующей строке (он всегда показывает то место на экране, куда будет помещен очередной вводимый символ программы).

Курсор можно смещать по экрану при помощи клавиш передвижения. Если вы ошиблись при вводе очередного символа, его можно стереть с помощью клавиши **Backspace**. Клавиша **Del** стирает символ, на который в данный момент указывает курсор, а команда **Ctrl+Y** – всю строку, на которой располагается курсор.

Чтобы «разрезать» строку, следует подвести курсор к нужному месту и нажать клавишу **Enter**; чтобы «склеить» соседние строки, нужно установить курсор в конец первой строки (для этого удобно использовать клавишу **End**) и нажать клавишу **Del**, или поместить курсор в начало следующей строки (клавишей **Home**) и нажать клавишу **Backspace**.

Нормальный режим работы редактора – режим вставки, в котором каждый вновь вводимый символ как бы раздвигает текст на экране, смещая вправо остаток строки. Следует учитывать, что «разрезание» и последующая вставка пропущенных строк возможны только в этом режиме. Редактор может также работать в режиме замены, когда новый символ заменяет собой тот, на который

указывает курсор, а остаток строки справа от курсора не смещается. Для перехода к режиму наложения нужно нажать клавишу **Ins**; если нажать эту клавишу еще раз, вновь восстановится режим вставки. Признаком того, в каком режиме работает редактор, является форма курсора: в режиме вставки курсор похож на мигающий символ подчеркивания, а в режиме замены он представляет собой крупный мигающий прямоугольник, заслоняющий символ целиком.

И еще об одной возможности редактора. Обычно он работает в режиме автоотступа, когда каждая новая строка начинается в той же позиции на экране, что и предыдущая. Режим автоотступа поддерживает хороший стиль оформления текстов программ. Отступы от левого края выделяют тело условного или составного оператора и делают программу более наглядной. Отказаться от автоотступа можно командой **Ctrl+O+1** (при нажатой клавише **Ctrl** нажимается сначала клавиша **O**, затем **O** отпускается и нажимается клавиша **1**), повторная команда **Ctrl+O+1** восстановит режим автоотступа.

Смещение курсора

Итак, перемещать курсор можно с помощью перечисленных далее клавиш:

- **PgUp** – на страницу вверх;
- **PgDn** – на страницу вниз;
- **Home** – в начало строки;
- **End** – в конец строки;
- **Ctrl+PgUp** – в начало текста;
- **Ctrl+PgDn** – в конец текста.

Команды редактирования

Работать в текстовом редакторе удобно, используя следующие клавиши:

- **Backspace** – стереть символ слева от курсора;
- **Del** – убрать символ, отмеченный курсором;
- **Ctrl+Y** – стереть строку, на которой располагается курсор;
- **Enter** – вставить новую строку, разрезать старую;
- **Ctrl+Q+L** – восстановить текущую строку (действует, если курсор не покидал измененную строку).

Работа с блоком

При подготовке текстов программ часто возникает необходимость перенести фрагмент текста в другое место или удалить его. Для такого рода операций удобно использовать блоки – фрагменты текста, рассматриваемые как единое целое. Длина блока может быть достаточно большой (до 64 Кб), и тогда он занимает несколько экранных страниц. В каждый момент в среде может быть объявлен только один блок в одном окне редактора. Обмен блоками между окнами возможен только через буфер редактора (см. опцию **EDIT** в главном меню).

Клавиши, предназначенные для работы с блоком, таковы:

- **Ctrl+K+B** – пометить начало блока;
- **Ctrl+K+K** – пометить конец блока;
- **Ctrl+K+T** – пометить в качестве блока слово слева от курсора;
- **Ctrl+K+Y** – стереть блок;
- **Ctrl+K+C** – копировать блок;
- **Ctrl+K+V** – переместить блок;
- **Ctrl+K+W** – записать блок в дисковый файл;
- **Ctrl+K+R** – прочитать блок из дискового файла;
- **Ctrl+K+P** – напечатать блок;
- **Ctrl+K+I** – сдвинуть блок вправо;
- **Ctrl+K+U** – сдвинуть блок влево.
- **Ctrl+K+H** – убрать выделение блока цветом; повторное использование **Ctrl+K+H** вновь выделит блок.

Работа с файлами

Как уже говорилось, сразу после запуска Турбо Паскаля среда автоматически переходит в режим редактирования текста, в котором можно подготовить новую программу или исправить существующую.

Основной формой хранения текстов программ вне среды являются файлы. После завершения работы с Турбо Паскалем можно сохранить текст новой программы в дисковом файле с тем, чтобы использовать его в следующий раз. Для обмена данными между дисковыми файлами и редактором среды предназначены клавиши **F2** (запись в файл) и **F3** (чтение из файла). Если вы создаете новую программу, то среда еще не знает имя того файла, в который вы захотите поместить ее текст, и поэтому она присваивает тексту стандартное имя **NONAMEOO.PAS** (**NO NAME** – нет имени). Для сохранения текста программы в файле нужно нажать на клавишу **F2**. В этот момент среда проверит имя и, если это стандартное имя **NONAME**, спросит, нужно ли его изменять: на экране появится небольшое окно запроса с надписью в верхней части:

Save file as
(Сохранить в файле с именем)

Ниже надписи располагается поле для ввода имени файла, в котором можно написать любое имя и нажать клавишу **Enter** – текст будет сохранен в файле. Если в имени файла опущено расширение, среда присвоит ему стандартное расширение **.PAS**.

Если завершена работа с Турбо Паскалем (команда **Alt+X**), но не сохранен текст программы на диске, на экране появится окно с запросом:

NONAMEOO.PAS has been modified. Save?
(Файл NONAMEOO.PAS был изменен. Сохранить?)

В ответ следует нажать Y (Yes – да), если необходимо сохранить текст в файле, или N (No – нет), если делать этого не нужно.

Выполнение и отладка программы

После подготовки текста программы можно попытаться исполнить ее, то есть откомпилировать программу, связать ее (если необходимо) с библиотекой стандартных процедур и функций, загрузить в оперативную память и передать ей управление. Вся эта последовательность действий называется прогоном программы и реализуется командой **Ctrl+F9**.

Если в программе нет синтаксических ошибок, то все действия выполняются последовательно одно за другим, при этом на экране сообщается о количестве строк откомпилированной программы и объеме доступной оперативной памяти. Перед передачей управления загруженной программе среда очищает экран (точнее, выводит на экран окно прогона программы), а после завершения работы программы вновь берет управление компьютером на себя и восстанавливает на экране окно редактора.

Если на каком-либо этапе среда обнаружила ошибку, она прекращает дальнейшие действия, восстанавливает окно редактора и помещает курсор на ту строку программы, при компиляции или исполнении которой обнаружена ошибка. При этом в верхней строке редактора появляется диагностическое сообщение о причине ошибки. Таким образом, существует возможность очень быстро отладить программу, то есть устранить в ней синтаксические ошибки и добиться ее правильной работы.

Если ошибка возникла на этапе работы программы, простое указание того места, где она обнаружена, может не дать нужной информации в том случае, если ошибка является следствием неправильной подготовки данных. Например, если ошибка возникла при извлечении корня из отрицательного числа, будет указан оператор, в котором осуществлялась сама операция извлечения корня, хотя ясно, что первопричину ошибки следует искать где-то раньше – там, где соответствующей переменной присваивается отрицательное значение. В таких ситуациях обычно прибегают к пошаговому исполнению программы с помощью команд, связанных с клавишами **F4**, **F7** и **F8**. Пока еще не накоплен достаточный опыт отладки, можно пользоваться одной клавишей **F7**, после нажатия на которую среда осуществит компиляцию, компоновку (связь с библиотекой стандартных процедур и функций) и загрузку программы, а затем остановит прогон перед исполнением первого оператора. Строка программы, содержащая этот оператор, выделится на экране указателем (цветом). Теперь каждое новое нажатие на **F7** будет вызывать исполнение всех операций, запрограммированных в текущей строке, и смещение указателя к следующей строке программы. В подозрительном месте программы можно просмотреть значения нужных вам переменных или выражений, действуя следующим образом: установите курсор в то место текущей строки, где написано имя интересующей

вас переменной, и нажмите **Ctrl+F4**. На экране откроется диалоговое окно, состоящее из трех полей; в верхнем поле будет стоять имя переменной. После этого нажмите на клавишу **Enter**, чтобы получить в среднем поле текущее значение этой переменной. Если перед командой **Ctrl+F4** курсор стоял на пустом участке строки или указывал на другую переменную, верхнее поле также окажется пустым или будет содержать имя этой другой переменной. В таком случае следует ввести с помощью клавиатуры интересующее имя в верхнем поле и нажать клавишу **Enter**. Кстати, таким образом можно вводить не только имена прослеживаемых переменных, но и выражения с их участием – среда вычислит и покажет значение этого выражения.

Справочная служба Турбо Паскаля

Неотъемлемой составной частью среды Турбо Паскаля является встроенная справочная служба. Если вы достаточно хорошо владеете английским языком, у вас не будет проблем при работе с Турбо Паскалем: в затруднительной ситуации достаточно нажать на клавишу **F1**, и на экране высветится необходимая справка, которая зависит от текущего состояния среды (такую справочную службу называют контекстно-зависимой). Например, если нажать на **F1** в момент, когда среда обнаружила ошибку в программе, в справке будут сообщены дополнительные сведения о причинах появления этой ошибки и рекомендации по ее устранению.

Существуют четыре способа обращения к справочной службе непосредственно из окна редактора:

- **F1** – получение контекстно-зависимой справки;
- **Shift+F1** – выбор справки из списка доступных справочных сообщений;
- **Ctrl+F1** – получение справки о нужной стандартной процедуре, функции, о стандартной константе или переменной;
- **Alt+F1** – получение предыдущей справки;
- при использовании команды **Shift+F1** на экране появляется справочное окно, содержащее упорядоченный по алфавиту список стандартных процедур, функций, констант и переменных, для которых можно получить справочную информацию. В этот момент клавишами смещения курсора следует передвинуть указатель в окне к нужному слову и нажать клавишу **Enter**, чтобы получить справку.

Справку можно получить и другим способом: напечатать на экране имя стандартной процедуры (функции, константы, переменной) или подвести курсор к имеющемуся уже в тексте программы стандартному имени и нажать **Ctrl+F1**. Среда проанализирует ближайшее окружение курсора, выделит стандартное имя и даст нужную справку.

Доступ к справочной службе возможен и через главное меню Турбо Паскаля (подробнее о работе с меню см. далее).

Функциональные клавиши

Избежать лишней траты времени можно, используя в работе функциональные клавиши:

- **F1** – обратиться за справкой к встроенной справочной службе;
- **F2** – записать редактируемый текст в дисковый файл;
- **F3** – прочитать текст из дискового файла в окно редактора;
- **F4** – используется в отладочном режиме: начать или продолжить исполнение программы и остановиться перед исполнением той ее строки, на которой стоит курсор;
- **F5** – распахнуть активное окно на весь экран;
- **F6** – сделать активным следующее окно;
- **F7** – используется в отладочном режиме: выполнить следующую строку программы; если в строке есть обращение к процедуре (функции), войти в эту процедуру и остановиться перед исполнением первого ее оператора;
- **F8** – применяется в отладочном режиме: выполнить следующую строку программы; если в строке есть обращение к процедуре (функции), исполнить ее и не проследивать ее работу;
- **F9** – компилировать программу, но не выполнять ее;
- **F10** – перейти к диалоговому выбору режима работы с помощью главного меню;
- **Ctrl+F9** – выполнить прогон программы: компилировать программу, находящуюся в редакторе, загрузить ее в оперативную память и выполнить, после чего вернуться в среду Турбо Паскаля;
- **Alt+F5** – сменить окно редактора на окно вывода результатов работы (прогона) программы.

Работа с меню

Для перехода из состояния редактирования к выбору из главного меню используется клавиша **F10**, для возврата в редактор – клавиша **Esc**. В активном меню указателем (цветом или оттенком) выделяется очередная опция. Для выбора того или иного приложения нужно переместить клавишами смещения курсора указатель к нужной опции и нажать **Enter**.

Следует учесть, что детальную информацию на английском языке о том или ином приложении (опции) можно получить с помощью справочной службы, если клавишами перемещения курсора сместить указатель к этой опции и нажать клавишу **F1**.

Все управление средой Турбо Паскаля осуществляется в основном с помощью системы последовательно разворачивающихся меню. Лишь одно из них – **главное меню** – постоянно присутствует на экране, остальные разворачиваются по мере выбора приложений.

Главное меню содержит фактически лишь оглавление дополнительных меню. В последних сгруппированы близкие по своему роду действия, условное название которых и служит наименованием соответствующего элемента главного меню:

- **File** (Файл) – действия с файлами и выход из системы;
- **Edit** (Редактировать) – восстановление испорченной строки и операции с временным буфером;
- **Search** (Искать) – поиск текста, процедуры, функции или места ошибки;
- **Run** (Работа) – прогон программы;
- **Compile** (Компилировать) – компиляция программы;
- **Debug** (Отладка) – отладка программы;
- **Tools** (Инструменты) – вызов вспомогательных программ (утилит);
- **Options** (Варианты) – установка параметров среды;
- **Window** (Окно) – работа с окнами;
- **Help** (помощь) – обращение к справочной службе.

При обращении к некоторым опциям открывается *диалоговое окно*. Сразу после развертывания последнего активизируется то или иное поле, которое выделяется цветом (оттенком) в активных полях ввода, переключаемых опций или выбора файла; кроме того, виден мигающий курсор. Основные правила при работе с диалоговым окном:

- для перехода от одного поля к другому предназначена клавиша табуляции **Tab**;
- для перехода внутри поля используются клавиши смещения курсора;
- закрыть диалоговое окно можно клавишей **Esc** (в этом случае не происходит никаких действий, связанных с окном) или клавишей **Enter** (тогда выполняются все, указанные в окне установки, или выбирается указанный файл);
- если по смыслу того или иного исполняемого действия необходимо ввести текстовую строку (например, имя файла), то сразу после раскрытия диалогового окна активизируется поле ввода с мигающим курсором. Следует набрать нужный текст и нажать **Enter**: текст будет введен, диалоговое окно закроется, но, если вы измените свое решение, достаточно нажать **Esc**, чтобы закрыть диалоговое окно без ввода текста;
- переключаемые опции задают выбор нужной настройки среды из двух или нескольких вариантов. Последние могут быть связаны с включением или отключением какого-либо параметра среды. Например, можно потребовать от компилятора использовать арифметический сопроцессор или не использовать его. Слева от таких опций в диалоговом окне имеется небольшое поле выбора, выделенное квадратными скобками. Включенный параметр отмечается символом [X] в этом поле. Если последнее пустое [], то данный параметр не задействован. Если опция задает выбор из нескольких вариантов, слева от указателя каждого из них имеется поле выбора, выделенное двумя круглыми скобками, причем выбранный вариант

- в диалоговом окне обязательно имеется несколько командных полей, которые располагаются в правой или нижней части окна и выделяются цветом. С каждым таким полем связана некоторая команда, которую можно выполнить, если активизировать поле клавишей табуляции и нажать **Enter**.

NEW. Создает и открывает новое окно редактора с именем NONAME00.PAS. Порядковый номер XX окна зависит от количества окон со стандартным именем NONAME, открытых к моменту обращения к опции.

OPEN. Открывает новое окно редактора и помещает в него указанный дисковый файл. В диалоговом окне, которое открывается при обращении к этой опции, можно в поле ввода написать нужное имя файла. Если в имени опущено расширение, среда добавит стандартное расширение .PAS. Имени файла может предшествовать путь. Нужный файл вы можете также выбрать из поля выбора, предварительно активизировав это поле клавишей **Tab**. Поле **OPEN** (открыть) используется для команды чтения файла в новое редакторское окно, **REPLACE** (заменить) – для замены существующего в активном редакторском окне текста на текст, считанный из файла, **CANCEL** – закрыть диалоговое окно и не выполнять никаких действий. Опция вызывается непосредственно из редактора клавишей **F3** (рис. П.2).

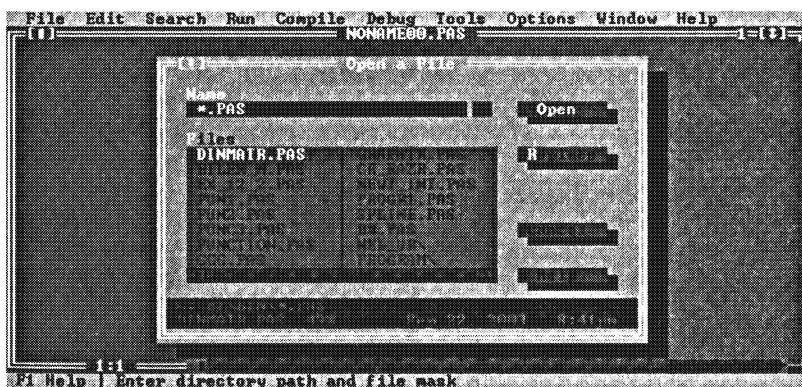


Рис. П.2 ▼ Диалоговое окно OPEN

SAVE. Записывает содержимое активного окна редактора в дисковый файл. Если это окно связано с именем NONAMExx.PAS, среда запросит новое имя файла (см. ниже опцию **SAVE AS**). Опция вызывается непосредственно из редактора клавишей **F2**.

SAVE AS. Записывает содержимое активного окна редактора в дисковый файл под другим именем. В диалоговом окне этой опции в поле ввода нужно написать имя того файла, в который будет переписано содержимое активного окна редактора. Вы можете выбрать уже существующий файл из поля выбора. В таком случае, в зависимости от настройки среды, старое содержимое файла будет уничтожено или сохранено в виде страховочной копии с расширением .BAK (настройку среды см. в опции OPTIONS/ENVIRONMENT).

SAVE ALL. Записывает содержимое всех окон редактора в соответствующие дисковые файлы.

CHANGE DIR. Позволяет изменить текущий каталог пользователя. В поле выбора диалогового окна этой опции приводится дерево каталогов текущего диска, перемещаясь по которому, можно указать на нужный каталог, после чего с помощью командного поля **CHDIR** (изменить каталог) сменить текущий каталог. Если выбран указатель **DRIVES** (дисководы), можно изменить также текущий диск. Командное поле **REVERT** (возвращаться) позволит восстановить прежний текущий каталог, если вы решите отказаться от сделанного изменения (эта команда действует до момента закрытия окна).

PRINT. Печатает содержимое активного окна редактора на принтере или выводит его в файл.

PRINT SETUP. Настраивает среду на печать текущего файла.

DOS SHELL. Обеспечивает временный выход в DOS. Турбо Паскаль остается резидентным в оперативной памяти и занимает значительную ее часть, поэтому под управлением DOS в таком состоянии могут выполняться только сравнительно небольшие по объему программы. Для возврата в Турбо Паскаль нужно в ответ на запрос DOS напечатать слово **EXIT** и нажать **Enter**.

EXIT. Завершает работу с Турбо Паскалем. Опция вызывается непосредственно из редактора командой **Alt+X**.

Меню EDIT

UNDO. В активном окне редактора восстанавливает только что уничтоженную командой **Ctrl+Y** или измененную строку. Турбо Паскаль создает специальный буфер изменений для каждой страницы редактора. Последовательное использование опции **UNDO** может отменить все сделанные изменения текста. Опция вызывается непосредственно из окна редактора клавишами **Alt+Backspace**.

REDO. Отменяет выполнение предыдущей команды.

CUT. Удаляет выделенный блок из окна редактора и переносит его в буфер обмена Clipboard. Опция вызывается непосредственно из редактора командой **Shift+Del**.

COPY. Копирует выделенный блок из окна редактора в буфер обмена Clipboard. Активируется командой **Ctrl+Ins**.

PASTE. Копирует содержимое буфера обмена Clipboard в окно редактора. Содержимое буфера остается без изменений и может использоваться повторно. Опция вызывается непосредственно из редактора нажатием клавиш **Shift+Ins**.

CLEAR. Удаляет из окна редактора выделенный блок, но не помещает его в буфер, то есть фрагмент безвозвратно теряется. Активизируется командой **Ctrl+Del**.

SHOW CLIPBOARD. Показывает содержимое буфера обмена.

Меню **SEARCH**

FIND. Обеспечивает поиск нужного фрагмента текста в активном окне редактора. В момент обращения к этой опции в поле выбора диалогового окна содержится слово, отмеченное курсором в активном окне редактора. Можно ввести новое слово или текстовую строку, положение которой в редактируемом файле вам необходимо найти, нажать **Enter**, после чего редактор обнаружит этот фрагмент в тексте и установит курсор на его начало.

SEARCH AGAIN. Повторяет поиск или поиск и замену фрагмента текста для ранее установленных параметров.

GO TO LINE NUMBER. Осуществляет позиционирование курсора в окне редактора на первую строку с указанным номером.

SHOW LAST COMPILER ERROR. Показывает строку текста программы, в которой была обнаружена синтаксическая ошибка при последнем прогоне компилятора.

FIND ERROR. Отыскивает в тексте программы строку, вызвавшую ошибку периода исполнения программы.

FIND PROCEDURE. Позволяет в режиме отладки обнаружить в тексте программы нужную процедуру или функцию.

Меню **RUN**

RUN. Осуществляет компиляцию, компоновку и исполнение (прогон) программы из файла редактора. Если программа уже откомпилирована к этому моменту, то среда сразу начнет ее прогон. Опция вызывается непосредственно из редактора командой **Ctrl+F9**.

GO TO CURSOR. Начинает или продолжает режим отладки исполняемой программы под управлением встроенного отладчика. Вначале осуществляются все действия по компиляции и компоновке программы, затем последняя начинает работать обычным образом (экран переходит в режим воспроизведения окна программы) и останавливается перед выполнением первого оператора из той строки, на которую указывает курсор. В этот момент экран возвращается в режим воспроизведения окна редактора, а строка с курсором выделяется цветным прямоугольником. Можно перевести курсор к новой строке и вновь выбрать данную опцию – программа остановится перед выполнением нового оператора и т.д. В этом режиме доступны все средства встроенного отладчика. Для прекращения отладки нажмите клавиши **Ctrl+F2**. Опция вызывается непосредственно из редактора клавишей **F4**.

TRACE INTO. Начинает или продолжает режим отладки исполняемой программы под управлением встроенного отладчика. Если к моменту обращения к этой опции режим отладки не был запущен, он включается точно так, как

если бы была вызвана опция **GO TO CURSOR**, однако программа останавливается перед первым исполняемым оператором, то есть указатель будет отмечать слово *Begin*, открывающее раздел операторов основной программы. Если режим отладки уже был запущен, вызов этой опции приведет к выполнению всех действий, запрограммированных в текущей строке, и указатель сместится к следующей строке программы. Если текущая строка содержит обращение к процедуре или функции, управление будет передано внутрь этой процедуры (функции), и программа остановится перед исполнением ее первого оператора. Таким образом, с помощью данной опции можно по шагам проследивать исполнение всех нестандартных процедур (функций). Опция вызывается непосредственно из редактора клавишей **F7**.

STEP OVER. Так же, как и предыдущая опция, начинает или продолжает пошаговое проследивание работы программы, но работа вызываемых процедур и функции не проследивается. Опция вызывается непосредственно из редактора клавишей **F8**.

PROGRAM RESET. Сбрасывает все ранее задействованные отладочные средства и прекращает отладку программы. Удаляет исполнявшуюся программу из памяти и закрывает все открытые в ней в этот момент файлы. Активизируется нажатием клавиш **Ctrl+F2**.

PARAMETERS. Позволяет задать текстовую строку параметров, которые DOS передает вызываемой программе, находящейся в окне редактора, при ее прогоне.

Меню **COMPILE**

COMPILE. Компилирует программу или модуль, который загружен в данный момент в активное окно редактора. Если в этой программе (модуле) содержатся обращения к нестандартным модулям пользователя, последние уже должны быть откомпилированы и храниться на диске в виде TPU-файлов. Опция вызывается непосредственно из редактора командой **Alt+F9**.

MAKE. Создает программу, которая, возможно, содержит включаемые файлы и/или обращения к нестандартным модулям. Опция существенно упрощает процесс разработки многофайловых программ, так как всегда компилируется только тот минимум файлов, которых коснулись сделанные в программе изменения. Опция вызывается непосредственно из редактора клавишей **F9**.

BUILD. Эта опция аналогична опции **MAKE** за одним исключением: отыскивается соответствующий PAS-файл и осуществляется его перекомпиляция независимо от того, были ли сделаны в нем изменения или нет. После компиляции в этом режиме можно быть уверенным в том, что в полученной программе учтены все изменения.

DESTINATION. Эта опция управляет выходом компилятора.

PRIMARY FILE. Задает имя начального файла. Если имя задано, то вне зависимости от того, какая часть программы загружена в данный момент в окна редактора, ее компиляция в режимах **RUN**, **MAKE** и **BUILD** будет начинаться с этого файла.

CLEAR PRIMARY FILE. Очищает имя начального файла, заданное опцией **PRIMARY FILE**.

INFORMATION. Показывает статистику программы.

Меню **DEBUG**

BREAKPOINTS. Эта опция позволяет просмотреть все контрольные точки и при необходимости удалить, переместить любую из них или задать условия ее работы.

CALL STACK. Делает активным окно программного стека, в котором отображаются все вызовы процедур и функций. Внизу стека находится **PROGRAM**, то есть имя программы, в вершине стека – текущая процедура (функция). Каждое новое обращение к процедуре отображается в этом окне в виде имени подпрограммы и списка параметров вызова. Опция вызывается из редактора командой **Ctrl+F3**.

REGISTER. Делает активным окно регистров. В нем отображается текущее состояние всех регистров микропроцессора ПК.

WATCH. Вызывает окно отладки.

OUTPUT. Делает активным окно программы.

USER SCREEN. Открывает окно программы и распаивает его на весь экран. Вызывается из редактора командой **Alt+F5**.

EVALUATE/MODIFY. Эта опция дает возможность в процессе отладки просмотреть содержимое любой переменной или найти значение того или иного выражения. При необходимости с ее помощью можно установить новое значение любой переменной. При обращении к ней на экране разворачивается диалоговое окно, содержащее три поля **EXPRESSION** (выражение), **RESULT** (результат) и **NEW VALUE** (новое значение). В первом поле следует ввести имя любой переменной или некоторое выражение. Сразу после того, как вы нажмете **Enter**, в поле **RESULT** появится соответствующее значение или сообщение **unknown identifier** (неопределенный идентификатор), если такая переменная не определена в программе. К моменту вызова опции программа должна находиться в режиме отладки, в противном случае это сообщение будет даваться для любых переменных и выражений с их участием. Если было запрошено значение переменной, то можно перевести курсор в нижнее поле **NEW VALUE** и установить новое значение переменной, которое будет немедленно передано в программу. Для выхода из диалогового окна используйте клавишу **Esc** или поле **Cancel**. Опцию можно вызвать непосредственно из редактора командой **Ctrl+F4**. Отметим, что эта опция может использоваться как встроенный в Turbo Паскаль калькулятор.

ADD WATCH. С помощью данной опции можно указать отладчику те переменные и/или выражения, за изменением значений которых хочется понаблюдать при отладке программы. Указанные переменные и выражения вместе с их текущими значениями будут постоянно содержаться в окне наблюдения, доступ к которому возможен с помощью клавиши **F6**. Если это окно будет активным, то перемещаться в нем можно, вызывая при необходимости «прокрутку»

его содержимого. Опцию можно вызвать непосредственно из редактора командой **Ctrl+F7**.

ADD BREAKPOINT. С помощью этой опции можно установить в текущей строке (в ней находится курсор) контрольную точку. Если для нее установлена контрольная точка, строка выделяется цветом (яркостью). В программе можно указать произвольное количество контрольных точек. После запуска программы с установленными контрольными точками (точкой) отладчик прекратит исполнение программы перед выполнением того оператора, который содержится в первой (по логике работы программы) контрольной точке. При этом на экране появится окно редактора с контрольной точкой, и среда перейдет к режиму отладки программы. Если контрольная точка задана для строки, не содержащей исполняемого оператора (например, для строки со словом *begin*), программа остановится перед первым после этой строки исполняемым оператором. Останов в контрольной точке можно сделать условным. Однажды указанная контрольная точка действует на каждое очередное обращение к соответствующей строке программы. В диалоговом окне опции поле **CONDITION** задает условие останова: это может быть произвольное логическое выражение с использованием любых переменных, констант, вызовов функций. Если к моменту исполнения оператора с контрольной точкой это выражение имеет значение **TRUE**, произойдет останов прогона, и среда перейдет к режиму отладки. Поле **PASS COUNT** указывает количество обращений к оператору с контрольной точкой, после которого произойдет останов. С помощью команды **Ctrl+F8** контрольную точку можно установить/снять непосредственно из режима редактирования.

Меню **TOOLS**

MESSAGES. Активизирует окно сообщений, которое содержит вывод инструментальных программ типа **GREP** и позволяет использовать эти сообщения для поиска нужных фрагментов в текстах программ.

GO TO NEXT. Ищет фрагмент, заданный следующим сообщением в окне **Messages**. Опция вызывается непосредственно из окна редактора клавишами **Alt+8**.

GO TO PREVIOUS. Ищет фрагмент, заданный предыдущим сообщением в окне **Messages**. Опция вызывается непосредственно из окна редактора клавишами **Alt+P**.

GREP. Иницирует работу утилиты **GREP**. Опция вызывается непосредственно из окна редактора клавишами **Shift+F2**.

Меню **OPTIONS**

COMPILER. Эта опция задает несколько параметров, с помощью которых можно управлять генерацией машинного кода программы.

MEMORY SIZES. В диалоговом окне данной опции можно регулировать размеры памяти, которую занимает работающая программа.

LINKER. В диалоговом окне имеются две группы переключаемых опций, с помощью которых регулируется режим работы компоновщика Турбо Паскаля.

DEBUGGER. Эта опция определяет используемый отладчик и режим обновления экрана дисплея в процессе отладки. Если активна опция **Integrated**, к программе будет добавлена информация, необходимая для работы встроенного отладчика. Только в этом состоянии опции можно использовать контрольные точки и пошаговую отладку. При активизации опции **Standalone** к ехе-файлу программы будут добавлены соответствующие таблицы, которые позволят вести отладку программы вне среды Турбо Паскаля с помощью внешнего отладчика TD.EXE. Три других опции сообщают среде, в каких случаях следует переключать экран с воспроизведения окна редактора на окно программы.

OPEN. Здесь можно указать имя конфигурационного файла, из которого среда должна получить информацию о своей настройке.

SAVE. Сохраняет текущую настройку среды в конфигурационном файле.

SAVE AS. С помощью этой опции можно указать каталог и файл, в котором среда будет сохранять свою настройку (по умолчанию это файл TURBO.TP).

Меню WINDOW

TILE. Располагает окна так, чтобы каждое было видно на экране, и все они имели бы приблизительно одинаковые размеры.

CASCADE. Располагает окна редактора таким образом, чтобы были видны рамки каждого из них.

CLOSE ALL. Закрывает все открытые окна.

REFRESH DISPLAY. Удаляет следы вывода программы, работавшей в режиме отладки с усыновленной опцией **OPTIONS/DEBUGGER/DISPLAY SWAPPING/NONE**.

SIZE/MOVE. Эта опция обеспечивает перемещение окна по экрану и/или изменение его размеров. Вызывается из редактора командой **Ctrl+F5**.

ZOOM. Распахивает активное окно на весь экран или возвращает ему прежний вид. Вызов из редактора клавишей **F5**.

NEXT. Активизирует очередное окно. Вызывается из редактора клавишей **F6**.

PREVIOUS. Активизирует предыдущее активное окно. Вызывается из редактора командой **Shift+F6**.

CLOSE. Закрывает активное окно. Вызывается из редактора командой **Alt+F3**.

LIST. Выводит на экран список всех открытых окон среды. Вызывается из редактора командой **Alt+0**.

Меню HELP

CONTENTS. Открывает содержание справочной службы.

INDEX. Выводит на экран алфавитный список всех ссылок справочной службы. Вызывается из редактора командой **Shift+F1**.

TOPIC SEARCH. Вызов справки о служебном слове под курсором. Активизируется командой **Ctrl+F1**.

PREVIOUS TOPIC. Выводит на экран предыдущее справочное сообщение. Вызывается из редактора командой **Alt+F1**.

HELP ON HELP. Дает справку о том, как пользоваться справочной службой.

FILES. С помощью этой опции можно установить нужные файлы справочной службы.

COMPILER RERECEIVES. Показывает справку о директивах компилятора.

RESERVED WORDS. Дает справку о зарезервированных словах.

STANDARD UNITS. Показывает справку о стандартных модулях.

TURBO PASCAL LANGUAGE. Открывает доступ к справке о языке Турбо Паскаль.

ERROR MESSAGES. Показывает справку с сообщениями об ошибках.

ABOUT. Выводит информацию об авторских правах и версии Турбо Паскаля.

Директивы компилятора

В меню **OPTIONS/COMPILER** включены опции, с помощью которых можно управлять работой компилятора. В ряде случаев необходимо временно отменить действие той или иной опции при трансляции некоторого фрагмента программы. Особенно часто, например, такая необходимость возникает при обращении к диску: если программа пытается прочитать несуществующий файл или записать данные на защищенный диск, возникнет ошибка периода исполнения, и программа аварийно закончит свою работу. В то же время, если отключить опцию **I/O CHECKING**, программа сможет проанализировать последствия обращения к диску и предпринять альтернативные действия.

В Турбо Паскале можно использовать директивы компилятора, которые в виде особым образом оформленных комментариев вставляются в текст программы и модифицируют те или иные возможности компилятора в процессе его работы. Директивы могут быть переключающими, условными и параметрическими. Первые воздействуют на те опции, которые включены в диалоговое окно **OPTIONS/COMPILER**, условные директивы определяют условия, при которых компилируются те или иные фрагменты программы. Параметрические директивы задают параметры, которые должен учитывать компилятор.

Все директивы оформляются в виде особых комментариев: они обрамляются фигурными скобками, а за открывающей скобкой должен без пробелов следовать знак доллара. Как только в процессе разбора исходного текста программы компилятор встретит такого рода последовательность символов, он воспримет их как директиву и неявным образом изменит свою работу.

Переключающая директива содержит букву, обозначающую опцию, и знак «+» или «-». Символ «+» означает установку опции в активное состояние, знак «-» – в пассивное состояние. Например, директива **{\$1—}** означает временное отключение контроля ошибок ввода-вывода, директива **{\$R+}** – включение контроля границ диапазона. В одной директиве можно перечислить несколько опций.

Следует учесть, что директивы компилятора действуют от момента своего появления в тексте до конца текущего модуля, то есть локализируются в теле модуля. В то время как опции, установленные в самой среде, распространяются на все модули и основную программу. В случае конфликта между директивами и опциями предпочтение отдается директивам. Таким образом, правильно расставленные директивы обеспечивают нужную компиляцию программы независимо от настройки среды. Они особенно полезны в случае, когда данная процедура осуществляется автономным компилятором TPC.EXE.

Некоторые директивы компилятора могут действовать только на часть текста программы – они называются локальными. В отличие от них, глобальные директивы располагаются в самом начале текста программы (модуля) и влияют сразу на всю программу (модуль).

Ниже приводится список всех директив компилятора. В скобках дается действие директивы для знака «-». Знаком «!» отмечены локальные директивы:

- {A+} – выравнивать данные на границу слова (байта);
- {B+} – вычислять логические выражения полностью (до получения результата);
- {D+} – разрешить (запретить) работу со встроенным отладчиком;
- {E+} – включить (отключить) режим программной эмуляции сопроцессора;
- {F+} – использовать дальнюю (ближнюю) модель вызова;
- {C+} – использовать (не использовать) полный набор команд микропроцессора Intel 80286 (микропроцессора Intel 8088);
- {I+} – активизировать (не активизировать) контроль операций ввода-вывода;
- {L+} – включить (не включать) локальные символы в информацию для отладчика;
- {V+} – использовать числовой сопроцессор (реализовать операции с плавающей точкой программно);
- {O+} – разрешить (не разрешать) создание оверлейной структуры;
- {R+} – включить (отключить) контроль границ диапазона;
- {S+} – активизировать (не активизировать) контроль возможного переполнения стека;
- {V+} – включить (отключить) контроль длины строк при обращении к процедуре или функции;
- {X+} – использовать (не использовать) расширенный синтаксис.

Используемая литература

1. Алексеев Е. Р., Муранова И. В., Палкин А. Е. Массивы более 64 Кб в Турбо Паскале // Компьютеры + Программы. – 1996. – № 7. – С. 73–75.
2. Белецкий Ян. Турбо Паскаль с графикой для персональных компьютеров. – М.: Машиностроение, 1991. – 320 с.
3. Бородич Ю. С., Вальвачев А. Н., Кузьмич А. И. Паскаль для персональных компьютеров. – Минск: Вышэйшая школа, 1991. – 365 с.
4. Вержбицкий В. М. Основы численных методов. – М.: Высшая школа, 2002. – 840 с.
5. Гмурман В. Е. Теория вероятностей и математическая статистика. – М.: Высшая школа, 2002. – 479 с.
6. Демидович Б. П., Марон И. А. Основы вычислительной математики. – М.: Наука, 1970. – 664 с.
7. Линник Ю. В. Метод наименьших квадратов и основы математико-статистической теории обработки наблюдений. – Л.: Физматгиз, 1962. – 352 с.
8. Поляков Д. Б., Круглов И. Ю. Программирование в среде Турбо Паскаль (версия 5.5). – М.: МАИ, РОСВУЗНАУКА, 1992. – 576 с.
9. Фаронов В. В. Турбо Паскаль 7.0. Начальный курс. – М.: Нолидж, 1997. – 616 с.
10. Фаронов В. В. Турбо Паскаль 7.0. Практика программирования. – М.: Нолидж, 1997. – 432 с.

Предметный указатель

А

Адаптер 152
Алгоритм 11
 блок-схема 12
 линейный 14
 недостатки 17
 построения графика непрерывной функции 167
 разветвленной структуры 15
 циклической структуры 20

Б

Блок модификации 48

В

Выражение 31
 арифметические операции 31
 логические операции 31
 операции отношения 31

Д

Данные 28
 константы 28
 переменные 28
 типы данных 28
 скалярные 28, 29
 скалярные вещественные 29
 скалярные интервальные 30
 скалярные логические 29
 скалярные перечислимые 29

 скалярные символьные 29
 скалярные целочисленные 28
 структурированные 28, 30
 структурированные массивы 30
 структурированные строки 31

Драйвер 151

З

Заголовок программы 34

И

Идентификатор 27
Интегрирование 241
 метод прямоугольников 241
 метод Симпсона 244
 метод трапеций 243
Интерполяция 221
 канонический полином 222
 линейная 237
 полином Лагранжа 225
 сплайн 226

К

Куча 125

М

Массив 63
 динамический 127
Матрица 82
Метод Гаусса 189

- вычисление 198
- вычисление обратной матрицы 195
- решение систем линейных уравнений 190
- Метод касательных (Ньютона) 59
- Метод наименьших квадратов 205
- Метод половинного деления 56
- Метод простой итерации 60
- Метод хорд 57
- Модуль 133, 178
 - CRT 134
 - PRINTER 141
 - заголовок 178
 - иницирующая часть 179
 - интерфейсная часть 178
 - исполняемая часть 179

О

- Оператор
 - break 49
 - continue 49
 - exit 50
 - halt 50
 - варианта case 42
 - ввода 36
 - вывода 37
 - присваивания 36
 - составной 39
 - условный if 40
 - цикла for ... do 47
 - цикла repeat ... until 45
 - цикла while ... do 43
- Операторы 35
 - простые 36
 - структурированные 36

П

- Память
 - динамическая 123
 - сегмент данных 123
 - указатель 124
 - нетипизированный 124
 - типизированный 124

- ячейка
 - адрес 124
- Подпрограмма 96
 - переменные
 - глобальные 101
 - локальные 101
 - процедуры 96
 - функции 96
- Процедура 96
 - Arc 162
 - Bar 163
 - Bar3D 163
 - Circle 158
 - ClearDevice 158
 - ClearViewPort 158
 - CloseGraph 154
 - cleoln 136
 - delay 138
 - delete 143
 - delline 136
 - DetectGraph 155
 - Ellipse 162
 - FillEllipse 164
 - Floodfill 163
 - GetAspectRatio 158
 - GetImage 166
 - GetModeRange 155
 - gotoxy 136
 - highvideo 136
 - InitGraph 152
 - insert 143
 - Line 157
 - LineRel 160
 - LineTo 160
 - lowvideo 136
 - MoveRel 157
 - MoveTo 156
 - normvideo 136
 - nosound 138
 - OutText 164
 - OutTextXY 165
 - PieSlise 164
 - PutImage 167
 - PutPixel 157

Rectangle 158
RestoreCRTMode 154
Sector 164
SetAspectRatio 158
SetBkColor 157
SetColor 157
SetFillStyle 162
SetGraphMode 155
SetLineStyle 161
SetTextStyle 165
SetViewPort 157
sound 138
str 143
textbackground 135
textcolor 135
textmode 134
val 144
window 135
вызов 97
заголовок 97
описание 97
параметры
 открытые массивы 102
 параметр-константа 103
 параметры-значения 98
 параметры-переменные 98
 фактические 97
 формальные 97

Процедурные типы 103

Р

Расширенный синтаксис 104

С

Строка 142
Структура программы 35

Ф

Файл

буфер файла 109
закрыть 109
запись в файл 110

компонент 108
конец файла 110
открыть 109
переменная
 файловая 109
текстовый 108
 дозапись 119
текущая позиция 112
типизированный 108
удаление компонентов 115
удалить 110
установить указатель файла 113
чтение из файла 110

Функции

 стандартные 32

Функция

 chr(x) 134
 copy 143
 GetGraphMode 155
 GetMaxX 156
 GetMaxY 156
 GetPixel 160
 GetX 156
 GetY 156
 GraphResult 154
 ImageSize 166
 length 143
 ord(x) 134
 Str 165
 wherex 136
 wherey 136
 обращение 101
 описание 100

Ц

Цикл 20

 параметр цикла 21
 с известным числом повторений 21
 с неизвестным числом повторений 23
 с постусловием 21
 с предусловием 21
 тело цикла 20

Для заметок

Алексеев Евгений Ростиславович,
Чеснокова Оксана Витальевна,
Павлыш Владимир Николаевич,
Славинская Людмила Васильевна

Турбо Паскаль 7.0

Главный редактор *Захаров И. М.*
editor-in-chief@ntpress.ru

Выпускающий редактор *Мирошникова Е. Г.*
Верстка *Дудатий А. М.*
Графика *Салимонов Р. В.*
Дизайн обложки *Клубничкин Д. Е.*

Общероссийский классификатор продукции
ОК-005-93, том 2; 953005 — учебная литература
Санитарно-эпидемиологическое заключение
№ 77.99.02.953.Д.000577.02.04 от 03.02.2004 г.

ООО «Издательство АСТ»
667000, Республика Тыва, г. Кызыл, ул. Кочетова, д. 28
Наши электронные адреса: WWW.AST.RU E-mail: astpub@aha.ru

Издательство «НТ Пресс», 105023, Москва, пл. Журавлева, д. 2/8.

Отпечатано с готовых диапозитивов
в ООО «Типография ИПО профсоюзов Профиздат».
109044, Москва. Крутицкий вал, 18.

АЛЕКСЕЕВ Е. Р., ЧЕСНОКОВА О. В.,
ПАВЛЫШ В. Н., СЛАВИНСКАЯ Л. В.

ТУРБО ПАСКАЛЬ 7.0

В книге описана методика составления алгоритмов с помощью блок-схем, содержится описание языка программирования Турбо Паскаль версии 7.0. Изложены методы решения нелинейных уравнений, систем линейных алгебраических уравнений, обработки экспериментальных данных, численные методы интегрирования. Приведены практические примеры программирования, ряд функциональных модулей для рисования графиков, работы с массивами и упражнения для самостоятельной работы.

Для школьников и студентов, изучающих программирование, а также для всех желающих изучить язык Турбо Паскаль 7.0.



ЧИСЛЕННЫЕ МЕТОДЫ