

Ю.А. Алексеев, А.С. Ваулин,
А.В. Куров

Практикум по программированию

Обработка ЧИСЛОВЫХ ДАННЫХ

Под редакцией Б.Г. Трусова

*Допущено Учебно-методическим объединением вузов
по университетскому политехническому образованию
в качестве учебного пособия
для студентов высших учебных заведений
машиностроительного и приборостроительного профиля,
изучающих курс «Информатика» в соответствии
с Государственными образовательными стандартами*

Москва
Издательство МГТУ им. Н.Э. Баумана
2008

УДК 681.3.06(075)
ББК 22.18
А47

Рецензенты:

кафедра «Автоматизированные системы управления» МАДИ (ГТУ)
(зав. кафедрой, д-р техн. наук, проф. *А.Б. Николаев*);
д-р техн. наук, проф. *И.П. Норенков*;
канд. техн. наук, доц. *Г.С. Иванова*

Алексеев Ю.Е., Ваулин А.С., Куров А.В.

А47 Практикум по программированию: Обработка числовых данных: Учеб. пособие / Под ред. Б.Г. Трусова. — М.: Изд-во МГТУ им. Н.Э. Баумана, 2008. — 288 с.

ISBN 978-5-7038-3159-5

Приведены обработка числовых типов данных, краткие теоретические сведения, примеры программ реализации изучаемых алгоритмов, а также задания для выполнения лабораторных работ по каждой из рассматриваемых в курсе тем.

Пособие ориентировано на среду программирования Delphi и содержит большое количество важных алгоритмов решения инженерных задач. Представлены полные комплекты заданий (не менее 25 вариантов), имеющих разнообразный характер, но одинаковый уровень сложности.

Материал пособия авторы используют при проведении практических занятий в МГТУ им. Н.Э. Баумана.

Для студентов первого курса машино- и приборостроительных специальностей. Может быть полезно преподавателям как сборник заданий при проведении лабораторных работ.

УДК 681.3.06(075)
ББК 22.18

ISBN 978-5-7038-3159-5

© Алексеев Ю.Е., Ваулин А.С.,
Куров А.В., 2008
© Оформление. Изд-во МГТУ
им. Н.Э. Баумана, 2008

Оглавление

Введение	6
1. Краткая справка по языку Object Pascal и разработке консольных приложений в среде Delphi	9
2. Программы линейной структуры	21
2.1. Средства разработки программ линейной структуры	21
Целые типы данных	21
Вещественные типы данных	23
Стандартные функции для обработки числовых данных	24
Арифметические выражения	25
Оператор присваивания	26
Ввод данных с клавиатуры	27
Вывод данных в окно программы	28
2.2. Приемы, используемые для минимизации вычислений	31
2.3. Примеры выполнения задания	32
2.4. Задания А для самостоятельной работы	36
2.5. Задания Б для самостоятельной работы	39
3. Программы разветвляющейся структуры	44
3.1. Средства разработки программ разветвляющейся структуры ..	44
Условные операторы	44
Булев тип	46
3.2. Примеры выполнения задания	48
3.3. Задания для самостоятельной работы	51
4. Программы циклической структуры	57
4.1. Средства разработки программ циклической структуры	57
Цикл с параметром	57
Цикл с предусловием	58
Цикл с постусловием	59
4.2. Вычисление и вывод данных в виде таблицы	60
4.3. Пример выполнения задания с использованием цикла while ..	62
4.4. Пример выполнения задания с использованием цикла for	64
4.5. Задания для самостоятельной работы	65
4.6. Сохранение результатов вычислений в массиве	78
4.7. Пример выполнения задания	80
4.8. Задания для самостоятельной работы	83
4.9. Приемы вычисления сумм, произведений и экстремальных значений	86
Вычисление суммы и произведения	86
Нахождение наибольшего или наименьшего значения	87
4.10. Пример выполнения задания А	90
4.11. Задания А для самостоятельной работы	92
4.12. Пример выполнения задания Б	95
4.13. Задания Б для самостоятельной работы	96
4.14. Вычисление суммы бесконечного ряда с заданной точностью	99

4.15. Вывод рекуррентной формулы для вычисления члена ряда ...	100
Способы вычисления значения члена ряда	101
4.16. Примеры выполнения задания	102
4.17. Задания для самостоятельной работы	105
4.18. Уточнение корней уравнений	111
Метод простых итераций	111
Метод половинного деления	115
Метод касательных	116
4.19. Пример выполнения задания	117
4.20. Задания для самостоятельной работы	119
4.21. Вычисление определенных интегралов	121
4.22. Пример выполнения задания	125
4.23. Задания для самостоятельной работы	127
5. Организация программ со структурой вложенных циклов	129
5.1. Вычисление определенного интеграла с заданной точностью	132
5.2. Задания для самостоятельной работы	137
5.3. Вычисление наибольшего (наименьшего) значения функции с заданной точностью на заданном интервале	138
5.4. Задания для самостоятельной работы	144
5.5. Обработка матриц	146
5.6. Примеры выполнения задания на обработку матриц	150
5.7. Задания для самостоятельной работы	158
5.8. Методы сортировки массивов	163
Метод включения с сохранением упорядоченности (метод прямого включения или сортировка вставками)	163
Метод прямого обмена (метод пузырька)	165
Метод прямого выбора (сортировки посредством выбора) и его модификации	166
5.9. Пример выполнения задания	168
5.10. Задания для самостоятельной работы	169
6. Программирование с использованием подпрограмм	172
6.1. Процедуры	172
6.2. Пример выполнения задания	178
6.3. Задания для самостоятельной работы	180
6.4. Функции	184
6.5. Пример выполнения задания	188
6.6. Задания для самостоятельной работы	191
6.7. Рекурсивные подпрограммы	196
6.8. Пример выполнения задания на составление рекурсивной подпрограммы	199
6.9. Задания для самостоятельной работы	202
6.10. Дополнительные сведения о подпрограммах и массивах	208
Параметры — открытые массивы	209
Динамические массивы	212
Перегружаемые подпрограммы	215
Параметры со значениями по умолчанию	216
Примеры организации программ с подпрограммами	217

7. Модули пользователей	225
7.1. Создание и использование модулей	225
7.2. Пример выполнения задания	240
7.3. Задания для самостоятельной работы	251
7.4. Пример выполнения задания	258
7.5. Задания для самостоятельной работы	265
Приложение 1. Функция перекодировки кириллицы	273
Приложение 2. Дополнительные сведения по обработке исключений	274
Приложение 3. Основные формулы, используемые при решении геометрических задач	280
Список литературы	282
Алфавитный указатель	283

Введение

В связи с ростом роли информатики в жизни современного общества больше внимания уделяется и преподаванию этой дисциплины в вузах. По сложившейся традиции значительное место в курсе информатики в технических университетах занимает раздел, связанный с изучением языков программирования и разработкой на изучаемом языке алгоритмов решения важнейших инженерных задач.

Изучение алгоритмов решения основных инженерных задач (характерных приемов программирования) рассматривается как база для дальнейшего освоения дисциплины — студенты учатся работать с различными типами и структурами данных, создавать алгоритмы решения разнообразных задач. Конкретный язык программирования, на котором реализуются рассматриваемые алгоритмы, выступает в этом случае как конкретный инструмент для практического воплощения основных теоретических положений.

В ходе последующего изучения дисциплины при решении более сложных задач и обработке разных типов и структур данных объясняется значение простейших алгоритмов как своего рода строительных блоков, на базе которых разрабатывают алгоритм решения поставленной задачи. Уяснив постановку задачи и разрабатывая алгоритм ее решения, студенты должны выделить основные этапы решения, которые чаще всего представляют собой рассмотренные ранее приемы программирования и алгоритмы. Разработка и реализация алгоритмов решения задач позволяет попутно добиться еще одного важного результата — формирования основ логического мышления.

Многолетняя практика преподавания информатики первокурсникам свидетельствует о том, что уровень начальной подготовки студентов различается существенным образом, у многих из них отсутствуют умения и навыки логического построения алгоритма решения поставленной задачи. Проблема усугубляется в дальнейшем еще и тем, что в силу разных причин студенты должны образом не осваивают раздел, посвященный разработке и реализации основных типов алгоритмов и характерных приемов программирования.

Цель данного учебного пособия — изложить в краткой форме основные принципы, правила построения и программирования алгоритмов различных типов (линейных, разветвляющихся, циклических, в том числе и с вложенными циклами), а также характерных приемов; показать возможности этих алгоритмов при решении практических задач. Теоретический материал сопровождается примерами программ, а также заданиями для выполнения лабораторных работ. В отличие от подобных пособий авторы стремились составить более сложные и интересные задания, в рамках одной темы выдержать одинаковый уровень сложности для разных вариантов. Систематическое выполнение предлагаемых заданий позволит студентам успешно решать задачи, предлагаемые при проведении рубежного контроля и на экзамене.

В пособии рассматривается программирование основных алгоритмических структур: линейной, разветвляющейся, циклической (циклы с заранее известным и неизвестным числом повторений). Изложены принципы организации простых и вложенных циклов, а также следующие характерные приемы программирования: 1) нахождение сумм и произведений; 2) нахождение минимального и максимального значения функции на заданном интервале; 3) вычисление суммы бесконечного ряда и уточнение корней уравнений; 4) запоминание результатов вычислений; 5) сортировка элементов массива (по убыванию или возрастанию).

Реализация указанных простейших алгоритмов требует умения программировать основные алгоритмические структуры и уже на базе этих простых алгоритмов решать более сложные инженерные задачи. Так, определение суммы позволяет вычислить значение определенного интеграла; умение найти значение интеграла и организовать вложенные циклы — вычислить значение интеграла с заданной точностью; умение находить минимальный (максимальный) элемент массива в сочетании с программированием вложенных циклов — отсортировать массивы и т. д.

В заключительном разделе пособия рассмотрено использование подпрограмм. Приведены основные правила организации подпрограмм-функций и подпрограмм-процедур, виды параметров и механизмы их передачи в подпрограммы. Изложены возможности языка программирования по созданию библиотек подпрограмм, одним из средств разработки которых является механизм модулей, а также правила организации модулей и примеры их использования при разработке достаточно объемных приложений.

В качестве инструментального средства программной реализации алгоритмов применяется язык Pascal, который изучается на большинстве специальностей университета. Авторы сочли необходимым включить раздел, содержащий основные сведения о среде программирования Delphi и подготовке в ней консольных приложений, поскольку именно эта среда используется при изучении курса «Информатика».

1. Краткая справка по языку Object Pascal и разработке консольных приложений в среде Delphi

Среда программирования Delphi предоставляет возможность разработки и отладки различных программных продуктов, в том числе приложений, работающих как с использованием графического интерфейса пользователя, так и в консольном режиме. Последний имеет интерфейс пользователя в виде текстового окна, называемого *окном программы*, в котором последовательно, строка за строкой, отображаются данные, вводимые пользователем с клавиатуры и выводимые программой. Позицию начала ввода или вывода в окне программы указывает *курсор* — мигающий символ, имеющий вид подчеркивания в режиме вставки или прямоугольника — в режиме замены. По умолчанию длина строки равна 80 знакам, а количество строк — 50. Изменить эти и другие параметры окна программы позволяет диалоговое окно, открывающееся при вводе команды *Свойства* в системном меню.

При вводе пользователь имеет возможность редактировать последние вводимые данные, используя символьные клавиши, а также клавиши BackSpace (удаление последнего введенного символа), Delete (удаление символа справа от курсора), Insert (переключение режимов вставки и замены), Стрелка вверх (удаление всех введенных символов), Стрелка влево (перемещение курсора в предыдущую позицию), Стрелка вправо (перемещение курсора в следующую позицию). Если в диалоговом окне команды *Свойства* установить на вкладке *Общие* флажок *Выделение мышью*, то становится возможным выделять части текста буксировкой мыши, копировать выделенное в буфер обмена щелчком правой клавиши и затем вставлять в позицию курсора щелчком правой клавиши. Завершается ввод нажатием клавиши Enter, при этом курсор перемещается в начало новой строки. Максимальная длина вводимой последовательности символов равна 254.

Вывод данных из программы выполняется в виде текста, символ за символом при автоматическом перемещении курсора в очередную позицию строки, а при достижении ее конца — в начало новой строки.

Консольный режим обычно используется, когда необходимо минимизировать время счета и расход оперативной памяти. Кроме того, консольный режим удобен для быстрой проверки и

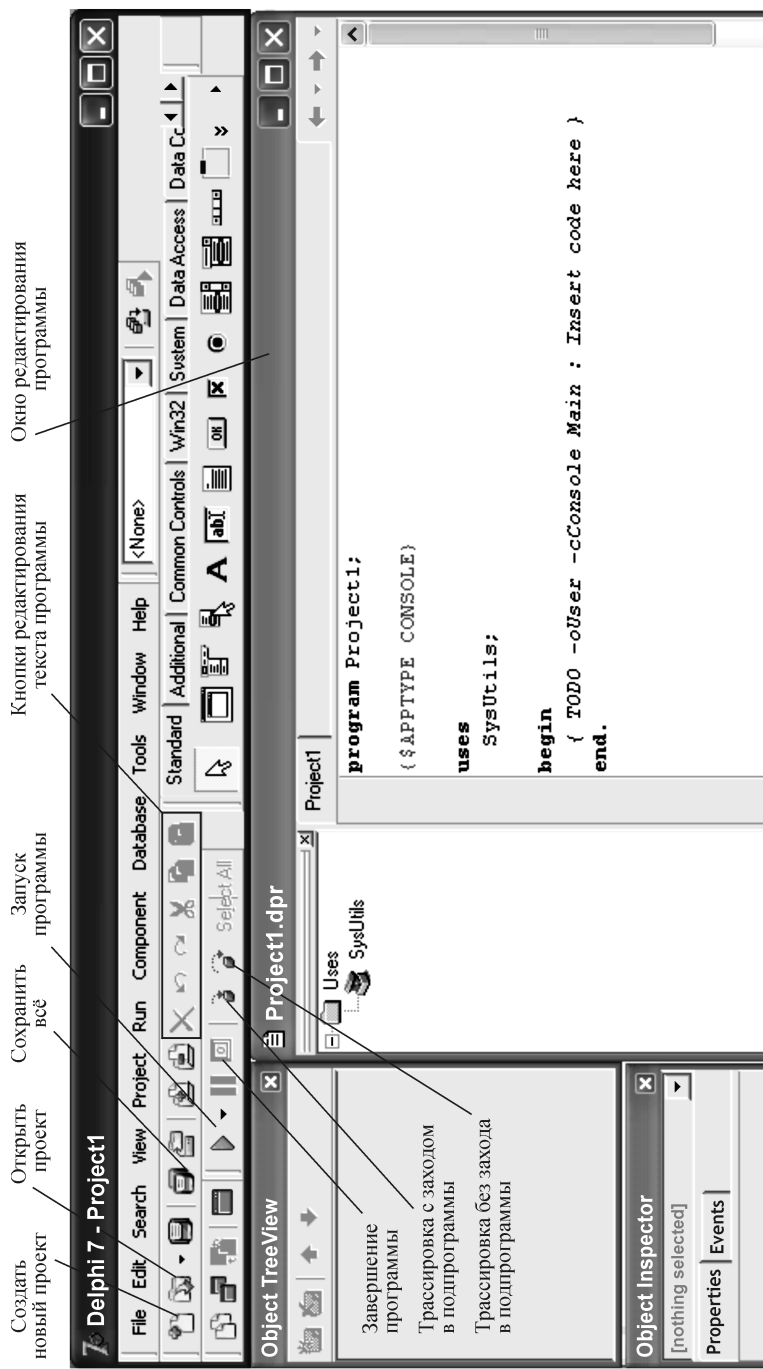



Рис. 1.1

отладки отдельных алгоритмов. Так как данный практикум ориентирован на развитие начальных навыков алгоритмизации и отладки небольших программ, предполагается использование консольного режима.

Для создания программы, работающей в консольном режиме, следует после запуска Delphi ввести команду *File/New/Other...* и выбрать вариант *Console Application*. При этом изменится набор окон, как показано на рис. 1.1 (окна Delphi при создании консольного приложения). В окне *редактирования* программы будет представлен стандартный шаблон консольной программы. Это работающая программа, которая при запуске только отображает окно и сразу же прекращает работу, закрывая его. Запуск программы в среде Delphi выполняется командой *Run/Run* или щелчком на кнопке  панели инструментов, или нажатием клавиши F9. Чтобы задержать окно программы до нажатия клавиши Enter, следует вставить в шаблон оператор ReadLn перед **end**. Впрочем, закрыть окно и завершить работу программы можно и любым другим способом, предусмотренным Windows.

Элементами шаблона программы являются:

заголовок,

директива {*\$APPTYPE CONSOLE*} ,

предложение использования **uses** SysUtils,

пустой *раздел операторов* — пустой *составной оператор*¹, состоящий только из операторных скобок **begin** и **end**,

символ «точка» (.), указывающий компилятору, что текст программы закончен.

Внутри раздела операторов (между ключевыми словами² **begin** и **end**) находится *комментарий* — текст, предлагающий разместить в этом месте *операторы*, реализующие алгоритм программы. Комментарии в отличие от операторов не порождают команд процессора при обработке компилятором, он их просто пропускает. Назначение комментариев состоит в том, чтобы облегчить понимание алгоритма программы. Оформить текст в виде комментария можно одним из следующих способов:

поместив два символа *//* перед текстом строки,

заклучив текст, который может занимать несколько строк, в фигурные скобки { },

¹ Составным оператором называется конструкция, состоящая из операторных скобок **begin** и **end**, между которыми располагают выполняемые последовательно операторы.

² Ключевыми называют слова, используемые как обязательные элементы синтаксических конструкций языка программирования (**begin**, **end**, **uses**, **type**, **var**, **if**, **for**, **do** и др.), иное их использование запрещено. В тексте эти слова выделяются автоматически жирным шрифтом.

заклучив текст, который может занимать несколько строк, в скобки вида (* и *).

Заголовок программы содержит ее имя. Delphi дает программе стандартное имя Project1, или Project2, или Project3 и так далее в порядке создания очередной программы командой *File/New...* Заданное по умолчанию имя можно заменить на другое при сохранении текста программы командой *File/Save All...*, когда Delphi предложит сохранить ее в файле с расширением .dpr. Одновременно Delphi создаст и сохранит файлы с тем же именем и расширением .cfg и .drf. Вместе эти три файла образуют *проект* и для их хранения желательно создать отдельную папку. В этой папке будет размещена и исполняемая программа, имеющая расширение .exe.

Директива {*\$APPTYPE* *CONSOLE*} указывает компилятору, что он должен создать консольное приложение.

Предложение использования *uses SysUtils*; предписывает подключить к программе *стандартный* (входящий в систему программирования Delphi) модуль³ с именем SysUtils. Этот модуль в свою очередь содержит предложение использования, подключающее другие стандартные модули, и все, что объявлено в них, становится доступным в разрабатываемой программе. В частности, подключение модулей необходимо для создания консольного приложения и использования в нем некоторых стандартных типов данных⁴ (например, типа Integer для объявления переменных целого типа, типа Real для объявления переменных вещественного типа), стандартных подпрограмм⁵ — стандартных процедур (на-

³ Модуль — это специальным образом оформленный набор именованных констант, типов, переменных, подпрограмм и директив, которые могут быть доступны в другом модуле или программе, если имя модуля включить в список предложения использования *uses*. Исходный текст модуля хранится в файле с расширением .pas, а откомпилированный модуль — в файле с расширением .dcl. Delphi позволяет создавать новые модули, автоматически или по запросу включая их в проект (см. гл. Модули пользователей).

⁴ Типом данных называют набор характеристик, которыми обладают *переменные*, объявленные с помощью этого типа. К числу характеристик простых типов относятся правила записи констант, набор операций, набор стандартных подпрограмм. Можно объявлять в программе или в модуле новые типы данных на основе стандартных или ранее объявленных. Объявление новых именованных типов дается в разделе объявления типов, начинающемся со слова **type**, например, *<type tMas=array[1..10] of Real;>* объявляет тип с именем tMas, типом массивов из 10 вещественных чисел. Переменные объявляются в разделе, начинающемся со слова **var**, например, *<var N:Integer; X Y:tMas;>* объявляет переменную N целого типа и массивы X и Y типа tMas.

⁵ Подпрограммой называют оформленный специальным образом алгоритм, для выполнения которого достаточно сделать вызов подпрограммы, указав после ее имени список данных, подлежащих обработке. В языке Object Pascal есть два

пример, Write для вывода данных в окно программы или в файл, Read для чтения данных с клавиатуры или из файла) и стандартных функций (например, Sin для вычисления синуса, Sqrt для вычисления квадратного корня). Дополнительный набор стандартных подпрограмм (процедур и функций) содержит стандартный модуль с именем Math, который следует подключить к программе, добавив в предложение использования: **uses** SysUtils, Math;

Объявления *именованных констант*⁶, типов, переменных и подпрограмм, сделанные в подключенных к программе модулях, считаются сделанными до аналогичных объявлений в программе, что соответствует правилу языка Object Pascal: в любом объявлении можно использовать только ранее объявленные имена.

Объявления и следующий за ними раздел операторов называют *блоком*. Используя этот термин, структуру консольной программы можно определить как последовательность, включающую заголовок программы, директиву {**\$APPTYPE** **CONSOLE**}, предложение использования модулей, блок и точку. Кроме того, в тексте программы можно размещать директивы, управляющие работой компилятора.

Рассмотрим следующий пример программы:

```
program Project1;  
{$APPTYPE CONSOLE}  
uses SysUtils;  
{РАЗДЕЛ ОБЪЯВЛЕНИЯ КОНСТАНТ}  
const  
    //Именованная константа, представляющая ускорение  
    g=9.8;  
{РАЗДЕЛ ОБЪЯВЛЕНИЯ ПЕРЕМЕННЫХ}  
var  
    //Переменная целого типа, представляющая длину пути  
    S: Integer;
```

вида подпрограмм: процедуры и функции. Вызовы процедур являются отдельными операторами, а вызовы функций обычно используют внутри операторов в выражениях. Например, для вывода значения $\sin\left(\frac{\pi}{4}\right)$ следует записать вызов стандартной процедуры Write со стандартной функцией Sin, вычисляющей значение синуса, и стандартной функции Pi, вычисляющей значение числа π : Write(Sin(Pi/4)). Пользователь может создавать свои подпрограммы, размещая их тексты в любом месте перед разделом операторов (см. гл. Программирование с использованием подпрограмм).

⁶ Именованные константы объявляются в разделе, начинающемся со слова **const**. Например, **<const Nmax=25;>**. Имена таких констант представляют в программе соответствующие значения. Использование именованных констант облегчает понимание алгоритма программы и позволяет минимизировать ошибки при ее модернизации: если имя константы используется в разных частях программы, то достаточно изменить значение только в объявлении именованной константы.

```
//Переменная вещественного типа, представляющая время
T: Real;
{РАЗДЕЛ ОПЕРАТОРОВ}
begin
    //Вывод приглашения к вводу значения переменной S
    Write('Enter S: ');
    //Ввод значения переменной S
    ReadLn(S);
    //Вычисление времени движения тела на заданном пути
    T:=Sqrt(S*2/g);
    //Вывод результата с пояснениями
    WriteLn('T =', T);
    //Задержка закрытия окна программы до нажатия клавиши
Enter
    ReadLn;
end.
```

Программа вычисляет время движения тела с ускорением $9,8 \text{ м/с}^2$ на пути заданной длины, вводимой с клавиатуры. В блоке программы используется раздел объявления именованных констант (константа g), раздел объявления переменных (переменные T и S) и раздел операторов. Протокол ввода-вывода программы представлен в окне программы на рис. 1.2.

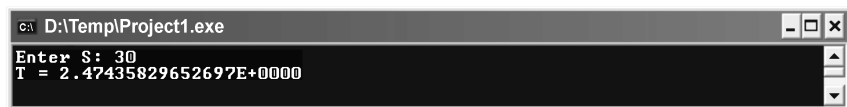


Рис. 1.2

Результат выводится с поясняющим кратким текстом, набранным латинскими буквами.


Цвет фона (черный) и символов (белый) в окне программы можно изменить следующим образом: раскрыть системное меню окна щелчком на кнопке , выбрать пункт Свойства, в появившемся диалоговом окне, на вкладке Цвета из представленной палитры выбрать для фона и символов желаемые цвета, нажать кнопку ОК, выделить в появившемся окне Изменение свойств кнопку Сохранить свойства для других окон с тем же именем и нажать ОК. Например, при выборе белого цвета для фона и черного для символов окно программы примет вид рис. 1.3.



Рис. 1.3

Используемая в Delphi кодировка символов не обеспечивает правильный вывод букв кириллицы, что представляет неудобство для русскоязычных пользователей. Наиболее простым способом вывода текстов на русском языке является включение в блок программы, до раздела операторов, объявления функции перекодирования символов (см. приложение 1). Включив объявление этой функции, можно повысить информативность программы, используя следующий раздел операторов:

```
{РАЗДЕЛ ОПЕРАТОРОВ}
begin
  //Вывод пояснения о назначении программы
  WriteLn(Rus('Программа вычисляет время, '
    + 'за которое тело в свободном падении '
    + 'пройдёт заданный путь.'));
  //Вывод приглашения к вводу значения переменной S
  Write(Rus('Введите длину пути в метрах: '));
  //Ввод значения переменной S
  ReadLn(S);
  //Вычисление времени движения тела на заданном пути
  T:=Sqrt(S*2/g);
  //Вывод результата с пояснениями
  WriteLn(Rus('Путь будет пройден за '
    ,T,Rus(' секунд.'));
  //Задержка закрытия окна программы
  //до нажатия клавиши Enter
  ReadLn;
end.
```

Протокол ввода-вывода программы представлен в окне программы на рис. 1.4.

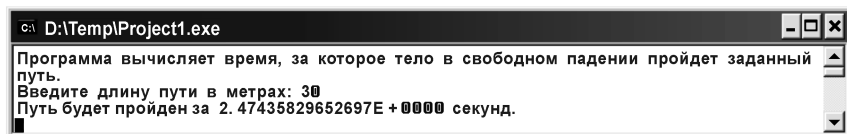


Рис. 1.4

Перекодирование символов кириллицы для вывода можно выполнить и с помощью стандартных средств Delphi. Чтобы не загромождать тексты примеров программ, использовать функцию перекодирования для выводимых русских текстов в основном не будем.

Исполняемая программа может быть создана, только если подключены необходимые модули и устранены *синтаксические*

ошибки (ошибки, связанные с нарушением в конструкциях программы правил языка программирования, которые обнаруживает компилятор). При обнаружении таких ошибок Delphi выдает сообщения с кратким пояснением причины в специальном окне, появляющемся под окном редактирования программы. Ошибки следует исправить и вновь выполнить команду *Run/Run*.

Устранением ошибок разработка программы не заканчивается. Это только начало важного этапа, называемого *отладкой*. Суть отладки состоит в приведении программы к полному соответствию заданию на разработку. Программа должна на любых допустимых входных данных и их последовательностях давать требуемый результат.

На этапе отладки должны быть выявлены и устранены ошибки в ее алгоритме и другие, проявляющиеся только во время работы программы. Для их обнаружения при отладке обычно используют вывод промежуточных результатов вычислений, проверку конечных результатов (если они не очевидны), иные методы и приемы.

Сократить время отладки позволяет использование *начальных значений переменных* (значений, заданных при объявлении переменных, которые они имеют в момент старта программы), что особенно заметно при вводе большого числа исходных данных, например в программах обработки массивов. Использование начальных значений переменных облегчает также подготовку исходных данных для тестирования программы (если требуется изменять лишь их часть), представление данных в удобном для восприятия виде (например, в виде матрицы) рядом с переменными, быстрое изменение данных (так как после перезагрузки сразу становится активным окно редактирования программы). На время отладки части программы, обеспечивающие ввод данных, превращают в комментарии, например, заключая между скобками (* и /*). В представленной ранее программе выполнить это можно так:

```
var
  S:Integer=15;//Переменная целого типа,
                    //представляющая длину пути
  T:Real;//Переменная вещественного типа,
                    //представляющая время
{РАЗДЕЛ ОПЕРАТОРОВ}
begin
  //Вывод пояснения о назначении программы
  WriteLn(Rus('Программа вычисляет время, '
```



```

      +'за которое тело в свободном падении '
      +'пройдёт заданный путь.'));
(*
  //Вывод приглашения к вводу значения переменной S
  Write(Rus('Введите длину пути в метрах: '));
  //Ввод значения переменной S
  ReadLn(S);
//*)

```

После завершения отладки ввод данных восстанавливается превращением скобки (* в комментарий /*(*. Для отключения операторов на время отладки или добавления отладочных операторов, которых не должно быть в готовой программе, удобно использовать *условную компиляцию*. Для этого в программу вводят конструкции вида

```

{$IFDEF <ИМЯ>}
. . . . //операторы, используемые при отладке
{$ENDIF}

```

или

```

{$IFDEF <ИМЯ>}
. . . . //операторы, используемые при отладке
{$ELSE}
. . . . //операторы, используемые в готовой программе
{$ENDIF}

```

Обе конструкции обрабатываются компилятором следующим образом. Если ранее в исходном тексте программы встретилась директива {\$DEFINE <ИМЯ>}, объявляющая некоторое уникальное имя, то компилятор включает в исполняемую программу операторы, расположенные между директивами {\$IFDEF <ИМЯ>} и {\$ENDIF} или {\$IFDEF <ИМЯ>} и {\$ELSE}. Если же директиву {\$DEFINE <ИМЯ>} удалить либо превратить в комментарий, например так /*{\$DEFINE <ИМЯ>}, то компилятор включит в исполняемую программу операторы, расположенные между директивами {\$ ELSE} и {\$ENDIF}⁷. Если на время отладки в начале программы поместить директиву

```

{$DEFINE Debug }

```

и в разделе операторов

⁷ Есть и другие директивы условной компиляции, в частности, {\$IFDEF <ИМЯ>} и {\$UNDEF <ИМЯ>}, имеющие противоположный смысл по сравнению с директивами {\$IFDEF <ИМЯ>} и {\$DEFINE <ИМЯ>}.

```
{IFDEF Debug}
  Write(Rus('При длине пути = '), S
    ,Rus(' метров'));
{$ELSE}
  //Вывод приглашения к вводу значения переменной S
  Write(Rus('Введите длину пути в метрах: '));
  //Ввод значения переменной S
  ReadLn(S);
{$ENDIF}
```

то при запуске в ее окне увидим текст, приведенный на рис. 1.5.

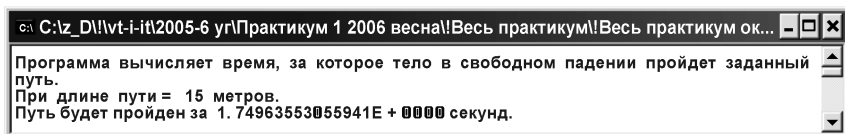






Рис. 1.5


Рассмотренный прием ускорения отладки программы рекомендуется использовать по умолчанию при выполнении заданий.


Среда Delphi также предоставляет средства отладки. Они позволяют остановить выполнение программы на любой строке ее исходного текста и просмотреть текущие значения переменных и выражений, установить или отменить точки останова и продолжить работу программы. Рассмотрим некоторые из этих средств:


- трассировка с заходом в подпрограммы (команда  *Run/Step Into*. Быстрая клавиша F7). Ввод одной команды приводит к выполнению операторов одной строки программы и останову до ввода очередной команды продолжения работы. Если в строке есть вызов подпрограммы, то вводимые далее такие команды будут выполнять операторы строк подпрограммы. В исходном тексте очередная строка, подлежащая исполнению, будет выделена и отмечена значком стрелки, например  **Result: '=';**


- трассировка без захода в подпрограммы (команда  *Run/Step Over*. Быстрая клавиша F8). То же, что и *Run/Step Into*, но строка с вызовами подпрограмм выполняется за один шаг трассировки;

- выполнить до курсора (команда  *Run/Run to Cursor*. Быстрая клавиша F4). Выполнение программы до строки текста, в которой расположен курсор ввода;

- точка безусловного останова. Устанавливается щелчком в полосе перед строкой операторов на значке . В результате зна-

чок и строка будут выделены. Например,  `T:= Sqrt(S*2/g);`. Удаляется повторным щелчком на значке. При работе программы ее останов будет перед выполнением операторов строк, отмеченных как точки безусловного останова;

– перезагрузка программы (команда  *Run/program Reset*. Быстрые клавиши Ctrl+F2). Выполняет выход из режима отладки и готовит программу к новому запуску;

– окно наблюдения (команда  *Run/Add Watch...* Быстрые клавиши Ctrl+F5). При вводе команды открывается окно диалога, в котором можно указать, значения какой переменной или выражения должны отображаться в окне наблюдения в момент ожидания системой ввода очередной команды выполнения программы. Выражения могут содержать переменные и именованные константы программы, обычные константы, обращения к некоторым стандартным функциям. Допускаются любые выражения, не обязательно те, что есть в программе. Окно наблюдения можно буксировкой разместить в окне редактирования программы или вне его. Значения переменных и выражений можно также просматривать, подводя к ним в окне редактирования программы курсор мыши.

Пример окна редактирования программы, вычисляющей выражение $\sqrt{2S/g}$, представлен на рис. 1.6. В нижней его части расположено окно наблюдения.

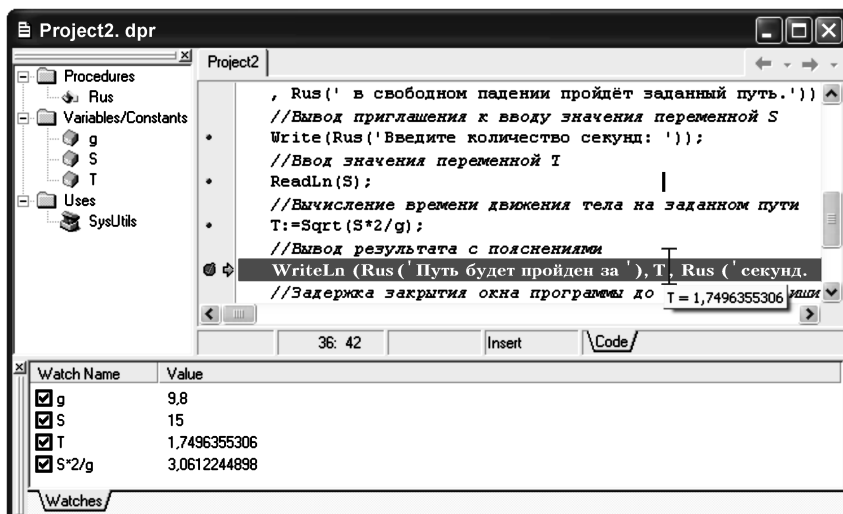



Рис. 1.6

Программа после запуска остановлена перед выполнением оператора `WriteLn` (эта строка объявлена точкой безусловного останова), и в окне наблюдения отображены значение именованной константы, текущие значения переменных и выражения. На вкладке с текстом программы виден курсор мыши, подведенный к переменной `T`, и под ним — значение этой переменной.

При выполнении программы могут возникать ошибки, связанные с ограничениями технических средств и особенностями выполнения некоторых операций, что приводит к неверным результатам работы программы или ее прекращению. Краткие сведения по обработке указанных ситуаций приведены в приложении 2.

Устранение всех видов ошибок требует, как правило, многократной корректировки текста программы, поэтому желательно настроить панель инструментов Delphi, добавив в нее кнопки типовых команд редактора текста, показанных на рис. 1.1. Делается это так. Вводится команда *View/Toolbars/Customize...*; в открывшемся окне *Customize* на вкладке *Commands* выбирается строка *Edit*; значки из списка *Commands* буксируются на панель инструментов. Для удаления значка с панели инструментов нужно просто, находясь в этом режиме, отбуксировать его за ее пределы.

Аналогично можно добавить кнопки, используемые при отладке, раскрыв список *Run* на вкладке *Commands* окна *Customize*. На рис. 1.1 к расположенным по умолчанию кнопкам на панели инструментов добавлена кнопка перезагрузки программы  (команда *Run/program Reset*, быстрые клавиши *Ctrl+F2*).

2. Программы линейной структуры

Программой линейной структуры называют такую, каждый оператор которой выполняется один и только один раз. Она может строиться только из простых операторов, не меняющих естественный порядок вычислений: из операторов присваивания и процедур. Из числа последних в этом разделе нас будут интересовать только операторы процедур ввода и вывода для стандартных устройств — клавиатуры и монитора.

2.1. Средства разработки программ линейной структуры

Рассмотрение алгоритмизации задач и приемов программирования удобнее всего проводить на примерах обработки числовых данных. Рассмотрим в первую очередь *стандартные типы* (имеющиеся в Delphi и не требующие объявления в программе) числовых данных.

Целые типы данных

К числу стандартных целых типов относят:

Int64 — представляющий целые со знаком от -2^{63} до $+2^{63} - 1$, занимает 8 байт;

Integer — представляющий целые со знаком от -2147483648 до $+2147483647$, занимает 4 байта;

LongInt — эквивалентный типу Integer;

SmallInt — представляющий целые со знаком от -32768 до $+32767$, занимает 2 байта;

ShortInt — представляющий целые со знаком от -128 до $+127$, занимает 1 байт;

Byte — представляющий целые без знака от 0 до 255, занимает 1 байт;

Word — представляющий целые без знака от 0 до 65535, занимает 2 байта;

LongWord — представляющий целые без знака от 0 до 4294967295, занимает 4 байта;

Cardinal — эквивалентный типу LongWord .

Например, чтобы объявить переменные с именами I и K как переменные типа Integer и N как Byte, в программе следует записать

```
var
//Объявление целых переменных I и K
//типа Integer
I, K:Integer;
//Объявление целой переменной N типа Byte
N:Byte
```

Константы целого типа записывают в виде последовательности цифр, перед которой может стоять знак числа. Знак + перед положительным числом можно не писать. Например, константы +25 и 25 представляют одно и то же значение. Тип целой константы определяется как стандартный целый тип с наименьшим диапазоном значений, включающим значение константы.

Именованные константы (имя такой константы представляет значение) объявляют в разделе **const**, связывая имя и значение знаком =, например:

```
const
Nmax=10; //Объявление именованной константы Nmax
```

Для данных целого типа определены следующие арифметические операции, результат выполнения которых также будет иметь целый тип с минимальным диапазоном, включающим вычисленное значение:

- сложение (знак +),
- изменение знака (унарный минус -),
- вычитание (знак -),
- умножение (знак *),
- целочисленное деление (знак **div**),
- взятие по модулю (знак **mod**).

Результатом выполнения операции **div** является целая часть частного, а операции **mod** — остаток от целочисленного деления (знак остатка всегда совпадает со знаком делимого). Например, выполнение `-5 div -2` даст значение 2, а после выполнения `-5 mod -2` получим -1.

Над целыми допустима также операция деления (знак /), приводящая к вещественному значению. Так, результатом выполнения `-5/-2` будет вещественное число 2,5.

К числу целых относят также интервальные (диапазонные) типы, объявляемые в программе. Например, в фрагменте программы

```
type
tBall = 2..5; //Объявление типа tBall
tIndex = 1..10; //Объявление типа tIndex
```

объявлены тип `tBall` с диапазоном значений 2..5 и тип `tIndex` с диапазоном 1..10. Значение, представляющее начало диапазона,

должно быть меньше представляющего конец диапазона, а разделителем является *составной символ* из двух точек.

Вещественные типы данных

К числу стандартных вещественных (действительных) типов относят:

Extended — вещественный тип с повышенной точностью, допускающий множество значений с 19—20 десятичными цифрами в диапазоне абсолютных значений от $3,6 \cdot 10^{-4951}$... $1,1 \cdot 10^{+4932}$, занимает 10 байт;

Double — допускающий множество значений с 15—16 десятичными цифрами в диапазоне абсолютных значений $5 \cdot 10^{-324}$... $1,7 \cdot 10^{+308}$, занимает 8 байт;

Real — эквивалентный типу **Double**;

Real48 — допускающий множество значений с 11—12 десятичными цифрами в диапазоне абсолютных значений $2,9 \cdot 10^{-39}$... $1,7 \cdot 10^{+38}$, занимает 6 байт;

Single — допускающий множество значений с 7—8 десятичными цифрами в диапазоне абсолютных значений от $1,5 \cdot 10^{-45}$... $3,4 \cdot 10^{+38}$, занимает 4 байта;

Comp — допускающий множество целых значений с 19—20 десятичными цифрами в диапазоне $-2^{63} + 1$... $2^{63} - 1$, занимает 8 байт;

Currency — допускающий множество значений с 19—20 десятичными цифрами в диапазоне от $-922337203685477,5808$... $+922337203685477,5807$, занимает 8 байт.

Константы вещественного типа записывают либо в *естественной* форме, например, -12.345 , либо в *экспоненциальной* форме, в которой то же самое число можно записать по-разному, например, $-0.12345E+2$, или $-0.12345E2$, или $-0.12345e+2$, или $-1.2345E+1$, или $-1.2345E1$, или $-12.345E0$, или $-1234.5E-2$, или $-12345E-3$ и т. д. При представлении числа в такой форме безразлично, используется строчная или прописная латинская буква *E*. Чтобы получить значение числа, представленного в экспоненциальной форме, нужно умножить *мантиссу*, т. е. то, что стоит перед символом *E*, на *порядок*, т. е. на 10 в степени, значение которой представляет целое число, записанное после *E*. Так, константу $-0.12345E+2$ следует читать как $-0,12345 \cdot 10^{+2}$, а константу $-1234.5E-2$ — как $-1234,5 \cdot 10^{-2}$.

Следующий фрагмент программы представляет объявления вещественных переменных *X* и *Y* типа **Real**, *Z* типа **Extended** и именованной константы *H* со значением 0,00000025:

```
var
  X, Y: Real;
  Z: Extended;
const
  H = 2.5E-7;
```

Для данных вещественного типа определены следующие арифметических операции, результат выполнения которых также будет иметь вещественный тип с минимальным диапазоном, включающим вычисленное значение:

- сложение (знак +),
- изменение знака (унарный минус −),
- вычитание (знак −),
- умножение (знак *),
- деление (знак /).

В отличие от языков программирования BASIC и Fortran, в Object Pascal нет операции возведения в степень.

Стандартные функции для обработки числовых данных

Стандартные функции, предназначенные для вычисления некоторых математических и тригонометрических функций, а также для преобразования данных вещественного типа к целому, содержатся в модуле System, подключаемом к программе по умолчанию, и в модуле Math.

Стандартные функции можно разбить на группы по типу их аргументов (т. е. фактическим параметрам, которыми могут быть константы, переменные, выражения) и типу возвращаемого результата.

Стандартные функции, не меняющие тип (для целого аргумента возвращается целое значение, для вещественного — вещественное):

- Abs(X) — возвращает абсолютное значение аргумента;
- Sqr(X) — возвращает квадрат аргумента;
- Max(X,Y) — возвращает максимальное из X и Y;
- Min(X,Y) — возвращает минимальное из X и Y.

Стандартные функции, возвращающие в любом случае (аргумент целый, вещественный или без аргумента) вещественное значение:

- Tan(X) — возвращает значение тангенса аргумента;
- ArcTan(X) — возвращает значение арктангенса аргумента;
- Sin(X) — возвращает значение синуса аргумента;
- ArcSin(X) — возвращает значение арксинуса аргумента;

$\text{Cos}(X)$ — возвращает значение косинуса аргумента;

$\text{ArcCos}(X)$ — возвращает значение арккосинуса аргумента;

$\text{Exp}(X)$ — возвращает значение e^x ;

$\text{Frac}(X)$ — возвращает дробную часть аргумента;

$\text{Int}(X)$ — возвращает целую часть аргумента;

$\text{Ln}(X)$ — возвращает значение натурального логарифма аргумента;

$\text{Log10}(X)$ — возвращает значение логарифма аргумента X по основанию 10;

$\text{LogN}(N, X)$ — возвращает значение логарифма аргумента X по основанию N ;

Pi — возвращает значение числа π ;

$\text{Sqrt}(X)$ — возвращает значение квадратного корня аргумента;

$\text{IntPower}(X, N)$ — возвращает значение X , возведенное в целую степень N ;

$\text{Power}(X, Y)$ — возвращает значение X , возведенное в степень Y (если Y не целое, то значение X должно быть не отрицательным).

Стандартные функции, возвращающие значение целого типа:

$\text{Round}(X)$ — возвращает округленное значение вещественного аргумента;

$\text{Trunc}(X)$ — возвращает значение вещественного аргумента после отбрасывания его дробной части;

$\text{Sign}(X)$ — возвращает значение 1 для $X > 0$, значение 0 — для $X = 0$, значение -1 — для $X < 0$.

Дополнительные сведения о подпрограммах модулей `System` и `Math` можно найти в справочных разделах `Delphi Arithmetic routines` и `Trigonometry routines`.

Арифметические выражения

Арифметическое выражение записывается в виде последовательности целых или вещественных констант, переменных и обращений к функциям, разделенных знаками арифметических операций и круглыми скобками. В арифметических выражениях могут использоваться одновременно переменные, константы и функции разных целых и вещественных типов (такую возможность обозначают термином «совместимость типов в выражениях»).

Результат вычисления выражения будет целого типа, если все константы, переменные и функции имеют целый тип и не используется знак операции $/$, иначе — вещественного.

В выражениях в первую очередь вычисляются обращения к функциям и содержимое круглых скобок, затем — операции типа

умножения (*, /, **div**, **mod**) в порядке слева направо, после чего — операции типа сложения (+ и –) в порядке слева направо.

Например, для вычисления выражения

$$\frac{\sin^2(X) \cos(Y^3) \cdot 12 \cdot 10^{-5}}{\sqrt{X} \cdot Y Z^{2/3}}$$

в программе можно записать

```
Sqr(Sin(X)) * Cos(IntPower(Y, 3)) *  
1.2E-4 / Sqrt(X) / Y / Power(Z, 2/3).
```

Оператор присваивания

В процессе выполнения программы переменные могут получать новые значения либо с помощью процедур ввода, либо с помощью операторов присваивания, либо как выходные параметры подпрограмм (об этом позже).

Оператор присваивания записывается в виде, показанном на рис. 2.1.



Рис. 2.1

Выражение, записанное справа от знака присваивания := (состоит из знаков «двоеточие» и «равно»), вычисляется, преобразуется (при необходимости) к типу переменной, стоящей слева от знака присваивания, после чего эта переменная получает вычисленное значение. Например, для вещественной переменной X выполнение оператора

```
X:=5 mod 2;
```

будет складываться из следующих действий: вычисления выражения $5 \bmod 2$ целого типа, результатом чего станет значение 1, приведения этого значения к вещественному типу и сохранения преобразованного значения в ячейке памяти, соответствующей переменной X .

Возможность преобразования в операторе присваивания значения одного типа к другому называют *совместимостью по присваиванию*. Допустимо оно не во всех случаях. Например, целое можно преобразовать присваиванием к вещественному, но не наоборот. В последнем случае следует воспользоваться стандартными функциями Round или Trunc. Например, для переменной K целого типа допустимо использование операторов:

```
//Переменная K получит значение 6
```

```
K:=Round(5.6);
```

или

```
//Переменная K получит значение 5,
```

```
K:=Trunc(5.6);
```

но запрещено

```
K:=5.6;
```

Ввод данных с клавиатуры

В консольных приложениях для ввода данных с клавиатуры есть два оператора вызова процедур с именами `Read` и `ReadLn` (далее будем их называть просто операторами ввода или операторами `Read` и `ReadLn`). Параметры этих процедур — переменные, значения которых располагаются вслед за именами процедур в круглых скобках в виде списка через запятую, который называют *списком ввода*. Элементами списка ввода могут быть переменные только числовых типов, символьные и строковые. Данные для числовых типов, набираемые на клавиатуре, должны разделяться или пробелами, или символами табуляции (клавиша `TAB`), или символами конца строки (клавиша `Enter`). Набор данных заканчивается нажатием клавиши `Enter`. Набранные данные последовательно присваиваются переменным списка ввода. Возможно также, что число элементов списка ввода будет меньше или больше количества набранных на клавиатуре данных. Тогда, при использовании оператора `Read`, в первом случае не использованные этим оператором числа могут быть введены следующим оператором ввода, а во втором — следующий оператор ввода будет ждать набора на клавиатуре недостающих данных и нажатия клавиши `Enter`.

Например, ввести значения вещественных переменных X и Y оператором `Read(X,Y)` можно, подготовив вводимые числа на одной строке:

```
5.21 1e-3↵
```

или в двух строках:

```
5.21↵
```

```
1e-3↵
```

где символ `↵` обозначает нажатие клавиши `Enter`.

Использование для тех же целей двух операторов:

```
Read(X); Read(Y);
```

также допустимо при наборе данных в одной или двух строках.

Оператор `ReadLn` отличается от оператора `Read` тем, что после завершения ввода оставшиеся неиспользованными набранные числа пропадают (не могут быть использованы следующими операторами ввода). Например, для ввода операторами

```
ReadLn(X); ReadLn(Y);
```

тех же данных их следует набирать в двух строках, как показано в предыдущем примере.

Вывод данных в окно программы

В консольных приложениях для вывода данных на экран есть два оператора вызова процедур с именами `Write` и `WriteLn` (далее их будем называть просто операторами вывода или операторами `Write` и `WriteLn`). Оператор `WriteLn` отличается от оператора `Write` тем, что после завершения вывода курсор в окне программы переводится в начало следующей строки, и если текущая строка была последней, то внизу окна программы появляется новая, пустая, а все другие смещаются вверх.

Подлежащие выводу данные перечисляются в списке вывода, который записывается в круглых скобках непосредственно за словами `Write` или `WriteLn`. Список вывода содержит выражения, в частности константы, переменные, обращения к функциям, разделенные запятыми. Оператор `WriteLn` записывают и без списка вывода и скобок, если требуется просто перевести курсор в начало новой строки.

Элементы списка вывода могут иметь любой из простых типов: числовой, булевый и символьный. Пока ограничимся рассмотрением только числовых типов и строковых констант.

Пример. Составить программу вычисления с повышенной точностью (Extended) тангенса угла, значение которого (целое число) в градусах вводится с клавиатуры в ответ на приглашение. Вывести на экран с пояснениями введенное значение угла в градусах, соответствующее ему значение в радианах и вычисленное значение тангенса этого угла.

```
program Project1;  
{$APPTYPE CONSOLE}  
uses  
SysUtils, Math;  
var  
R:Extended;
```

```
Fi:Integer;
begin
  //Вывод приглашения к вводу угла в градусах
  Write('Введите значение угла в градусах: ');
  //Ввод значения угла в переменную Fi
  ReadLn(Fi);
  //Перевод угла в радианы
  //и присвоение переменной R
  R:=Fi*Pi/180;
  //Вывод значения R
  WriteLn('Значение угла в радианах = ',R);
  WriteLn;           //Пропуск строки
  //Вывод R и tg(R) с поясняющими текстами
  WriteLn('tg(', R, ') = ', Tan(R));
  ReadLn;
end.
```

Здесь заключенные в апострофы тексты, например, 'Fi=', 'tg(' и ') = ' — строковые константы, R и выражение Tan(R) представляют данные типа Extended.

Протокол ввода-вывода при выполнении этой программы при вводе для Fi значения 30 будет иметь вид:

```
Введите значение угла в градусах: 30
Значение угла в радианах = 5.23598775598200E-0001
tg(5.23598775598200E-0001) = 5.77350269189626E-0001,
```

а курсор перейдет в начало следующей строки.

Первый оператор Write выводит приглашение к вводу, в ответ на которое пользователь вводит значение угла, в данном случае 30. Ввод данных заканчивается нажатием клавиши Enter, что приведет к переводу курсора в начало новой строки. Следующий далее оператор WriteLn('Значение угла в радианах = ',R) выводит значение угла в радианах и переводит курсор на начало следующей, пустой, строки, оператор WriteLn без параметров оставит эту строку пустой и переведет курсор в начало еще одной новой пустой строки, в которую оператор WriteLn('tg(', R, ') = ', Tan(R)) выведет последовательность символов:

```
tg( 5.23598775598299E-0001) = 5.77350269189626E-0001.
```

В этом примере для всех типов данных использовались принятые по умолчанию (что не всегда удобно) форматы вывода: вещественные в экспоненциальной форме в 23 позициях с четырьмя цифрами порядка; целые и строковые занимают мини-

мально необходимое число позиций, причем целые положительные изображаются без знака +.

Программисту, чтобы определить свою форму вывода, следует использовать один из форматов вывода, записываемый непосредственно за элементом списка вывода либо в виде :n — для любого типа данных, либо в виде :n:m — при выводе вещественного числа в естественной форме (<целая часть><точка><дробная часть>), где n — выражение целого типа, значение которого задает длину поля (число знакомест) в строке на экране, отводимую для изображения значения, а m — выражение целого типа, значение которого задает количество цифр в дробной части числа.

Выражение n может быть больше или меньше требуемого для представления значения количества знакомест. В первом случае выводимое значение будет расположено в правой части поля вывода. Во втором случае под вывод значения отводится минимально необходимое число позиций (для вещественных в экспоненциальной форме оно равно 10).

Например, при выполнении следующей программы:

```
program Project2;
{$APPTYPE CONSOLE}
uses
  SysUtils, Math;
var
  R:Extended;
  Fi:Integer;
begin
  //Вывод приглашения к вводу угла в градусах
  Write('Введите значение угла в градусах: ');
  //Ввод значения угла в переменную Fi
  ReadLn(Fi);
  //Перевод угла в радианы
  //и присвоение переменной R
  R:=Fi*Pi/180;
  //Вывод значения R
  WriteLn('Значение угла в радианах = ',R:0);
  WriteLn;           //Пропуск строки
  //Вывод R и Tan(R) с поясняющими текстами
  WriteLn('tg(', R:0:2, ') = ', Tan(R):14);
  ReadLn;
end.
```

протокол ввода-вывода при вводе для Fi значения 30 будет иметь вид

Введите значение угла в градусах: 30

Значение угла в радианах = 5.2E-0001

$\text{tg}(0.52) = 5.77350\text{E}-0001$

Значение R первый раз выведено в экспоненциальной форме в минимально необходимое число позиций (формат вывода :0), а второй раз — в естественной форме также в минимально необходимое число позиций (формат вывода :0:2), так как в обоих случаях длина поля вывода указана равной нулю, т. е. меньше минимально необходимой. Значение выражения $\text{Tan}(R)$ выведено в экспоненциальной форме в поле из 14 позиций с шестью значащими цифрами мантиссы (формат вывода :14).

2.2. Приемы, используемые для минимизации вычислений

Одним из критериев, характеризующих качество составленной программы, является объем выполняемых для достижения требуемого результата вычислений. Чем он меньше, тем быстрее будет работать программа. Существуют разные приемы, позволяющие сократить объем вычислений за счет уменьшения количества вызовов функций, операций типа умножение и операций типа сложение. Вот некоторые из них, рассмотренные отдельно (как правило, используются в сочетании).

Вынесение общих множителей за скобки. Например, вместо оператора

```
Z:=Sin(X)*Y-Sin(X)*Sqr(Y)*Y+Sqr(X)*X+X;
```

лучше использовать оператор

```
Z:=Sin(X)*Y*(1-Sqr(Y))+X*(Sqr(X)+1);
```

Использование схемы Горнера для полиномов. Например, полином

$$2X^5 - 5X^4 + 2X^3 + 7X^2 - 4X + 6,$$

преобразованный по схеме Горнера, примет вид

$$(((2X - 5)X + 2)X + 7)X - 4)X + 6,$$

где явно меньше число операций умножения, если считать, что возведение в степень выполняется через умножение. В программе по второму варианту записи полинома будем иметь выражение

$$(((2X-5)*X+2)*X+7)*X-4)*X+6,$$

а по первому —

$$2*\text{IntPower}(X, 5) - 5*\text{IntPower}(X, 4) + 2*\text{IntPower}(X, 3) + 7*\text{IntPower}(X, 2) - 4*X + 6,$$

здесь помимо такого же числа операций умножения требуется выполнить четыре обращения к функции возведения в степень.

Другой пример, где также применима схема Горнера:

$$1 + 2! + 3! + 4! + 5! = 1 + 2(1 + 3(1 + 4(1 + 5))).$$

Использование дополнительных переменных. Например, при расчете значения функции $Z = (X - Y) \frac{1 + (X - Y)^3}{1 + (X - Y)^2}$ целесообразно предварительно вычислить $A = X - Y$ и $B = A^2$, а исходную формулу преобразовать к виду $Z = A \frac{1 + AB}{1 + B}$. В этом случае в программе будут использованы три оператора присваивания:

```
A:=X-Y;
B:=Sqr(A);
Z:=A*(1+A*B)/(1+B);
```

2.3. Примеры выполнения задания

1. Найти коэффициенты k_0, k_1, k_2, k_3 представления числа X ($0 \leq X \leq 80$) в троичной системе счисления:

$$X = k_3 \cdot 3^3 + k_2 \cdot 3^2 + k_1 \cdot 3 + k_0, \quad (2.1)$$

используя операции **mod** и **div**. Для контроля результатов выполнить вычисление X непосредственно по формуле (2.1) для найденных коэффициентов, а также после преобразования выражения в формуле (2.1) по схеме Горнера. Вывести все результаты вычислений в наглядной форме с поясняющими текстами. Проверить работу программы на значениях $X = 0; 1; 2; 10; 27; 48; 80$.

```
program Project1_1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var
  X, k0, k1, k2, k3, X1, X2:Byte;
begin
  {Ввод исходных данных}
  Write('Введите X : '); ReadLn(X);
  {Вычисление коэффициентов разложения}
  k0:=X mod 3; X:=X div 3; //k0:=X mod 3;
  k1:=X mod 3; X:=X div 3; //k1:=X div 3 mod 3;
  k2:=X mod 3; //k2:=X div 9 mod 3;
  k3:=X div 3; //k3:=X div 27;
```



```

{Вывод вычисленных коэффициентов разложения
  числа X по степеням 3 с поясняющими текстами}
WriteLn;
WriteLn('Коэффициенты разложения '
        , 'введённого числа по степеням 3');
WriteLn('k3 = ', k3, 'k2 = ':7, k2, 'k1 = ':7
        , k1, 'k0 = ':7, k0);
{Вычисление непосредственно по формуле (2.1)}
X1:=k3*27+k2*9+k1*3+k0;
{Вывод результатов}
WriteLn('Вычислено по формуле (2.1) непосредственно');
      WriteLn(k3, '*27+', k2, '*9+', k1, '*3+', k0, ' = ', X1);
{Вычисление по формуле (1.1),
  преобразованной по схеме Горнера}
X2:=((k3*3+k2)*3+k1)*3+k0;
{Вывод результатов}
WriteLn ('Вычислено по формуле(2.1) '
        , 'представленной по схеме Горнера');
      WriteLn('((', k3, '*3+', k2, ')*3+', k1, ')*3+'
        , k0, ' = ', X2);

ReadLn;
end.

```

2. Найти значение функции

$$Y(X) = \frac{\left(\frac{a}{2}\right)^x - \lg\left(\frac{a}{2} + 1\right)}{\left(\frac{a}{2}\right)^3 - \left(\frac{a}{2}\right)^2}, \quad (2.2)$$

упростив вычисления за счет использования скобочных форм и/или дополнительных переменных (здесь и далее конструкция «А и/или Б» обозначает «или А, или Б, или А и Б одновременно»). Для контроля правильности результата выполнить вычисление по формуле (2.2) без использования скобочных форм и дополнительных переменных.

Проверить работу программы на значениях $A = (1; -1; 2; -2; 4; -4)$, $X = (0,5; 2)$.

```

program Project1_2;
{$APPTYPE CONSOLE}
uses
  SysUtils, Math;
var
  A, B, C, X, Y1, Y2: Real;

```

```

begin
    {Ввод исходных данных}
    Write('Введите X и A : ');
    ReadLn(X,A);
    B:=A/2;
    C:=Sqr(B);
    {Вычисление выражения}
    { - с использованием дополнительных переменных}
    Y1:=(Power(B,X)-Log10(B+1))/C/(B-1);
    { - непосредственно по формуле (1.2)}
    Y2:=(Power(A/2,X)-Log10(A/2+1))
        / (IntPower(A/2,3)-Sqr(A/2));
    {Вывод вычисленных значений с надписями}
    WriteLn('      Y1              Y2 ');
    WriteLn(Y1:12:7, '      ', Y2:12:7);
    ReadLn;
end.

```

Представленная программа не предусматривает обработку исключений (см. приложение 2), поэтому введем в задание дополнительные условия: при возникновении любого исключения, связанного с вычислением функции, предусмотреть его обработку с выводом типа исключения и завершением работы нажатием клавиши Enter. В соответствии с этим в программу следует включить обработку всех (из числа рассмотренных в приложении 2) исключений для вещественных данных, например, так:

```

program Project1_2;
{$APPTYPE CONSOLE}
uses
    SysUtils, Math;
var
    A, B, C, X, Y1, Y2: Real;
begin
    {Ввод исходных данных}
    Write('Введите X и A : ');
    ReadLn(X,A);
    B:=A/2;
    C:=Sqr(B);
    try
        {Попытка вычислить выражения}
        { - с использованием дополнительных переменных}
        Y1:=(Power(B,X)-Log10(B+1))/C/(B-1);

```

```
{ - непосредственно по формуле (2.2)}
Y2:=(Power(A/2,X)-Log10(A/2+1))
    /(IntPower(A/2,3)-Sqr(A/2));
except
{Вычислить выражение не удалось}
on EZeroDivide do
begin
    WriteLn('Исключение типа "деление на 0" ');
    WriteLn('Невозможно вычислить Y при A =',A
        , ' и X =',X);
    WriteLn('Нажмите Enter для завершения'
        , ' работы программы.');
```

ReadLn;

Halt; //Завершить выполнение программы

```
end;
on EInvalidOp do
begin
    WriteLn('Исключение типа'
        , ' "невыполнимая операция" ');
    WriteLn('Невозможно вычислить Y при A =',A
        , ' и X =',X);
    WriteLn('Нажмите Enter'
        , ' для завершения работы программы.');
```

ReadLn;

Halt; //Завершить выполнение программы

```
end;
on EOverflow do
begin
    WriteLn('Исключение типа "переполнение'
        , ' вещественной переменной" ');
    WriteLn('Переменная не может хранить'
        , ' вычисленное значение');
```

WriteLn('Нажмите Enter'
 , ' для завершения работы программы.');

ReadLn;

Halt; //Завершить выполнение программы

```
end;
end; //try except
{Вывод вычисленных значений с надписями}
WriteLn('          Y1          Y2 ');
WriteLn(Y1:12:7,' ',Y2:12:7);
ReadLn;
end.
```

Это позволит по каждому исключению вывести краткое пояснение на русском языке о причине его возникновения, однако потребует дублирования операторов

```
WriteLn('Нажмите Enter'
      , ' для завершения работы программы.');
```

```
ReadLn;
Halt;
```

так как запрещено совмещение с оператором **on** других операторов (в блоке **except** допускается использовать только операторы **on** либо только другие).

2.4. Задания А для самостоятельной работы

В заданиях 1—28:

— вычислить, упростив за счет использования скобочных форм и/или дополнительные переменные, значения по заданным формулам;

— для контроля правильности результатов выполнить вычисления по формулам без использования скобочных форм и дополнительных переменных;

— проверить результаты на комбинациях заданных значений аргументов и при возникновении исключений дополнить программу обработкой их с выводом типа и предложением закончить выполнение программы нажатием клавиши Enter.

1. $Z = X^2Y^2 + 3XY^2 - 5X^2Y + X^2 - 2Y^2 + 4XY - X + Y$; $X = 2; -2$; $Y = 4; -3$.

2. $B = A + 2$; $C = (A + 3)/(A + 2)$; $D = (A + 4)/(A + 3)$; $E = (A + 5)/(A + 4)$; $A = 1; 2; -2; 3; 4$.

3. $Z = (X + 2) \frac{(X + 2)^2 + 3}{(X + 2)^4 + (X + 2)^2 + 3}$; $X = 0; 1; 2; -2; 4$.

4. $B = \sin A$; $C = \lg A$; $D = e^A$; $E = |A|$; $S = (A + B)(A + B + C)(A + B + C + D)(A + B + C + D + E)$; $A = 8; -2; 4; -5$.

5. $B = A + 5$; $C = A - 2$; $D = B + C$; $E = A - C$; $P1 = \frac{A}{B}$;

$P2 = \frac{AC}{B}$; $P3 = \frac{AC}{BD}$; $P4 = \frac{ACE}{BD}$; $A = -15; -5; 0; 7; 14$.

6. $B = A - 2$; $C = A + 3$; $D = B + C$; $E = A - 2$; $P1 = AB$; $P2 = ABC$; $P3 = ABCD$; $P4 = ABCDE$; $A = -4; 0; 4; 7$.

$$7. Y = \frac{3}{1+3+X} - \frac{3 \cdot 5}{1+3+5+2X} + \frac{3 \cdot 5 \cdot 7}{1+3+5+7+3X} - \frac{3 \cdot 5 \cdot 7 \cdot 9}{1+3+5+7+9+4X}; X = -9; -4; 0; 3; 9.$$

$$8. Y = -X^6 + AX^5 - A^2X^4 + A^3X^3 - A^4X^2 + A^5X - A^6; X = -3; 5, A = -3; 5.$$

$$9. Y = A^X - \frac{A^{2X}}{3} + \frac{A^{3X} \cdot 4}{3 \cdot 5} - \frac{A^{4X} \cdot 6}{3 \cdot 5 \cdot 7}; X = -3; 0; 3, A = 4.$$

$$10. Y = X + \frac{2 \cdot 4X^2}{2+4} + \frac{2 \cdot 4 \cdot 6X^3}{2+4+6} + \frac{2 \cdot 4 \cdot 6 \cdot 8X^4}{2+4+6+8} + \frac{2 \cdot 4 \cdot 6 \cdot 8 \cdot 10X^5}{2+4+6+8+10}; X = -7; -2; 0; 2; 7.$$

$$11. Y = \left(\frac{\sqrt[3]{A-B} \cdot (A+B)^{2/3} X^2}{A^4 + B^4 - 2A^2B^2 - X^4} \right)^{\frac{1}{X}}; X = 0,5; 1; 2, A = 4, B = 3.$$

$$12. Y = \left(\frac{A}{B} \right)^X + \left(\frac{A}{B} \right)^{2X} + 2 \left(\frac{A}{B} \right)^{-2/3} \log_2 \left(\frac{A}{B} \right); X = 2,5; 5; 7; 10, A = 4, B = 3.$$

$$13. Y = \frac{-3,3 \cdot 10^{-4} \operatorname{tg} X \lg (X^2 - 5) \sqrt{|\operatorname{tg} X|}}{\sqrt[3]{X^2 - 5} \cdot X e^{-2X}}; X = 1; 2,5; 5; 7; 10.$$

$$14. Y = \frac{\sqrt[3]{(X+1)^2} \cdot (X+2) \ln(X+1)}{(X+2)^3 - (X+2)^2 + (X+1)^2}; X = -0,5; 5; 10; 25.$$

$$15. Y = \frac{\sqrt[3]{X^4 + X^3 - X^2 - X + 1}}{X^4 + 2X^3 - 2X - 1}; X = -15; -5; -2; 2; 5.$$

$$16. Y = \frac{X + X^3 - 3}{2X^2 - 4X + 2 + 6X^3 - 4X^5}; X = -5; -2; 2; 5.$$

$$17. Y = \frac{\cos^4 X}{4} - \frac{\cos^2 X}{2} - \ln(|\cos X|); X = 0; 30^\circ; 45^\circ; 60^\circ; 90^\circ.$$

$$18. Y = \left[\frac{1}{2} + (X-1) - \frac{(X-1)^2}{2} + \frac{(X-1)^4}{3} \right] / (X^2 - 1); X = 0; 1; 2; 3.$$

$$19. Y = \frac{\left(\frac{1}{2} + \sin^2 X \right) \ln |\sin X|}{\frac{\pi}{3} - \arcsin X}; X = 0,001; 0,1; 0,3; 0,5; 0,9; 1,8.$$

$$20. Y = \frac{e^X - e^{2X} + e^{3X} \arccos^2 X}{e^X - e^{2X} + e^{3X} \arcsin^{0,5} X}; X = 0,001; 0,02; 0,1; 0,9.$$

$$21. Y = \frac{\left| \frac{1}{4} - A^2 \operatorname{tg} X \right|}{(\sin X + \cos X)(\operatorname{tg}^2 X + 1)}; X = 0,001^\circ; 15^\circ; 30^\circ; 60^\circ; 135^\circ.$$

$$22. Y = \frac{|\sin X \cos X| + \operatorname{tg} X}{\sin X + \sin X \cos X + \cos X};$$

$X = 0,001^\circ; 15^\circ; 30^\circ; 60^\circ; 270^\circ.$

$$23. Y = \frac{2X^3 + 6X^2 - 8X + 4}{-4X^3 + 8X^2 - X^5 + 2X^4}; X = -5; -2; 2; 5.$$

$$24. Y = \frac{1 - \sqrt{|\log_2 X|} + 25 \cdot 10^{-5} \cdot \log_{10} X}{\log_2 X + 0,00025 \log_{10} X}; X = 0,001; 0,1; -1; 1; 4.$$

$$25. Y = \frac{X \lg(X+1) + \lg(X+1) + X \ln A + \ln A + A^{X+1} \lg(X+1) + A^{X+1} \ln A}{\ln A + \ln(X+1)};$$

$X = 0,001; 0,1; -1; 1; 4; A = 3.$

$$26. Y = \frac{A^X X^A + 2A^{2X} X^A - 2A^X X^{2A} - 4A^{2X} X^{2A}}{\lg A + \lg X};$$

$X = 0,001; 0,1; 1; 4; A = 1,5.$

$$27. Y = \frac{A^{X^A} X^{A^{AX}}}{A^{AX} + X^{AX}}; X = 0,001; 0,1; 1; 4 \text{ и } A = 2.$$

$$28. Y = 1 + \frac{X^2}{3^2} + \frac{X^2}{7^2} + \frac{X^2}{11^2} + \frac{X^4}{3^2 7^2} + \frac{X^4}{3^2 11^2} + \frac{X^4}{7^2 11^2} + \frac{X^6}{3^2 7^2 11^2};$$

$X = -4; 0; 4; 11.$

В заданиях 29 и 30 найти коэффициенты k_0, k_1, k_2 , представления числа X ($0 \leq X < P^N$) в позиционной системе счисления с основанием P , используя операции **mod** и **div**. Для контроля результатов выполнить вычисление X непосредственно по заданной формуле разложения X по степеням P для найденных коэффициентов, а также после преобразования выражения в формуле по схеме Горнера. Вывести все результаты вычислений в наглядной форме с поясняющими текстами. Проверить работу программы при вводимых значениях X из набора M .

29. $P = 8; N = 4; X = k_4 \cdot 8^4 + k_3 \cdot 8^3 + k_2 \cdot 8^2 + k_1 \cdot 8 + k_0;$
 $M = 0; 1; 2; 4; 7; 8; 65; 1023; 4095.$

30. $P = 16; N = 3; X = k_3 \cdot 16^3 + k_2 \cdot 16^2 + k_1 \cdot 16 + k_0;$
 $M = 0; 1; 15; 64; 127; 255; 2047; 4095.$

31. Найти среднее геометрическое абсолютных значений частных от целочисленного деления X, X_2, X_3 на Y и среднее арифметическое остатков от целочисленного деления X, X_2, X_3 на Y .

Для контроля результатов целочисленного деления выводить на экран с поясняющими надписями делимое, делитель, частное, абсолютное значение частного, остаток. Также с поясняющими текстами вывести найденные средние геометрические и средние арифметические. Проверить работу программы при вводе значений $X = (-5; 5)$ и $Y = (-3; 3)$.

32. Координаты вершин параллелепипеда заданы положительными значениями $X1, X2, Y1, Y2, Z1, Z2$ ($X1 < X2, Y1 < Y2, Z1 < Z2$), имеющими ненулевые дробные части. Требуется найти целочисленные координаты $I1, I2, J1, J2, K1, K2$ вершин такого параллелепипеда, который находится внутри заданного и имеет наибольший объем. Найти также объемы этих параллелепипедов и их отношение. Все значения $X1, X2, Y1, Y2, Z1, Z2$ и $I1, I2, J1, J2, K1, K2$ вывести на экран с поясняющими надписями, а найденные объемы и их отношения вывести с предшествующими поясняющими текстами. Проверить работу программы на вводимых $X1 = (2,7; 5,2), X2 = 2 X1, Y1 = X1 - 1, Y2 = 2 Y1, Z1 = X1/2, Z2 = 3 Z1$.

2.5. Задания Б для самостоятельной работы

Во всех заданиях вводимые и выводимые данные сопровождать краткими пояснениями; для проверки значений результатов предусмотреть в программе соответствующие вычисления.

В задачах, отмеченных звездочкой предусмотреть обработку возможных исключений при вычислении арифметических выражений с выводом сообщения о их типе. При невозможности продолжить поиск решения вывести соответствующее сообщение и завершить работу программы после нажатия клавиши Enter (см. приложение 2).

1*. Решить систему из двух линейных уравнений и проверить найденное решение подстановкой результатов в уравнения.

2. Вычислить площадь S остроугольного треугольника, заданного координатами вершин на плоскости, по формуле Герона, а затем — углы, используя соотношение $S = \frac{La \cdot Lb \cdot \sin C}{2}$, где C — угол между сторонами с длинами La и Lb . Для проверки результатов вычислить сумму углов.

3. Вычислить координаты точек на плоскости, которые делят отрезок прямой, заданный координатами концов, в отношении $m : n : k$.

4*. Решить квадратное уравнение, полагая, что оно имеет только вещественные корни. Проверить результаты подстановкой корней в уравнение.

5*. Вычислить коэффициенты уравнения прямой $Y = K \cdot X + B$, проходящей через точки с координатами $(X1, Y1)$ и $(X2, Y2)$, и найти точку пересечения этой прямой с осью абсцисс. Проверить результаты подстановкой точек в уравнение для заданных координат.

6*. Вычислить координаты точки пересечения двух прямых, заданных уравнениями $A \cdot X + B \cdot Y = C$ и $D \cdot X + E \cdot Y = F$. Проверить результаты подстановкой в уравнения.

7. Точка имеет координаты $X0, Y0$. Вычислить ее координаты после поворота осей относительно начала на угол A против часовой стрелки. Проверить работу программы для:

а) $A = \arctg(Y0/X0)$; б) $A = \pi$; в) $A = \arctg(Y0/X0) - \pi/2$.

8*. Вычислить координаты вершин треугольника, находящихся на пересечении прямых $Y = k1 \cdot X + b1$ и $Y = k2 \cdot X + b2$ между собой и с осью X . Найти площадь S этого треугольника, а также длины сторон. Проверить работу программы вводом данных для уравнений $Y = X + 1$ и $Y = -X + 1$.

9*. Вычислить координаты точек пересечения прямой и окружности на плоскости: $A \cdot X + B \cdot Y = C$, $X^2 + Y^2 = R^2$. Проверить результаты подстановкой в уравнения.

10*. Вычислить площадь S равнобедренного треугольника, вписанного в окружность радиусом R , если известна длина La его стороны, не равная длинам других сторон. Найти также длины других сторон треугольника и угол A между ними. Проверить работу программы на равностороннем треугольнике по его площади, которую следует вычислить заранее.

11*. Вычислить координаты точки пересечения эллипса $A \cdot X^2 + B \cdot Y^2 = R^2$ и гиперболы $Y = C/X$. Проверить результаты: при $A = B = C = 1$ и $R^2 = 2$ должно быть $X1 = X3 = 1$, $Y1 = Y3 = 1$, $X2 = X4 = -1$, $Y2 = Y4 = -1$.

12*. Найти числа X и Y , произведение которых равно A , а разность составляет B . Вывести найденные значения, а также для контроля — их произведение и разность. Проверить работу программы при $A = 1$ и $B = 0$, где решение очевидно.

13. Вычислить площадь треугольника, заданного координатами вершин в пространстве, по формуле Герона. Подобрать два варианта исходных данных для проверки работы программы.

14*. Найти числа X и Y , сумма которых равна A , а сумма квадратов равна B . Вывести найденные значения, а также для контроля — их сумму и сумму квадратов. Проверить работу программы при вводе $A = 1$ и $B = 1$, где решение очевидно.

15*. Для треугольника, заданного длинами сторон La , Lb , Lc , найти угол, противоположный стороне длины La , используя соотношение $\sin \frac{\alpha}{2} = \sqrt{\frac{(P-Lb)(P-Lc)}{Lb \cdot Lc}}$, где P — полупериметр треугольника. Найти другие углы. Проверить результаты для различных исходных данных по сумме углов.

16*. Вычислить координаты точек пересечения кривых, заданных уравнениями $Y = X + C$ и $\left(\frac{X}{2}\right)^2 + Y^2 = 1$. Проверить результаты подстановкой в исходные уравнения.

17. Вычислить площадь правильного N -угольника, вписанного в окружность радиусом R . Найти относительные ошибки замены площади круга площадью такого N -угольника при значениях N , равных 6, 60 и 360. Проверить правильность решения: при $N = 4$ и любом R относительная ошибка должна быть равна 0,363.

18. Вычислить площадь правильного N -угольника, в который вписана окружность диаметра D . Найти относительные ошибки замены площади такого N -угольника площадью круга при N , равном 12, 120, 720. Проверить правильность решения: при $N = 4$ и любом D относительная ошибка должна быть равна 0,274.

19. Вычислить площадь равнобедренного треугольника, вписанного в окружность радиусом R , если известен угол A между его сторонами равной длины. Вычислить отношение площади круга радиусом R к площади треугольника. Проверить работу программы при вычислении отношения площади круга радиусом R к площади треугольника при вводе следующих значений угла A :

а) $\pi/3$, когда площадь треугольника $\frac{3\sqrt{3}R^2}{4}$;

б) $\pi/2$, когда площадь треугольника R^2 .

20. Найти числа X и Y , сумма которых равна A , а разность равна B . Вывести найденные значения, а для контроля — их сумму и разность. Проверить работу программы при вводе $A = 1$ и $B = 1$, где решение очевидно.

21*. Для треугольника, заданного длинами сторон La , Lb , Lc , найти угол α , противоположный стороне длины La , используя соотношение $\cos \frac{\alpha}{2} = \sqrt{\frac{P(P-La)}{Lb \cdot Lc}}$, где P — полупериметр треугольника. Найти другие углы. Проверить результаты для различных исходных данных по сумме углов.

22. На плоскости найти угол A между двумя сторонами (1, 2) и (1, 3) остроугольного треугольника, заданного координатами вершин $X1$, $Y1$, $X2$, $Y2$, $X3$, $Y3$ ($X1 < X2 < X3$), длины $L12$, $L13$ сторон, и затем площадь треугольника S по формуле $S = (L12 \cdot L13 \cdot \sin A)/2$. Значение угла A вывести в градусах. Проверить работу программы при вводе $X1 = 0$, $Y1 = 0$, $X2 = 1$, $Y2 = 0$, $X3 = 2$ и $Y3 = (2, -2)$.

23*. Вычислить площадь S равнобедренного треугольника, в который вписана окружность радиусом R , если известна длина La его стороны, не равная длинам других сторон. Найти длину L других сторон треугольника и его углы. Проверить работу программы на равностороннем треугольнике по его площади, которую вычислить заранее.

24. Найти площадь прямоугольного треугольника, в который вписана окружность радиусом R , а также значения его углов, если известна длина La его катета Ka . Для проверки работы программы предусмотреть вычисление La по найденной длине Lb другого катета. Проверить работу программы при $R = 1$ и $La = 2 + \sqrt{2}$, когда прямоугольник равнобедренный.

25. Найти координаты центра тяжести треугольника на плоскости, т. е. координаты точки, лежащей на медиане и отстоящей на $2/3$ длины от вершины, из которой медиана проведена. Для проверки результата выполнить вычисления для всех трех медиан. Проверить работу программы для равнобедренного прямоугольного треугольника с координатами вершин $(0; 0)$, $(3; 0)$, $(0; 3)$, где решение очевидно.

26. Вычислить S — площадь остроугольного треугольника по формуле $S = \frac{La \cdot Lb \cdot \sin C}{2}$, где La и Lb — длины сторон, а C — угол между ними. Затем вычислить длину третьей стороны Lc , используя соотношение $Lc^2 = La^2 + Lb^2 - 2 La \cdot Lb \cdot \cos C$ и остальные углы, используя соотношение $\sin A/\sin C = La/Lc$. Проверить результаты для различных исходных данных по сумме углов.

27. Вычислить:

- а) уравнение прямой $Y = k_2 \cdot X + b_2$, проходящей через точку (X_0, Y_0) и перпендикулярной заданной прямой $Y = k_1 \cdot X + b_1$;
- б) точку (X_1, Y_1) пересечения этих прямых;
- с) площадь и длины сторон треугольника, вершинами которого являются точки (X_1, Y_1) , (X_0, Y_0) и точка (X_2, Y_2) пересечения оси Y с заданной прямой.

Проверить результаты, предварительно вычислив площадь треугольника с вершинами в этих точках при вводе $k_1 = 1$, $b_1 = 1$, $X_0 = 0$, $Y_0 = 2$.

28. Дано уравнение $A \cdot X^2 + B \cdot Y^2 + C \cdot X + D \cdot Y + E = 0$. Вычислить коэффициенты уравнения $A_1 \cdot X^2 + B_1 \cdot Y^2 + C_1 \cdot X + D_1 \cdot Y + E_1 = 0$ после переноса начала координат в точку X_1, Y_1 и проверить результаты. Выполнить проверку решения обратным преобразованием координат.

3. Программы разветвляющейся структуры

3.1. Средства разработки программ разветвляющейся структуры

Программой разветвляющейся структуры называют такую, в которой в зависимости от исходных данных возможны различные последовательности выполнения операторов, причем на любой последовательности каждый оператор выполняется только один раз.

Для реализации программ или фрагментов программ с разветвляющейся структурой используют *сложные* операторы¹: условные **if** и выбора **case**. В этом разделе ограничимся рассмотрением полной формы условного оператора — оператора **if then else** и его сокращенной формы — оператора **if then**.

В случае применения условных операторов ветвление алгоритма обусловлено проверками логических выражений (в языке Object Pascal их называют булевыми выражениями), результатом вычисления которых могут быть лишь два значения: *истина* и *ложь*. В условных операторах используются как простейшие булевы выражения, основанные на сравнении выражений других типов, так и сложные, с логическими операциями.

Условные операторы

Оператор **if then else** имеет синтаксическую диаграмму, показанную на рис. 3.1, где БВ — *булево выражение* (см. ниже), значением которого может быть либо *истина*, либо *ложь*. Оп1 и Оп2 — операторы, каждый из которых может быть пустым.

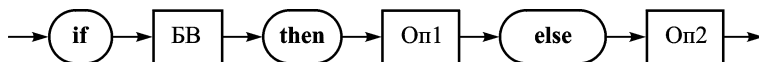


Рис. 3.1

При выполнении оператора **if then else** вначале вычисляется выражение БВ, и если результат *истина*, то — оператор Оп1, если результат имеет значение *ложь* — оператор Оп2.

¹ Сложные операторы включают в себя другие операторы.

Оператор **if then** имеет синтаксическую диаграмму, показанную на рис. 3.2, где БВ — булево выражение, Оп1 — оператор.

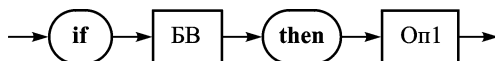


Рис. 3.2

При выполнении оператора **if then** вначале вычисляют выражение БВ, и если результат — *истина*, то переходят к оператору Оп1, если результат — *ложь*, управление передается следующему по порядку оператору программы.

Простейшими булевыми выражениями являются отношения. Знаки отношений записывают следующим образом: $>$, $<$, $=$ — так же, как в математике; знаки \leq , \geq , \neq — парами символов $<=$, $>=$, $<>$. Более сложные булевы выражения рассмотрены далее.

Пример. Требуется записать условный оператор, вычисляющий новое значение Y исходя из заданных значений A , B , X , Y по формуле

$$Y = \begin{cases} 2, & \text{если } (A > B) \& B \geq 3; \\ Y, & \text{если } (A > B) \& B < 3; \\ X & - \text{ в остальных случаях,} \end{cases}$$

т. е. в соответствии с алгоритмом, показанным на рис. 3.3.

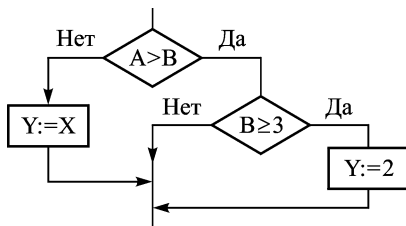


Рис. 3.3

Вот этот оператор:

```
if A > B then
  if B >= 3 then
    Y:=2
  else
else
  Y:=X;
```

Оператору **if then else** подчинен оператор присваивания $Y:=X$ и еще один оператор **if then else**, который, в свою очередь, содержит *пустой оператор* (после первого **else**) и оператор присваи-

вания $Y:=2$. Необходимость использования **else** во вложенном условном операторе вытекает из следующего правила: **else** относится к ближайшему предшествующему **if**, у которого нет части **else**. Можно было не использовать **else** во вложенном условном операторе, но тогда пришлось бы заключить его в *операторные скобки*, т. е. заменить его оператором

begin if $B \geq 3$ then $Y:=2$ end.

Пример. Для функции предыдущего примера можно составить другой алгоритм (рис. 3.4).

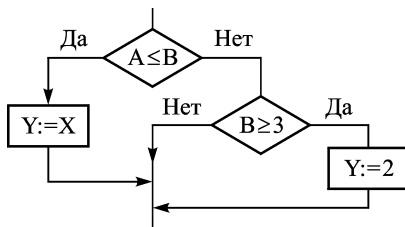


Рис. 3.4

Тогда соответствующим ему оператором **if then else** будет

```

if A ≤ B then
    Y:=X
else
    if B ≥ 3 then
        Y:=2
  
```

и вложенный в него условный оператор следует использовать в сокращенной форме.

Булев тип

Стандартный тип Boolean. Он представляет всего два значения: *ложь* и *истина* (соответствующие булевы константы — False и True). Операциями над данными типа Boolean являются:

- *отрицание* (другое название — *не*, знак операции **not**),
- *конъюнкция* (другие названия — *логическое произведение* или просто *И*, знак операции **and**),
- *дизъюнкция* (другие названия — *логическая сумма* или просто *ИЛИ*, знак операции **or**),
- *неравнозначность* (другие названия — *неэквивалентность* или *сумма по модулю два*, знак операции **xor**).

Отрицание является унарной операцией, остальные — бинарными.

В математике операции **not** соответствует знак \neg перед аргументом ($\neg X$) или черта над аргументом (\bar{X}); операции **and** соответствует знак $\&$ или \cdot , или \wedge ($X \& Y$ или $X \cdot Y$ или $X \wedge Y$); операции **or** соответствует знак \vee ($X \vee Y$); операции **xor** соответствует знак ∇ ($X \nabla Y$).

Ниже представлены их таблицы истинности.

X	not X	X	Y	X and Y	X or Y	X xor Y
False	True	False	False	False	False	False
True	False	True	False	False	True	True
		False	True	False	True	True
		True	True	True	True	False

В общем случае *булевы выражения* содержат отношения, булевы константы, переменные и функции, возвращающие булевы значения, разделенные знаками булевых операций и круглыми скобками. Операция **not** имеет наивысший приоритет и выполняется в первую очередь, операция **and** имеет тот же приоритет, что и арифметические операции типа умножения, операции **or** и **xor** имеют тот же приоритет, что и арифметические операции типа сложения и, наконец, в последнюю очередь вычисляются отношения.

Пример. Составить условный оператор для вычисления нового значения Y по формуле

$$Y = \begin{cases} A, & \text{если } (A \cdot B > 1) \& (A > 0), \\ B, & \text{если } (A + B > 1) \& (A < 0), \\ Y & - \text{ в остальных случаях.} \end{cases}$$

Как видно из задания, при истинности одного из условий другие будут иметь значение *ложь*, поэтому для вычисления лучше использовать не два, а один условный оператор **if then else**, что приведет к сокращению вычислений:

```

if (A*B>1) and (A>0) then
    Y:=A
else
    if (A+B>1) and (A<0) then
        Y:=B;
```

Во избежание ошибок компиляции отношения, входящие в сложные булевы выражения, следует заключать в скобки, так как приоритет операций отношений ниже приоритетов других операций.

3.2. Примеры выполнения задания

1. Составить программу вывода значений функции $Y(X)$, заданной графиком (рис. 3.5) (функция не определена при $|X| > 3$)

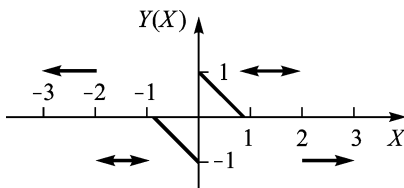


Рис. 3.5

или, что то же самое, формулой

$$Y(X) = \begin{cases} Z(1 - |X|), & \text{если } 0 \leq |X| \leq 1, \\ Z, & \text{если } 1 < |X| < 2, \\ -Z, & \text{если } 2 \leq |X| < 3, \\ \text{в остальных случаях} & \text{– не определена,} \end{cases}$$

где $Z = -1$, если $X < 0$; $Z = 0$, если $X = 0$; $Z = 1$, если $X > 0$, двумя способами:

а) с помощью минимального числа операторов **if then else**, без применения булевых операций (**not**, **and**, **or**, **xor**),

б) с помощью минимального числа операторов **if then** (без **else**) с применением булевых операций.

Вывести с поясняющими текстами значение X и вычисленные значения функции:

```
program Project2;
{$APPTYPE CONSOLE}
uses
  SysUtils, Math;
var
  X, A, Z, Y: Real;
  F: Boolean;
begin
  Write('Введите значение аргумента: ');
  ReadLn(X);
  //Вычисление абсолютного значения X
  A:=Abs(X);
  Z:=Sign(X); //Вычисление Z
  { Вычисление без применения булевых операций }
  if A>=3 then
    WriteLn('Функция не определена. ')
```



```

else
begin
  if A<=1 then
    Y:=Z*(1-A)
  else if A<2 then
    Y:=Z
  else
    Y:=-Z;
  WriteLn('Y = ',Y:4:2);
end;
{ с применением булевых операций }
if A>=3 then
  WriteLn('Функция не определена. ');
if A<=1 then
  WriteLn('Y = ',Z*(1-A):4:2);
if (A>1) and (A<2) then
  WriteLn('Y = ',Z:4:2);
if (A>=2) and (A<3) then
  WriteLn('Y = ',-Z:4:2);
ReadLn;
end.

```

2. Составить программу вычисления Z — номера области (см. рис. 3.6), в которую попадает точка с координатами (X, Y) , с помощью одного оператора **if then else** двумя способами:

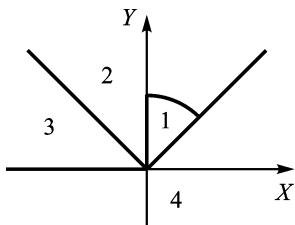


Рис. 3.6

а) без применения булевых операций (**not**, **and**, **or**, **xor**) и с сохранением результата в переменной $Z1$,

б) с применением булевых операций и сохранением результата в переменной $Z2$.

Все области, кроме области с номером 1 с границей в виде дуги окружности радиусом $R = 5$, бесконечны. Точку, лежащую на границе областей, можно считать принадлежащей любой из них.

Вывести с поясняющими текстами значения X , Y и вычисленные значения $Z1$ и $Z2$:

```

program Project2;
{$APPTYPE CONSOLE}
uses
  SysUtils;
const
  R=5;
var

```

```
X,Y:Real;
Z1,Z2:Byte;
begin
//Ввод значения аргумента функции
  Write('X, Y: '); ReadLn(X, Y);
//Определение Z1 - номера области, которой
//принадлежит точка, без применения булевых
//операций.
//Если точка лежит в нижней полуплоскости
//или на оси X,
  if Y<=0 then //то
    Z1:=4 //переменной Z1 присвоить значение 4,
  else //иначе (то есть точка лежит в верхней
    //полуплоскости),
    if Y<=X then //если Y не больше X, то
      Z1:=4 //переменной Z1 присвоить значение 4,
    else //иначе (то есть точка вне области 4),
      if Y<-X then //если точка лежит ниже прямой,
        //разделяющей области 2 и 3, то
        Z1:=3 //переменной Z1 присвоить значение 3,
      else //иначе (то есть точка лежит выше или
        //на прямой, разделяющей области 2 и 3),
        if X<=0 then //если X<=0, то переменной
          Z1:=2 //Z1 присвоить значение 2,
        else //иначе (то есть точка лежит в первой
          //четверти в области 1 или 2),
          if Sqr(X)+Sqr(Y)<=Sqr(R) then //если
            //расстояние до точки от начала
            //координат не превосходит R, то
            Z1:=1 //точка лежит в области 1,
          else //иначе (то есть расстояние до точки
            //от начала координат превосходит R),
            //значит
            Z1:=2; //точка лежит в области 2.
    WriteLn('Z1 = ', Z1);
//Определение номера Z2 - области, которой принадлежит
//точка, с применением булевых операций.
  if (Y<=0)or(Y<=X) then //Если точка принадлежит
    //области 4, то
    Z2:=4 //переменной Z2 присвоить значение 4,
  else //иначе (то есть точка вне области 4),
    if Y<-X then //если точка принадлежит области 3, то
      Z2:=3 //переменной Z2 присвоить значение 3,
    else //иначе (то есть точка вне областей 3 и 4),
```

```

    if (Sqr(X)+Sqr(Y)<=Sqr(R)) and (X>=0) then
        // если точка принадлежит области 1, то
        Z2:=1 //переменной Z2 присвоить значение 1,
    else //иначе (то есть точка вне областей 1, 3 и 4),
        Z2:=2; //переменной Z2 присвоить значение 2.
WriteLn('Z2 = ', Z2);
ReadLn;
end.

```

3.3. Задания для самостоятельной работы

В заданиях от 1 до 25 требуется для зависимости $Y(X)$, заданной аналитически или графически, составить программу вычисления для вводимого X :

- $Y1 = Y(X)$ — с помощью минимального числа операторов **if then else**, без применения булевых операций (**not**, **and**, **or**, **xor**),
 - $Y2 = Y(X)$ — с помощью минимального числа операторов **if then** (без **else**), с применением булевых операций,
- и вывести с поясняющими текстами вычисленные значения $Y1$ и $Y2$.

Для значений аргумента, при которых функция не определена, выводить соответствующие сообщения.

В заданиях с графиками функций (см. пример 1 выполнения задания):

- стрелка на линии графика указывает открытую границу интервала, в котором функция имеет заданное положением линии значение,
- в точках отсутствия линии графика функция не определена.

$$1. Y(X) = \begin{cases} 1 & \text{при } X < -2, \\ X/2 & \text{при } -2 \leq X < 0, \\ \text{не определена} & \text{при } X = 0, \\ 2 & \text{— в остальных случаях.} \end{cases}$$

$$2. Y(X) = \begin{cases} 0 & \text{при } -1 > X, \\ 1 & \text{при } -1 \leq X < 0, \\ -1 & \text{при } 0 \leq X < 2, \\ 1 & \text{при } 2 \leq X. \end{cases}$$

$$3. Y(X) = \begin{cases} X & \text{при } 0 > X + X^2 > -0,2, \\ X^2 & \text{при } 0 < X + X^2, \\ \text{иначе} & \text{— не определена.} \end{cases}$$

$$4. Y(X) = \begin{cases} 1 & \text{при } -1 > X, \\ 0 & \text{при } -1 \leq X < 0, \\ -1 & \text{при } 0 \leq X < 1, \\ 0 & \text{при } 1 \leq X. \end{cases}$$

$$5. Y(X) = \begin{cases} -1 & \text{при } -2 > X, \\ X+1 & \text{при } -2 \leq X < 0, \\ 1 & \text{при } 0 \leq X < 1, \\ 0 & \text{при } 1 \leq X. \end{cases}$$

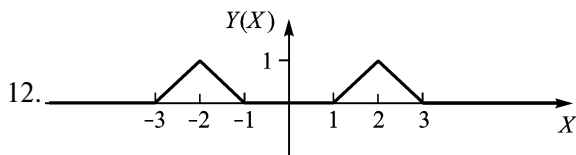
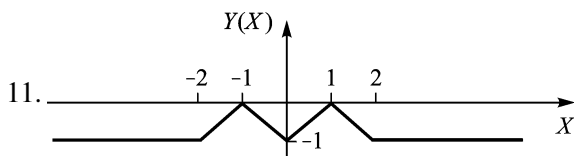
$$6. Y(X) = \begin{cases} 0 & \text{при } X < -2, \\ 1 & \text{при } -2 \leq X < -1, \\ 0 & \text{при } -1 \leq X < 0, \\ 1 & \text{при } 0 \leq X. \end{cases}$$

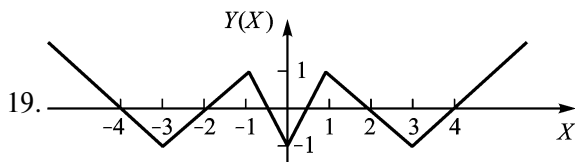
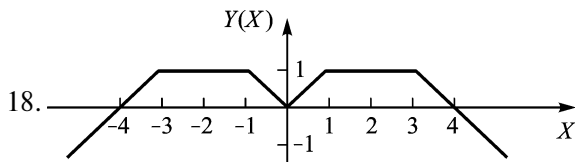
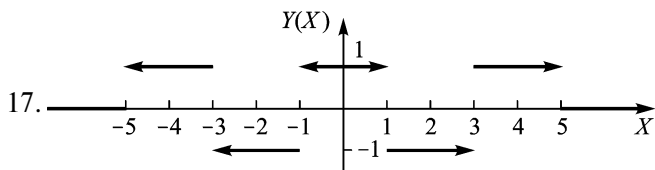
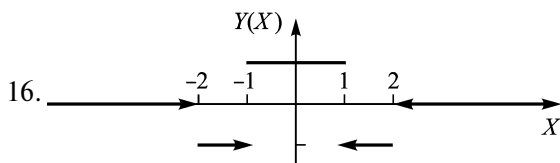
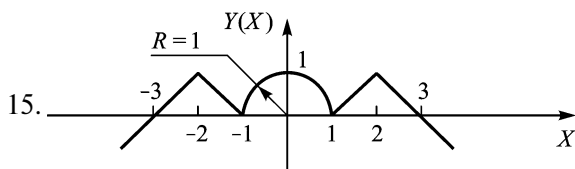
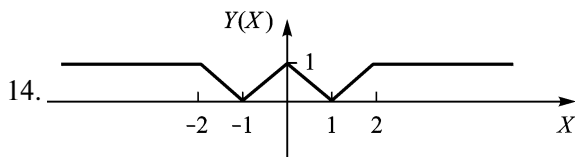
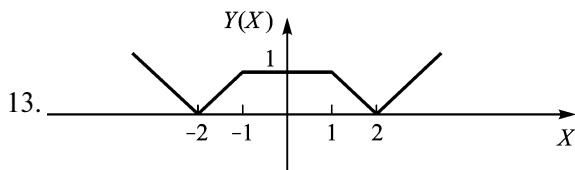
$$7. Y(X) = \begin{cases} 0 & \text{при } X < -1, \\ X+1 & \text{при } -1 < X < 0, \\ X & \text{при } 0 \leq X < 1, \\ 0 & \text{при } 1 \leq X. \end{cases}$$

$$8. Y(X) = \begin{cases} -1 & \text{при } X < -1, \\ X & \text{при } -1 \leq X < 1, \\ -X+2 & \text{при } 1 \leq X < 2, \\ 0 & \text{при } 2 \leq X. \end{cases}$$

$$9. Y(X) = \begin{cases} -1/X & \text{при } X < -3, \\ \sqrt{-X} & \text{при } -3 \leq X < 0, \\ X^2 & \text{при } 0 \leq X < 1, \\ \sqrt{X} & \text{при } 1 \leq X. \end{cases}$$

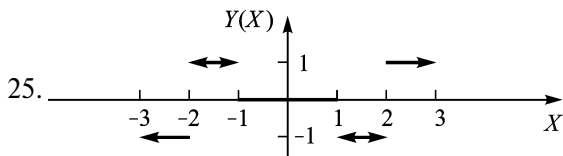
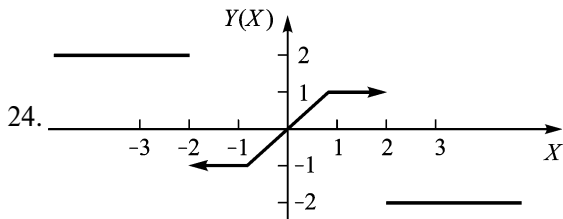
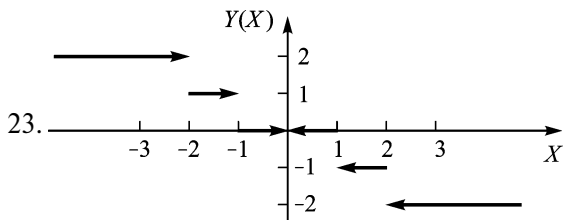
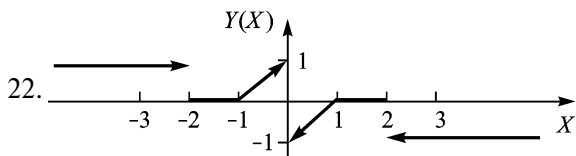
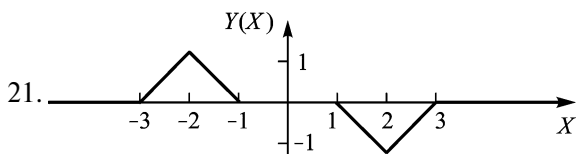
$$10. Y(X) = \begin{cases} -2-X & \text{при } X \leq 0, \\ 0 & \text{при } 0 < X < 1, \\ X & \text{при } 1 \leq X < 3, \\ 1-X & \text{при } 3 \leq X. \end{cases}$$





$$20. Y(X) = \begin{cases} 0, & \text{если } |X| < 3, \text{ иначе} \\ 1, & \text{если } \lceil |X| \rceil \oplus 2 \text{ четное, иначе} \\ -1, & \text{если } \lceil |X| \rceil \oplus 2 \text{ нечетное,} \end{cases}$$

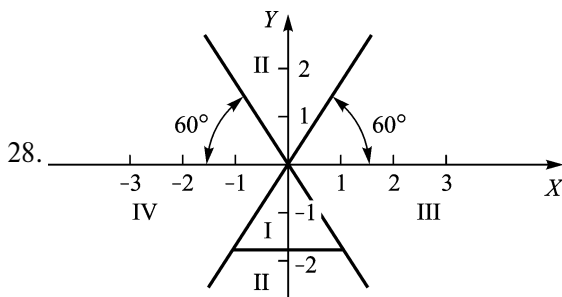
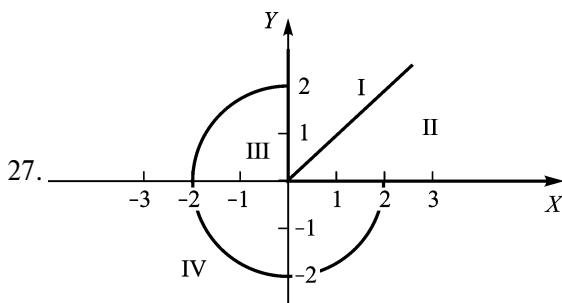
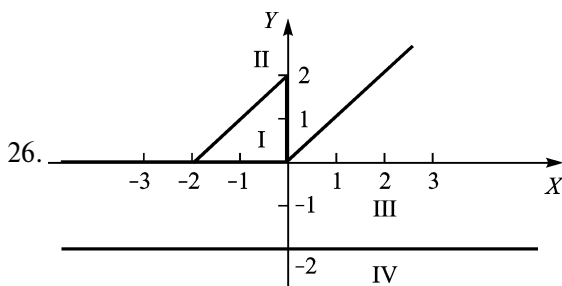
где скобки $\lceil \rceil$ обозначают целую часть числа, а знак \oplus — остаток от деления целого числа на 2.

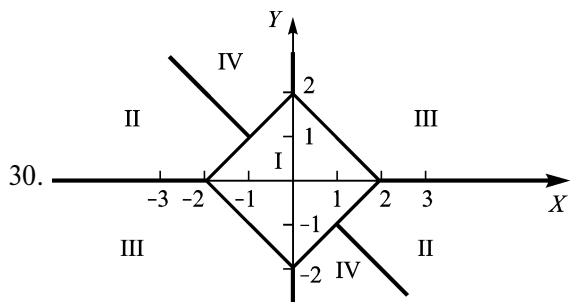
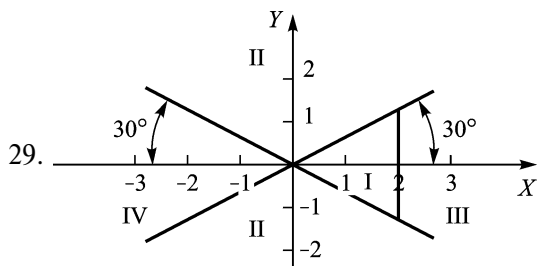


В заданиях 26—30 для рисунков, на которых области обозначены римскими цифрами, требуется составить программу вычисления для вводимых X и Y :

- $Z1$ — номера области с помощью минимального числа операторов **if then else**, без применения булевых операций (**not**, **and**, **or**, **xor**),

- $Z2$ — номера области с помощью минимального числа операторов **if then** (без **else**), с применением булевых операций и вывода с поясняющими текстами вычисленных значений $Z1$ и $Z2$. Точку, лежащую на границе областей, можно считать принадлежащей любой из них.





4. Программы циклической структуры

4.1. Средства разработки программ циклической структуры

Программой циклической структуры называют такую, в которой операторы могут повторно, при изменяющихся значениях *переменных* выполняться несколько раз, образуя *цикл*. Различают следующие виды циклов (для их организации используют специальные сложные операторы — *операторы циклов*):

- *цикл с заданным числом повторений* или *цикл с параметром* (операторы цикла **for: for to** и **for downto**),
- *цикл с предусловием* (оператор цикла **while**),
- *цикл с постусловием* (оператор цикла **repeat until**).

В циклах можно выделить управляющие части, определяющие начало и условия выполнения, и части из одного или нескольких операторов (*тело*), выполняющие необходимые преобразования данных. Цикл называют *простым*, если в его теле нет других циклов.

Цикл с параметром

Структура оператора цикла **for to** описывается синтаксической диаграммой (рис. 4.1), где используют следующие обозначения: I — параметр цикла — переменная *ординального* (*порядкового*), в частности целого, типа; B1 и B2 — выражения того же типа, что и параметр цикла, или совместимые с ним; Оп — оператор, выполняемый внутри цикла.

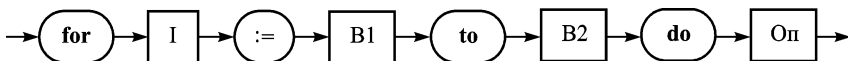


Рис. 4.1

Часть, предшествующая оператору Оп, — *заголовок цикла* — является управляющей, а сам оператор Оп — *телом цикла*. Оператор Оп будет последовательно выполняться при автоматическом увеличении с минимальным шагом значения параметра цикла I от B1 до B2 включительно (для целых типов шаг равен 1). При B1 > B2 оператор Оп не будет выполняться вообще.

Например, в цикле

```
for I:=0 to 6 do
  WriteLn(I*10:2,Sin(I/18*Pi):8:2);
```

оператор **WriteLn** будет выполняться 7 раз при *I*, изменяющемся от 0 до 6 с шагом 1.

На экран будет выведена таблица, в первом столбце которой целые числа 0, 10, 20, ..., 60, представляющие углы в градусах, а во втором — соответствующие им значения синуса:

0	0.00
10	0.17
20	0.34
30	0.50
40	0.64
50	0.77
60	0.87

Структура оператора цикла **for downto** описывается синтаксической диаграммой (рис. 4.2), а его работа отличается от оператора **for to** тем, что параметр цикла *I* не увеличивается, а уменьшается от *B1* до *B2*, а оператор *Оп* не будет выполняться вообще при *B1* < *B2*.

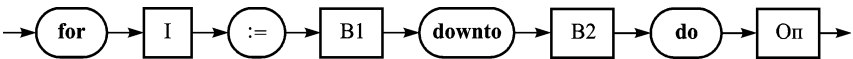


Рис. 4.2

Цикл с предусловием

Структура оператора цикла **while** описывается синтаксической диаграммой (рис. 4.3), где *БВ* — булево выражение, *Оп* — оператор, выполняемый внутри цикла (тело цикла).

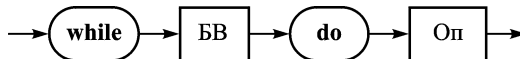


Рис. 4.3

Заголовок цикла — конструкция, предшествующая оператору *Оп*, управляет выполнением цикла следующим образом: оператор *Оп* будет последовательно выполняться, пока выражение *БВ* имеет значение **True**, или не будет выполняться вообще, если до выполнения оператора **while** *БВ* имеет значение **False**.

Например, фрагмент программы

```
N:=0;  
while N<=60 do  
begin  
  WriteLn(N:2,Sin(N/180*Pi):8:2);  
  N:=N+10  
end
```

будет выполнять ту же работу, что и оператор **for to** в предыдущем примере.

Цикл с постусловием

Структура оператора цикла **repeat until** описывается синтаксической диаграммой (рис. 4.4).

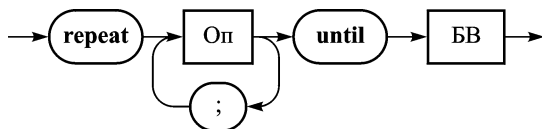


Рис. 4.4

Внутри такого цикла может находиться произвольное число операторов Оп, которые будут выполняться один или более раз до получения булевым выражением БВ значения True. Например, такую же таблицу, что и в первом примере с применением оператора **for to**, будет выводить следующий фрагмент программы:

```
N:=0;  
repeat  
  WriteLn(N:2,Sin(N/180*Pi):8:2);  
  N:=N+10  
until N>60;
```

В приведенных примерах переменные *I* и *N* изменялись по закону арифметической прогрессии. Нередко возникает необходимость иметь в цикле переменную — *дополнительный параметр цикла*, изменяющуюся по требуемому закону. Сделать это можно так: до входа в цикл этой переменной дается начальное значение, а внутри цикла значение переменной изменяется нужным образом с помощью оператора присваивания.

В фрагменте программы

```
R:=5;  
for K:=1 to N do
```

```
begin  
    . . . . .  
    R:=R*1.2;  
    . . . . .  
end;
```

R — дополнительный параметр, который в цикле при $N = 4$ будет последовательно получать значения 5; 6; 7,2; 8,64, изменяясь по закону геометрической прогрессии умножением предыдущего значения на 1,2.

Выход из цикла по условию, объявленному в его управляющей части, будем называть *естественным*. При этом для циклов с параметром (организованным операторами **for**) рекомендуется считать, что значение параметра становится неопределенным.

Существует возможность и досрочного выхода из любого цикла, организованного рассмотренными операторами, либо с помощью оператора безусловного перехода **goto** (их мы не будем использовать), либо с помощью оператора Break. В этом случае текущее значение параметра цикла **for** сохраняется (считается определенным), и его можно использовать в дальнейших вычислениях.

В теле любого из рассмотренных циклов допускается использовать оператор Continue. Его действие сводится к тому, что сразу происходит переход к очередному выполнению тела цикла (в циклах **for** с очередным значением параметра) или выход из цикла, если выполнено условие его завершения.

4.2. Вычисление и вывод данных в виде таблицы

Простейшими примерами применения операторов цикла на практике являются программы вычисления значений функций при изменяющихся значениях аргумента и вывода данных в виде таблиц с заголовками. В качестве аргумента обычно выступает переменная — дополнительный параметр, изменяющаяся в цикле по требуемому закону. В циклах **while** и **repeat** эта переменная используется в условиях завершения цикла.

При работе с вещественными данными необходимо иметь в виду, что их значения могут представляться с ошибкой, что эти ошибки могут зависеть от типа переменных, накапливаться при выполнении арифметических операций и приводить к непредусмотренному программистом выполнению программы. Рассмотрим два фрагмента программы вывода таблицы значений аргумента и функции, где аргумент — дополнительный параметр в цикле — является вещественной переменной с именем X :

1)

```
//Все переменные типа Extended
X:=0; H:=1/3; Xk:=5/3;
while X<=Xk do
begin
  WriteLn(X:6:2,Sin(X):8:2);
  X:=X+H;
end;
```

2)

```
H:=1/3;X:=-H; Xk:=5/3;
repeat X:=X+H;
  WriteLn(X:6:2,Sin(X):8:2);
until X=Xk;
```

Казалось бы, оба фрагмента выведут на экран таблицы значений синуса для аргумента X , изменяющегося от 0 до $5/3$ включительно с шагом $1/3$. Но в действительности первый фрагмент не выведет значения $5/3$ и $\sin(5/3)$, а второй не обеспечит завершение выполнения цикла при $X = 5/3$, так что цикл продолжит выполняться и при больших значениях X .

Во избежание подобных ситуаций следует вместо проверок вещественных данных на равенство использовать проверки на $<$ (меньше) и/или $>$ (больше) с некоторым достаточно малым запасом, превышающим точность представления чисел данного типа и не нарушающим логику работы программы. Например, рассмотренные фрагменты программы можно изменить так:

1)

```
X:=0; H:=1/3;
//Запас в H/2=1/6 больше ошибки
//представления вещественных чисел
Xk:=5/3+H/2;
while X<Xk do //и при X=5/3 цикл будет выполнен.
begin
  WriteLn(X:6:2,Sin(X):8:2);
  X:=X+H;
end;
```

2)

```
H:=1/3;X:=-H;
//Запас в H/2 больше ошибки
//представления вещественных чисел
Xk:=5/3-H/2;
repeat X:=X+H;
```

```
WriteLn(X:6:2,Sin(X):8:2);
until X>Xk;//и выход из цикла будет, и будет
//только при X, равном 5/3, что больше Xk.
```

Рассмотрим решение этой же задачи вывода таблицы значений синуса, но при использовании оператора цикла **for**. Потребуется заранее вычислить, сколько раз цикл должен выполняться, для чего воспользуемся формулой

$$N = \left\lceil \frac{X1 - X0}{H} \right\rceil + 1,$$

где скобки обозначают округление; N — искомое число повторений цикла; $X0$ и $X1$ — начальное и конечное значения аргумента; H — шаг изменения аргумента.

Так как в нашем случае $X0 = 0$, $X1 = 5/3$ и $H = 1/3$, фрагмент программы с циклом **for** будет следующим:

```
H:=1/3; X:=0;
N:=Round(5/3/H)+1;
for i:=1 to N do
begin
  WriteLn(X:6:2,Sin(X):8:2);
  X:=X+H;
end;
```

4.3. Пример выполнения задания с использованием цикла while

Упростив вычисления за счет использования дополнительных переменных и/или скобочных форм, вычислить значения функции

$$Y(X) = \frac{e^{-X^2}}{X} \sin(10X)$$

и ее производной

$$Y'(X) = \frac{X[e^{-X^2} 10 \cos(10X) - 2Xe^{-X^2} \sin(10X)] - e^{-X^2} \sin(10X)}{X^2}$$

на интервале значений X 40...50° с шагом $(1/3)^\circ$.

Для проверки правильности вычислений $Y'(X)$ найти ее значение по формуле без преобразований, а также по *разностной схеме*

$$Y'(X) = \frac{Y(X + \Delta X/2) - Y(X - \Delta X/2)}{\Delta X}$$

при $\Delta X = 10^{-8}$.

Вычисленные значения вывести с предшествующими порядковыми номерами и соответствующими значениями аргумента X в виде таблицы с заголовками столбцов:

```

program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils, Math;
const
  A=40/180*Pi;           //Начальное значение аргумента
  H=Pi/180;              //Шаг изменения аргумента
  B=50/180*Pi+H/2;      //Конечное значение аргумента
var  F1,
     X,Y,P,P1,C,D,E,S:Extended;
     I:Integer;
begin
  //Вывод заголовка таблицы
  WriteLn('#':3,'X':8,'Y(X)':12
          , 'P':11,'P1':11,'F1':14);
  X:=A; //Переменная X будет представлять
        //текущее значение аргумента
  I:=0; //Переменная I будет представлять
        //номер строки таблицы
  while X<B do
  begin
    //Увеличение значения счетчика строк таблицы
    I:=I+1;
    //Вычисление значений дополнительных переменных
    C:=Sqr(X); D:=10*X; E:=Exp(-C); S:=Sin(D);
    //Вычисление Y - значения функции и P - ее производной
    //с использованием дополнительных переменных.
    Y:=E*S/X;
    P:= E*(D*Cos(D)-S*(1+2*C))/C;
    //Вычисление P1 - контрольного значения производной
    //без использования дополнительных переменных.
    P1:=((Exp(-Sqr(X))*10*Cos(10*X)-2*X*Exp(-Sqr(X))
          *Sin(10*X))*X-Exp(-Sqr(X))*Sin(10*X))/Sqr(X);
    //Вычисление F1 - контрольного значения
    //производной по разностной схеме.
    F1:= ( Exp(-Sqr((X+5E-9)))*Sin(10*(X+5E-9))/(X+5E-9)
          -Exp(-Sqr((X-5E-9)))*Sin(10*(X-5E-9))/(X-5E-9)
          )/1E-8;
    //Вывод в строку таблицы вычисленных значений

```

```

WriteLn(I:3, X*180/Pi:10:2,Y:12:5
      ,P:14:5,P1:10:5, F1:14:5 );
X:=X+H;      //Увеличение значения аргумента
end; //while
ReadLn;
end.

```

4.4. Пример выполнения задания с использованием цикла for

Для функции

$$F(X) = \frac{2X(X-1) - (X^2-1)\ln(X^2-1)}{(X^2-1)(X-1)^2}$$

при десяти значениях приращения аргумента $DX = 0,5; 0,25; 0,125; \dots$ вычислить:

1) точные значения приращений первообразной

$$DP(X) = \frac{\ln[(X+DX)^2-1]}{(X+DX)-1} - \frac{\ln(X^2-1)}{X-1};$$

2) по формуле $F(X+DX/2) \cdot DX$ — приближенные значения:

а) $DP1$, упростив вычисления за счет дополнительных переменных,

б) $DP2$, не используя дополнительных переменных, и

3) $|DP - DP1|$ — абсолютные ошибки найденных приближенных значений.

Результаты вычислений и соответствующие значения DX вывести в виде таблицы с заголовками столбцов и номерами строк:

```

program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
const
  X=1.5;
var
  DP, DP1, DP2, D, R, K, X1, DX: Extended;
  I, N: Integer;
begin
  //Вывод заголовка таблицы
  WriteLn(' #      DX              DP              DP1 '
        , '      DP2              |DP-DP1| ');
  //Переменная DX будет представлять

```



```

//текущее приращение аргумента
DX:=0.5;
for I:=1 to 10 do
begin
  //Вычисление точного значения DP(X)
  DP:=Ln(Sqr(X+DX)-1)/(X+DX-1)-Ln(Sqr(X)-1)/(X-1);
  //Вычисление значений дополнительных переменных
  X1:=X+DX/2; K:=Sqr(X1); R:=K-1; D:=X1-1;
  //Вычисление DP(X) с использованием
  //дополнительных переменных
  DP1:=(2 * X1 * D - R * Ln(R)) / R / Sqr(D)*DX;
  //Вычисление DP(X)
  //без использования дополнительных переменных
  DP2:=(2*(X+DX/2)*(X+DX/2-1)
        - (Sqr(X+DX/2)-1)*Ln(Sqr(X+DX/2)-1))
        / (Sqr(X+DX/2)-1)/Sqr(X+DX/2-1)*DX;
  //Вывод в строку таблицы вычисленных значений
  WriteLn(i:3,DX:10:5,' ',DP:12:6,' ', DP1:10:6
          , ' ', DP2:9:6,' ',Abs(DP-DP2):12);
  //Изменение значения приращения аргумента
  DX:=DX/2;
end; //for i
ReadLn;
end.

```

4.5. Задания для самостоятельной работы

Во всех заданиях использовать только простые циклы.

1. Вычислить для первых 20 значений $X = \frac{1}{2}, 1 - \frac{2}{3}, 1 - \frac{3}{4}, \dots$ и вывести в виде таблицы с заголовками:

- значения функции $\ln(1 + X)$;
- приближенные значения функции по формуле

$$X - \frac{X^2}{2} + \frac{X^3}{3} - \frac{X^4}{4} + \frac{X^5}{5},$$

используя скобочные формы и/или дополнительные переменные;

- приближенные значения функции по этой же формуле, не используя скобочные формы и дополнительные переменные;
- абсолютную и относительную ошибки приближенных значений.

Для организации цикла использовать оператор **for downto**; при вычислениях приближенных значений — только операции сложения, вычитания, умножения, деления и стандартную функцию $\text{Sqr}(X)$.

2. Вычислить при $X = -0,5; -0,25; 0; 0,25; 0,5; 0,75; 1$ и вывести в виде таблицы с заголовками:

- значения функции e^X ;
- приближенные значения функции по формуле

$$1 + X + \frac{X^2}{2!} + \frac{X^3}{3!} + \frac{X^4}{4!} + \frac{X^5}{5!},$$

используя скобочные формы и/или дополнительные переменные;

- приближенные значения функции по этой же формуле, не используя скобочные формы и дополнительные переменные;
- абсолютную и относительную ошибки приближенных значений.

Для организации цикла использовать оператор **while**; при вычислениях приближенных значений — только операции сложения, вычитания, умножения, деления и стандартную функцию $\text{Sqr}(X)$.

3. Вычислить при X , изменяющемся от $0,1$ до $\pi/3$ с шагом $0,05$, и вывести в виде таблицы с заголовками:

- значения функции $\sin(X)$;
- приближенные значения функции по формуле

$$X - \frac{X^3}{3!} + \frac{X^5}{5!} - \frac{X^7}{7!} + \frac{X^9}{9!},$$

используя скобочные формы и/или дополнительные переменные;

- приближенные значения функции по этой же формуле, не используя скобочные формы и дополнительные переменные;
- абсолютную и относительную ошибки приближенных значений.

Для организации цикла использовать оператор **for to**; при вычислениях приближенных значений — только операции сложения, вычитания, умножения, деления и стандартную функцию $\text{Sqr}(X)$.

4. Вычислить в цикле **repeat until** при X , изменяющемся от 0 до $\pi/4$ с шагом $0,1$, и вывести в виде таблицы с заголовками:

- значения функции $\cos(X)$;
- приближенные значения функции по формуле

$$1 - \frac{X^2}{2!} + \frac{X^4}{4!} - \frac{X^6}{6!} + \frac{X^8}{8!},$$

используя скобочные формы и/или дополнительные переменные;

- приближенные значения функции по этой же формуле, не используя скобочные формы и дополнительные переменные;
- абсолютную и относительную ошибки приближенных значений.

При вычислениях приближенных значений использовать только операции сложения, вычитания, умножения, деления и стандартную функцию $\text{Sqr}(X)$.

5. Вычислить при X , изменяющемся от A до B с шагом H , и вывести в виде таблицы с заголовками:

- значения функции $\text{tg}(X)$;
- приближенные значения функции по формуле

$$X + \frac{X^3}{3} + \frac{2X^5}{15} + \frac{17X^7}{315} + \frac{62X^9}{2835},$$

используя скобочные формы и/или дополнительные переменные;

- приближенные значения функции по этой же формуле, не используя скобочные формы и дополнительные переменные;
- абсолютную и относительную ошибки приближенных значений.

Для организации цикла использовать оператор **for downto**; при вычислениях приближенных значений — только операции сложения, вычитания, умножения, деления и стандартную функцию $\text{Sqr}(X)$.

6. Вычислить при M , изменяющемся от 0 до 6 с шагом 0,5, и вывести в виде таблицы с заголовками:

- значения функции $(1 + X)^M$;
- приближенные значения функции по формуле

$$1 + M X + \frac{M(M-1)X^2}{2!} + \frac{M(M-1)(M-2)X^3}{3!} + \\ + \frac{M(M-1)(M-2)(M-3)X^4}{4!},$$

используя скобочные формы и/или дополнительные переменные;

- приближенные значения функции по этой же формуле, не используя скобочные формы и дополнительные переменные;

- абсолютную и относительную ошибки приближенных значений.

Для организации цикла использовать оператор **while**; при вычислениях приближенных значений — только операции сложения, вычитания, умножения, деления и стандартную функцию $\text{Sqr}(X)$.

7. Вычислить при $X = 1; 0,5; 0,25; 0,125; 0,0625; 0,03125; 0,015625$ и вывести в виде таблицы с заголовками:

- значения функции $\sqrt{1+X}$;
- приближенные значения функции по формуле

$$1 + \frac{X}{2} - \frac{X^2}{2 \cdot 4} + \frac{3X^3}{2 \cdot 4 \cdot 6} - \frac{3 \cdot 5X^4}{2 \cdot 4 \cdot 6 \cdot 8},$$

используя скобочные формы и/или дополнительные переменные;

- приближенные значения функции по этой же формуле, не используя скобочные формы и дополнительные переменные;
- абсолютную и относительную ошибки приближенных значений.

Для организации цикла использовать оператор **for to**; при вычислениях приближенных значений — только операции сложения, вычитания, умножения, деления и стандартную функцию $\text{Sqr}(X)$.

8. Вычислить при $X = \sin 5^\circ, \sin 10^\circ, \dots, \sin 60^\circ$ и вывести в виде таблицы с заголовками:

- значения функции $\arcsin(X)$;
- приближенные значения функции по формуле

$$X + \frac{X^3}{2 \cdot 3} - \frac{3X^5}{2 \cdot 4 \cdot 5} + \frac{3 \cdot 5X^7}{2 \cdot 4 \cdot 6 \cdot 7} - \frac{3 \cdot 5 \cdot 7X^9}{2 \cdot 4 \cdot 6 \cdot 8 \cdot 9},$$

используя скобочные формы и/или дополнительные переменные;

- приближенные значения функции по этой же формуле, не используя скобочные формы и дополнительные переменные;
- абсолютную и относительную ошибки приближенных значений.

Для организации цикла использовать оператор **for to**; при вычислениях приближенных значений — только операции сложения, вычитания, умножения, деления и стандартную функцию $\text{Sqr}(X)$.

9. Вычислить в цикле **repeat until** при первых 15 значениях

$$X = \operatorname{tg} \left(\frac{1}{2} 45^\circ \right), \operatorname{tg} \left(\frac{1}{3} 45^\circ \right), \operatorname{tg} \left(\frac{1}{4} 45^\circ \right), \dots$$

и вывести в виде таблицы с заголовками:

- значения функции $\operatorname{arctg}(X)$;
- приближенные значения функции по формуле

$$X - \frac{X^3}{3} + \frac{X^5}{5} - \frac{X^7}{7} + \frac{X^9}{9},$$

используя скобочные формы и/или дополнительные переменные;

- приближенные значения функции по этой же формуле, не используя скобочные формы и дополнительные переменные;
- абсолютную и относительную ошибки приближенных значений.

При вычислениях приближенных значений использовать только операции сложения, вычитания, умножения, деления и стандартную функцию $\operatorname{Sqr}(X)$.

10. Вычислить при X , изменяющемся от X_0 до X_1 с шагом H , и вывести в виде таблицы с заголовками:

- значения функции $\frac{e^X - e^{-X}}{2}$;
- приближенные значения функции по формуле

$$X + \frac{X^3}{3!} + \frac{X^5}{5!} + \frac{X^7}{7!} + \frac{X^9}{9!},$$

используя скобочные формы и/или дополнительные переменные;

- приближенные значения функции по этой же формуле, не используя скобочные формы и дополнительные переменные;
- абсолютную и относительную ошибки приближенных значений.

Для организации цикла использовать оператор **for downto**; при вычислениях приближенных значений — только операции сложения, вычитания, умножения, деления и стандартную функцию $\operatorname{Sqr}(X)$.

11. Для функции $Y = \frac{Xe^{-X^2}}{1+X}$ и вводимого значения X вычислить:

- точное значение производной $Y' = \frac{e^{-X^2} [(1-2X^2)(1+X) - X]}{(1+X)^2}$;

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных;

для 8-ми значений $DX = 0,2; 0,04; 0,008; \dots$

– приближенные значения приращений функции $DY = Y(X + DX) - Y(X)$;

– приближенные значения производной по отношению DY/DX ;

– абсолютные ошибки приближенных значений производной.

Для организации цикла использовать оператор **while**. Результаты вычислений и соответствующие значения DX вывести в виде таблицы с заголовками столбцов.

12. Для функции $Y = \frac{1+2X}{X^2-1}$ и вводимого значения X вычислить:

– точное значение производной $Y' = \frac{2(X^2-1) - 2X(1+2X)}{(X^2-1)^2}$;

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных;

для значений $DX = 0,0001; 0,001; 0,01; 0,1$:

– приближенные значения приращений функции $DY = Y(X + DX/2) - Y(X - DX/2)$;

– приближенные значения производной по отношению DY/DX ;

– абсолютные ошибки приближенных значений производной.

Для организации цикла использовать оператор **for to**. Результаты вычислений и соответствующие значения DX вывести в виде таблицы с заголовками столбцов.

13. Для функции $Y = \frac{\ln(1+\cos X)}{2^X+1}$ и вводимого значения X вычислить:

– точное значение производной

$$Y' = \frac{-\sin X}{(1+\cos X)(2^X+1)} - \frac{\ln(1+\cos X) \cdot 2^X \ln 2}{(2^X+1)^2}.$$

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных;

для значений $DX = 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}$:

– приближенные значения приращений функции $DY = Y(X + DX) - Y(X)$;

– приближенные значения производной по отношению DY/DX ;

– абсолютные ошибки приближенных значений производной.

Для организации цикла использовать оператор **for to**. Результаты вычислений и соответствующие значения DX вывести в виде таблицы с заголовками столбцов.

14. Для функции $Y = \frac{2^X}{1 + \ln [2 + \cos(X)]}$ и вводимого значения X вычислить:

– точное значение производной

$$Y' = \frac{2^X \ln 2 \cdot [1 + \ln (2 + \cos X)] + (2^X \sin X) / (2 + \cos X)}{(1 + \ln [2 + \cos X])^2};$$

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных;

в цикле **repeat until** для значений $DX = 0,00001; 0,0001; 0,001; 0,01; 0,1$:

– приближенные значения приращений функции $DY = Y(X + DX/2) - Y(X - DX/2)$;

– приближенные значения производной по отношению DY/DX ;

– абсолютные ошибки приближенных значений производной.

Полученные значения и соответствующие значения DX вывести в виде таблицы с заголовками столбцов.

15. Для функции $Y = \ln(1 + X^2) \operatorname{tg}(X^2)$ в точке $X = 0,3$ вычислить:

– точное значение производной

$$Y' = \frac{2X \operatorname{tg} X^2}{1 + X^2} + \frac{2X \ln(1 + X^2)}{\cos^2 X^2};$$

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных;

в цикле **for downto** для значений $DX = 0,00000025; 0,000005; 0,0001; 0,002; 0,04; 0,8$:

– приближенные значения приращений функции $DY = Y(X + DX) - Y(X)$;

– приближенные значения производной по отношению DY/DX ;

– абсолютные ошибки приближенных значений производной.

Полученные значения и соответствующие значения DX вывести в виде таблицы с заголовками столбцов.

16. Для функции $Y = \frac{\sin X}{\ln(2 + \sin^2 X)}$ и вводимого значения X вычислить:

– точное значение производной

$$Y' = \frac{\cos X \ln(2 + \sin^2 X) - (2 \sin^2 X \cos X) / (2 + \sin^2 X)}{\ln^2(2 + \sin^2 X)};$$

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных,

в цикле **while** для значений $DX = 0,0005; 0,001; 0,002; 0,004; 0,008; 0,016$:

– приближенные значения приращений функции $DY = Y(X + DX/2) - Y(X - DX/2)$;

– приближенные значения производной по отношению DY/DX ;

– абсолютные ошибки приближенных значений производной.

Полученные значения и соответствующие значения DX вывести в виде таблицы с заголовками столбцов.

17. Для функции $Y = \arctg \sqrt{\frac{X+1}{1-X}}$ и вводимого значения X вычислить:

– точное значение производной

$$Y' = \frac{1}{(1-X)^2 \left(1 + \frac{1+X}{1-X}\right) \sqrt{\frac{1+X}{1-X}}}:$$

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных;

в цикле **for to** для семи значений $DX = 0,000001; 0,000004; 0,000016; 0,000064; \dots$:

– приближенные значения приращений функции $DY = Y(X + DX) - Y(X)$;

– приближенные значения производной по отношению DY/DX ;

– абсолютные ошибки приближенных значений производной.

Полученные значения и соответствующие значения DX вывести в виде таблицы с заголовками столбцов.

18. Для функции $Y = \frac{X+1}{(X+2)(X+3)}$ и вводимого значения X вычислить:

– точное значение производной $Y' = \frac{1 - (X+2)X}{(X+2)^2(X+3)^2}$:

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных;

в цикле **for to** для 12 значений $DX = 1/3, 1/9, 1/27, 1/81, \dots$:

- приближенные значения приращений функции $DY = Y(X + DX) - Y(X)$;
- приближенные значения производной по отношению DY/DX ;
- абсолютные ошибки приближенных значений производной.

Полученные значения и соответствующие значения DX вывести в виде таблицы с заголовками столбцов.

19. Упростив вычисления за счет использования дополнительных переменных и/или скобочных форм, найти в цикле **repeat until** значения функции

$$Y(X) = \frac{\operatorname{tg}^4 X}{4} - \frac{\operatorname{tg}^2 X}{2} - \ln(\cos^2 X)$$

и ее производной

$$Y'(X) = \frac{\operatorname{tg}^3 X}{\cos^2 X} - \frac{\operatorname{tg} X}{\cos^2 X} - 2\operatorname{tg} X$$

на интервале от $-7,5^\circ \dots 7,5^\circ$ с шагом $0,75^\circ$. Для проверки правильности результата вычислить также значение производной по заданной формуле без преобразований. Найденные значения вывести в виде таблицы с предшествующими порядковым номером и соответствующим значением аргумента X .

20. Упростив вычисления за счет использования дополнительных переменных и/или скобочных форм, определить значения функции

$$Y(X) = \left[\frac{X-1}{2} - \frac{(X-1)^2}{2} + \frac{(X-1)^4}{3} \right] (X^2 - 1)$$

и ее производной

$$Y'(X) = \left[\frac{1}{2} - (X-1) + \frac{4(X-1)^3}{3} \right] (X^2 - 1) + \left[\frac{(X-1)}{2} - \frac{(1-X)^2}{2} + \frac{(1-X)^4}{3} \right] 2X$$

на интервале от $-1,1 \dots 1,0$ с шагом $0,1$. Для проверки правильности результата вычислить также значение производной по заданной формуле без преобразований. Вычисленные значения вывести в виде таблицы с предшествующими порядковым номером и соответствующим значением аргумента X . Для организации цикла использовать оператор **for downto**.

21. Для функции $F(X) = \frac{2X(1-X) + (1+X^2)}{2(1-X)^2} \sqrt{\frac{1-X}{1+X^2}}$ при $X =$

$= 0,5$ и K приращениях аргумента $DX = 0,0005; 0,001; 0,002; 0,004; 0,008; \dots$ вычислить:

- точное значение приращения первообразной

$$DP = \sqrt{\frac{1 + (X + DX)^2}{1 - (X + DX)}} - \sqrt{\frac{1 + X^2}{1 - X}};$$

- по формуле $F(X + DX/2) \cdot DX$ — приближенные значения приращения первообразной:
 - а) упростив вычисления за счет дополнительных переменных,
 - б) не используя дополнительных переменных;
- абсолютные и относительные ошибки в процентах для полученных приближенных значений.

Результаты вычислений и соответствующие значения DX вывести в виде таблицы с заголовками столбцов. Для организации цикла использовать оператор **while**.

22. Для функции $F(X) = \frac{e^X - e^{-X}}{e^X + e^{-X}}$ и вводимого значения X при N приращениях аргумента $DX = -0,1; -0,1/4; -0,1/16; \dots$ вычислить:

- точное значение приращения первообразной

$$DP = \ln(e^{X+DX} + e^{-(X+DX)}) - \ln(e^X + e^{-X});$$
- по формуле $F(X) \cdot DX$ — приближенные значения приращения первообразной:
 - а) упростив вычисления за счет дополнительных переменных,
 - б) не используя дополнительных переменных;
- абсолютные ошибки и относительные ошибки в процентах для вычисленных приближенных значений.

Результаты вычислений и соответствующие значения DX вывести в виде таблицы с заголовками столбцов. Для организации цикла использовать оператор **for to**.

23. Для функции $F(X) = \frac{\sin^3 X + 1}{\sin^2 X}$ и вводимого значения X при приращениях аргумента $DX = -0,0005; +0,001; -0,002; +0,004; -0,008; +0,016$ вычислить:

- точное значение приращения первообразной

$$DP = -\cos(X + DX) - \operatorname{ctg}(X + DX) + \cos X + \operatorname{ctg} X;$$
- по формуле $\frac{F(X + DX) + F(X)}{2} DX$ — приближенные значения приращения первообразной:
 - а) упростив вычисления за счет дополнительных переменных,

- б) не используя дополнительных переменных;
- абсолютные и относительные ошибки в процентах для полученных приближенных значений.

Результаты вычислений и соответствующие значения DX вывести в виде таблицы с заголовками столбцов. Для организации цикла использовать оператор **for to**.

24. Для функции $F(X) = \frac{-2Xe^{-X^2}}{1 + e^{-X^2}}$ и вводимого значения X при K приращениях аргумента $DX = -0,0005; -0,001; -0,002; -0,004; \dots$ вычислить в цикле **repeat until**:

- точное значение приращения первообразной:

$$DP = \ln[1 + e^{-(X+DX)^2}] - \ln(1 + e^{-X^2});$$

- по формуле $F(X + DX/2) \cdot DX$ — приближенные значения приращения первообразной:
 - а) упростив вычисления за счет дополнительных переменных,
 - б) не используя дополнительных переменных;
- абсолютные и относительные ошибки в процентах для найденных приближенных значений.

Результаты вычислений и соответствующие значения DX вывести в виде таблицы с заголовками столбцов.

25. Для функции $F(X) = \frac{2}{(2X-1)^2 \sqrt{1 - \left(\frac{1}{2X-1}\right)^2}}$ при $X = 1,5$ и K приращениях аргумента $DX = 5 \cdot 10^{-1}; 5 \cdot 10^{-2}; 5 \cdot 10^{-3}; 5 \cdot 10^{-4}; \dots$ вычислить:

- точное значение приращения первообразной

$$DP = \arccos \frac{1}{2(X+DX)-1} - \arccos \frac{1}{2X-1};$$

- по формуле $F(X) \cdot DX$ — приближенные значения приращения первообразной:
 - а) упростив вычисления за счет дополнительных переменных,
 - б) не используя дополнительных переменных;
- абсолютные и относительные ошибки в процентах для вычисленных приближенных значений.

Результаты вычислений и соответствующие значения DX вывести в виде таблицы с заголовками столбцов. Для организации цикла использовать оператор **for downto**.

26. Для функции $F(X) = -\frac{2X}{(X^2 - 1)^2 + 1}$ и вводимого значения X при K приращениях аргумента $DX = 0,1; -0,05; 0,025; -0,0125; \dots$ вычислить:

– точное значение приращения первообразной

$$DP = \arctg \frac{1}{(X + DX)^2 - 1} - \arctg \frac{1}{X^2 - 1};$$

– по формуле $\frac{F(X + DX) + F(X)}{2} DX$ — приближенные значения приращения первообразной:

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных;

– абсолютные и относительные ошибки в процентах для вычисленных приближенных значений.

Результаты вычислений и соответствующие значения DX вывести в виде таблицы с заголовками столбцов. Для организации цикла использовать оператор **while**.

27. Для функции $F(X) = \cos X - \frac{1}{\cos^2 X}$ и вводимого значения X при K приращениях аргумента $DX = 0,08; 0,04; 0,02; \dots$ вычислить:

– точное значение приращения первообразной

$$DP = [\sin(X + DX) - \tg(X + DX)] - (\sin X - \tg X);$$

– по формуле $F(X + DX/2) \cdot DX$ — приближенные значения приращения первообразной:

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных;

– абсолютные и относительные ошибки в процентах для вычисленных приближенных значений.

Результаты вычислений и соответствующие значения DX вывести в виде таблицы с заголовками столбцов. Для организации цикла использовать оператор **for to**.

28. Для функции $F(X) = \frac{-1}{(X - 1)^2 \sqrt{1 - \left(\frac{1}{X - 1}\right)^2}}$ при $X = 10$ и 12

приращениях аргумента $DX = 1/4, 1/6, 1/8, \dots$ вычислить:

– точное значение приращения первообразной

$$DP = \arcsin [1/(X + DX - 1)] - \arcsin [1/(X - 1)];$$

— по формуле $F(X) \cdot DX$ — приближенные значения приращения первообразной:

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных;

— абсолютные ошибки и в процентах относительные ошибки для вычисленных приближенных значений.

Результаты вычислений и соответствующие значения DX вывести в виде таблицы с заголовками столбцов. Для организации цикла использовать оператор **for to**.

29. Для функции $F(X) = \frac{X}{\sqrt{X^2 + 1}}$ при $X = 0,95$ и приращениях аргумента $DX = 0,0005; 0,001; 0,002; 0,04; 0,08; 0,016; 0,032$ вычислить в цикле **repeat until**:

— точное значение приращения первообразной,

$$DP = \sqrt{(X + DX)^2 + 1} - \sqrt{X^2 + 1};$$

— по формуле $\frac{F(X + DX) + F(X)}{2} DX$ — приближенные значения приращения первообразной:

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных;

— абсолютные и относительные ошибки в процентах для вычисленных приближенных значений.

Результаты вычислений и соответствующие значения DX вывести в виде таблицы с заголовками столбцов.

30. Упростив вычисления за счет использования дополнительных переменных и/или скобочных форм, определить значения функции

$$Y(X) = \frac{a^{X^2-1} + a^{X-1}}{X-1}$$

и ее производной

$$Y'(X) = \frac{(2Xa^{X^2-1} + a^{X-1}) \ln(a)(X-1) - (a^{X^2-1} + a^{X-1})}{(X-1)^2}$$

на 20 значениях.

Для проверки правильности вычислений Y' найти также ее значение по заданной формуле без преобразований. Вычисленные значения вывести с предшествующими порядковыми номерами и соответствующим значением аргумента X в виде таблицы с заголовками столбцов. Для организации цикла использовать оператор **for downto**.

4.6. Сохранение результатов вычислений в массиве

Массивами называют структурные переменные, представляющие набор однотипных элементов — *элементов массива*. Доступ к элементам массива, называемым *индексными переменными*, осуществляется по имени массива и набору индексов, однозначно определяющих положение элемента в массиве. Если элементы массива сами не являются массивами, то такой массив называют *одномерным* и для обращения к элементам используют только один индекс, иначе его называют многомерным (двумерным, трехмерным и т. д.).

В качестве индексов могут выступать выражения (в частности, константы и переменные) *ординальных (порядковых) типов* (в том числе булевых и целых), которые записывают после имени массива в квадратных скобках через запятую. В этом разделе ограничимся рассмотрением одномерных *статических массивов* с числовыми индексами.

В общем случае *описатель типа статического массива*, создаваемого пользователем, определяется следующей синтаксической диаграммой (рис. 4.5).

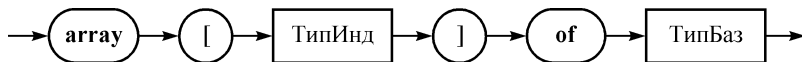


Рис. 4.5

На рис. 4.5 ТипИнд — тип индекса (имя стандартного или ранее объявленного в программе ординального типа или диапазон ординального типа, задающие диапазон значений индексов); ТипБаз — базовый тип (имя стандартного или ранее объявленного типа или описатель типа, задающие тип и допустимые значения элементов массива).

Для одномерных числовых массивов ТипБаз должен представлять простой тип.

Имена, представляющие типы-массивы, объявляют, как и имена прочих пользовательских типов, в разделе **type**. Например,

type

```

tX=array[1..9] of Byte;
tY=array[-2..2] of Real;

```

Здесь тип tX объявлен как тип целочисленного одномерного массива из 9 элементов, значениями которых могут быть числа из диапазона 0...255, а индексами — выражения целого типа со

значениями от 1 до 9; тип `tY` объявлен как тип одномерного массива вещественных данных (типа `Real`) из пяти элементов, индексами которых могут быть выражения целого типа со значениями -2; -1; 0; 1; 2.

Переменные-массивы объявляют, как и переменные прочих типов, в разделе **var**. Им можно задавать начальные значения (что удобно при отладке программ), записав их списком в круглых скобках через запятую, причем длина списка должна быть равна размеру массива. Например, в объявлениях

```
var
  X, Z: tX;
  Y: tY = (1.2, 4.2, -5.1, 4.4, -1.5);
```

переменные `X` и `Z` будут иметь тип `tX`, а переменная `Y` — тип `tY` с начальными значениями своих элементов $Y[-2]=1.2$, $Y[-1]=4.2$, $Y[0]=-5.1$, $Y[1]=4.4$, $Y[2]=-1.5$.

При решении задач, связанных с обработкой множеств значений в массивах или получаемых в результате вычисления значений функций, может потребоваться сохранение результатов вычислений для дальнейшего использования. В таких случаях необходимо выделить память для хранения результатов, объявив в разделе **var** массив соответствующего типа, и очередное вычисленное значение сохранять в соответствующей ячейке массива, присваивая его индексной переменной — элементу этого массива с очередным значением индекса.

Объявление статических массивов приводит к резервированию области памяти для хранения значений элементов, размеры которой нельзя изменить во время работы программы. Поэтому для обеспечения массовости алгоритма программы необходимо объявлять массивы с максимально возможными размерами исходя из условий применения программы, обычно формулируемыми в задании на разработку.

Свойство *массовость* алгоритма предполагает его применимость к различным, заранее оговоренным, наборам данных, в частности задаваемым при вводе. В рассмотренном выше примере объявления массива `X`, если он будет представлять исходные данные для какого-либо алгоритма, свойство «массовость» обозначает, что могут обрабатываться любые наборы от одного до девяти чисел с любыми значениями от 0 до 255. Например, все девять элементов массива или только N первых, $N < 9$.

В дальнейшем в заданиях на обработку массивов будут использоваться следующие сокращенные обозначения:

$X(20)$ — будет обозначать, что для хранения данных должен использоваться одномерный массив, в котором подлежат обработке 20 последовательно расположенных элементов;

$X(N)$, $N \leq 20$ — будет обозначать, что для хранения данных должен использоваться одномерный массив, в котором подлежат обработке N последовательно расположенных элементов. Иногда при постановке задач удобно использовать слово *вектор* или *последовательность*, имея в виду размещаемые в последовательных ячейках массива данные.

Пример. Составить программу вычисления и сохранения в массиве Y значений функции $y = \sin x$ и в массиве X — соответствующих значений аргумента. Аргумент должен изменяться с шагом dX от начального значения X_0 :

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
type
  tMas=array[1..10] of Real;
var
  i:Integer;
  X0, dX:Real;
  X, Y:tMas;
begin
  Write('Введите начальное значение X и шаг dX: ');
  ReadLn(X0, dX);
  for i:=1 to 10 do
    begin
      X[i]:=X0;
      Y[i]:=Sin(X0);
      X0:=X0+dX;
    end;
  for i:=1 to 10 do
    WriteLn(X[i]:10:4,Y[i]:10:4);
  ReadLn;
end.
```

4.7. Пример выполнения задания

Составить программу для подсчета и сохранения в массиве $M(10)$ количеств значений целочисленного массива $X(N)$, $N \leq 500$, попадающих в интервалы с номерами от 1 до 10 шириной

$h = (X_{\max} - X_{\min} + 1)/10$, где $X_{\max} = 50$ и $X_{\min} = 1$ — соответственно максимальное и минимальное значения в массиве X . Массив X заполнить случайными числами с равномерным распределением в диапазоне $X_{\min}...X_{\max}$, используя стандартную функцию `Random`. Использовать также процедуру `Randomize`, чтобы генерировать новый набор случайных чисел при каждом запуске программы. Объявить X_{\min} , X_{\max} и h в разделе констант. Полученные результаты использовать для вывода в виде гистограммы (рис. 4.6).

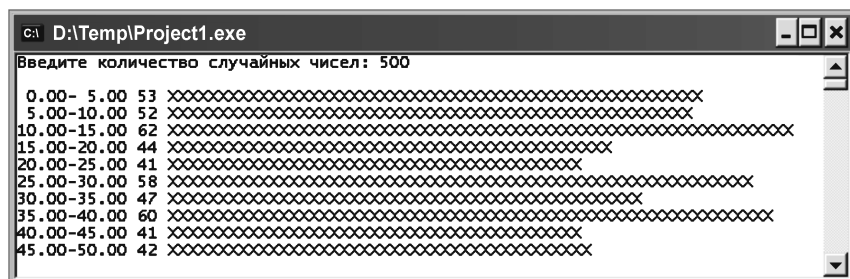


Рис. 4.6

Вычислить также для помещенных в массив X случайных чисел среднее значение MX и дисперсию DX по формулам

$$MX = \frac{\sum_{i=1}^N X_i}{N} \quad \text{и} \quad DX = \frac{\sum_{i=1}^N (X_i - MX)^2}{N}:$$

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
const
  Nmax=500;
  Xmax=50;
  Xmin=1;
  h=5;
  NX=Xmax-Xmin+1;
var
  X:array[1..Nmax] of Word;
  M:array[1..10] of Word;
```

```
i, j, N: Word;
MX, DX: Real;
begin
  Write('Введите количество случайных чисел: ');
  ReadLn(N);
  Randomize;
  for i:=1 to N do
    //Сохраниение случайных чисел в массиве X
    X[i]:=Random(NX)+ Xmin;
  //Накопление в ячейках 1..10 массива M количеств
  //попаданий значений из массива X в интервалы
  //[1..5], [6..10], ..., [46..50]
  for i:=1 to N do
    //Увеличить на 1 значение элемента массива M
    //с индексом (X[i]-Xmin) div h + 1
    Inc(M[(X[i]-Xmin) div h + 1]);
  //Вывод результатов в виде гистограммы
  WriteLn;
  for i:=1 to 10 do
    begin
      Write((i-1)*h+1:3, ' -', i*h:3, M[i]:3, ' ');
      for j:=1 to M[i] do
        Write('X');
      WriteLn;
    end;
  //Вычисление среднего значения
  MX:=0;
  for i:=1 to N do
    MX:=MX+X[i];
  MX:=MX/N;
  //Вычисление дисперсии
  DX:=0;
  for i:=1 to N do
    DX:=DX+Sqr(X[i]-MX);
  DX:=DX/N;
  WriteLn('Среднее значение = ', MX);
  WriteLn('          Дисперсия = ', DX);
  ReadLn;
end.
```

4.8. Задания для самостоятельной работы

Во всех заданиях использовать только простые циклы.

1. В массиве $M(5)$ хранятся в порядке возрастания значения 1, 5, 10, 50, 100. Требуется найти для положительного целого числа N и сохранить в массиве $K(5)$ коэффициенты разложения $N = K_1 \cdot M_1 + K_2 \cdot M_2 + K_3 \cdot M_3 + K_4 \cdot M_4 + K_5 \cdot M_5$, при котором сумма $\sum_{i=1}^5 K_i$ будет минимальна (использовать операции **mod** и **div**).

2. В целочисленном массиве $M(N)$, $N \leq 20$ содержатся разные числа от 1 до k ($k < N$), а в массиве $S(k)$ — не повторяющиеся числа от 1 до k в произвольном порядке. Требуется зашифровать данные массива M следующим образом: новым значением элемента массива M будет значение элемента массива S , индекс которого равен значению этого элемента массива M . Расшифровать i -е значение массива M и присвоить результат переменной P .

3. Выполнить циклический сдвиг элементов массива $X(N)$ ($N \leq 10$), в результате которого значение последнего элемента должно оказаться на месте первого, а остальные — сдвинуться на одну позицию в сторону увеличения индекса.

4. На заданном отрезке вычислить с заданным шагом изменения аргумента и поместить в массив F 30 значений функции $e^{-x} \sin(6x)$, деленные на ее последнее положительное значение.

5. S является последовательностью нулей и единиц длиной $L \leq 30$. Требуется сохранить в массиве Y информацию, представленную S , в виде $Y_0 = S_1$, а также числа, представляющие длины локальных подпоследовательностей с одинаковыми значениями. Подсчитать количество записанных в массив Y чисел.

6. Восстановить последовательность S (см. п. 5 задания) по данным из массива Y и количеству записанных в массив Y чисел.

7. Последовательность S из нулей и единиц длиной $L < 30$ зашифровать и поместить в массив D . Шифровать по следующему правилу: положить $D_1 = S_1$, потом $D_i = 1$, если $S_i = S_{i-1}$, иначе — 0. Затем по данным из D расшифровать последовательность и поместить в массив R .

8. В массиве $X(4)$ хранятся в порядке возрастания значений положительные вещественные числа. Требуется найти и сохранить в целочисленном массиве $K(4)$ коэффициенты разложения переменной R : $R = D + K_1 \cdot X_1 + K_2 \cdot X_2 + K_3 \cdot X_3 + K_4 \cdot X_4$, где $D < X_1$, в котором сумма $\sum_{i=1}^4 K_i$ будет минимальна.

9. Из массива $X(N)$, $N \leq 20$, упорядоченного по невозрастанию значений элементов, переписать в массив Y без повторов значения элементов с четными индексами, меньшие C , сохранив упорядоченность.

10. Изменяя X от заданного начального значения с заданным шагом H , вычислить и поместить в массив F 20 значений разности функции и ее значения в точке первого локального минимума.

11. В массиве $V(10)$, заданном начальными значениями, содержатся числа от 0 до 9 в произвольном порядке. Требуется поместить в массив D зашифрованную произвольную последовательность S длиной $L \leq 30$ из целых чисел от 0 до 9. Шифрование выполнить по следующему правилу: $D_i = i - V_{S_i}$. По данным из D расшифровать k -ю цифру и поместить в R .

12. Найти и сохранить в массиве N коэффициенты $n_0, n_1, n_2, n_3, n_4, n_5$ разложения целого числа K ($0 < K < 10^6$) по степеням числа 10.

13. Выполнить циклический сдвиг элементов массива $X(N)$, $N \leq 20$, на K позиций, в результате которого последние K элементов займут место в начале массива, а остальные будут сдвинуты на K позиций в сторону увеличения индекса. Использовать дополнительный массив D .

14. На заданном отрезке, с заданным шагом изменения аргумента вычислить и поместить в массив $X(20)$ значения аргумента функции $e^{-x} \sin(3x) - 0,2$, предшествующие изменению ее знака, и подсчитать количество. Вычисления проводить либо до достижения границы интервала, либо до заполнения массива.

15. В массив $X(N)$, $N \leq 20$, упорядоченный по возрастанию значений элементов, добавить новое число так, чтобы не нарушить упорядоченность.

16. S является последовательностью из чисел 1, 2, 3 и 4 длиной $L \leq 20$. Требуется сохранить в массивах K и N информацию, представленную S в виде K_i — число из i -й подпоследовательности из одинаковых чисел в S , N_i — длина этой подпоследовательности, а также количество записанных в массивы K и N чисел.

17. Из массива X , упорядоченного по невозрастанию значений элементов, переписать в массив Y числа, исключив их повторы и обеспечив упорядоченность по возрастанию.

18. Поместить положительные элементы массива X в начало массива Y , а следом — его отрицательные элементы.

19. Из целочисленного массива $X(N)$, $N \leq 20$, удалить числа, кратные K , поместив остальные в его начало без пропусков и не изменив их взаимного расположения. Вывести количество оставленных в массиве чисел и сами числа.

20. Найти и сохранить в массиве $K(N)$, $N \leq 14$, старшие N цифр правильной дроби R при представлении ее в десятичной системе счисления, а в переменной D — часть числа R , меньшую 10^{-N} . Использовать стандартные функции Frac и Int .

21. На заданном отрезке с заданным шагом изменения аргумента вычислить и поместить в массив $X(50)$ значения аргумента функции $e^{-3x} \sin^2 20x$, предшествующие первому локальному экстремуму функции типа максимум, а в массив Y — соответствующие значения функции. Если за 50 шагов экстремум не будет найден, то вывести соответствующее сообщение, иначе вывести помещенные в массивы X и Y значения в виде таблицы.

22. Из массива $X(20)$, упорядоченного по неубыванию значений элементов, переписать в массив Y числа, исключив их повторы и добавив новое вводимое значение P так, чтобы не нарушить упорядоченность.

23. На заданном отрезке с заданным шагом изменения аргумента вычислить и поместить в массив $X(12)$ значения аргумента функции $e^{-x/3} \sin^2 5x$, непосредственно предшествующие локальным максимальным приращениям функции. Если до достижения верхней границы интервала массив окажется заполненным, то вычисления прекратить и сопроводить вывод результатов соответствующим сообщением.

24. Поместить элементы массива X в начало массива Y в обратном порядке, исключив превосходящие по абсолютной величине вводимое значение R .

25. В массиве K с индексами от 0 до 9, заданном начальными значениями, содержатся разные числа (от 0 до 9) в произвольном порядке. Требуется поместить в массив Y зашифрованную произвольную последовательность X длиной $L \leq 30$ из целых чисел от 0 до 9. Шифрование выполнить по следующему правилу: $Y_i = i - K_{X_i}$. Затем по данным из Y расшифровать последовательность и поместить в массив P . Использовать дополнительный массив T с начальными значениями, заданными следующим образом: T_i равно номеру ячейки массива K со значением i .

4.9. Приемы вычисления сумм, произведений и экстремальных значений

Вычисление суммы и произведения

Прием *накопления суммы* часто используется в разных приложениях. Суммируемыми могут быть элементы массива или вычисляемые значения функции. В любом случае накопление суммы производится в цикле по рекуррентной формуле $S = S + Y$, где S — промежуточная сумма, Y — слагаемое. Перед циклом в большинстве случаев начальное значение S должно быть равно нулю.

Фрагмент программы для вычисления суммы n элементов массива Y будет иметь вид

```
S:=0;
for i:=1 to n do
  S:= S+Y[i];
```

Фрагмент программы для вычисления среднего первых n значений функции $Y(x)$, вычисляемых при значениях x от x_0 с шагом dx , будет иметь вид

```
S:=0;
x:=x0;
for i:=1 to n do
begin
  S:=S+Y(x)
  x:=x+dx
end;
S:=S/n;
```

Произведение вычисляется аналогичным образом, но перед циклом в большинстве случаев начальное значение $P:=1$, а в цикле — накапливается по рекуррентной формуле $P:=P*Y$.

Пример. Вычислить значение суммы элементов массива $X(n)$, $n \leq 30$:

```
const
  nMax=30;
var S:Real;
    X: array [1..nMax] of Real;
    i:Integer;
begin
  Write(' Введите количество элементов: ');
  ReadLn (n);
```

```
WriteLn(' Введите элементы массива: ');
for i:=1 to n do
  Read (X[i]);
ReadLn;
S:=0; //Начальное значение суммы
for i:=1 to n do
  S:= S+X[i]; //Накопление суммы
WriteLn('Сумма элементов массива равна ', S:6:2);
. . . . .
end.
```

Нахождение наибольшего или наименьшего значения

Поиск наибольшего или наименьшего значения может выполняться в имеющемся массиве или при вычислении значения функции.

При поиске в массиве начальному значению наибольшего X_{\max} (начальному значению наименьшего X_{\min}) присваивается значение первого элемента массива, а затем в цикле для очередного элемента массива X_i , $i = 2, 3, \dots, n$, производится проверка: если $X_i > X_{\max}$ ($X_i < X_{\min}$), то переменной X_{\max} (переменной X_{\min}) присваивается значение X_i , иначе значение X_{\max} (X_{\min}) не изменяется.

Поиск максимального и минимального значений в массиве представляют следующие фрагменты программ:

```
//Поиск максимального
Xmax= X[1];
for i:=2 to n do
  if X[i] > Xmax then
    Xmax:=X[i];
WriteLn('Xmax=', Xmax:6:2);

//Поиск минимального
Xmin = X[1];
for i:=2 to n do
  if X[i] < Xmin then
    Xmin:=X[i];
WriteLn('Xmin=', Xmin:6:2);
```

При поиске наибольшего Y_{\max} (наименьшего Y_{\min}) из вычисляемых значений функции $F(x)$ переменной Y_{\max} (Y_{\min}), которая после вычислений будет представлять максимальное (минимальное) значение функции, до входа в цикл следует присвоить одно из вычисляемых значений $F(x)$, а в цикле для каждого

очередного значения функции $F(x)$ проверять: если $F(x) > Y_{\max}$ ($F(x) < Y_{\min}$), то Y_{\max} (Y_{\min}) присвоить это значение функции.

Ниже представлены фрагменты программ поиска максимального и минимального значений функции $y = e^{-x} \sin 5x$, вычисляемых при x , изменяющихся от a до b с шагом dx .

```
//Поиск максимального значения функции
```

```
Ymax:=Exp(-A)*Sin(5*A);
```

```
X:=A+dx; Xk:=B+dx/2;
```

```
while X < Xk do
```

```
begin
```

```
    Y:= Exp(-X)*Sin(5*X);
```

```
    if Y > Ymax then
```

```
        Ymax:=Y;
```

```
end;
```

```
WriteLn('Ymax = ',Ymax);
```

```
//Поиск минимального значения функции
```

```
Ymin:=Exp(-A)*Sin(5*A);
```

```
X:=A+dx; Xk:=B+dx/2;
```

```
while X < Xk do
```

```
begin
```

```
    Y:= Exp(-X)*Sin(5*X);
```

```
    if Y < Ymin then
```

```
        Ymin:=Y;
```

```
end;
```

```
WriteLn('Ymin = ',Ymin);
```

Как видно из представленных фрагментов программ, их алгоритмы можно получить один из другого заменой знака отношения, например, больше на меньше.

Если заранее известно, что максимальное (минимальное) значение функции на заданном интервале больше (меньше) некоторой величины, например, -10^{15} (10^{15}), то эту величину можно использовать в качестве начального значения при поиске максимального (минимального) значения функции, например, $Y_{\max} := -1E15$ ($Y_{\min} := 1E15$).

Наряду с нахождением максимального/минимального значения функции (максимального/минимального значения в массиве) может потребоваться определение значения аргумента (индекса элемента массива), при котором оно достигается. Нахождение указанных значений при поиске максимума поясняют следующие фрагменты программ:


```
//Поиск максимального значения функции
//и соответствующего значения аргумента
Ymax:=Exp(-A)*Sin(5*A);Xmax:=A;
X:=A+dx; Xk:=B+dx/2;
while X < Xk do
begin
  Y:= Exp(-X)*Sin(5*X);
  if Y > Ymax then
    begin
      Ymax:=Y;
      Xmax:=X;
    end;
end
WriteLn('Ymax  = ',Ymax);
WriteLn('Xmax  = ',Xmax);

//Поиск максимального элемента
//массива и его индекса
Xmin = X[1]; Imin:=1;
for i:=2 to n do
  if X[i] > Xmax then
    begin
      Xmax:=X[i];
      Imax:=i;
    end;
WriteLn('Xmax = ', Xmax:6:2);
WriteLn('Imax = ', Imax);
```

Рассмотренный выше прием определения на заданном интервале максимума/минимума функции и точек (значений аргумента), на которых они достигаются, не следует путать с поиском экстремумов функции, так как искомый максимум/минимум может находиться на границе интервала, где производная функции не равна нулю.

Поиск приближенных значений точек *экстремумов* функции $F(X)$ и ее значения в этих точках, а также точек перегибов (экстремумов $F'(X)$), точек экстремальной кривизны (экстремума $|F''(X)|/\{1+[F'(X)]^2\}^{3/2}$) могут рассматриваться как самостоятельные задачи. Их следует решать рассмотренными методами при вычисляемых по известным аналитическим выражениям значениям производных. Однако поскольку аналитические выражения производных не всегда известны, приходится вычислять значения производных по разностным схемам или использовать более простые приемы.

Например, приближенное значение точки экстремума-максимума (локального максимума) будет найдено, если на очередном значении аргумента, изменяемом с заданным шагом, окажется, что слева и справа от него функция имеет меньшие значения. Проверку можно упростить: если заранее известно, что в начале исследуемого интервала функция растет, то точкой экстремума будет значение аргумента, после которого значение функции начнет уменьшаться.

4.10. Пример выполнения задания А

Составить программу вычисления среднего значения элементов целочисленного массива $X(N)$, $N \leq 15$, кратных K , и поместить округленное найденное среднее на место максимального положительного, меньшего M . Прекратить выполнение программы и вывести соответствующее сообщение, если в массиве не найдется элементов, кратных K , или элементов, меньших M . Вывести с пояснениями найденное среднее значение, максимальное значение и преобразованный массив.

```
program Project1;
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
const
  Nmax=15;
type
  tX=array[1..Nmax] of Integer;
var
  //Начальные значения переменных можно
  //использовать при отладке программы.
  X:tX=(11,21,36, 5,84
        ,99,51, 3,44,22
        ,55,77,12,23,45);
  N:Integer=9;
  K:Integer=3;
  M:Integer=30;
  S:Real;
  i,Xmax,iMax,L:Integer;
begin
  //ВВОД ИСХОДНЫХ ДАННЫХ
  Write('Введите количество элементов массива: ');
```

```
ReadLn(N);
WriteLn('Введите элементы массива');
for i:=1 to N do
    Read(X[i]);
ReadLn;
Write('Введите коэффициент кратности: ');
ReadLn(K);
//ВЫЧИСЛЕНИЕ СРЕДНЕГО ЗНАЧЕНИЯ
//Начальное значение для поиска суммы
S:=0;//элементов массива, кратных K.
//Начальное значение счётчика элементов
L:=0;//массива, включенных в сумму.
for i:=1 to N do
    if X[i] mod K = 0 then
        begin
            S:=S+X[i];
            L:=L+1;
        end;
//ПРОВЕРКА КОЛИЧЕСТВА ЭЛЕМЕНТОВ, ВКЛЮЧЕННЫХ В S
if L=0 then
    begin
        Write('в массиве нет элементов, кратных ',K);
        ReadLn;
        Halt //Завершение работы программы
    end;
//ВЫЧИСЛЕНИЕ И ВЫВОД СРЕДНЕГО ЗНАЧЕНИЯ
S:=S/L;//Найденное среднее значение
WriteLn(S:6:1,' - среднее элементов массива, ',
        ', кратных ',K);
//ПОИСК МАКСИМАЛЬНОГО И ЕГО ИНДЕКСА
Write('Введите верхнюю грань ',
        ', искомого максимального: ');
ReadLn(M);
Xmax:=0;
for i:=1 to N do
    if (X[i] < M) and (X[i]>Xmax) then
        begin
            Xmax:=X[i];
            iMax:=i;
        end;
//ПРОВЕРКА, НАЙДЕН ЛИ МАКСИМАЛЬНЫЙ ЭЛЕМЕНТ
if Xmax=0 then
```

```
begin
  Write('в массиве нет положительных элементов, '
        , 'меньших ', M);
  ReadLn;
  Halt //Завершение работы программы
end;
//ЗАМЕНА В МАССИВЕ МАКСИМАЛЬНОГО СРЕДНИМ
X[iMax]:=Round(S);
//ВЫВОД РЕЗУЛЬТАТОВ
WriteLn(Xmax:6, ' - максимальное, меньшее ', M);
WriteLn(iMax:6, ' - индекс максимального');
WriteLn;
WriteLn('Преобразованный массив');
for i:=1 to N do
  Write(X[i]:4);
ReadLn;
end.
```

4.11. Задания А для самостоятельной работы

Выполнение программы прекратить и вывести соответствующие сообщения, если в массиве не будут найдены требуемые значения.

1. Вычислить среднее арифметическое, наименьшее значение среди положительных элементов и произведение отрицательных в массиве $D(n)$, $n \leq 25$. Вывести массив, среднее арифметическое, наименьшее значение и произведение.

2. Найти наибольшее и наименьшее значения, их индексы и среднее арифметическое элементов, расположенных между ними в массиве $D(n)$, $n \leq 25$. Вывести массив, среднее арифметическое, наименьшее и наибольшее значения и их индексы.

3. Найти наибольшее и наименьшее значения и их индексы в массиве $D(n)$, $n \leq 25$. Если индекс наименьшего значения меньше индекса наибольшего, то вычислить сумму элементов, в противном случае — произведение. Вывести массив, наименьшее и наибольшее значения и их индексы, сумму или произведение.

4. Вычислить среднее арифметическое значение элементов, удовлетворяющих условию $a \leq D_i < b$, найти наименьшее значение среди положительных элементов и его индекс в массиве $D(n)$, $n \leq 25$. Вывести массив, среднее арифметическое и наименьшее значение и его индекс.

5. Вычислить сумму положительных элементов до первого отрицательного и произведение отрицательных до первого положительного элемента в массиве $D(n)$, $n \leq 25$. Вывести массив, сумму и произведение.

6. Из массива $D(n)$, $n \leq 25$ переписать элементы подряд в массив P , расположив вначале положительные, а затем отрицательные. Определить, в каком из массивов наименьший элемент встретился первым. Вывести массивы, наименьшие значения и их индексы.

7. Вычислить среднее арифметическое положительных элементов, кратных двум, и произведение отрицательных элементов, кратных трем, в массиве $D(n)$, $n \leq 25$. Вывести массив, среднее арифметическое и произведения.

8. Вычислить количество положительных и отрицательных элементов массива $D(n)$, $n \leq 25$. Если количество положительных элементов больше, то вычислить их среднее арифметическое, в противном случае вычислить их произведение. Вывести массив, количество положительных и отрицательных чисел, среднее арифметическое или произведение.

9. Найти наибольшее и наименьшее значения и их индексы в массиве $D(n)$, $n \leq 25$. Первый элемент массива заменить наименьшим, а последний — наибольшим значением в массиве. Вывести массив, наименьшее и наибольшее значения, их индексы.

10. Найти наибольшее и наименьшее значения и их индексы в массиве $D(n)$, $n \leq 25$. Вычислить их среднее значение и произведение элементов, значения которых превышают среднее. Вывести массив, наименьшее, наибольшее значения и их индексы, произведение.

11. Вычислить среднее арифметическое положительных элементов с четными индексами и произведение отрицательных с нечетными индексами в массиве $D(n)$, $n \leq 25$. Вывести массив, среднее арифметическое и произведение.

12. В массивах $X(n)$ и $Y(n)$, $n \leq 25$ вычислить произведения соответствующих элементов $X_i Y_i$ и найти наибольшее и наименьшее значения. Вывести массивы, произведения, наибольшее и наименьшее значения.

13. В массивах $X(n)$ и $Y(n)$, $n \leq 25$ вычислить средние значения соответствующих элементов $(X_i + Y_i)/2$ и найти среди них наибольшее и наименьшее. Вывести массивы, средние значения, наибольшее и наименьшее значения.

14. Вычислить отношение $C = A/B$, где A — произведение положительных элементов с четными индексами, а B — сумма элементов по абсолютному значению с нечетными индексами массива $D(n)$, $n \leq 25$. Вывести массив, произведение, сумму и их отношение.

15. Вычислить разность $C = A - B$, где A — произведение положительных элементов, а B — сумма элементов по абсолютному значению массива $D(n)$, $n \leq 25$. Вывести массив, произведение, сумму и их разность.

16. Найти наибольшее и наименьшее значения произведений соседних элементов $X_i Y_{i+1}$ (для последнего элемента $X_n Y_1$) в массиве $X(n)$, $n \leq 25$. Вывести массив, наименьшее и наибольшее значения произведений.

17. В массивах $X(n)$ и $Y(n)$, $n \leq 25$ вычислить произведения пар элементов $X_i Y_i > A$ и найти среди них наибольшее и наименьшее значения. Вывести массивы, произведения и наибольшее и наименьшее значения.

18. Вычислить среднее арифметическое положительных элементов и произведение элементов, превышающих среднее арифметическое по абсолютному значению в массиве $D(n)$, $n \leq 25$. Вывести массив, среднее арифметическое и произведение.

19. Для массивов $X(n)$ и $Y(n)$, $n \leq 25$ вычислить модуль разности соответствующих элементов массивов $|X_i - Y_i|$ и найти среди них наибольшее и наименьшее значения. Вывести массивы, разности, наибольшее и наименьшее значения.

20. Вычислить среднее геометрическое положительных элементов, кратных двум, и сумму отрицательных с нечетными индексами в массиве $D(n)$, $n \leq 25$. Вывести массив, среднее геометрическое и сумму.

21. Суммы стоящих рядом элементов массива $X(n)$, $n \leq 25$ $X_i + X_{i+1}$ (для последнего элемента $X_n + X_1$) записать в массив $Y(n)$, $n \leq 25$ и найти наибольшее и наименьшее значения в массивах $X(n)$ и $Y(n)$. Вывести массивы, наибольшее и наименьшее значения.

22. Из массива $D(n)$, $n \leq 25$ переписать числа в массив P , расположив подряд вначале отрицательные, а затем положительные. Определить, в каком из массивов наименьший по модулю элемент встретится первым. Вывести массивы, наименьшие значения и их индексы.

23. Вычислить сумму, количество положительных элементов, которые превышают B , и произведение элементов по абсолютно-

му значению массива $D(n)$, $n \leq 25$. Вывести массив, произведение, сумму и количество.

24. В массивах $X(n)$ и $Y(n)$ $n \leq 25$ вычислить количество равенств соответствующих элементов X_i и Y_i и найти пары элементов, имеющих в сумме наибольшее и наименьшее значения. Вывести массивы, количество равенств, наибольшее и наименьшее значения.

25. В массиве $D(n)$, $n \leq 25$ вычислить среднее арифметическое положительных элементов до первого отрицательного и найти среди них количество элементов, превышающих среднее арифметическое. Вывести массив, среднее арифметическое и количество элементов.

26. В массиве $D(n)$, $n \leq 25$ вычислить произведение положительных элементов до первого отрицательного и найти среди них наибольшее и наименьшее значения. Вывести массив, произведение, наибольшее и наименьшее значения.

4.12. Пример выполнения задания Б

Известно, что функция $y = \frac{-\sin(5x)x^2 \sin x}{x-3}$ на интервале значений аргумента $-1,6 \dots 2,0$ имеет несколько точек перегиба, в которых значение хотя бы одной производной больше $-1 \cdot 10^{30}$. Требуется найти точку перегиба с максимальным значением производной, изменяя аргумент с шагом 0,001. Вывести найденные приближенные значения производной и соответствующей точки перегиба:

```
program Project2;
{$APPTYPE CONSOLE}
uses
  SysUtils;
const
  dX=0.001;
var
  Xmax, dYmax, X, dYL, dY, dYR:Extended;
begin
  X:=-1.6;
  //Приращение функции в точке слева от текущей
  dYL:= Sin(5*X) *Sin(X) *Sqr(X) / (X-3)
    -Sin(5*(X+dX)) *Sin(X+dX) *Sqr(X+dX) / (X+dX-3);
  X:=X+dX;
```

```

//Приращение функции в текущей точке
dY:= Sin(5*X)*Sin(X)*Sqr(X)/(X-3)
      -Sin(5*(X+dX))*Sin(X+dX)*Sqr(X+dX)/(X+dX-3);
Xmax:=X; dYmax:=-1e30;
repeat
  X:=X+dX;
  //Приращение функции в точке справа от текущей
  dYR:= Sin(5*X)*Sin(X)*Sqr(X)/(X-3)
        -Sin(5*(X+dX))*Sin(X+dX)*Sqr(X+dX)/(X+dX-3);
  //Если X - точка перегиба
  if (dY>dYL) and (dY>dYR) or
     (dY<dYL) and (dY<dYR) then
    if dY>dYmax then //и dY в ней больше dYmax,
      begin //то текущей точкой перегиба
        //с максимальной производной станет
        Xmax:=X-dX; //текущее значение X, а текущим
          //максимальным приращением
        dYmax:=dY; //в точке перегиба - значение dY.
      end;
    dYL:=dY;
    dY:=dYR;
  until X>1.9995;
WriteLn(Xmax, ' - точка перегиба, '
        , 'в которой достигается', #13#10
        , dYmax/dX, ' - максимум производной');
ReadLn;
end.

```

4.13. Задания Б для самостоятельной работы

Во всех заданиях не использовать аналитических формул производных заданных функций. Ввод исходных данных выполнить с применением переменных $X_{нач}$, $X_{кон}$, h_X , используя числовые значения вариантов заданий. Найденные значения выводить с поясняющими текстами.

1. Составить программу вычисления максимального и минимального значений функции $Y = X^3 - 18X^2 - 10X + 7$ и соответствующих значений аргумента при его изменении от -4 до 16 с шагом $0,01$.

2. Составить программу вычисления значения аргумента, при котором функция $X \sin^5(3X)$ имеет минимальное по абсолютной величине значение производной, изменяя его от -1 до $2,5$ с шагом $0,001$.

3. Составить программу вычисления максимального значения экстремума-минимума функции $X^{1/3} \sin^2(10X)$ и соответствующего значения аргумента при его изменении на интервале $0,06...2,32$ с шагом $0,001$.

4. Составить программу вычисления минимального расстояния между точками экстремумов-максимумов функции $\cos X \sqrt{1 + 9 \sin^2 X}$ и соответствующих значений функции при изменении X на интервале $8...18$ с шагом $0,001$.

5. Произвольные значения от $-3,4$ до $1,1$ аргумента функции $Y = X^5 - 18X^3 - 22X^2$ находятся в массиве $X(n)$, $n \leq 20$. Составить программу вычисления максимального и минимального значений функции, а также соответствующих значений элементов массива X и их индексов.

6. Известно, что в интервале $-2...8,5$ уравнение $\cos(2,5X) \times \sin^2 X + 0,2 = 0$ имеет несколько корней и в каждом корне производная функции меньше -1000 . Составить программу нахождения корня, в котором производная функции имеет максимальное значение.

7. Известно, что в интервале от -14 до 19 функция $Y = e^{-2,5(X-1)^2} \cos X$ имеет несколько точек перегиба со значениями производной больше -500 . Составить программу нахождения точки перегиба, в которой производная функции имеет максимальное значение.

8. Составить программу вычисления минимального расстояния между соседними корнями уравнения $\frac{1}{1,5 + X \cos(1,5X)} - 0,6 = 0$, изменяя X на интервале $1,2...16$ с шагом $0,0001$.

9. Составить программу вычисления значения аргумента, изменяя его на интервале $6...12$ с шагом $0,001$, при котором производная функции $Y = X^{0,2} \sin^2 X \cos(3X)$ имеет минимальное по абсолютной величине значение в точке перегиба.

10. На интервале $-0,5...0,3$ функция $\sqrt[3]{X} \sin^2(20X) - 0,2X$ имеет несколько экстремумов. Требуется найти, изменяя аргумент с шагом dX , пару соседних точек экстремума, разность значений функции в которых минимальна.

11. Составить программу вычисления максимального расстояния между точками экстремумов-минимумов функции $Y = \sqrt{1 + 9 \sin^2(5X)} \cdot \sin(3X)$ и соответствующих значений функции при изменении X на интервале $2...8$ с шагом $0,001$.

12. На интервале $-1,8...1,9$ функция $Y = \cos(5X) \sin^2 X$ имеет несколько экстремумов. Требуется найти, изменяя аргумент с шагом dX , точку экстремума-минимума с максимальным значением функции.

13. Составить программу вычисления минимального положительного значения функции $Y = 10 - (2X^3 + 7X^2 - 3X^4) \sin(12X)$ и соответствующих значений аргумента при его изменении на интервале $-1,5...2,2$ с шагом $0,001$.

14. Найти локальное минимальное приращение расстояния от точки с координатами (Xt, Yt) до кривой $Y = X^5 - 18X^3 - 22X^2$, изменяя X на интервале $-3...0,2$ с шагом $0,05$.

15. Составить программу вычисления максимального расстояния между корнями уравнения $2\cos(2X) + X\sin X + 0,4 = 0$ с положительным приращением функции в соседних точках, изменяя X на интервале $-2...3$ с шагом $0,0001$.

16. На интервале $8...16$ функция $Y = \cos(5X) \sin^2 X$ имеет несколько экстремумов. Требуется найти, изменяя аргумент с шагом dX , точку экстремума с максимальным значением функции.

17. В массивах $X(N)$, $Y(N)$, $N \leq 30$ заданы координаты точек на плоскости. Найти такое $i \leq N$, для которого расстояние

$$d = \left| \frac{aX_i + bY_i + c}{\sqrt{a^2 + b^2}} \right|$$

от точки (X_i, Y_i) до прямой $aX + bY + c = 0$

минимально.

18. Изменяя аргумент функций $Y1 = X \sin(5X)$ и $Y2 = e^x \cos^2(2X)$ на интервале $0...4,15$ с шагом $0,0001$, найти минимальное расстояние между соседними точками их экстремумов.

19. Изменяя аргумент функции $Y = X \cos(12X) - X \sin(X)$ на интервале $-1...1$ с шагом $0,0001$, найти минимальное и максимальное приращения и соответствующие им значения аргумента.

20. Найти минимальное расстояние от точки с координатами (Xt, Yt) до прямых $a_iX + b_iY + c_i = 0$, $i = 1, 2, \dots, 10$, используя

$$\text{формулу } d = \left| \frac{aXt + bYt + c}{\sqrt{a^2 + b^2}} \right|$$

расстояния от точки (Xt, Yt) до прямой $aX + bY + c = 0$.

21. На интервале $-2...6$ функция $Y = \cos(2,5X) \sin^2 X$ имеет несколько экстремумов. Требуется найти, изменяя аргумент с шагом dX , точку экстремума-максимума с минимальным значением функции.

22. Составить программу вычисления максимального отрицательного значения функции $Y = \sin^5(3X) + 15X \sin^4(3X) \cos(3X)$ и соответствующее значение аргумента при его изменении на интервале $-4 \dots 16$ с шагом 0,001.

23. Составить программу вычисления минимального расстояния между корнями уравнения $1/[2\cos X + X \sin(2X)] - 0,4 = 0$ с положительным приращением в их окрестностях, изменяя X на интервале от $-1,5 \dots 7$ с шагом 0,0001.

24. На интервале $-1 \dots 8$ функция $Y = \cos(2,5X) \sin^2 X + 0,5$ имеет несколько экстремумов-минимумов. Требуется найти, изменяя аргумент с шагом dX , минимальный положительный из таких экстремумов и соответствующее значение X .

25. Найти минимальное расстояние от точки с координатами (Xt, Yt) до кривой $Y = X^2 \sin(9X)$, изменяя аргумент с шагом dX на интервале $X1 \dots X2$, а также соответствующую точку (X_{\min}, Y_{\min}) на этой кривой.

4.14. Вычисление суммы бесконечного ряда с заданной точностью

Пусть задана последовательность чисел $R_1, R_2, R_3, \dots, R_n, \dots$. Выражение $R_1 + R_2 + R_3 + \dots + R_n + \dots$ называют *бесконечным рядом*, или просто *рядом*, а числа R_1, R_2, R_3, \dots — *членами ряда*. При этом имеют в виду, что накопление суммы ряда начинается с первых

его членов. Сумма $S_n = \sum_{i=1}^n R_i$ называется *частичной суммой ряда*: при $n = 1$ — первой частичной суммой, при $n = 2$ — второй частичной суммой и так далее.

Ряд называется *сходящимся*, если последовательность его частичных сумм имеет предел, и *расходящимся* — в противном случае. Понятие суммы ряда можно расширить [5], и тогда некоторые расходящиеся ряды также будут обладать суммами. Именно *расширенное понимание суммы ряда* будет использовано при разработке алгоритмов для следующей постановки задачи: накопление суммы следует выполнять до тех пор, пока очередной член ряда по абсолютной величине не будет больше заданной величины ϵ .

В общем случае все или часть членов ряда могут быть заданы выражениями, зависящими от номера члена ряда и переменных. Например,

$$S = 1 - X + \frac{X^2}{2!} - \frac{X^3}{3!} + \frac{X^4}{4!} - \frac{X^5}{5!} + \dots = 1 + \sum_{i=1}^{\infty} (-1)^i \frac{X^i}{i!}.$$

Тогда возникает вопрос, как минимизировать объем вычислений — вычислять значение очередного члена ряда по *общей формуле члена ряда* (в приведенном примере ее представляет выражение под знаком суммы), по рекуррентной формуле (ее вывод представлен ниже) или использовать рекуррентные формулы лишь для частей выражения члена ряда (см. ниже).

4.15. Вывод рекуррентной формулы для вычисления члена ряда

Пусть требуется найти ряд чисел R_1, R_2, R_3, \dots , последовательно вычисляя их по формулам

$$R_1 = \frac{1}{2} X, \quad R_2 = \frac{1 \cdot 3}{2 \cdot 4} X^2, \dots,$$

$$R_N = \frac{1 \cdot 3 \cdot 5 \cdot 7 \cdot \dots \cdot (2N-3)(2N-1)}{2 \cdot 4 \cdot 6 \cdot 8 \cdot 9 \cdot \dots \cdot (2N-2)(2N)} X^N. \quad (4.1)$$

Для сокращения вычислений в данном случае удобно воспользоваться *рекуррентной формулой* вида $R_N = \alpha(X, N) R_{N-1}$, позволяющей вычислить R_N при $N > 1$, зная значение предыдущего члена ряда R_{N-1} , где $\alpha(X, N)$ — выражение, которое можно получить после упрощения отношения в формуле (4.1) N к $N-1$:

$$\begin{aligned} \alpha(X, N) &= \frac{1 \cdot 3 \cdot 5 \cdot 7 \cdot \dots \cdot (2N-3)(2N-1)}{2 \cdot 4 \cdot 6 \cdot 8 \cdot \dots \cdot (2N-2)(2N)} X^N \\ &= \frac{1 \cdot 3 \cdot 5 \cdot 7 \cdot \dots \cdot [2(N-1)-3][2(N-1)-1]}{2 \cdot 4 \cdot 6 \cdot 8 \cdot \dots \cdot [2(N-1)-2][2(N-1)]} X^{N-1} \\ &= \frac{1 \cdot 3 \cdot \dots \cdot (2N-3)(2N-1) X^N \cdot 2 \cdot 4 \cdot \dots \cdot [2(N-1)-2][2(N-1)]}{1 \cdot 3 \cdot \dots \cdot [2(N-1)-3][2(N-1)-1] X^{N-1} \cdot 2 \cdot 4 \cdot \dots \cdot (2N-2) \cdot 2N} = \\ &= \frac{1 \cdot 3 \cdot \dots \cdot (2N-3)(2N-1) X^{N-1} X \cdot 2 \cdot 4 \cdot \dots \cdot (2N-4)(2N-2)}{1 \cdot 3 \cdot \dots \cdot (2N-5)(2N-3) X^{N-1} \cdot 2 \cdot 4 \cdot \dots \cdot (2N-2) \cdot 2N} = \\ &= \frac{(2N-1)X}{2N}. \end{aligned}$$

Таким образом, рекуррентная формула примет вид

$$R_N = \frac{(2N-1)X}{2N} R_{N-1}. \quad (4.2)$$

Из сравнения общей формулы члена ряда (4.1) и рекуррентной (4.2) видно, что рекуррентная формула значительно упрощает вычисления. Применим ее для $N = 2, 3$ и 4 , зная, что $R_1 = \frac{1}{2}X$:

$$R_2 = \frac{(2 \cdot 2 - 1)X}{2 \cdot 2} \cdot \frac{1}{2}X = \frac{1 \cdot 3}{2 \cdot 4}X^2;$$

$$R_3 = \frac{(2 \cdot 3 - 1)X}{2 \cdot 3} \cdot \frac{1 \cdot 3}{2 \cdot 4}X^2 = \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6}X^3;$$

$$R_4 = \frac{(2 \cdot 4 - 1)X}{2 \cdot 4} \cdot \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6}X^3 = \frac{1 \cdot 3 \cdot 5 \cdot 7}{2 \cdot 4 \cdot 6 \cdot 8}X^4.$$

Способы вычисления значения члена ряда

Для вычисления значения члена ряда, в зависимости от его вида, может оказаться предпочтительнее использование либо общей формулы члена ряда, либо рекуррентной формулы, либо *смешанного способа вычисления значения члена ряда*, когда для одной или нескольких частей члена ряда используют рекуррентные формулы, затем их значения подставляют в общую формулу члена ряда. Например,

для ряда $1 + \frac{1}{2 \cdot 3} + \frac{1}{3 \cdot 5} + \dots + \frac{1}{N(2N-1)} + \dots$ проще вычислять

значение члена ряда R_N по его общей формуле $R_N = \frac{1}{N(2N-1)}$

(сравните с $R_N = \frac{(N-1)(2N-3)}{N(2N-1)}R_{N-1}$ по рекуррентной формуле);

для ряда $1 + \frac{2}{1 \cdot 2} + \frac{2^3}{1 \cdot 2 \cdot 3} + \dots + \frac{2^N}{N!} + \dots$ лучше воспользоваться

рекуррентной формулой $R_N = \frac{2}{N}R_{N-1}$;

для ряда $\frac{X^3 \ln(X^3)}{1} + \frac{X^6 \ln(X^6)}{1 \cdot 2} + \dots + \frac{X^{3N} \ln(X^{3N})}{N!} + \dots$ следу-

ет применить смешанный способ, вычисляя $A_N = X^{3N}$ по рекуррентной формуле $A_N = X^3 A_{N-1}$, $N = 2, 3, \dots$ при $A_1 = 1$; $B_N = N!$ — также по рекуррентной формуле $B_N = N B_{N-1}$, $N = 2, 3, \dots$ при $B_1 = 1$, а затем член ряда R_N — по общей формуле, которая примет вид

$$R_N = \frac{A_N \ln(A_N)}{B_N}.$$

4.16. Примеры выполнения задания

1. Вычислить с точностью ϵ для $0 \leq X \leq 45^\circ$:

– приближенное значение функции $\cos X$ по формуле

$$S = 1 - \frac{X^2}{2!} + \frac{X^4}{4!} - \dots + (-1)^{2N+1} \frac{X^{2N}}{(2N)!} + \dots,$$

используя рекуррентную формулу для вычисления члена

$$\text{ряда } R_N = \frac{-R_{N-1}X^2}{(2N-1)(2N)};$$

– точное значение функции $\cos X$;

– абсолютную и относительную ошибки приближенного значения.

```

program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  Math;
{$DEFINE DBG}
const
  //Коэффициент для перевода из градусов в радианы
  K=Pi/180;
var
  Eps: Extended =1E-8;
  X   : Extended =15;
  R, S, Y, D: Extended;
  N:Word;
begin
{$IFNDEF DBG}
  //Операторы, не используемые при отладке
  Write('Введите требуемую точность: ');
  ReadLn(Eps);
  Write('Введите значение угла в градусах: ');
  ReadLn(X);
{$ENDIF}
  //Перевод X в радианы и возведение в квадрат
  D:=Sqr(K*X);
  //Задание начальных значений переменным
  N:=0;
  R:=1;
  S:=0;
  //Цикл для вычисления членов ряда

```

```

//и накопления их суммы.
//Выполнять,
//пока модуль очередного члена ряда больше Eps.
while Abs(R)>Eps do
begin
  S:=S+R;
{$IFDEF DBG}
  if N<10 then //Вывод, используемый при отладке
    WriteLn('N=', N, ' R=', R:14:11
      , ' S=', S:14:11);
{$ENDIF}
  N:=N+2;
  R:=-R*D/N/(N-1)
end;
WriteLn;
//Вывод результатов вычислений:
WriteLn(N:14
  , ' = Число шагов, за которое достигнута '
  , ' заданная точность');
WriteLn(S:14:11
  , ' = Приближенное значение функции');
WriteLn(Cos(K*X):14:11
  , ' = Точное значение функции');
WriteLn(Abs(Cos(K*X)-S):14:11
  , ' = Абсолютная ошибка');
WriteLn(Abs((Cos(K*X)-S)/Cos(K*X)):14:11,
  ' = Относительная ошибка');
ReadLn;
end.

```

2. Вычислить с точностью ϵ для $0 \leq X \leq 1$:

— приближенное значение функции $\ln(1+X)$ по формуле

$$S = X - \frac{X^2}{2} + \frac{X^3}{3} - \frac{X^4}{4} + \dots + (-1)^{N+1} \frac{X^N}{N} + \dots,$$

используя смешанный способ вычисления члена ряда;

— точное значение функции $\ln(1+X)$;

— абсолютную и относительную ошибки приближенного значения.

Предусмотреть обработку исключения, возникающего при вычислениях вследствие ошибок в исходных данных, обеспечивающую вывод типа исключения и числа шагов вычисления суммы ряда, на котором оно возникло:

```

program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  Math;
var
  //Объявленные ниже переменные будут представлять:
  X, // - аргумент функции,
  Eps, // - требуемую точность,
  D, // - часть  $(-1)^{(N+1)} * X^N$  члена ряда,
      // вычисляемую по рекуррентной формуле,
  R, // - член ряда,
  S, // - сумму ряда,
  Y // - значение функции  $\ln(1+X)$ ,
      :Extended;
  N // - номер члена ряда.
      :Integer;
begin
  Write('Введите требуемую точность: ');
  ReadLn(Eps);
  Write('Введите значение аргумента функции: ');
  ReadLn(X);
  D:=1; //Начальное значение для вычисления числителя
      // члена ряда по рекуррентной формуле
  N:=1; //Нумерация членов ряда начинается с 1
  R:=X; //Первый член ряда
  S:=0; //Начальное значение для накопления суммы
      //членов ряда
  //Цикл для вычисления членов ряда
  //и накопления их суммы.
  //Выполнять, пока модуль очередного члена ряда
  //больше требуемой точности.
  while Abs(R)>Eps do
  try
    S:=S+R; //Включение очередного члена ряда в сумму
    // (*)
    if N<5 then //Вывод, используемый при отладке
      WriteLn('N=', N, ' R=', R:14:11
        , ' S=', S:14:11);
    // *)
    N:=N+1; //Увеличение номера члена ряда
    D:=-D*X; //Рекуррентная формула вычисления
      //числителя члена ряда
    R:=D/N //Вычисление N-го члена ряда

```



```

except
  on E:Exception do //Обработать любое исключение
  begin
    WriteLn(E.Message); //Вывод типа исключения
    WriteLn('Исключение возникло на шаге N = ',N);
    ReadLn;
    Halt
  end;
end; //while try
//Вывод результатов вычислений:
//Число шагов, за которое достигнута
//заданная точность
WriteLn('Число шагов, за которое достигнута'
      , ' заданная точность = ',N);
//Приближенное значение функции ln(1+X)
WriteLn('Приближенное значение функции ln(1+X) = '
      ,S:14:11);
//Точное значение функции ln(1+X)
WriteLn('ln(1+X) = ',Ln(1+X):14:11);
//Абсолютная ошибка
WriteLn('Абсолютная ошибка = '
      ,Abs(Ln(1+X)-S):14:11);
//Относительная ошибка
WriteLn('Относительная ошибка = '
      ,Abs((Ln(1+X)-S)/Ln(1+X)):14:11, '%');
ReadLn;
end.

```

4.17. Задания для самостоятельной работы

Составить программу вычисления суммы ряда с заданной точностью ε . Анализируя код программы, выявить возможные причины возникновения исключений и провести их обработку, обеспечивающую вывод типа исключения и пояснение причины возникновения.

1. Вычислить с точностью ε :

– приближенное значение функции $\ln(1+X)/X$ по формуле

$$S = 1 - \frac{X}{2} + \frac{X^2}{3} - \frac{X^3}{4} + \dots + (-1)^{N+1} \frac{X^{N-1}}{N} + \dots,$$

используя смешанный способ вычисления члена ряда;

– точное значение функции $\ln(1+X)/X$;

– абсолютную и относительную ошибки приближенного значения.

2. Вычислить с точностью ϵ :

- приближенное значение функции e^X по формуле

$$S = 1 + X + \frac{X^2}{2!} + \frac{X^3}{3!} + \frac{X^4}{4!} + \frac{X^5}{5!} + \dots,$$

используя рекуррентную формулу для вычисления члена ряда;

- точное значение функции e^X ;
- абсолютную и относительную ошибки приближенного значения.

3. Вычислить с точностью ϵ :

- приближенное значение функции $\sin X$ по формуле

$$S = X - \frac{X^3}{3!} + \frac{X^5}{5!} - \frac{X^7}{7!} + \frac{X^9}{9!} + \dots,$$

используя рекуррентную формулу для вычисления члена ряда;

- точное значение функции $\sin X$;
- абсолютную и относительную ошибки приближенного значения.

4. Вычислить с точностью ϵ :

- приближенное значение функции $\sqrt{1+X}$ по формуле

$$S = 1 + \frac{X}{2} - \frac{X^2}{2 \cdot 4} + \frac{3X^3}{2 \cdot 4 \cdot 6} - \frac{3 \cdot 5 X^4}{2 \cdot 4 \cdot 6 \cdot 8} + \dots,$$

используя рекуррентную формулу для вычисления члена ряда;

- точное значение функции $\sqrt{1+X}$;
- абсолютную и относительную ошибки приближенного значения.

5. Вычислить с точностью ϵ :

- приближенное значение функции $\arcsin X$ по формуле

$$S = X + \frac{X^3}{2 \cdot 3} - \frac{3X^5}{2 \cdot 4 \cdot 5} + \frac{3 \cdot 5 X^7}{2 \cdot 4 \cdot 6 \cdot 7} - \frac{3 \cdot 5 \cdot 7 X^9}{2 \cdot 4 \cdot 6 \cdot 8 \cdot 9} + \dots,$$

используя рекуррентную формулу для вычисления члена ряда;

- точное значение функции $\arcsin X$;
- абсолютную и относительную ошибки приближенного значения.

6. Вычислить с точностью ϵ :

- приближенное значение функции $\operatorname{arctg} X$ по формуле

$$S = X - \frac{X^3}{3} + \frac{X^5}{5} - \frac{X^7}{7} + \frac{X^9}{9} + \dots,$$

используя смешанный способ вычисления члена ряда;

- точное значение функции $\operatorname{arctg} X$;
- абсолютную и относительную ошибки приближенного значения.

7. Вычислить с точностью ϵ :

- приближенное значение функции $\frac{e^X - e^{-X}}{2}$ по формуле

$$S = X + \frac{X^3}{3!} + \frac{X^5}{5!} + \frac{X^7}{7!} + \frac{X^9}{9!} + \dots,$$

используя рекуррентную формулу для вычисления члена ряда;

- точное значение функции $\frac{e^X - e^{-X}}{2}$;
- абсолютную и относительную ошибки приближенного значения.

8. Вычислить с точностью ϵ :

- приближенное значение функции $\frac{e^X + e^{-X}}{2}$ по формуле

$$S = 1 + \frac{X^2}{2!} + \frac{X^4}{4!} + \frac{X^6}{6!} + \frac{X^8}{8!} + \dots,$$

используя рекуррентную формулу для вычисления члена ряда;

- точное значение функции $\frac{e^X - e^{-X}}{2}$;
- абсолютную и относительную ошибки приближенного значения.

9. Вычислить с точностью ϵ :

- приближенное значение функции $\ln(1 - X)$ по формуле

$$S = -X - \frac{X^2}{2} - \frac{X^3}{3} - \frac{X^4}{4} - \frac{X^5}{5} + \dots,$$

используя смешанный способ вычисления члена ряда;

- точное значение функции $\ln(1 - X)$;
- абсолютную и относительную ошибки приближенного значения.

10. Вычислить с точностью ϵ :

- приближенное значение функции $\ln \frac{1+X}{1-X}$ по формуле

$$S = 2 \cdot \left(X + \frac{X^3}{3} + \frac{X^5}{5} + \frac{X^7}{7} + \frac{X^9}{9} + \dots \right),$$

используя смешанный способ вычисления члена ряда;

- точное значение функции $\ln \frac{1+X}{1-X}$;
- абсолютную и относительную ошибки приближенного значения.

11. Вычислить с точностью ϵ :

- приближенное значение функции $(1+X)^{-3}$ по формуле

$$S = 1 - \frac{2 \cdot 3}{2} X + \frac{3 \cdot 4 X^2}{2} - \frac{4 \cdot 5 X^3}{2} + \frac{5 \cdot 6 X^4}{2} - \frac{6 \cdot 7 X^5}{2} + \dots,$$

используя смешанный способ вычисления члена ряда;

- точное значение функции $(1+X)^{-3}$;
- абсолютную и относительную ошибки приближенного значения.

12. Вычислить с точностью ϵ :

- приближенное значение функции $\ln(X + \sqrt{1+X^2})$ по формуле

$$S = X - \frac{1}{2} \cdot \frac{X^3}{3} + \frac{1}{2} \cdot \frac{3}{4} \cdot \frac{X^5}{5} - \frac{1}{2} \cdot \frac{3}{4} \cdot \frac{5}{6} \cdot \frac{X^7}{7} + \frac{1}{2} \cdot \frac{3}{4} \cdot \frac{5}{6} \cdot \frac{7}{8} \cdot \frac{X^9}{9} - \dots,$$

используя смешанный способ вычисления члена ряда;

- точное значение функции $\ln(X + \sqrt{1+X^2})$;
- абсолютную и относительную ошибки приближенного значения.

13. Вычислить с точностью ϵ :

- приближенное значение функции e^{-X^2} по формуле

$$S = 1 - \frac{X^2}{1!} + \frac{X^4}{2!} - \frac{X^6}{3!} + \frac{X^8}{4!} - \dots + (-1)^N \frac{X^{2N}}{N!} + \dots,$$

используя рекуррентную формулу для вычисления члена ряда;

- точное значение функции e^{-X^2} ;
- абсолютную и относительную ошибки приближенного значения.

14. Вычислить с точностью ϵ :

- приближенное значение функции $(1+X)^{-2}$ по формуле

$$S = 1 - 2X + 3X^2 - 4X^3 + 5X^4 - \dots,$$

используя смешанный способ вычисления члена ряда;

- точное значение функции $(1+X)^{-2}$;
- абсолютную и относительную ошибки приближенного значения.

15. Вычислить с точностью ϵ :

- приближенное значение функции $\frac{1}{\sqrt{1+X}}$ по формуле

$$S = 1 - \frac{1}{2}X + \frac{1 \cdot 3}{2 \cdot 4}X^2 - \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6}X^3 + \frac{1 \cdot 3 \cdot 5 \cdot 7}{2 \cdot 4 \cdot 6 \cdot 8}X^4 - \dots,$$

используя рекуррентную формулу для вычисления члена ряда;

- точное значение функции $\frac{1}{\sqrt{1+X}}$;
- абсолютную и относительную ошибки приближенного значения.

16. Вычислить с точностью ϵ :

- приближенное значение π по формуле

$$S = 4 \cdot \left[1 - \frac{1}{3} + \frac{1}{5} - \dots + (-1)^{N-1} \frac{1}{2N-1} + \dots \right],$$

используя смешанный способ вычисления члена ряда;

- точное значение π с помощью стандартной функции Pi ;
- абсолютную и относительную ошибки приближенного значения.

17. Вычислить с точностью ϵ :

- приближенное значение $\frac{1}{\sqrt{1-X^2}}$ по формуле

$$S = 1 + \frac{1}{2}X^2 + \frac{1 \cdot 3}{2 \cdot 4}X^4 + \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6}X^6 + \frac{1 \cdot 3 \cdot 5 \cdot 7}{2 \cdot 4 \cdot 6 \cdot 8}X^8 + \dots,$$

используя рекуррентную формулу для вычисления члена ряда;

- точное значение функции $\frac{1}{\sqrt{1-X^2}}$;
- абсолютную и относительную ошибки приближенного значения.

18. Вычислить с точностью ϵ сумму бесконечного ряда:

$$S = \sum_{N=0}^{\infty} (-1)^{N-1} \frac{X^{2N+1}}{2N+1},$$

- используя смешанный способ вычисления члена ряда;
- используя общую формулу для вычисления члена ряда.

19. Вычислить с точностью ϵ сумму бесконечного ряда:

$$S = \sum_{N=0}^{\infty} (-1)^N \frac{X^{2N-1}}{4N^2-1},$$

- используя смешанный способ вычисления члена ряда;
- используя общую формулу для вычисления члена ряда.

20. Вычислить с точностью ϵ сумму бесконечного ряда:

$$S = \sum_{N=0}^{\infty} (-1)^{N+1} \frac{X^{2N} \ln(1+2N)}{4N^2},$$

- используя смешанный способ вычисления члена ряда;
- используя общую формулу для вычисления члена ряда.

21. Вычислить с точностью ϵ сумму бесконечного ряда:

$$S = \sum_{N=1}^{\infty} (-1)^{N+1} \frac{X^{2N} (2^N - 1)}{2^N (2N - 1)},$$

- используя смешанный способ вычисления члена ряда;
- используя общую формулу для вычисления члена ряда.

22. Вычислить с точностью ϵ сумму бесконечного ряда:

$$S = \sum_{N=1}^{\infty} (-1)^{N+1} \frac{(2^{2N} - 1)X^{-2N}}{2^{2N} 2N},$$

- используя смешанный способ вычисления члена ряда;
- используя общую формулу для вычисления члена ряда.

23. Вычислить с точностью ϵ сумму бесконечного ряда:

$$S = \sum_{N=1}^{\infty} (-1)^N \frac{(2N - 1)X^{-N}}{(N + 5)2N},$$

- используя смешанный способ вычисления члена ряда;
- используя общую формулу для вычисления члена ряда.

24. Вычислить с точностью ϵ сумму бесконечного ряда:

$$S = \sum_{N=1}^{\infty} \frac{X^{3N} (3N + 1)}{N!},$$

- используя смешанный способ вычисления члена ряда;
- используя рекуррентную формулу для вычисления члена ряда.

25. Вычислить с точностью ϵ сумму бесконечного ряда:

$$S = \frac{1 \cdot 2}{1} X + \frac{3 \cdot 4 X^3}{3} + \frac{5 \cdot 6 X^5}{3 \cdot 7} + \frac{7 \cdot 8 X^7}{3 \cdot 7 \cdot 11} + \frac{9 \cdot 10 X^9}{3 \cdot 7 \cdot 11 \cdot 15} + \dots +$$

$$+ \frac{9 \cdot 10 X^9}{3 \cdot 7 \cdot 11 \cdot 15} + \dots + \frac{(2N - 1)(2N) X^{2N-1}}{1 \cdot 3 \cdot 7 \cdot 11 \cdot 15 \cdot \dots \cdot (4N - 5)} + \dots,$$

- используя смешанный способ вычисления члена ряда;
- используя рекуррентную формулу для вычисления члена ряда.

26. Вычислить с точностью ϵ сумму бесконечного ряда:

$$S = \frac{1}{3 \lg X} + \frac{1+2}{(3 \lg X)^2} + \frac{1+2+3}{(3 \lg X)^3} + \dots + \frac{1+2+\dots+N}{(3 \lg X)^N} + \dots,$$

используя смешанный способ вычисления члена ряда.

27. Вычислить с точностью ϵ сумму бесконечного ряда:

$$S = X + \frac{X^3}{Y^3 + X} + \frac{X^5}{Y^6 + X^2} + \dots + \frac{X^{2N+1}}{Y^{3N} + X^N} + \dots,$$

– используя рекуррентную формулу для вычисления члена ряда;

– используя смешанный способ вычисления члена ряда.

4.18. Уточнение корней уравнений

Для численного решения алгебраических уравнений разработано множество *итерационных методов* (методов последовательного приближения к точному значению) уточнения корня. Задача ставится так: при заданном одном или двух (зависит от метода) начальных приближениях корня уравнения $F(X) = 0$ получить приближение корня с заданной точностью ϵ .

Требуемая точность ϵ определяет условие завершения итерационного процесса, которое задается отношением $|X_n - X_{n-1}| < \epsilon$, где X_n и X_{n-1} — соседние приближения корня, полученные на $(n-1)$ -м и n -м шагах его уточнения, а начальные (грубые) приближения корней можно найти, например, по результатам табулирования функции $F(X)$.

Метод простых итераций

Для уточнения корня уравнения вида $F(X) = 0$ его следует преобразовать к уравнению $X = G(X)$. Исходными данными для уточнения корня являются требуемая точность ϵ и начальное приближение X_0 . Очередное приближение X_1 корня вычисляется на основе текущего приближения X_0 по формуле $X_1 = G(X_0)$ (на первом шаге уточнения корня X_0 представляет начальное приближение), после чего X_0 получает значение X_1 и процесс повторяется, пока модуль разности между X_0 и X_1 больше ϵ . Применение метода приводит к решению, если $|G'(X)| < 1$ внутри интервала, содержащего корень уравнения.

Пример. Пусть известно, что при заданном начальном приближении корня X_0 метод простых итераций обеспечит получение решения уравнения $X = (X - 0,1)^4 + 0,1$. Тогда для нахождения

ния корня с заданной точностью ε можно использовать следующий фрагмент программы:

```
uses
  SysUtils, Math;
var
  X0, X1, Eps, dX: Extended;
begin
  ReadLn(X0,Eps);
  repeat
    X1:=IntPower(X0-0.1, 4)+0.1;
    dX:=Abs(X0-X1);
    X0:=X1
  until dX<Eps;
  WriteLn('С точностью ',Eps, ' корень равен ',X0);
  .....
end.
```

Метод не всегда обеспечивает нахождение корня. Так, при $\varepsilon=10^{-3}$ и $X0 < 1,1$, где в окрестности корня $0,1 |G'(X)| = |4(X-0,1)^3| < 1$, будет найден искомый корень (как проходит уточнение корня, показано стрелками на рис. 4.7). Но в окрестности корня $1,1$ ($1,1$ — второй корень уравнения), где $|G'(X)| > 1$,

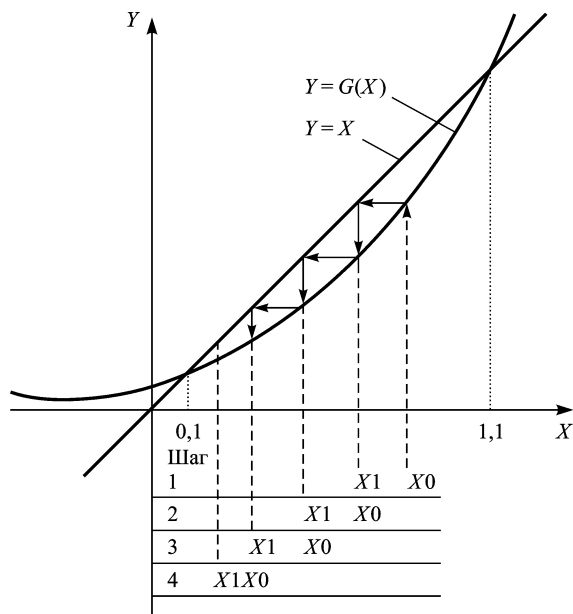


Рис. 4.7

каждый шаг процесса будет приводить к удалению от уточняемого корня, что, в конечном счете, приведет к нахождению другого корня ($X = 0,1$ при $X_0 < 1,1$) или переполнению разрядной сетки машины и аварийному останову (при $X_0 < 1,1$).

Чтобы найти корень уравнения $X = G(X)$ при условии $|G'(X)| > 1$, его следует преобразовать к виду $X = H(X)$, где $H(X)$ — обратная относительно $G(X)$ функция, и тогда использовать для поиска корня. С учетом сказанного изменим текст фрагмента программы, чтобы иметь возможность находить все корни уравнения $X = (X - 0,1)^4 + 0,1$ при любых начальных приближениях, когда $|G'(X)| \neq 0$:

```
uses
  SysUtils, Math;
var
  X0, X1, Eps, dX, P: Extended;
begin
  ReadLn(X0, Eps);
  P := Abs(4 * IntPower(X0 - 0.1, 3));
  if P = 0 then
    begin
      WriteLn('Нарушено условие '
        , 'применимости метода!');
      ReadLn;
      Halt;
    end
  else if P < 1 then
    //Использование исходной функции
    repeat
      X1 := IntPower(X0 - 0.1, 4) + 0.1;
      dX := Abs(X0 - X1);
      X0 := X1
    until dX < Eps
  else if P > 1 then
    //Использование обратной функции
    repeat
      X1 := Power(X0 - 0.1, 0.25) + 0.1 ;
      dX := Abs(X0 - X1);
      X0 := X1
    until dX < Eps;
  WriteLn('С точностью ', Eps, ' корень равен ', X0);
  . . . . .
end.
```

В некоторых случаях возможны заикливание — бесконечное выполнение цикла программы (например, для уравнения $X = 1/X$) — или медленная сходимость процесса (например, для уравнения $X = 1/(X - 10^{-6})$).

Для обеспечения информативности программ при заикливании или очень медленной сходимости вводят ограничения на число итераций. Если за заданное число шагов заданная точность не достигается, то процесс останавливается и выдается соответствующее сообщение.

Пример. Составить фрагмент программы для решения следующей задачи. Найти корень уравнения $X = 1/(X - 10^{-6})$ с заданной точностью ϵ . Если за заданное число N шагов точность не будет достигнута, то прекратить выполнение программы и вывести с пояснениями модуль разности между двумя последними приближениями ϵ и N , иначе вывести найденное значение корня и число шагов, за которое оно было найдено:

```
var
  X0, X1, Eps:Extended;
  I, N:Integer;
  F:Boolean;
begin
  ReadLn(X1,Eps,N);
  F:=False;
  for I:=1 to N do
    begin
      X0:=X1;
      X1:=1/(X0-1E-6);
      F:=Abs(X0-X1)<Eps;
      if F then
        //Решение найдено
        Break;//Выход из цикла
      end;
      if F then
        WriteLn('Корень X0 = ',X0:12,' найден за '
          ,i,' шагов')
      else
        WriteLn('Заданная точность ',Eps:0
          , ' не достигнута за ',N,' шагов');
      . . . . .
    end.
```

Выполнив эту программу при $X_0 = 0,5$, $\epsilon = 10^{-3}$, $N = 5000000$, можно убедиться, что после 5000000 итераций заданная точность достигнута не будет.

Метод половинного деления

Исходными данными для уточнения корня уравнения вида $F(X) = 0$ являются требуемая точность ϵ и два начальных приближения: XL и XR , между которыми должен находиться корень. Поэтому необходимым условием применения метода является истинность отношения $F(XL) \cdot F(XR) < 0$, т. е. метод непригоден в тех случаях, когда график $F(X)$ лишь касается оси абсцисс, не пересекая ее, например, в случае уравнения $X^2 = 0$. Один шаг итерационного процесса уточнения корня состоит в перемещении правой (XR) или левой (XL) границы отрезка (XL, XR) в середину в соответствии со следующим правилом: если знак $F[(XR + XL)/2]$ совпадает со знаком $F(XL)$, то XL получит значение $(XR + XL)/2$, иначе это значение получит XR (рис. 4.8). Процесс повторяется, пока модуль разности между XR и XL больше ϵ .

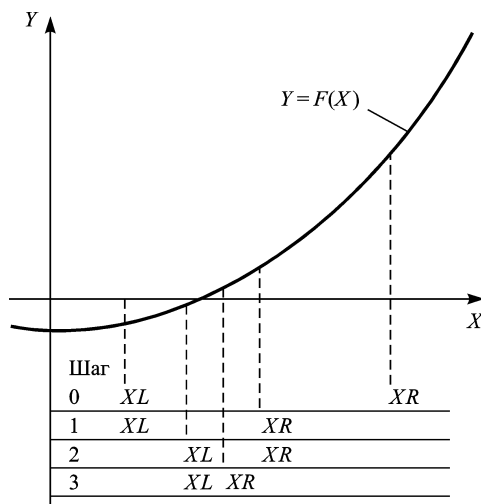


Рис. 4.8

Пример. Составить фрагмент программы уточнения корня уравнения $\arctg(X) - X = 0$ с заданной точностью ϵ при начальных приближениях корня XL и XR методом половинного деления:

```

ReadLn (XL, XR, Eps) ;
YL:=ArcTan (XL) -XL;
repeat
    X:=(XL+XR) /2;
    Y:= ArcTan (X) -X;
    if Y*YL>0 then

```

```

XL:=X
else
  XR:=X;
until Abs (XR-XL)<Eps;
WriteLn('Корень уравнения равен ', X:12);

```

Метод касательных

Исходными данными для уточнения корня уравнения вида $F(X) = 0$ являются требуемая точность ϵ и начальное приближение X_0 . Необходимым условием применения метода является истинность отношения $F(X_0) \cdot F''(X_0) > 0$. Один шаг итерационного процесса уточнения корня состоит в вычислении очередного приближения по формуле $X_1 = X_0 - F(X_0)/F'(X_0)$, после чего X_0 получает значение X_1 (рис. 4.9). Процесс повторяется, пока модуль разности между X_0 и X_1 больше ϵ .

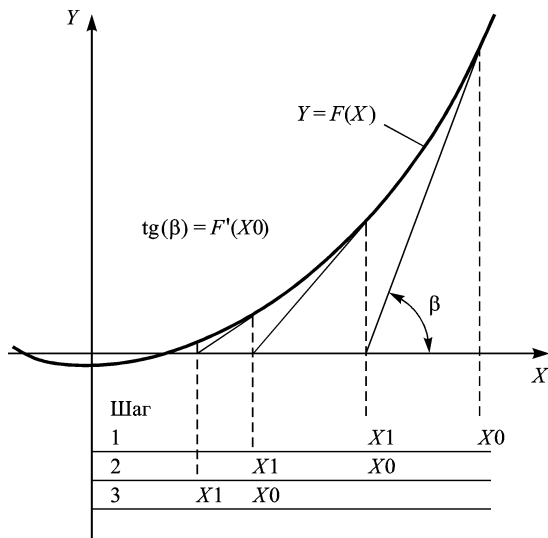


Рис. 4.9

Пример. Составить фрагмент программы уточнения корня уравнения

$$(X - 0,1)^4 - X + 0,1 = 0$$

с заданной точностью ϵ при начальном приближении корня X_0 :

```

ReadLn (X0, Eps);
repeat
  dX:=(IntPower(X0-0.1, 4)-X0+0.1)
    / (4*IntPower(X0-0.1, 3)-1);

```

```

X1:=X0-dX;
X0:=X1
until Abs(dX)<Eps;
WriteLn('Корень уравнения равен ', X0:12);

```

В этом фрагменте использовалось найденное заранее выражение $4(X - 0,1)^3 - 1$ первой производной для $(X - 0,1)^4 - X + 0,1$. С точки зрения объема и точности вычислений такое решение предпочтительнее использования для этих целей разностного отношения, как, например, в следующем операторе:

```

dX1:=(IntPower(X0-0.1, 4) - X0+0.1)
      /((( IntPower(X0+1e-8-0.1, 4) - X0+1e-8+0.1)
          -(IntPower(X0-0.1,4) - X0+0.1) )/1e-8);

```

где для вычисления приближенного значения производной использовалась формула $F'(X0) \approx \frac{F(X0 + \Delta X) - F(X0)}{\Delta X}$ и $\Delta X = 10^{-8}$.

4.19. Пример выполнения задания

Программа составлена по условию варианта задания № 30 (см. ниже). В реализации метода касательных используют выражения производной $9X^2 - 10X + 1$ и второй производной $18X - 10$ выражения, входящего в уравнение $3X^3 - 5X^2 + X + 0,4 = 0$:

```

program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils, Math;
var
  X0,X1,Eps,dX,X,XL,XR,YL,Y:Extended;
  i,N:Integer;
begin
  //Метод касательных
  WriteLn('Метод касательных');
  Write('Введите X0, Eps и N: ');
  ReadLn(X0,Eps,N);
  //Проверка применимости метода
  if (3*IntPower(X0,3)-5*Sqr(X0)+X0+0.4)
    *(18*X0-10) <= 0 then
    WriteLn('Не выполнено условие применимости'
      , ' метода касательных!')
  else
    begin
      i:=0;
      repeat

```

```

i:=i+1;
dX := (3*IntPower(X0,3)-5*Sqr(X0)+X0+0.4)
      / (9* Sqr(X0)-10*X0+1);
X1:=X0-dX;
X0:=X1
until (Abs(dX)<Eps) or (i=N);
if Abs(dX)<Eps then
  WriteLn('Корень X уравнения ',X0:14
          , ' найден за ',i, ' шагов, Y(X) = '
          ,3* IntPower(X0,3)-5* Sqr(X0)+X0+0.4:14)
else
  WriteLn('Корень уравнения не найден!');
end;
WriteLn;
//Метод половинного деления
WriteLn('Метод половинного деления');
Write('Введите XL, XR, Eps и N: ');
ReadLn(XL, XR, Eps, N);
//Проверка применимости метода
if (3*IntPower(XL,3)-5*Sqr(XL)+XL+0.4)
  * (3*IntPower(XR,3)-5*Sqr(XR)+XR+0.4) > 0 then
  WriteLn('Не выполнено условие применимости'
          , ' метода половинного деления!')
else
begin
  YL:=3*IntPower(XL,3)-5*Sqr(XL)+XL+0.4;
  i:=0;
  repeat
    X:=(XL+XR)/2;
    Y:= 3*IntPower(X,3)-5*Sqr(X)+X+0.4;
    if Y*YL>0 then
      XL:=X
    else
      XR:=X;
    inc(i);
  until (Abs(XR-XL)<Eps) or (i=N);
  if Abs(XR-XL)<Eps then
    WriteLn('Корень уравнения ',X:14, ' найден за '
            ,i, ' шагов, Y(X) = ',Y:14)
  else
    WriteLn('Корень уравнения не найден');
  end;
  ReadLn;
end.

```

4.20. Задания для самостоятельной работы

Составить программу нахождения корня уравнения (см. таблицу) с заданной точностью ϵ двумя указанными методами. Если за заданное число N шагов точность не будет достигнута, то вывести соответствующее сообщение, иначе вывести найденное значение корня, число шагов, за которое оно было найдено, и значение функции в корне.

В программе проверить возможность использования метода при введенном начальном приближении (для метода половинного деления — приближения слева и справа от корня). В правом столбце таблицы для каждого уравнения приведены приближенные значения корней, на которых требуется проверить работу программы (начальное приближение для поиска корня следует брать несколько меньше и/или несколько больше такого значения). Следует иметь в виду, что не каждый метод и не при каждом начальном приближении приводит к ближайшему корню, некоторые корни вообще не могут быть найдены избранным методом, разные методы при одинаковых начальных приближениях могут приводить к разным результатам, а также возможны исключения, которые следует обработать, не прерывая работы программы.

№ п/п	Методы	Уравнение	Начальные приближения
1	Итераций и касательных	$0,75 \cdot X - \sqrt[3]{X} = 0$	-1,5; 0; 1,5
2	Итераций и половинного деления	$\frac{X}{2} - \sqrt[5]{X} + 0,2 = 0$	-2,9; 0; 2,2
3	Касательных и половинного деления	$X - \ln X - 2 = 0$	0,15; 3,2
4	Итераций и касательных	$X^3 - X = 0$	-1; 0; 1
5	Итераций и половинного деления	$X^5 - X + 0,2 = 0$	-1; 0,2; 0,95
6	Касательных и половинного деления	$\operatorname{tg} \frac{X}{3} - X + 0,5 = 0$	-4; 0,76
7	Итераций и касательных	$\operatorname{tg} \frac{X}{2} - X = 0$	-2,3; 0; 2,3
8	Итераций и половинного деления	$\cos(4X) - 0,5X = 0$	1,4; 1,7
9	Касательных и половинного деления	$X^5 \cos X - X + 1 = 0$	-4,7; 1,5; 4,7

№ п/п	Методы	Уравнение	Начальные приближения
10	Итераций и касательных	$\frac{X^4}{8} - X \sin(14X)$	0,9; 1,1; 1,38
11	Итераций и половинного деления	$\sin X - 4\sqrt{\frac{X}{2}} + 2 + 2 = 0$	-1,8; -1,15
12	Касательных и половинного деления	$X^2 - X - 2 = 0$	-1; 2
13	Итераций и касательных	$e^{X/5} - X = 0$	1,3; 12,7
14	Итераций и половинного деления	$e^{X/2} - X - 3 = 0$	-2,75; 3,8
15	Касательных и половинного деления	$X + \ln X = 0$	0,57
16	Итераций и касательных	$X + \ln \frac{X}{3} = 0$	1
17	Итераций и половинного деления	$2X - \ln(X + 2) = 0$	-1,98; 0,45
18	Касательных и половинного деления	$e^{-X} - X = 0$	0,57
19	Итераций и касательных	$e^{-X} - 2 + X = 0$	-1,15; 1,84
20	Итераций и половинного деления	$\sin(2X) - X = 0$	-0,95; 0; 0,95
21	Касательных и половинного деления	$\sin X - X + 2 = 0$	2,55
22	Итераций и касательных	$e^X + \ln \frac{X}{10} + 2 = 0$	0,33
23	Итераций и половинного деления	$e^{-3X} - \sin X - 1,5 = 0$	-0,17
24	Касательных и половинного деления	$\operatorname{arctg} X - 0,5 + (X - 1)^3 = 0$	0,61
25	Итераций и касательных	$\arcsin X - \sqrt{X + 0,5} = 0$	0,93
26	Итераций и половинного деления	$5 \sin(2X) - \ln(X + 1) = 0$	-0,98; 0; 1,47
27	Касательных и половинного деления	$e^{-X} - X^2 + 2 = 0$	1,49
28	Итераций и касательных	$(X^2 - 1)^{-1} - 2^{1-X} = 0$	-1,1; 1,57; 6,25
29	Итераций и половинного деления	$2X^2 + 5X - 10 = 0$	-3,8; 1,3
30	Касательных и половинного деления	$3X^3 - 5X^2 + X + 0,4 = 0$	-0,19; 0,51; 1,3

4.21. Вычисление определенных интегралов

Для вычисления значений определенных интегралов существует множество методов. Рассмотрим три из них: *прямоугольников*, *трапеций* и *парабол (метод Симпсона)* на примерах при следующей постановке задачи. Составить фрагмент программы для вычисления приближенного значения определенного интеграла

$$z = \int_a^b f(x)dx \text{ при заданных подынтегральной функции } f(x), \text{ пре-}$$

делах интегрирования a и b и числе N разбиений интервала на подынтервалы. При этом шаг изменения аргумента Δx следует найти по формуле $\Delta x = (b - a)/N$.

Суть этих методов в накоплении, с учетом знаков, сумм площадей прямоугольников, трапеций или параболических трапеций, заменяющих на каждом подынтервале в общем случае криволинейную трапецию.

Замену криволинейной трапеции прямоугольником можно осуществить одним из трех способов. В первом случае (рис. 4.10) построение прямоугольников начинается с левой границы интервала интегрирования, при этом основание каждого прямоугольника равно Δx , а высота численно равна значению подынтегральной функции на левой границе подынтервала (левые прямоугольники).

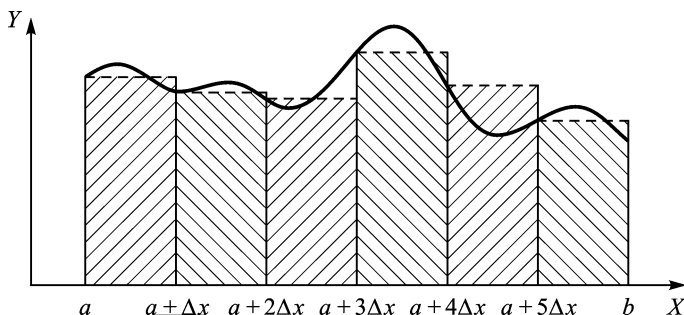


Рис. 4.10

Во втором случае (рис. 4.11) построение прямоугольников начинается с правой границы интервала интегрирования, при этом основание каждого прямоугольника равно Δx , а высота численно равна значению подынтегральной функции на правой границе подынтервала (правые прямоугольники). Оба этих способа дают одинаковую погрешность при вычислении интеграла.

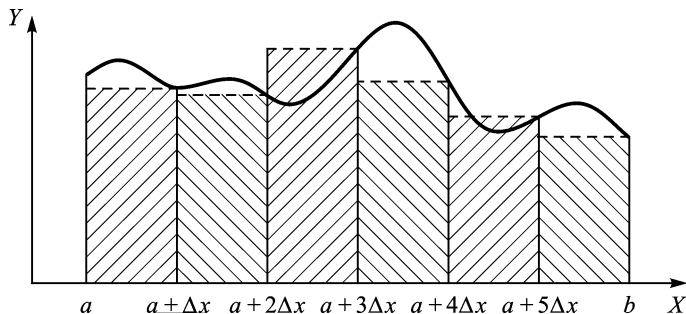


Рис. 4.11

В третьем случае (рис. 4.12) верхнее основание прямоугольника находится на точке пересечения перпендикуляра к оси абсцисс, проведенного через середину подынтервала, с кривой графика подынтегральной функции. При этом основание каждого прямоугольника равно Δx , а высота численно равна значению подынтегральной функции в середине подынтервала (средние прямоугольники). Этот способ дает более точный результат и обычно применяется на практике.

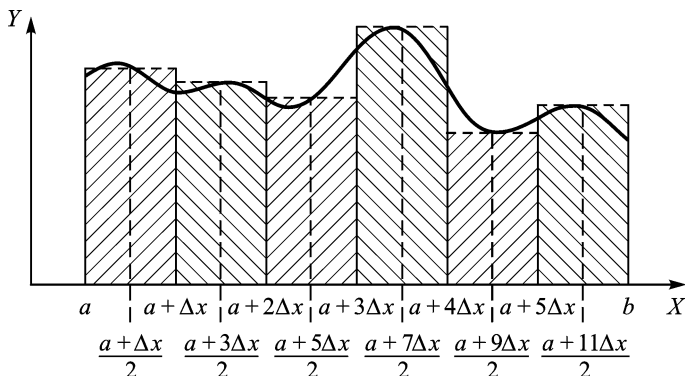


Рис. 4.12

В методе трапеций (рис. 4.13) основания каждой трапеции образуют перпендикуляры к оси абсцисс, проведенные на концах подынтервала и заключенные между точкой пересечения с осью абсцисс и точкой пересечения с кривой графика подынтегральной функции. Одну боковую сторону образует отрезок оси абсцисс, а другую — отрезок, соединяющий точки пересечения оснований с кривой графика функции.

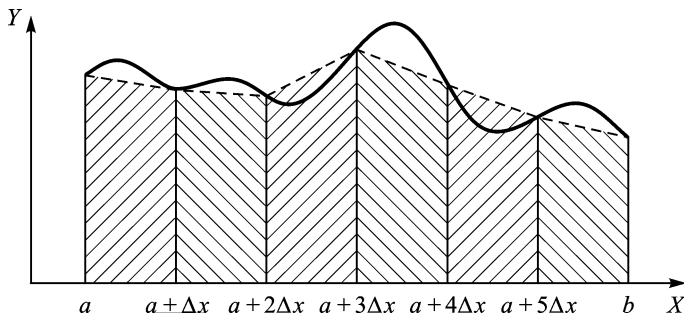


Рис. 4.13

Метод парабол (рис. 4.14) во многом совпадает с методом трапеций, отличие состоит в том, что точки пересечения перпендикуляров с кривой графика подынтегральной функции соединяются не отрезком прямой, а другой параболы, которая проходит через три точки, являющиеся точками пересечения перпендикуляров к оси абсцисс, проведенных через концы и середину подынтервала, с кривой графика функции.

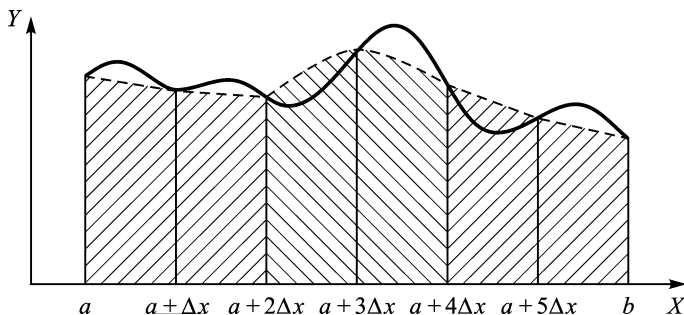


Рис. 4.14

Пример 1. Использование метода прямоугольников с вычислением высот прямоугольников в серединах подынтервалов. В этом методе формула приближенного значения определенного интеграла представляется в виде

$$z = \sum_{i=1}^N f(x_i) \Delta x,$$

где $x_i = a + \Delta x / 2 + (i - 1)\Delta x$.

Для уменьшения объема вычислений множитель Δx следует вынести за знак суммы: $z = \Delta x \sum_{i=1}^N f(x_i)$, а для вычисления теку-

щих значений центров x_i подынтервалов будем использовать прием накопления суммы:

```

z:=0;
dx:=(b-a)/N;
x:=a+dx/2;//Середина первого подынтервала
for i:=1 to N do
begin
  z:=z+Sin(x);
  x:=x+dx
end;
z:=z*dx;

```

Пример 2. Использование метода трапеций. В этом методе формула приближенного значения определенного интеграла представляется в виде

$$z = \sum_{i=0}^{N-1} \frac{f(x_i) + f(x_{i+1})}{2} \Delta x,$$

где $x_i = a + i\Delta x$, $x_{i+1} = a + (i+1)\Delta x$.

Преобразование ее к виду

$$\begin{aligned}
 z &= \sum_{i=0}^{N-1} \frac{f(x_i) + f(x_{i+1})}{2} \Delta x = \left[\sum_{i=0}^{N-1} \frac{f(x_i)}{2} + \sum_{i=1}^N \frac{f(x_i)}{2} \right] \Delta x = \\
 &= \left[\frac{f(a) + f(b)}{2} + \sum_{i=1}^{N-1} f(x_i) \right] \Delta x
 \end{aligned}$$

позволяет исключить повторные вычисления высот трапеций на внутренних подынтервалах и таким образом сократить объем вычислений:

```

z:=(Sin(a)+Sin(b))/2;
dx:=(b-a)/N;
x:=a+dx;
for i:=1 to N-1 do
begin
  z:=z+Sin(x);
  x:=x+dx
end;
z:=z*dx;

```

Пример 3. Использование метода параболических трапеций (Симпсона). В этом методе формула приближенного значения определенного интеграла представляется в виде

$$z = \frac{\Delta x}{3} \left[\frac{f(a) + f(b)}{2} + \sum_{i=1}^{N-1} f(x_i) + 2 \sum_{i=1}^N f(x_i - \Delta x / 2) \right], \quad x_i = a + i \cdot \Delta x$$

или, взяв N в 2 раза большим, т. е. разбив весь интервал на четное количество участков, в 2 раза меньшей длины:

$$z = \frac{\Delta x}{3} \left[f(a) + f(b) + 4 \sum_{i=1}^{N/2} f(x_{2i-1}) + 2 \sum_{i=1}^{N/2-1} f(x_{2i}) \right],$$

где $x_{2i-1} = a + (2i - 1)\Delta x$; $x_{2i} = a + 2i\Delta x$.

Используем вторую формулу в следующем фрагменте программы:

```
ReadLn(a, b, N);
Integ:=Sin(a);
dx:=(b-a)/N;
for i:=1 to N div 2 do
begin
  x:=a+2*i*dx;
  Integ:=Integ+2*Sin(x)+4*Sin(x-dx);
end;
Integ:=(Integ-Sin(b))*dx/3;
WriteLn(Integ:10:5);
Itoch:=- (Cos(b)-Cos(a));
WriteLn(Itoch:10:5);
ReadLn;
```

4.22. Пример выполнения задания

Составить программу вычисления приближенных значений определенного интеграла $\int_a^b e^{qx} (q^2 \cos x - 2q \sin x - \cos x) dx$ методом прямоугольников и методом Симпсона, а также точное его значение по первообразной $e^{qx} (q \cos x - \sin x)$. Вычислить также абсолютную и относительную ошибки для каждого приближенного метода. Пределы интегрирования a и b , а также число N подынтервалов задавать при вводе. Для метода Симпсона используем первую формулу из предыдущего примера:

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
```

```

var
  A,B,Q,dX,dX2,X,Z,Z0:Extended;
  i,N:Integer;
begin
  Write('Введите A, B, Q и N : ');ReadLn(A,B,Q,N);
  //МЕТОД ПРЯМОУГОЛЬНИКОВ
  WriteLn('Метод прямоугольников при N = ',N);
  Z:=0;
  dX:=(B-A)/N;
  X:=A+dX/2;
  for i:=1 to N do
  begin
    Z:=Z+Exp(Q*X)*(Sqr(Q)*Cos(X)-2*Sin(X)*Q-Cos(X));
    X:=X+dX
  end;
  Z:=Z*dX;
  WriteLn(Z,' - приближенное значение интеграла');
  Z0:=Exp(Q*B)*(Q*Cos(B)-Sin(B))-Exp(Q*A)
    *(Q*Cos(A)-Sin(A));
  WriteLn(Z0,' - точное значение интеграла');
  WriteLn(Abs(Z0-Z),' - абсолютная ошибка');
  WriteLn(Abs((Z0-Z)/Z0),' - относительная ошибка');
  WriteLn;
  //МЕТОД ПАРАБОЛ
  WriteLn('Метод Симпсона при N = ',N);
  dX:=(B-A)/N;
  dX2:=dX/2;
  Z:=( Exp(Q*A)*(Sqr(Q)*Cos(A)-2*Sin(A)*Q-Cos(A))
    +Exp(Q*B)*(Sqr(Q)*Cos(B)-2*Sin(B)*Q-Cos(B)))/2
    +2*Exp(Q*(B-dX2))*( Sqr(Q)*Cos(B-dX2)
    -2*Sin(B-dX2)*Q-Cos(B-dX2));
  X:=A+dX;
  for i:=1 to N-1 do
  begin
    Z:=Z+Exp(Q*X)*(Sqr(Q)*Cos(X)-2*Sin(X)*Q-Cos(X))
      +2*Exp(Q*(X-dX2))
        *(Sqr(Q)*Cos(X-dX2)-2*Sin(X-dX2)*Q-Cos(X-dX2));
    X:=X+dX
  end;
  Z:=Z*dX/3;
  WriteLn(Z,' - приближенное значение интеграла');
  Z0:=Exp(Q*B)*(Q*Cos(B)-Sin(B))
    -Exp(Q*A)*(Q*Cos(A)-Sin(A));

```

```

WriteLn(Z0, ' - точное значение интеграла');
WriteLn(Abs(Z0-Z), ' - абсолютная ошибка');
WriteLn(Abs((Z0-Z)/Z0), ' - относительная ошибка');
ReadLn;
end.

```

4.23. Задания для самостоятельной работы

Составить программу вычисления приближенных значений определенного интеграла (см. таблицу) двумя предложенными методами, а также точное его значение по первообразной. Вычислить абсолютную и относительную ошибки для каждого приближенного метода. Пределы интегрирования a и b , а также число N подынтервалов задавать при вводе. Выполняя программу при вводимых $N = 10k$, $k = 1, 2, \dots, 8$, установить зависимость величин ошибок от N .

№ п/п	Методы	Подынтегральная функция	Первообразная подынтегральной функции
1	Парабол и прямоугольников	$\frac{1}{1-x^2}$	$\frac{1}{2} \ln \frac{x+1}{1-x}$
2	Трапечий и парабол	$\sin^2 x$	$\frac{1}{2} x - \frac{1}{4} \sin 2x$
3	Трапечий и прямоугольников	$\frac{1}{x}$	$\ln x $
4	Парабол и прямоугольников	$\sqrt{\frac{x+1}{1-x}}$	$-\sqrt{1-x^2} + \arcsin x$
5	Трапечий и парабол	$\sin 3x \cos 2x$	$-\frac{\cos 5x}{10} - \frac{\cos x}{2}$
6	Трапечий и прямоугольников	$\frac{1}{x\sqrt{x^2+1}}$	$\ln \frac{x}{1+\sqrt{x^2+1}}$
7	Парабол и прямоугольников	$\ln^2 x/x$	$\ln^3 x/3$
8	Трапечий и парабол	$e^x \sin x$	$e^x (\sin x - \cos x)/2$
9	Трапечий и прямоугольников	$\frac{1}{1+x^2}$	$\arctg x$
10	Парабол и прямоугольников	$\frac{x}{(1+x)^3}$	$\frac{1}{2(1+x)^2} - \frac{1}{1+x}$

№ п/п	Методы	Подынтегральная функция	Первообразная подынтегральной функции
11	Парабол и прямоугольников	$\sin^3 x \cos x$	$\frac{\sin^4 x}{4}$
12	Трапечий и парабол	$(e^x - e^{-x})/2$	$(e^x + e^{-x})/2$
13	Трапечий и прямоугольников	$\operatorname{tg} x$	$-\ln \cos x$
14	Парабол и прямоугольников	$x\sqrt{x^2 + 1}$	$\sqrt{(x^2 + 1)^3}/3$
15	Трапечий и парабол	$\frac{1}{x(1 + x^2)}$	$\frac{1}{2} \ln \frac{x^2}{1 + x^2}$
16	Трапечий и прямоугольников	$\sqrt{1 + x}$	$2\sqrt{(1 + x)^3}/3$
17	Парабол и прямоугольников	$\sin x$	$-\cos x$
18	Трапечий и парабол	8^x	$8^x/\ln 8$
19	Парабол и прямоугольников	$\frac{1}{x(1 + x)}$	$-\ln \frac{1 + x}{x}$
20	Парабол и прямоугольников	$1/\sqrt{1 - x^2}$	$\arcsin x$
21	Трапечий и парабол	$\ln x$	$x \ln x - x$
22	Трапечий и прямоугольников	$\frac{e^x - e^{-x}}{e^x + e^{-x}}$	$\ln \frac{e^x + e^{-x}}{2}$
23	Парабол и прямоугольников	$\frac{1}{x^2\sqrt{x^2 - 1}}$	$\frac{\sqrt{x^2 - 1}}{x}$
24	Трапечий и парабол	$x^3 \ln x$	$x^4 \left(\frac{\ln x}{4} - \frac{1}{16} \right)$
25	Трапечий и прямоугольников	$\frac{x}{1 + x}$	$1 + x - \ln(1 + x)$
26	Трапечий и парабол	$\frac{\sin x}{1 + \cos x}$	$-\ln 1 + \cos x $
27	Парабол и прямоугольников	$\ln(x + \sqrt{x^2 + 4})$	$-\sqrt{x^2 + 4} +$ $+ x \ln(x + \sqrt{x^2 + 4})$
28	Трапечий и прямоугольников	$\frac{1}{x \ln x}$	$\ln(\ln x)$
29	Трапечий и парабол	$\frac{1}{\sin x}$	$\ln \left \operatorname{tg} \frac{x}{2} \right $
30	Парабол и прямоугольников	$-2e^x \sin x$	$e^x (\cos x - \sin x)$

5. Организация программ со структурой вложенных циклов

Структурой с вложенным циклом называют такую, в которой внутри одного цикла находится один или несколько других. Цикл, расположенный внутри другого, называют *внутренним*. Цикл, внутри которого находятся другие, называют *внешним*. Таким образом, один и тот же цикл может выступать и в роли внешнего (если он содержит внутри себя другие циклы), и в роли внутреннего (если он расположен внутри другого цикла). Правильная организация вложенного цикла состоит в том, что внутренний цикл должен целиком располагаться внутри внешнего.

Допустимыми являются следующие варианты организации вложенных циклов. Первый вариант вложенного цикла — внутри внешнего цикла последовательно расположено несколько внутренних:

```
for I:=1 to N do //внешний цикл
begin
    . . . . .
    for J:=1 to M do //первый внутренний цикл
    begin
        . . . . .
    end;      // конец первого внутреннего цикла
    . . . . .
    for K:=1 to L do //второй внутренний цикл
    begin
        . . . . .
    end;      // конец второго внутреннего цикла
    . . . . .
end;  // конец внешнего цикла
```

Частным случаем первого варианта организации вложенных циклов является следующий:

```
for I:=1 to N do
    for J:=1 to M do
        for K:=1 to L do
            A[I,J,K] := I*J*K;
```

В приведенном примере все циклы оканчиваются в одном и том же месте.

Второй вариант организации вложенного цикла — иерархическое расположение (каждый внутренний цикл расположен внутри предыдущего):

```
for I:=1 to N do //внешний цикл
begin
    . . . . .
    for J:=1 to M do // первый внутренний цикл
    begin
        . . . . .
        for K:=1 to L do //второй внутренний цикл
        begin
            . . . . .
        end; // конец второго внутреннего цикла
        . . . . .
    end; // конец первого внутреннего цикла
    . . . . .
end; // конец внешнего цикла
```

Следует иметь в виду, что во вложенном цикле параметры каждого из них изменяются одновременно, т. е. при очередном (фиксированном) значении параметра внешнего цикла параметр внутреннего последовательно принимает все возможные значения. Затем параметр внешнего цикла принимает следующее по порядку значение, при этом параметр внутреннего цикла вновь принимает все возможные значения. Поэтому вложенный цикл часто называют циклом с одновременно изменяющимися параметрами. Таким образом, если внутренний цикл должен выполняться M раз, а внешний — N раз, то операторы, стоящие во внутреннем цикле, выполнятся в общей сложности $M \cdot N$ раз, т. е. трудоемкость выполнения вложенных циклов может быть весьма высокой, что надо учитывать при разработке программ. Например, если $M = N = 100$, то операторы внутреннего цикла будут выполняться 10 000 раз:

```
//Вычисление и вывод таблицы
//умножения чисел от 1 до 100
for I:=1 to 100 do
    for J:=1 to 100 do
    begin
        K:=I*J; // умножение выполняется 10000 раз
        Write(K:6);
    end;
```

Во вложенных циклах допустимо передавать управление из внутреннего цикла в любую точку внешнего. Передача управления из внешнего цикла в произвольную точку внутреннего запрещена, так как в этом случае вход во внутренний цикл осуществляется не через его начало. Такая передача управления в языке Паскаль возможна только с использованием оператора безусловного перехода. Поскольку применение этого оператора не отвечает принципам структурного программирования, то его, как правило, в программах не используют, и, следовательно, подобные структуры встречаться не будут.

При организации вложенных циклов необходимо обращать внимание на правильное задание вложенности. В некоторых программах порядок вложенности циклов не оказывает влияния на правильность получаемого результата. Например, при вычислении суммы всех элементов матрицы порядок следования циклов может быть произвольным:

Вариант 1

```
S:=0;  
for I:=1 to M do  
  for J:=1 to N do  
    S:=S+A[I,J];
```

Вариант 2

```
S:=0;  
for J:=1 to N do  
  for I:=1 to M do  
    S:=S+A[I,J];
```

Однако при решении многих задач изменение порядка вложенности циклов приводит к неверному результату. Например, при вычислении суммы элементов каждой строки матрицы изменение порядка расположения циклов приведет к тому, что будут вычисляться суммы элементов столбцов.

Вариант 1 (правильный)

```
for I:=1 to M do  
  begin  
    S:=0;  
    for J:=1 to N do  
      S:=S+A[I,J];  
    WriteLn(S:7:2);  
  end;
```

Вариант 2 (неправильный)

```
for J:=1 to N do  
  begin  
    S:=0;  
    for I:=1 to M do  
      S:=S+A[I,J];  
    WriteLn(S:7:2);  
  end;
```

Необходимость организации вложенных циклов возникает при решении таких широко распространенных задач, как вычисление определенного интеграла с заданной точностью, вычисление экстремума функции на заданном интервале с заданной точностью. Рассмотрим решение этих задач.

5.1. Вычисление определенного интеграла с заданной точностью

Задача вычисления определенного интеграла формулируется следующим образом: вычислить $I = \int_a^b f(x)dx$ с точностью ϵ при известных значениях пределов интегрирования a, b , известной точности ϵ и заданной подынтегральной функции $f(x)$. При вычислении интеграла с точностью ϵ будут использоваться изложенные ранее методы прямоугольников, трапеций и парабол для определения значения интеграла при заданном числе разбиений интервала интегрирования. Для оценки погрешности вычисления интеграла на практике используют правило Рунге [4]. Суть правила состоит в том, что выполняют вычисление интеграла с двумя разными шагами изменения переменной x , а затем сравнивают результаты и получают оценку точности. Наиболее часто используемое правило связано с вычислением интеграла дважды: с шагом Δx и шагом $\Delta x/2$.

Для методов прямоугольников и трапеций погрешность $R_{\Delta x/2}$ вычисления интеграла с шагом $\Delta x/2$ оценивается следующей формулой:

$$|R_{\Delta x/2}| = \frac{|I_{\Delta x/2} - I_{\Delta x}|}{3},$$

где $I_{\Delta x/2}$ — значение интеграла, вычисленное с шагом $\Delta x/2$; $I_{\Delta x}$ — значение интеграла, вычисленное с шагом Δx .

Для метода Симпсона погрешность оценивается в соответствии со следующим выражением:

$$|R_{\Delta x/2}| = \frac{|I_{\Delta x/2} - I_{\Delta x}|}{15}.$$

Шаг интегрирования для методов прямоугольников и трапеций пропорционален $\sqrt{\epsilon}$, поэтому в [4] рекомендовано начальное число разбиений выбирать согласно следующему выражению: $n = \left\lceil \frac{b-a}{\sqrt{\epsilon}} \right\rceil + 1$, где $\lceil \cdot \rceil$ — целая часть. Для метода парабол шаг интегрирования пропорционален $\sqrt[4]{\epsilon}$, поэтому начальное число шагов рекомендовано выбирать из выражения $n = \left\lceil \frac{b-a}{2\sqrt[4]{\epsilon}} \right\rceil + 1$.

В программе вычисления интеграла с точностью ϵ во внутреннем цикле находят значение определенного интеграла при

заданном (фиксированном) числе разбиений интервала интегрирования. Во внешнем цикле производится сравнение значений интегралов, вычисленных при числе шагов, равных n и $2n$ соответственно. Если требуемая точность не достигнута, то число разбиений удваивается, а в качестве предыдущего значения интеграла берут текущее и вычисление интеграла выполняется при новом числе разбиений.

В дальнейшем в программах при вычислении значений аргумента используют формулу арифметической прогрессии $X = a + (i - 1)\Delta x$, а не формулу накопления суммы $X = X + \Delta x$, так как в первом случае меньше погрешность вычислений [9].

Пример программы вычисления определенного интеграла с заданной точностью методом трапеций:

```
program integraltrap;
{Вычисление определенного интеграла
 с заданной точностью методом трапеций.
 Подынтегральная функция - X*Exp(X) .
 Первообразная функция (x-1)*Exp(X) .}
{$APPTYPE CONSOLE}
uses
  SysUtils,
  Math;
function Rus(S:String):String;
var I:Byte;
begin
  Result:='';
  for I:=1 to Length(S) do
    case S[I] of
      'A'..'n': Result:=Result+Chr(Ord(S[I])-64);
      'p'..'я': Result:=Result+Chr(Ord(S[I])-16);
      'Ё': Result:=Result+Chr(240);
      'ё': Result:=Result+Chr(241);
      else
        Result:=Result+S[I];
      end;
  end;
end;

var
  A,B,Eps,I1,I2,Itoch,X,Dx,S1:Real;
  I,N,M,K:Integer;
begin
  WriteLn(Rus('Введите пределы интегрирования '),
    Rus('и точность'));
```

```

ReadLn (A,B,Eps) ;
//вычисление начального количества разбиений
N:=Trunc ((B-A)/Sqrt (Eps) )+1;
//полусумма значений функции на нижнем и
// верхнем пределах интегрирования
S1:=(A*Exp (A) + B*Exp (B) )/2.0;
//начальное значение интеграла, вычисленное
//при начальном количестве разбиений
Dx:=(B-A)/N;
I2:=0;
for I:=1 to N-1 do
begin
    X:=A+I*Dx; // текущее значение аргумента
    //вычисление суммы значений функции
    //в узлах интегрирования
    I2:=I2+X*Exp (X) ;
end;
I2:=(S1+I2)*Dx;
//вычисление количества позиций при выводе
//числа с заданной точностью
M:=Trunc (-Log10 (Eps) )+2;
K:=Trunc (Log10 (Abs (I2) ) )+M+3;
Writeln (Rus ('Начальное количество разбиений='),
    ,N:4,Rus (' Интеграл='),I2:K:M);
//цикл вычисления интеграла с точностью
repeat
    //Переменной I1 присваивается текущее
    //приближенное значение интеграла
    I1:=I2;
    //Переменной I2 присваивается полусумма
    //значений функции на концах интервала
    //интегрирования
    I2:=S1;
    //Удвоение количества разбиений
    N:=N*2;
    //Шаг изменения переменной интегрирования
    Dx:=(B-A)/N;
    //Внутренний цикл для вычисления суммы
    //значений функции при фиксированном
    //количестве шагов
    for I:=1 to N-1 do
    begin
        //Вычисление текущего значения аргумента
        X:=A+I*Dx;

```

```

//Накопление суммы значений функции
//в узлах интегрирования
I2:=I2+X*Exp(X);
end;
//Текущее значение интеграла
I2:=I2*Dx;
//Анализ точности вычисления
until Abs(I2-I1)/3<Eps;
WriteLn(Rus('Значение интеграла ='),I2:K:M
        ,Rus(' при количестве разбиений, равном ')
        , N);
//Точное значение интеграла
Itoch:=(B-1)*Exp(B) - (A-1)*Exp(A);
WriteLn(Rus('Точное значение интеграла равно ')
        ,Itoch:K:M);
ReadLn;
end.

```

Метод трапеций удобен для вычисления интеграла по правилу Рунге, так как при увеличении числа разбиений в два раза каждый второй узел представляет собой ранее рассматривавшийся, поэтому повторно значения функции в этих точках вычислять не нужно. Для этого сумму значений функции в этих узлах следует сохранять в переменной *s2*.

Фрагмент модифицированной программы приведен ниже. В отличие от первого варианта программы до вложенного цикла необходимо написать цикл вычисления сумм значений функции при начальном разбиении интервала интегрирования для всех узлов, кроме первого и последнего:

```

ReadLn(A,B,Eps);
N:=Trunc((B-A)/Sqrt(Eps))+1;
S1:=(A*Exp(A) + B*Exp(B))/2.0;
S:=0;
Dx:=(B-A)/N;
//Цикл вычисления суммы значений функции
//в узлах интегрирования при начальном
//разбиении интервала интегрирования.
for I:=1 to N-1 do
begin
  X:=A+I*Dx;
  S:=S+ X*Exp(X);
end;
I2:=Dx*(S+S1);

```

```

S2:=S;
repeat
  I1:=I2;
  N:=N*2;
  Dx:=(B-A)/N;
//Координата первого нового узла интегрирования.
  X1:=A+Dx;
  S:=0;
  //Цикл вычисления суммы значений функции
  //в новых узлах интегрирования.
  for I:=1 to N div 2 do
  begin
    X:=X1+2*(I-1)*Dx;
    S:=S+ X*Exp(X);
  end;
  //Сумма значений функции в узлах интегрирования
  //при новом числе разбиений.
  S2:=S+S2;
  I2:=(S1+S2)*Dx;
until Abs(I2-I1)/3<Eps;

```

Таким же образом можно сократить количество выполняемых операций при вычислении интеграла по методу левых или правых прямоугольников. Однако на практике при использовании метода прямоугольников значение функции вычисляют не на конце интервала (левом или правом), а в середине. Тогда при удвоении числа разбиений ранее вычисленные значения функции (и их сумму) использовать не удастся. Для использования результатов ранее выполнявшихся вычислений при применении метода прямоугольников следует количество разбиений увеличивать в три раза и вычислять значения функции в точках с абсциссами $X_i = X_{\text{нач}} + \Delta x i/6 + \Delta x(i-1)$, $i = 1, n$; $X_j = X_{\text{нач}} + 5\Delta x j/6 + \Delta x(j-1)$, $j = 1, n$. В приведенных выражениях $X_{\text{нач}}$ — нижний предел интегрирования, Δx — шаг, n — количество разбиений на предыдущей итерации.

При вычислении интеграла по методу Симпсона можно непосредственно использовать суммы значений функции, вычислявшиеся на предыдущей итерации. При этом надо иметь в виду, что узлы с нечетными номерами получают на очередной итерации четные индексы, и сумма значений функции для этих узлов должна браться с коэффициентом 2 (на предыдущей итерации с коэффициентом 4). Узлы с четными номерами и при новой итерации останутся четными, поэтому сумма значений в этих узлах берется с тем же коэффициентом 2.

5.2. Задания для самостоятельной работы

В таблице приведены подынтегральные функции. Вычислить значение интеграла каждым из трех методов (прямоугольников, трапеций, парабол) с заданной точностью. Пределы интегрирования брать из интервала, установленного заданием. Сравнить скорость сходимости используемых методов. Обеспечить в программе ввод точности, пределов интегрирования, коэффициентов, присутствующих в функциях, вывод с поясняющим текстом вычисленных значений интеграла каждым методом, количество разбиений при использовании каждого метода.

№ п/п	Подынтегральная функция	Интервал для выбора пределов интегрирования
1	$X^3 e^{X^2}$	1...5
2	$X e^{X^2}$	1...7
3	$X^2 e^{aX}$	1...6
4	$e^{aX} \sin pX$	1...5
5	$X e^{aX}$	1...7
6	$X^3(\operatorname{tg} X + \operatorname{ctg} X)$	2...3
7	$\operatorname{tg}^3 X$	0,9...1,5
8	$\operatorname{tg}^2 X$	1...1,52
9	$e^{\operatorname{tg} X + \operatorname{ctg} X}$	1,7...3
10	$\operatorname{ctg}^2 X$	0,1...0,8
11	$\operatorname{ctg}^3 X$	0,1...0,8
12	$\sin(X)X^4$	1...2
13	$\cos(X)X^4$	1...3
14	$\operatorname{tg}(X)X^2$	1...1,52
15	$e^X + e^{-X}$	1...4
16	$e^X - e^{-X}$	-4...4
17	$e^{\sqrt{X}}$	3...16
18	$X^{2/3} e^{aX}$	2...5
19	$X^{2\sqrt{X^2+X}}$	2...10
20	$(\operatorname{tg} X + \ln X)X^{5/2}$	1,1...1,52
21	$(\operatorname{ctg} X + X)X^{-13/4}$	0,1...1
22	$e^{\operatorname{tg} X}$	0,9...1,5

№ п/п	Подынтегральная функция	Интервал для выбора пределов интегрирования
23	X^X	3...7
24	$X^{\frac{8}{3}\sqrt{X^3+X}}$	2...10
25	$X^{13/2} \sin X$	1,1...2,5
26	$X^{\frac{7}{2}} \log_2(X^3 - X)$	2...8
27	$\operatorname{ctg} X$	0,1...0,7
28	$X^{2\sqrt{X}}$	3...6
29	$X^4 - X^3 + X^2$	1...8
30	$\operatorname{tg} X$	1...1,52

5.3. Вычисление наибольшего (наименьшего) значения функции с заданной точностью на заданном интервале

Необходимость организации вложенного цикла возникает также и при решении задачи нахождения наибольшего (наименьшего) значения функции на заданном интервале с заданной точностью. Рассмотрим сначала задачу определения с заданной точностью аргумента, при котором функция достигает своего экстремального значения.

Данная задача может быть сформулирована следующим образом. Заданы некоторая функция $f(x)$ и некоторый интервал $[a, b]$. Известно, что на заданном интервале функция имеет один экстремум, известен и вид экстремума (максимум или минимум). Требуется с заданной точностью ε найти значение аргумента, при котором достигается экстремум функции.

Решить поставленную задачу можно, используя прием программирования — вычисление максимума или минимума. При этом шаг изменения аргумента следует задать равным требуемой точности ε . Однако такой подход может потребовать большого количества вычислений. Сократить количество выполняемых операций можно за счет использования следующего алгоритма. Сначала вычисляют значения функции при *грубом* значении шага изменения аргумента. При этом очередное значение функции сравнивают с ранее вычисленным максимальным значением (до-

пустим, что определяется максимум функции); если текущее значение превышает максимальное, т. е. $f(x_i) > f_{\max}$, то $f_{\max} := f(x_i)$, и производится вычисление значения функции в следующей точке. Если же на очередном шаге будет выполнено условие $f(x_i) < f_{\max}$, то это означает, что максимум функции уже пройден, т. е. находится на интервале $(x_i - 2h, x_i)$, где h — шаг изменения аргумента. В этом случае шаг изменения аргумента уменьшается (обычно в два раза) и на вновь полученном интервале вычисляют значения функции и максимум при новом шаге изменения аргумента. Процесс продолжается до тех пор, пока h не станет меньше или равным ε .

Таким образом, во внутреннем цикле вычисляют значения функции и максимум на заданном интервале изменения аргумента при заданном шаге. Во внешнем цикле задают новый интервал поиска максимума и новый шаг изменения аргумента. Фрагмент программы вычисления максимума функции $y = 2x^3 + 10x^2 + 6x - 20$ в интервале $[a, b]$ имеет вид

```
//Вычисление максимума функции с использованием
//итерационного (вложенного) цикла
N2:=0; //Количество вычислений значения функции
ReadLn (A,B,Eps,H);
Xn:=A;
while H>Eps do
begin
  I:=0;
  //Установка начального значения для максимума
  Ymax:=( (2*Xn+10)*Xn+6)*Xn-20;
  Xmax:=Xn;
  //Цикл определения максимума функции
  //на очередном интервале изменения аргумента
  repeat
    I:=I+1;
    //Вычисление текущего значения аргумента
    X:=Xn+(I-1)*H;
    //Вычисление значения функции в очередной точке
    Y:=( (2*X+10)*X+6)*X-20;
    N2:=N2+1;
    //Определение нового максимума
    //и соответствующего значения аргумента
    if Y>Ymax then
      begin
        Ymax:=Y;
        Xmax:=X;
      end;
```

```
until (Y<Ymax);  
//Определение левого конца нового интервала  
//вычисления максимума  
Xn:= Xmax-H;  
//Уменьшение шага изменения аргумента в два раза  
H:=H/2;  
end;
```

Поиск минимума функции можно свести к поиску максимума, если функцию $f(x)$ заменить функцией $-f(x)$.

Рассматриваемую задачу обобщим на случай, когда функция не имеет внутри заданного интервала экстремум, а изменяется внутри него монотонно, причем заранее характер поведения функции не известен. Тогда следует говорить о нахождении наибольшего (наименьшего) значения функции на заданном интервале. Наибольшее (наименьшее) значение функции достигается либо в критической точке (где первая производная равна нулю), либо на конце интервала.

В этом случае в программу необходимо добавить проверку аргумента на принадлежность заданному интервалу. При отсутствии такой проверки аргумент может принимать значения, не принадлежащие заданному интервалу (если наибольшее значение достигается на левом конце интервала, то возможен выход аргумента за левую границу, если на правом конце интервала, то возможен выход аргумента за правый конец). Тогда добавляю еще одно условие выхода из цикла и проводят проверку нового значения для начального значения аргумента:

```
until (Y<Ymax) or (X>=B);  
//Определение левой границы нового интервала  
//вычисления наибольшего значения.  
Xn:= Xmax-H;  
//Проверка выхода нового значения аргумента  
//за пределы заданного интервала.  
if Xn<A then Xn:=A;
```

Если известны аналитические выражения для первой и второй производных, то можно найти значение аргумента, при котором первая производная обращается в нуль (решить уравнение, например, методом деления отрезка пополам), и определить знак второй производной в найденной точке. Знак второй производной позволит определить вид экстремума (максимум или минимум) или его отсутствие на заданном интервале. При отсутствии экстремума следует сделать запрос о нахождении наибольшего или наименьшего значения функции.

Пример программы вычисления наибольшего (наименьшего) значения функции приведен ниже. В программе реализованы рассмотренные подходы к определению наибольшего (наименьшего) значения функции на заданном интервале. Она позволяет также оценить трудоемкость последних двух способов на основе сравнения количества вычислений значений функции:

```
program Extremum;
{
Вычисление наибольшего(наименьшего) значения функции
 $y=2x^3+10x^2+6x-20$  в интервале  $[a,b]$  с заданной точностью
eps и начальным шагом изменения аргумента h.
 $y'=6x^2+20x+6$  - первая производная функции,
 $y''=12x+20$  - вторая производная
}
{$APPTYPE CONSOLE}
uses
  SysUtils;
var
  X,A,B,Aa,Bb,H,Y,Ymax,Ya,Yb,Ysr:Real;
  Eps,Xmax,Dx,Xn,Xsr:Real;
  I,N1,N2,K,Mon:Integer;
begin
  WriteLn('Введите a,b,eps,h');
  ReadLn(A,B,Eps,H);
  //Признак монотонности:
  //0 - немонотонная,
  //1 - монотонная.
  Mon:=0;
  //Определение точки на заданном интервале,
  //в которой первая производная функции
  //обращается в нуль (уточнение корня
  //уравнения методом деления отрезка пополам).
  Ya:= 6*A*A+20*A+6;
  if Ya=0 then
    Xmax:=A
  else
    begin
      Yb:=6*B*B+20*B+6;
      if Yb=0 then
        Xmax:=B
      else if Ya*Yb>0 then
        begin
          WriteLn ('Функция на заданном интервале '
            +'изменяется монотонно');
```

```
WriteLn ('Для нахождения наибольшего значения'
        , ' введите 1, для наименьшего - -1');
ReadLn(K);
Mon:=1;
end
else
begin
  Aa:=A;
  Bb:=B;
  Xsr:=(Aa+Bb)/2;
  Ysr:= 6*Xsr*Xsr+20*Xsr+6;
  while (Abs(Bb-Aa)>Eps)and(Ysr*Ya<>0) do
  begin
    if Ya*Ysr>0 then Aa:=Xsr else Bb:=Xsr;
    Xsr:=(Aa+Bb)/2;
    Ysr:= 6*Xsr*Xsr+20*Xsr+6;
  end;
  Xmax:=Xsr;
end;
end;
//Определение знака второй производной в критической
//точке с целью определения вида экстремума:
//вторая производная отрицательна - максимум,
//вторая производная положительна - минимум.
if Mon=0 then
begin
  if 12*Xmax+20<0 then
    K:=1
  else
    //Поиск минимума сводится к поиску
    //максимума функции -f(x)
    K:=-1;
  //Вычисление значения функции в критической точке
  Ymax:=K*((2*Xmax+10)*Xmax+6)*Xmax-20;
  case k of
    1: WriteLn('ymax=',k*ymax:10:5
              , ' xmax=',xmax:10:5);
    -1: WriteLn('ymin=',k*ymax:10:5
              , ' xmin=',xmax:10:5);
  end;
  ReadLn;
end;
//Поиск максимума функции с использованием
//табулирования значений функции при шаге
//изменения аргумента, равном требуемой точности.
```

```

N1:=Trunc ((B-A)/Eps)+1;
//Установка начального значения для максимума
Ymax:=((2*A+10)*A+6)*A-20;
Xmax:=A;
Dx:=Eps;
for I:=1 to N1-1 do
begin
    X:=A+I*Dx;
    Y:= K*(((2*X+10)*X+6)*X-20);
    if Y>Ymax then
    begin
        Ymax:=Y;
        Xmax:=X;
    end;
end;
case k of      1: WriteLn('ymax=',k*ymax:10:5
                    , ' xmax=',xmax:10:5);
    -1: WriteLn('ymin=',k*ymax:10:5
                    , ' xmin=',xmax:10:5);
end;
WriteLn('n1=',n1);
ReadLn;
//Вычисление наибольшего значения функции с
//использованием итерационного (вложенного) цикла.
N2:=0; //Количество вычислений значения функции
Xn:=A;
while N>Eps do
begin
    //Установка начального значения
    //для наибольшего значения
    Ymax:=K*(((2*Xn+10)*Xn+6)*Xn-20);
    Xmax:=Xn;
    I:=0;
    //Цикл определения наибольшего значения функции
    //на очередном интервале изменения аргумента
    repeat
        I:=I+1;
        //Вычисление текущего значения аргумента.
        X:=Xn+(I-1)*N;
        //Вычисление значения функции в очередной точке.
        Y:=K*(((2*X+10)*X+6)*X-20);
        N2:=N2+1;
        if Y>Ymax then
            //Определение нового наибольшего значения и
            //соответствующего значения аргумента.

```

```
begin
    Ymax:=Y;
    Xmax:=X;
end;
until (Y<Ymax) or (X>=B);
//Определение левой границы нового интервала
//вычисления наибольшего значения.
Xn:= Xmax-H;
//Проверка выхода аргумента за пределы интервала
if Xn<A then
    Xn:=A;
//Уменьшение шага изменения аргумента в два раза
H:=H/2;
end;
case k of
    1: WriteLn('ymax=',k*ymax:10:5
        , ' xmax=',xmax:10:5);
    -1: WriteLn('ymin=',k*ymax:10:5
        , ' xmin=',xmax:10:5);
end;
WriteLn('n1=',n1);
ReadLn;
end.
```

5.4. Задания для самостоятельной работы

Составить программу для вычисления экстремума функции на заданном интервале с заданной точностью.

1. Найти аналитическое выражение для первой производной заданной в таблице функции. Одним из известных методов уточнения корня уравнения определить значение аргумента на заданном интервале, при котором первая производная обращается в нуль. Вычислить значение функции в полученной точке. С помощью второй производной определить вид экстремума (максимум или минимум).

2. Найти с помощью итерационного алгоритма экстремум той же функции и значение аргумента, при котором он достигается.

Обеспечить ввод в программе значения точности, значений, определяющих интервал поиска экстремума, начальный шаг изменения аргумента. Для обоих вариантов поиска вывести значение аргумента, при котором достигается экстремум функции, и значение функции в полученной точке.

№ п/п	Функция	Интервал поиска экстремума
1	$X^2 + 1/X^2$	[0,2; 2]
2	$X^3 - X^2 - 6X$	[-2; 0]
3	$\cos(2x)/2 - \sin(x) + 2$	$[\pi/3; 2\pi/3]$
4	$X + 2\sqrt{10 - X}$	[0; 10]
5	$3X^4 - 8X^3 + 6X^2 + 2$	[0,1; 2]
6	$\cos(2x) - 2\cos(x)$	$[-\pi/6; \pi/3]$
7	$5^X - 4X$	[0; 2]
8	$X^3 - X^2 - 6X$	[0; 3]
9	$X^4 - 8X^2 - 9$	[-1; 1]
10	$(4^x + 4^{-x})/\ln(4)$	[-1; 2]
11	$2X^2 + 7 X - 4 + 5$	[-1; 2]
12	$X^2\sqrt{5 - X}$	[1; 5]
13	$X^2 + X - 4 $	[-3; 3]
14	$1/(X^2 + X + 2)$	[-2; 2]
15	$2X^3 + 15X^2 + 36X - 30$	[-4; -2,4]
16	$X^4 - 6X^3 + 8$	[1; 6]
17	$\sqrt{X(8 - X)}$	[0; 6]
18	$X^3 - 5X^2 + 3X$	[1,5; 5]
19	$2X^3 - 3X^2$	[-2; 0,5]
20	$\log_2(X^2 - 2X + 5)$	[0,2; 2]
21	$6X - X^2 - 6$	[0; 6]
22	$\log_3(2X + 3 - X^2)$	[0; 2]
23	$X^4 - 8X^2 - 9$	[-4; -0,5]
24	$3X^4 - 8X^3 + 6X^2 + 2$	[-2; 0,1]
25	$\cos(2x)/2 - \sin(x) + 2$	$[-\pi/3; 9\pi/10]$
26	$2X^3 + 15X^2 + 36X - 30$	[-2,4; 0]
27	$X^3 - 5X^2 + 3X$	[-1; 1]
28	$2X^3 - 3X^2$	[0,5; 3]
29	$X^4 - 8X^2 - 9$	[0,2; 3,5]
30	$\log_2(3 - 4X - 4X^2)$	[-1,2; 0]
31	$0,5^{x^2 - 2x}$	[0; 2]
32	$8^x - 6 \cdot 4^x - 3 \cdot 2^x$	[0; 3]
33	$\sqrt{5 - 2x + 3x^2 - x^3}/3$	[0; 2,5]
34	$\sqrt{5 - 2x + 3x^2 - x^3}/3$	[2,5; 4]

5.5. Обработка матриц

Матрицы представляют собой один из наиболее удобных *математических* объектов, обработка которых ведет к необходимости программирования вложенных циклов. При программной реализации обработки матриц будем использовать двумерные массивы (элементы которых имеют два индекса), считая, что элементы X_{ij} матрицы X , $i = 1, 2, \dots, m, j = 1, 2, \dots, n$ размещаются в ячейках соответствующего массива с такими же индексами. Это позволит сделать текст программы удобным для понимания процесса обработки и уменьшить в некоторых случаях объем вычислений. Первый индекс массива будем называть номером строки, а второй — номером столбца.

Предполагая соответствие элементов матрицы элементам массива, им дают, как правило, одинаковые имена. Иногда в пояснениях к программе вместо термина *массив* будем использовать термин *матрица*, как бы отождествлять объекты предметной области с объектами программы для лучшего понимания процессов обработки.

При постановке задач в целях обеспечения применимости программ для обработки матриц, например матрицы X , размеры которых не должны превосходить заданных значений ($m \leq 12$ и $n \leq 14$), будем использовать сокращенное обозначение $X(m, n)$, $m \leq 12, n \leq 14$. Указанные максимальные значения индексов матрицы будем использовать в объявлении *типа* соответствующего *двумерного массива*, а сами максимальные значения индексов желательно объявить в виде именованных констант. Если учесть, что двумерный массив можно рассматривать и объявлять как одномерный массив с базовым типом *массив*, то объявление типа массива для хранения матриц с размерами, не превышающими 12×14 , будет таким:

```
const
    mMax=12; nMax=14;
type
    //Объявление типа строки двумерного массива типа tM
    tV=array[1..nMax] of Real;
    //Объявление типа двумерного массива
    tM=array[1..mMax] of tV;
```

(при объявлении типа двумерного массива необходимо задавать типы обоих индексов и самих элементов матрицы).

Двухэтапное объявление типа двумерного массива удобно тем, что строка массива будет совместима по присваиванию с одномерным массивом того же типа `tV`.

Задание в разделе констант максимальных значений для количества строк и столбцов улучшает стиль программирования, так как при необходимости изменения количества строк или столбцов программисту не придется внимательно изучать весь текст, а достаточно будет изменить значения соответствующих констант.

Переменная — двумерный массив — объявляется в разделе объявления переменных

```
var
```

```
  X:tM;
```

Менее предпочтительными, но допустимыми будут объявления типа

```
type
```

```
  tM=array[1..mMax, 1..nMax] of Real;
```

или

```
type
```

```
  tM=array[1..12, 1..14] of Real;
```

и объявление массива одной строкой

```
var
```

```
  X:array[1.. mMax, 1..nMax] of Real;
```

или

```
var
```

```
  X:array[1..12, 1..14] of Real;
```

Последние варианты менее предпочтительны, так как при использовании в подпрограммах типизированных параметров, представляющих матрицы и векторы, без предварительного объявления используемых типов обойтись невозможно.

При работе с матрицами надо помнить, что первый индекс соответствует номеру строки, а второй — номеру столбца.

Ввод-вывод матриц можно осуществить только с использованием вложенного цикла. Обычно принято вводить матрицу в виде матрицы, т. е. по строкам, как это принято записывать на бумаге. В этом случае во внутреннем цикле должен обеспечиваться ввод (вывод) одной строки матрицы. Внешний цикл будет задавать ввод (вывод) всех строк матрицы и переход в начало новой строки после ввода (вывода) очередной.

Фрагмент программы, обеспечивающей ввод и вывод матрицы $X(m, n)$, $m \leq 12$, $n \leq 14$, имеет следующий вид:

```
WriteLn('Введите количество строк и столбцов матрицы');
ReadLn(M,N);
WriteLn('Введите элементы матрицы по строкам');
for I:=1 to M do //Цикл ввода всех строк матрицы
begin
    //Цикл ввода очередной строки матрицы
    for J:=1 to N do
        Read(X[I,J]);
    //Переход в начало новой строки файла ввода
    ReadLn;
end;
WriteLn('Введенная матрица');
//Цикл вывода всех строк матрицы
for I:=1 to M do
begin
    //Цикл вывода очередной строки матрицы
    for J:=1 to N do
        Write(X[I,J]:6:1, ' ');
    //Переход в начало новой строки файла вывода
    WriteLn;
end;
```

Порядок расположения циклов при вводе-выводе имеет существенное значение. Если поменять местами заголовки циклов, то вводимые значения будут присваиваться элементам столбца, а не строки. В этом случае будет выведена фактически транспонированная матрица.

Также следует соблюдать аккуратность при использовании приемов программирования. Например, при вычислении сумм и произведений нужно следить за тем, в каком месте программы присваиваются начальные значения соответствующим переменным.

Приведенный ниже фрагмент программы позволяет вычислить сумму и произведение всех элементов матрицы:

```
S:=0; Pr:=1;
for I:=1 to M do
    for J:=1 to N do
        begin
            S:=S+A[I,J];
            Pr:=Pr*A[I,J];
        end;
WriteLn('s=',S:8:2, ' pr=',Pr:8:2);
```

При изменении места расположения операторов присваивания

S:=0; Pr:=1; СМЫСЛ ВЫЧИСЛЕНИЙ МЕНЯЕТСЯ:

```
for I:=1 to M do
begin
  S:=0; Pr:=1;
  for J:=1 to N do
  begin
    S:=S+A[I,J];
    Pr:=Pr*A[I,J];
  end;
  WriteLn('s=',S:8:2, ' pr=',Pr:8:2);
end;
```

Данный фрагмент программы позволит вычислить сумму и произведение элементов каждой строки матрицы, а не всех ее элементов.

При вычислении максимального и минимального элементов матрицы надо правильно устанавливать начальные значения вычисляемых переменных, а также правильно задавать пределы изменения параметров циклов:

```
Amax:=A[1,1]; Imax:=1; Jmax:=1;
Amin:=A[1,1]; Imin:=1; Jmin:=1;
for I:=1 to M do
  for J:=1 to N do
    if A[I,J]>Amax then
      begin
        Amax:=A[I,J];
        Imax:=I;
        Jmax:=J;
      end
    else if A[I,J]<Amin then
      begin
        Amin:=A[I,J];
        Imin:=I;
        Jmin:=J;
      end;
  end;
  WriteLn('amax=',Amax:8:2, ' imax=',Imax:2,
    ' jmax=',Jmax:2, ' amin=',Amin:8:2,
    ' imin=',Imin:2, ' jmin=',Jmin:2,);
```

В приведенном фрагменте программы нельзя опускать операторы присваивания начальных значений индексам минимального и максимального элементов (Imax:=1; Jmax:=1; Imin:=1; Jmin:=1;). В

случае отсутствия указанных операторов индексы искомым элементов будут не определены, если первый элемент $A[1, 1]$ является максимальным (минимальным).

При поиске минимального (максимального) элемента матрицы нельзя устанавливать начальное значение параметра цикла, равное двум, как это можно сделать при обработке одномерного массива. Следующий фрагмент программы будет приводить к неверному результату, так как будут исключены из рассмотрения все элементы первой строки, кроме первого, значение которого выбирается в качестве начального для максимума и минимума:

```
max:=A[1,1]; Imax:=1; Jmax:=1;
min:=A[1,1]; Imin:=1; Jmin:=1;
for I:=2 to M do
  for J:=1 to N do
    .....
```

Рассмотрим примеры обработки матриц.

5.6. Примеры выполнения задания на обработку матриц

1. В матрице $A(m, n)$, $m \leq 12$, $n \leq 10$ поменять местами строки с наибольшей и наименьшей суммами элементов.

Для решения поставленной задачи необходимо вычислить суммы элементов каждой строки матрицы, однако не требуется запоминать все эти суммы. Поэтому после вычисления суммы элементов очередной строки матрицы следует сравнить полученное с текущими значениями максимальной и минимальной сумм, изменить при необходимости текущие значения максимума и минимума на только что вычисленное, а также запомнить номер строки, для которой эта сумма минимальна или максимальна.

После нахождения номеров строк, подлежащих обмену местами, выполняется собственно обмен. Поскольку строка матрицы представляет собой одномерный массив (первый индекс имеет фиксированное значение), то эта операция выполняется с использованием обычного (не вложенного) цикла, а обмен местами — с промежуточной простой переменной b :

```
program Matr1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
const
  Mm=12; Nn=10;
```

```

type
  matr=array[1..Mm,1..Nn] of Real;
var
  A:matr;S,Smax,Smin,B:Real;
  Imax,Imin,I,J,M,N:Integer;
  .....
begin
  WriteLn('Введите количество строк и столбцов');
  ReadLn(M,N);
  WriteLn('Введите матрицу по строкам');
  for I:=1 to M do
    begin
      for J:=1 to N do
        Read(A[I,J]);
      ReadLn;
    end;
  WriteLn('Исходная матрица');
  for I:=1 to M do
    begin
      for J:=1 to N do
        Write(A[I,J]:6:1,' ');
      WriteLn;
    end;
  //Задание начальных значений
  Smax:=-1e30; //Очень маленькое число
  Smin:=1e30;  //Очень большое число
  //Строго говоря, самое маленькое и самое большое
  //значения для используемого типа данных
{
Другой вариант - нахождение суммы элементов
первой строки и задание этого значения для
максимума и минимума:
  S:=0;
  for J:=1 to N do
    S:=S+A[1,J];
  Smax:=S; Imax:=1;
  Smin:=S; Imin:=1;
Далее строки можно рассматривать, начиная со второй
}
  //Цикл поиска строк с максимальной и минимальной
  //суммами элементов
  for I:=1 to M do

```

```

begin
  S:=0; //Задание начального значения суммы
  //Цикл вычисления суммы элементов
  //очередной строки
  for J:=1 to N do
    S:=S+A[I,J];
  if S>Smax then
    begin
      Smax:=S;
      Imax:=I;
    end
  if S<Smin then
    begin
      Smin:=S;
      Imin:=I;
    end;
  end;
end;
//Цикл обмена местами найденных строк
for J:=1 to N do
  begin
    B:=A[Imin,J];
    A[Imin,J]:=A[Imax,J];
    A[Imax,J]:=B;
  end;
WriteLn('Полученная матрица');
for I:=1 to M do
  begin
    for J:=1 to N do
      Write(A[I,J]:6:1, ' ');
    WriteLn;
  end;
WriteLn('imin=', Imin:2, ' imax=', Imax);
ReadLn;
end.

```

Если матрицу описать как одномерный массив, элементами которого являются одномерные массивы, то обмен местами найденных строк может быть записан короче, без использования цикла:

```

type
  mas=array[1..Nn] of Real;
  matr=array[1..Mm] of mas;

```



```
var
  A:matr; S,Smax,Smin:Real; B:mas;
.....
  B:=A[Imin];
  A[Imin]:=A[Imax];
  A[Imax]:=B;
.....
```

2. Удалить из матрицы $B(m, n)$, $m \leq 10$, $n \leq 8$ строки, содержащие отрицательные элементы. Можно предложить два варианта решения задачи.

Если понимать условие буквально, то в случае наличия отрицательных элементов в очередной строке матрицы следует удалить именно эту строку. Выполняется эта операция путем перезаписи всех строк, расположенных после удаляемой, на одну строку в сторону уменьшения номеров строк и общего количества строк матрицы на единицу. В ходе анализа надо учитывать, что после удаления очередной строки на ее место переписывается новая, которая еще не анализировалась, поэтому номер строки изменять не следует. Если же текущая строка не удалялась, для перехода к следующей строке текущий номер следует увеличить на единицу. Поскольку в этом случае изменяется количество строк, а эта величина является конечным значением параметра цикла, то, учитывая правила организации цикла с параметром, использовать здесь такой цикл нельзя (внутри цикла нельзя изменять начальное, конечное и текущее значения параметра цикла). Поэтому следует использовать цикл с пред- или постусловием. Внутренний цикл, в котором определяется наличие или отсутствие отрицательных элементов в очередной строке матрицы, является циклом с заранее не известным числом повторений. Он должен выполняться, пока не обнаружен отрицательный элемент и не проверены все элементы строки.

Согласно второму варианту формально производится не удаление строк, содержащих отрицательные элементы, а оставление в матрице строк, не содержащих отрицательных элементов.

В представленной ниже программе использован метод, в котором очередная i -я строка, не содержащая отрицательных элементов, переписывается в начало матрицы на место k -й строки, где k — количество строк без отрицательных элементов на данный момент времени, включая найденную, если $i > k$, иначе строка остается на месте.

Фрагмент программы имеет следующий вид:

```

. . . . .
K:=0;
for I:=1 to M do
begin
  Pr:=True; //Отрицательных элементов нет
  J:=1;
  while Pr and (J<=N) do
    if B[I,J]<0 then
      Pr:=False
    else
      J:=J+1;
  if Pr and (I>K+1) then
  begin
    K:=K+1;
    for J:=1 to N do
      B[K,J]:=B[I,J];
    end;
  end;
  if K>0 then
  begin
    WriteLn('Полученная матрица');
    for I:=1 to K do
    begin
      for J:=1 to N do
        Write(B[I,J]:6:1, ' ');
      WriteLn;
    end;
  end
  else
    WriteLn('Полученная матрица пустая');
. . . . .

```

3. В инженерной деятельности часто приходится выполнять операцию умножения матриц. Напомним, что очередной элемент результирующей матрицы C вычисляется согласно следующей формуле:

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

где A , B — умножаемые матрицы; i, j — индексы очередного элемента матрицы C ; n — количество столбцов первой матрицы (строк второй матрицы).

Перемножение матриц возможно только в том случае, если количество столбцов первой матрицы равно количеству строк

второй. Умножение матриц приводит к необходимости программирования тройного цикла. Это связано с тем, что один элемент, представляющий собой сумму попарных произведений соответствующих элементов исходных матриц, определяется в цикле, а вычисление всех элементов матрицы требует организации еще одного вложенного цикла. Следующий фрагмент программы демонстрирует решение рассматриваемой задачи для матриц $A(m, n)$, $B(n, n_2)$, $m \leq 10$, $n \leq 8$, $n_2 \leq 12$:

```
program Matr3;
{$APPTYPE CONSOLE}
uses
  SysUtils;
const
  Mm=10;Nn=8; Nn2=12;
type
  matr1=array[1..Mm,1..Nn] of Real;
  matr2=array[1..Nn,1..Nn2] of Real;
  matrres=array[1..Mm,1..Nn2] of Real;
var
  A:matr1;B:matr2;C:matrres;
  I,J,M1,N1,M2,N2,K:Integer;
  S:Real;
. . . . .
begin
  //Ввод m1,n1, m2, n2 и матриц A и B
  . . . . .
  if N1<>M2 then
    WriteLn('Умножение матриц невозможно')
  else
    begin
      for I:=1 to M1 do
        for J:=1 to N2 do
          begin
            S:=0;
            for K:=1 to N1 do
              S:=S+A[I,K]*B[K,J];
            C[I,J]:=S;
          end;
        //Вывод матрицы C
      . . . . .
    end;
    ReadLn;
  end.
```

Наряду с прямоугольными матрицами часто приходится иметь дело и с квадратными, в которых количество строк равно количеству столбцов. В случае квадратных матриц часто необходимо обрабатывать элементы, расположенные в определенном месте (на главной диагонали, на побочной, под главной диагональю и т. д.). Чтобы избежать ненужных проверок индексов элементов матрицы, целесообразно знать закон изменения индексов для элементов, расположенных в определенном месте. Для матрицы A , имеющей n строк и столбцов, можно записать

- диагональные элементы имеют одинаковые индексы:
 $a[i, i], i = 1, n;$
- элементы, стоящие над главной диагональю (образующие верхнюю треугольную матрицу), не включая диагональ,
 $a[i, j], i = 1, n - 1; j = i + 1, n;$
- элементы, стоящие над главной диагональю, включая диагональ,
 $a[i, j], i = 1, n; j = i, n;$
- элементы, стоящие под главной диагональю (образующие нижнюю треугольную матрицу), не включая диагональ,
 $a[i, j], i = 2, n; j = 1, i - 1;$
- элементы, стоящие под главной диагональю, включая диагональ,
 $a[i, j], i = 1, n; j = 1, i;$
- элементы побочной диагонали
 $a[i, n - i + 1], i = 1, n;$
- элементы, стоящие над побочной диагональю, не включая диагональ,
 $a[i, j], i = 1, n - 1; j = 1, n - i;$
- элементы, стоящие над побочной диагональю, включая диагональ,
 $a[i, j], i = 1, n; j = 1, n - i + 1;$
- элементы, стоящие под побочной диагональю, не включая диагональ,
 $a[i, j], i = 2, n; j = n - i + 2, n;$
- элементы, стоящие под побочной диагональю, включая диагональ,
 $a[i, j], i = 1, n; j = n - i + 1, n.$

4. В квадратной матрице $F(k, k)$, $k \leq 9$ определить, что больше — модуль минимального отрицательного элемента, стоящего над побочной диагональю, или максимальный положительный элемент, стоящий под побочной диагональю. Ниже представлен фрагмент программы:

```
program Matr4;
{$APPTYPE CONSOLE}
uses
  SysUtils;
const
  Mm=9;
type
  matr=array[1..Mm,1..Mm] of Real;
var
  F:matr;
  I,J,M:Integer;
  Max,Min:Real;
  Pol,Otr:boolean;
  .....
begin
  Min:= 0; Otr:=False;
  for I:=1 to M-1 do
    for J:=1 to M-I do
      if (F[I,J]<Min) then
        begin
          Min:=F[I,J];
          Otr:=True;
        end;
  Max:= 0; Pol:=False;
  for I:=2 to M do
    for J:=M-I+2 to M do
      if (F[I,J]>Max) then
        begin
          Max:=F[I,J];
          Pol:=True;
        end;
  if not Otr then
    WriteLn('Отрицательных элементов нет')
  else if not Pol then
    WriteLn('Положительных элементов нет')
  else
    begin
      if Abs(Min)>Max then
        WriteLn('Модуль наименьшего отрицательного '
          +' больше наибольшего положительного '
          + ' Min=',Min:6:1, ' Max=',Max:6:1);
      if Abs(Min)<Max then
```

```
WriteLn('Модуль наименьшего отрицательного '  
      +' меньше наибольшего положительного '  
      , ' Min=',Min:6:1, ' Max=',Max:6:1);  
if Abs(Min)=Max then  
  WriteLn('Модуль наименьшего отрицательного '  
        +' равен наибольшему положительному '  
        , ' Min=',Min:6:1, ' Max=',Max:6:1);  
end;  
ReadLn;  
end.
```

5.7. Задания для самостоятельной работы

1. Составить программу, которая в матрице $A(m, n)$, $m \leq 10$, $n \leq 12$ меняет местами строку, содержащую максимальный элемент, со строкой, содержащей минимальный элемент. Предполагается, что искомые элементы единственные. Вывести исходную и преобразованную матрицы, минимальный и максимальный элементы, а также номера строк, в которых они расположены. Если минимальный и максимальный элементы расположены в одной строке, то поменять местами столбцы, содержащие эти элементы.

2. Составить программу, которая в каждой строке матрицы $B(m, n)$, $m \leq 10$, $n \leq 12$ находит модуль суммы отрицательных элементов и сумму положительных элементов. Из найденных элементов сформировать матрицу $C(m, 3)$, в каждой строке которой первые два элемента — найденные суммы, а третий элемент равен: минус единице, если первая сумма больше второй; нулю, если они равны; единице, если вторая сумма больше первой. Вывести исходную и полученную матрицы так, чтобы в каждой строке сначала располагалась строка исходной матрицы, а затем строка полученной.

3. Составить программу, которая в каждой строке матрицы $D(m, n)$, $m \leq 10$, $n \leq 12$ находит элемент, где модуль разности этого элемента и среднего арифметического элементов строки минимален. Вывести исходную матрицу так, чтобы после элементов строки матрицы располагались найденный элемент, среднее арифметическое и модуль их разности.

4. Составить программу, которая в матрице $K(m, n)$, $m \leq 10$, $n \leq 12$ меняет местами строки, содержащие максимальное количество четных и нечетных элементов. Если во всех строках

эти количества одинаковы, то поменять местами первую и последнюю строки матрицы. Вывести исходную и преобразованную матрицы, найденные количества и номера найденных строк.

5. Составить программу, которая в квадратной матрице $F(m, m)$, $m \leq 10$ находит сумму всех элементов верхней треугольной матрицы, больших всех элементов нижней. Вывести исходную матрицу и найденную сумму. Если верхняя треугольная матрица не содержит нужных элементов, то выдать соответствующее сообщение.

6. Составить программу, которая в каждой строке матрицы $H(m, n)$, $m \leq 10$, $n \leq 12$ находит максимальное из произведений вида h_{i1} , $h_{i1}h_{i2}$, $h_{i1}h_{i2}h_{i3}$, ..., $(h_{i1}h_{i2} \dots h_{in})$. Вывести исходную матрицу и рядом с каждой строкой найденное максимальное значение произведения.

7. Составить программу, которая в каждой строке матрицы $G(m, n)$, $m \leq 10$, $n \leq 12$ находит суммы элементов, расположенных до и после максимального элемента. Если сумма не может быть вычислена (нет элементов до или после максимального), то считать ее равной нулю. Вывести исходную матрицу, располагая в одной строке элементы строки матрицы, после которых вывести номер столбца максимального элемента и найденные суммы.

8. Составить программу, которая находит в каждой строке матрицы $S(k, l)$, $k \leq 12$, $l \leq 15$ самую длинную последовательность отрицательных чисел и произведение ее элементов. Если строка не содержит отрицательных чисел, то считать произведение равным нулю. Вывести исходную матрицу, располагая в одной строке элементы строки матрицы, после которых — длину найденной последовательности и ее произведение.

9. Составить программу, которая находит в каждой строке матрицы $P(k, l)$, $k \leq 12$, $l \leq 14$ суммы элементов с нечетными номерами столбцов и с четными. Найти максимальное значение первых сумм и минимальное вторых сумм. Вывести исходную матрицу, располагая в одной строке элементы строки матрицы, затем найденные суммы, максимальное и минимальное значения.

10. Составить программу, которая находит в каждой строке матрицы $Q(k, l)$, $k \leq 12$, $l \leq 14$ произведение элементов, расположенных между минимальным и максимальным элементами этой строки. Если произведение вычислить нельзя (нет элементов меж-

ду минимальным и максимальным), то считать его равным нулю. Вывести исходную матрицу, располагая в одной строке элементы строки матрицы, затем найденное произведение, минимальное и максимальное значения.

11. Составить программу, которая находит в каждой строке матрицы $Q(k, l)$, $k \leq 12$, $l \leq 14$ сумму положительных элементов, расположенных между первым и последним отрицательными элементами этой строки. Если сумму вычислить нельзя (нет положительных элементов между первым и последним отрицательными элементами), то считать ее равной нулю. Вывести исходную матрицу, располагая в одной строке элементы строки матрицы, затем найденную сумму, первый и последний отрицательные элементы.

12. Составить программу, которая находит в каждой строке матрицы $Q(k, l)$, $k \leq 12$, $l \leq 14$ среднее арифметическое максимального отрицательного и минимального положительного элементов. Найти максимальное среднее арифметическое и номер строки, для которой оно получено. Если среднее вычислено быть не может (нет отрицательных или положительных элементов в строке), то считать его равным нулю. Вывести исходную матрицу, располагая в одной строке элементы строки матрицы, затем найденные максимальное и минимальное, их среднее арифметическое. Под матрицей вывести максимальное среднее арифметическое и номер строки.

13. Составить программу, которая в каждой строке матрицы $D(m, n)$, $m \leq 10$, $n \leq 12$ находит элементы, где сумма предшествующих элементов больше суммы последующих. Для первого элемента сумму предшествующих элементов считать равной нулю. Для последнего элемента сумму последующих элементов считать равной нулю. Вывести матрицу в виде матрицы, располагая рядом с каждой строкой найденные элементы.

14. Составить программу, которая в каждой строке матрицы $D(m, n)$, $m \leq 10$, $n \leq 12$ находит элемент, для которого модуль разности элемента и среднего геометрического модулей всех элементов строки максимален. Предполагается, что матрица нулевых элементов не содержит. Вывести матрицу в виде матрицы, располагая рядом с каждой строкой найденный элемент и модуль искомой разности.

15. Составить программу, которая в матрице $D(m, n)$, $m \leq 10$, $n \leq 12$ находит все элементы, где сумма всех элементов строки,

стоящих до рассматриваемого, больше суммы элементов столбца, стоящих до рассматриваемого. Сумму предшествующих элементов считать равной нулю, если элемент является первым в строке или в столбце. Сформировать из найденных элементов массив. Вывести матрицу в виде матрицы, а под ней — элементы массива.

16. Составить программу, находящую в матрице $D(m, n)$, $m \leq 10$, $n \leq 12$ все элементы, модуль которых располагается в интервале между средним геометрическим и средним арифметическим модулей всех элементов матрицы. Из найденных элементов сформировать одномерный массив. Вывести матрицу в виде матрицы, а под ней — элементы массива. Предполагается, что матрица нулевых элементов не содержит.

17. Составить программу, находящую в матрице $D(m, n)$, $m \leq 10$, $n \leq 12$ элемент, для которого сумма его четырех ближайших соседей (двух элементов, стоящих перед ним в строке и в столбце, и двух, стоящих после него в строке и в столбце) максимальна. Если соседний элемент отсутствует, то считать его равным нулю. Вывести матрицу в виде матрицы, а под ней — найденный элемент, номера строки, столбца и сумму.

18. Составить программу, находящую в матрице $D(m, n)$, $m \leq 10$, $n \leq 12$ для элементов, сумма индексов которых нечетна, максимальный элемент и сумму элементов. Найти также максимальный элемент и сумму элементов, для которых сумма индексов четна. Вывести матрицу в виде матрицы, а под ней — найденные максимальные элементы, их индексы и две суммы.

19. Составить программу, находящую в матрице $D(m, n)$, $m \leq 10$, $n \leq 12$ все элементы, для которых максимальный среди предшествующих элементов строки, где стоит элемент, превышает максимальный среди предшествующих элементов столбца, где расположен элемент. Если предшествующие элементы отсутствуют, то считать максимальный равным нулю. Найденные элементы переписать в одномерный массив. Вывести матрицу в виде матрицы, а под ней — элементы сформированного массива.

20. Составить программу, которая в матрице $K(m, n)$, $m \leq 10$, $n \leq 12$ меняет местами строки, содержащие максимальный элемент, без остатка делящийся на заданное число L , и минимальный элемент, без остатка делящийся на то же число L . Если найденные элементы расположены в одной строке, то поменять местами столбцы, в которых они расположены. Если требуемых

элементов нет или он единственный, то поменять местами первую и последнюю строки матрицы. Вывести исходную и преобразованную матрицы, найденные элементы и их индексы.

21. Составить программу, которая в квадратной матрице $F(m, m)$, $m \leq 10$ находит одноименные строки и столбцы с равными суммами элементов. Номера найденных строк запомнить в массиве. Дополнительные массивов для сохранения значений сумм не использовать. Вывести исходную матрицу, номера найденных строк и их сумм. Если требуемые строки и столбцы отсутствуют, то выдать соответствующее сообщение.

22. Составить программу, находящую в матрице $D(m, n)$, $m \leq 10$, $n \leq 12$ номера строк, в которых максимальный среди элементов с четными индексами столбцов совпадает с максимальным из элементов с нечетными индексами столбцов. Вывести исходную матрицу, номера найденных строк и максимальные элементы. Если требуемых строк нет, то выдать соответствующее сообщение.

23. Составить программу, находящую в квадратной матрице $F(m, m)$, $m \leq 10$ произведение всех элементов нижней треугольной матрицы, которые меньше минимального элемента верхней. Вывести исходную матрицу и найденное произведение. Если нижняя треугольная матрица не содержит нужных элементов, то выдать соответствующее сообщение.

24. Составить программу, находящую в матрице $Q(k, l)$, $k \leq 12$, $l \leq 14$ все строки, произведение элементов которых больше их суммы. Определить среди найденных строку, для которой разность произведения и суммы максимальна. Вывести исходную матрицу, располагая рядом с элементами каждой строки найденные сумму и произведение. Вывести под матрицей номера найденных строк, номер строки с максимальной разностью или сообщение об отсутствии искомых строк.

25. Составить программу, находящую в матрице $D(m, n)$, $m \leq 10$, $n \leq 12$ номера строк, в которых каждый элемент больше максимального из того же столбца, расположенных до рассматриваемого. Вывести исходную матрицу, номера найденных строк. Если требуемых строк нет, то выдать соответствующее сообщение.

26. Составить программу, которая в матрице $D(m, n)$, $m \leq 10$, $n \leq 12$ находит номера строк с максимальным и минимальным значениями среднего квадратического отклонения. Среднее квадратическое отклонение элементов i -й строки вычисляют по сле-

дующей формуле: $\sigma_i = \sqrt{\frac{\sum_{j=1}^n (d_{ij} - d_{icp})^2}{n}}$, где $d_{icp} = \sum_{j=1}^n d_{ij} / n$. Вывести исходную матрицу, номера найденных строк и значения найденных минимального и максимального средних квадратических. Дополнительные массивы не использовать.

5.8. Методы сортировки массивов

Для *сортировки (упорядочения)* по возрастанию или убыванию значений в массиве разработано множество методов [6, 8]. Рассмотрим три из них, считая, для определенности, что первые n ($n = 6$) элементов массива X

X_1	X_2	X_3	X_4	X_5	X_6
34	21	15	18	25	40

требуется упорядочить по возрастанию, не используя дополнительных массивов.

Метод включения с сохранением упорядоченности (метод прямого включения или сортировка вставками)

Сортировка начинается со сравнения чисел в первой X_1 и второй X_2 ячейках массива. Если окажется, что $X_1 > X_2$, то их значения меняют местами:

X_1	X_2	X_3	X_4	X_5	X_6
21	34	15	18	25	40

На каждом следующем i -м шаге, $i = 2, 3, \dots, n - 1$, значение из $(i + 1)$ -й ячейки массива путем обмена положением с числом из предыдущей ячейки продвигают в сторону уменьшения индекса ячейки до тех пор, пока не окажется, что в предыдущей ячейке находится не большее число.

Из сказанного следует, что при реализации метода прямого включения внешний цикл должен выполняться $n - 1$ раз, а максимально возможное число повторений внутреннего цикла, в теле которого должны выполняться сравнения и перестановки чисел, будет увеличиваться от 1 до $n - 1$. Однако внутренний цикл следует организовать так, чтобы он заканчивался или вообще не выполнялся при наступлении условия: значение в предыдущей ячейке массива не больше, чем в текущей.

В нашем примере:

– при $i = 2$ число 15 из ячейки X_3 последовательно обменивается местами с числом 34 из ячейки X_2 , а затем с числом 21 из ячейки X_1 :

X_1	X_2	X_3	X_4	X_5	X_6
15	21	34	18	25	40

– при $i = 3$ число 18 из ячейки X_4 последовательно обменивается местами с числом 34 из ячейки X_3 , а затем с числом 21 из ячейки X_2 :

X_1	X_2	X_3	X_4	X_5	X_6
15	18	21	34	25	40

– при $i = 4$ число 25 из ячейки X_5 обменивается местами с числом 34 из ячейки X_3 :

X_1	X_2	X_3	X_4	X_5	X_6
15	18	21	25	34	40

– при $i = 5$ число 40 из ячейки X_6 не изменит своего положения, так как больше числа в ячейке X_5 :

X_1	X_2	X_3	X_4	X_5	X_6
15	18	21	25	34	40

Ниже представлен фрагмент программы упорядочения по возрастанию первых n элементов массива X методом *прямого включения* (включения с сохранением упорядоченности):

```
for i:=1 to n-1 do
begin
  j:=i;
  while (X[j+1]<X[j]) and (j>0) do
  begin
    R:=X[j];
    X[j]:=X[j+1];
    X[j+1]:=R;
    j:=j-1;
  end;
end;
```

Другая реализация метода основана не на перестановке соседних, а на сдвиге данных в массиве. На очередном шаге $i = 2, 3, \dots, n$: 1) копируют значение X_{i-k} в дополнительную переменную, например R ; 2) начиная с $k = 1$, пока $X_{i-k} > R$ и $k < i$, копируют X_{i-k} в X_{i-k+1} и увеличивают k на единицу; 3) копируют в X_{i-k+1} значение R .

Метод прямого обмена (метод пузырька)

Этот метод, как и предыдущий, основан на обмене значениями соседних ячеек массива, но с первого же шага в последовательном анализе при движении от одного конца массива к другому участвуют все пары соседних ячеек массива.

На первом шаге последовательно для $j = n, n - 1, \dots, 2$ сравнивают значения соседних ячеек массива, и при соблюдении условия $X_j < X_{j-1}$ выполняется их перестановка, в результате чего наименьшее число оказывается в ячейке X_1 .

В нашем примере после выполнения первого шага данные в массиве расположатся так:

X_1	X_2	X_3	X_4	X_5	X_6
15	34	21	18	25	40

На втором шаге процесс повторяется, но только для $j = n, n - 1, \dots, 3$:

X_1	X_2	X_3	X_4	X_5	X_6
15	18	34	21	25	40

На каждом следующем шаге число проверяемых пар ячеек будет уменьшаться на единицу. В общем случае на любом шаге i ($i = 1, 2, 3, \dots, n - 1$) процесс будет выполняться для j от n до $i + 1$, в частности, при $i = n - 1$ только один раз для n -й и $(n - 1)$ -й ячеек.

Из сказанного следует, что при реализации метода прямого обмена внешний цикл должен повторяться $n - 1$ раз, а число выполнений внутреннего цикла, в теле которого должны происходить сравнения и перестановки чисел, будет уменьшаться от $n - 1$ до 1.

Происхождение термина «метод пузырька» объясняется так: если представить вертикальное расположение ячеек массива с ростом индекса сверху вниз, то самое маленькое из рассматриваемых чисел будет подниматься вверх подобно пузырьку в воде.

В нашем примере:

– при $i = 3$ перестановки приведут к следующему состоянию массива:

X_1	X_2	X_3	X_4	X_5	X_6
15	18	21	34	25	40

– при $i = 4$

X_1	X_2	X_3	X_4	X_5	X_6
15	18	21	25	34	40

— при $i = 5$

X_1	X_2	X_3	X_4	X_5	X_6
15	18	21	25	34	40

При использовании метода пузырька не имеет значения, в какую сторону (увеличения или уменьшения индексов) продвигается анализ пар чисел в массиве. Вид упорядочения (по возрастанию или убыванию) определяется только условием перестановки чисел (меньшее должно расположиться за большим или наоборот).

Модифицированный метод прямого обмена (модифицированный метод пузырька). Как видно из приведенного выше числового примера, массив оказался упорядоченным уже после четвертого шага, т. е. возможно выполнение внешнего цикла не $n - 1$ раз, а меньше, пока не станет известно, что массив упорядочен. Такая проверка основывается на следующем: если при выполнении внутреннего цикла не было ни одной перестановки, значит массив уже упорядочен, и можно выйти из внешнего цикла. В качестве признака, выполнялась ли перестановка, используют переменную булевого типа: до входа во внутренний цикл ей дают одно значение, например False, а при выполнении перестановки — другое, например True.

Очевидно, эффект в ускорении процесса сортировки при использовании модифицированного метода пузырька по сравнению с немодифицированным будет наблюдаться, если исходная последовательность чисел близка к упорядоченности в нужном направлении. В предельном случае, когда массив уже упорядочен нужным образом, тело внешнего цикла будет выполнено только один раз.

Метод прямого выбора (сортировки посредством выбора) и его модификации

Все описываемые далее методы следует рассматривать как частные варианты метода, известного под названием *метод прямого выбора* (или *сортировка посредством выбора*). Общим для этих методов является нахождение (выбор) максимальных или минимальных элементов массива и размещение их в последовательных ячейках его.

Сортировка методом поиска минимального элемента. Суть этого метода (при выбранных ограничениях для рассматриваемого нами числового примера) состоит в следующем.

На первом шаге отыскивается и сохраняется в переменной, например X_{\min} , минимальное число среди всех чисел массива и его индекс, сохраняемый в другой переменной, например I_{\min} , а затем проводится обмен местами в массиве найденного минимального числа с первым элементом массива:

$X[I_{\min}] := X[1]; X[1] := X_{\min};$

В рассматриваемом примере минимальное число $X_{\min} = 15$ находится в ячейке $I_{\min} = 3$, и перестановка первого и минимального чисел приведет к следующему результату:

X_1	X_2	X_3	X_4	X_5	X_6
15	21	34	18	25	40

В общем случае на любом шаге i ($i = 1, 2, 3, \dots, n - 1$) отыскивается X_{\min} — минимальное число среди ячеек массива с индексами от i до n и его индекс I_{\min} , а затем проводится обмен местами в массиве найденного минимального числа с i -м элементом массива:

$X[I_{\min}] := X[i]; X[i] := X_{\min};$

В нашем примере:

— при $i = 2$ получим $X_{\min} = 18$ и $I_{\min} = 4$, после перестановки имеем:

X_1	X_2	X_3	X_4	X_5	X_6
15	18	34	21	25	40

— при $i = 3$ получим $X_{\min} = 21$ и $I_{\min} = 4$, после перестановки имеем:

X_1	X_2	X_3	X_4	X_5	X_6
15	18	21	34	25	40

— при $i = 4$ получим $X_{\min} = 25$ и $I_{\min} = 5$, после перестановки имеем:

X_1	X_2	X_3	X_4	X_5	X_6
15	18	21	25	34	40

— при $i = 5$ получим $X_{\min} = 34$ и $I_{\min} = 5$, после перестановки имеем:

X_1	X_2	X_3	X_4	X_5	X_6
15	18	21	25	34	40

Таким образом, внешний цикл должен выполняться $n - 1$ раз, а число повторений внутреннего цикла будет уменьшаться от $n - 1$ до 1. Чтобы упорядочить массив по убыванию, следует первое найденное минимальное число обменивать местами с последним, второе — с предпоследним и т. д.

Сортировка методом поиска максимального элемента. Этот метод отличается от предыдущего только тем, что отыскивают максимальные элементы. При одинаковой организации циклов реализации этих методов они дадут прямо противоположные результаты: если один приведет к возрастанию чисел в массиве, то другой — к убыванию и наоборот.

Сортировка методом поиска индекса минимального элемента. Этот метод отличается от метода поиска минимального элемента и его индекса тем, что внутренний цикл используется для поиска только индекса минимального элемента, поэтому перестановки чисел в массиве на каждом шаге i ($i = 1, 2, \dots, n-1$) придется выполнять с привлечением дополнительной переменной, например R :

```
R: R:=X[i]; X[i]:=X[Imin]; X[Imin]:=R;
```

Сортировка методом поиска индекса максимального элемента. Метод отличается от предыдущего тем, что отыскивают индекс максимального элемента. При одинаковой организации циклов при реализации этих методов они дадут прямо противоположные результаты: если один приведет к возрастанию чисел в массиве, то другой — к убыванию и наоборот.

5.9. Пример выполнения задания

Программа составлена по условию задания № 30 (см. ниже):

```
program Project1;  
{$APPTYPE CONSOLE}  
uses  
    SysUtils;  
{$DEFINE DBG}  
const  
    Nmax=12;  
type  
    tMas=array[1..Nmax] of Byte;  
var  
    X:tMas=(50,23,123,34,125,54,231,3,222,13,5,100);  
    N:Byte=10;  
    i, iTek, iMax, k:Byte;  
begin  
{IFDEF DBG}  
    //Операторы, не используемые при отладке  
    Write('Введите число элементов массива N: ');
```



```
ReadLn(N);
WriteLn('Введите элементы массива одной строкой');
for k:=1 to N do
    Read(X[k]);
ReadLn;
{$ENDIF}
i:=1;
repeat
    //Поиск индекса iMax максимального элемента
    //массива среди элементов с индексами от i до N
    iMax:=i;
    for iTek:=i+1 to N do
        if X[iTek]>X[iMax] then
            iMax:=iTek;
    //Взаимная перестановка X[i] с X[iMax]
    k:=X[i];
    X[i]:=X[iMax];
    X[iMax]:=k;
{$IFDEF DBG}
    //Операторы, используемые при отладке
    //Вывод массива после перестановки
    //элементов X[i] с X[iMax]
    for k:=1 to N do
        Write(X[k]:5);
    WriteLn;
{$ENDIF}
    i:=i+1;
until i=N;
for k:=1 to N do
    Write(X[k]:5);
ReadLn;
end.
```

5.10. Задания для самостоятельной работы

Составить программу упорядочения первых $N \leq 12$ элементов массива X . Вид, а также метод сортировки и операторы внешнего и внутреннего циклов, которые следует использовать в программе, указаны для каждого задания в таблице.

При отладке использовать начальные значения N массива X , а также выполнять форматный вывод первых N элементов массива одной строкой в конце каждого шага во внешнем цикле.

№ п/п	Вид сортировки	Метод сортировки	Оператор внешнего цикла	Оператор внутреннего цикла
1	По возрастанию	Прямого включения	while	while
2	По убыванию	Выбора максимального	for to	for to
3	По возрастанию	Выбора минимального	for to	for downto
4	По убыванию	Выбора индекса максимального	while	for downto
5	По возрастанию	Выбора индекса минимального	for downto	while
6	По убыванию	Прямого обмена (пузырька)	repeat	repeat
7	По возрастанию	Модифицированного прямого обмена (пузырька)	while	for downto
8	По убыванию	Прямого включения	for to	while
9	По возрастанию	Выбора максимального	for to	while
10	По убыванию	Выбора минимального	for downto	for to
11	По возрастанию	Выбора индекса максимального	for downto	for to
12	По убыванию	Выбора индекса минимального	for to	repeat
13	По возрастанию	Прямого обмена (пузырька)	for downto	while
14	По убыванию	Модифицированного прямого обмена (пузырька)	repeat	for downto
15	По возрастанию	Прямого включения	for downto	while
16	По убыванию	Выбора максимального	for downto	for downto
17	По возрастанию	Выбора минимального	for to	repeat
18	По убыванию	Выбора индекса максимального	for to	for to
19	По возрастанию	Выбора индекса минимального	while	repeat
20	По убыванию	Прямого обмена (пузырька)	for downto	for to
21	По возрастанию	Прямого включения	repeat	while
22	По убыванию	Выбора максимального	for to	for downto
23	По возрастанию	Выбора минимального	for downto	while
24	По возрастанию	Выбора индекса максимального	for downto	for downto
25	По убыванию	Выбора индекса минимального	repeat	while
26	По возрастанию	Прямого обмена (пузырька)	for to	for downto

№ п/п	Вид сортировки	Метод сортировки	Оператор внешнего цикла	Оператор внутрен- него цикла
27	По убыванию	Модифицированного пря- мого обмена (пузырька)	repeat	for to
28	По возрастанию	Выбора индекса мини- мального	for to	for to
29	По возрастанию	Модифицированного пря- мого обмена (пузырька)	while	repeat
30	По убыванию	Выбора индекса макси- мального	repeat	for to

6. Программирование с использованием подпрограмм

Подпрограмму можно определить как относительно самостоятельный фрагмент программы, оформленный таким образом, что его можно выполнять многократно, передавая ему управление из разных частей программы для обработки разных данных.

Использование подпрограмм позволяет уменьшить размер программы (если в различных частях программы необходимо выполнять обработку данных по одному алгоритму) и сделать ее исходный текст более удобным для понимания процесса обработки данных (если алгоритм подпрограммы обладает функциональной законченностью, а имя подпрограммы отражает ее назначение, как, например, у стандартных подпрограмм $\text{Sin}(X)$ или $\text{Abs}(X)$). Преимущества использования подпрограмм проявляются также при разработке больших программ, так как становится возможным распараллелить процесс создания программного продукта, поручив разработку отдельных подпрограмм разным исполнителям, и, что более важно, — упростить процесс написания и отладки.

Разбиение программы на подпрограммы производится прежде всего по функциональному признаку: подпрограмма должна реализовывать одну, но законченную функцию. При этом надо стремиться к сокращению количества межпрограммных связей (количеству передаваемых параметров). Рекомендуемый размер подпрограммы составляет 10–60 строк текста. Нецелесообразно создавать слишком короткие подпрограммы, а размещение подпрограммы в пределах одной страницы позволит программисту охватить весь текст одним взглядом и не тратить время на переключение внимания с одной страницы на другую.

В языке Object Pascal используются два вида подпрограмм — функции и процедуры. При общих принципах оформления функции обладают дополнительными возможностями. Поэтому сначала рассмотрим объявление и использование процедур, а затем особенности функций.

6.1. Процедуры

При работе с подпрограммами следует различать термины *объявление подпрограммы* (*описание подпрограммы*) и *обращение к подпрограмме* (*вызов подпрограммы*).

Объявление подпрограммы содержит ее имя и описывает процесс обработки данных, представленных параметрами (пока будем считать так, следуя рекомендациям структурного программирования). В объявлении подпрограммы параметры называют *формальными*. Имя подпрограммы задает ее разработчик, и оно должно быть уникальным в своем блоке.

Обращение к подпрограмме выполняется по ее имени и описывает данные (*фактические параметры*), обработку которых она должна выполнить. Выполнение обращения приводит к передаче управления подпрограмме. Вызовов одной подпрограммы может быть несколько, а фактические параметры в них — разными.

Объявления подпрограмм и их вызовы должны оформляться в соответствии с правилами, которые будут представлены в виде синтаксических диаграмм.

Имя процедуры (рис. 6.1) строится так же, как и прочие имена в языке Object Pascal. Блок процедуры, как и блок основной программы, может содержать объявления меток, констант, типов, переменных, подпрограмм и обязательно — составной оператор (**begin ... end**), представляющий алгоритм. *Список формальных параметров* содержит имена, используемые в теле подпрограммы для описания процесса обработки данных (рис. 6.2).

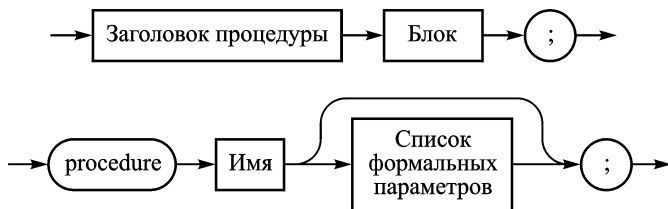


Рис. 6.1

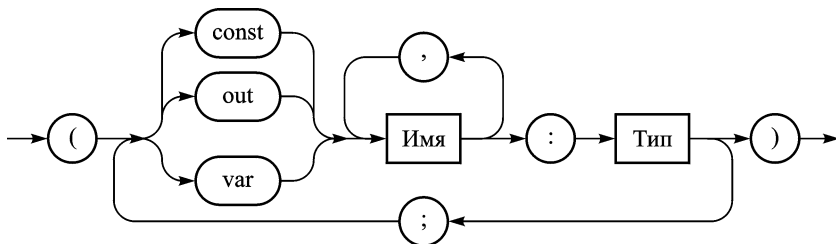


Рис. 6.2

Обращение к процедуре (рис. 6.3) является отдельным оператором программы. *Список фактических параметров* представляет реально существующие данные — константы, переменные, выраже-

ния, записываемые в круглых скобках через запятую. Соответствие фактических параметров формальным устанавливается порядком их следования в списках. При вызове подпрограммы фактические параметры как бы занимают в алгоритме места соответствующих формальных (уточнение будет дано позже), после чего алгоритм выполняется.

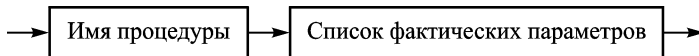


Рис. 6.3

Пример объявления и вызова процедуры, выводящей на экран первое из двух значений, представленных параметрами, кратное 5, или сообщение, что кратных нет:

```

//Объявление процедуры
procedure PutMod5(I,J:Integer);
// I и J - формальные параметры
begin
  if I mod 5 = 0 then
    WriteLn(I)
  else if J mod 5 = 0 then
    WriteLn(J)
  else
    WriteLn('Нет параметров, кратных 5');
end;
. . . . .
begin // РАЗДЕЛ ОПЕРАТОРОВ ПРОГРАММЫ
. . . . .
//Вызов процедуры
PutMod5(A+3, B); //A+3 и B - фактические параметры
. . . . .
end.
  
```

Рассмотрим на конкретных значениях переменных A и B , что будет выведено подпрограммой:

при $A=2$ и $B=10$ будет выведено значение 5;

при $A=3$ и $B=10$ будет выведено значение 10;

при $A=3$ и $B=11$ будет выведен текст «Нет параметров, кратных 5».

Тип в объявлении формального параметра пока будем считать обязательным и задавать только именем ранее объявленного или стандартного типа (другие возможности будут рассмотрены позже).

Прежде чем перейти к пояснению правил объявления формальных параметров, введем в рассмотрение два термина: *вход-*

ной параметр и *выходной параметр*. Параметр считается входным, если он представляет исходные данные для счета по алгоритму подпрограммы (в рассмотренном примере оба параметра являются входными). Параметр считается выходным, если он представляет результаты счета по подпрограмме. Допускается, что один и тот же параметр может одновременно представлять и исходные данные, и результат, т. е. быть одновременно и входным, и выходным.

В языке Object Pascal рассмотренное функциональное деление параметров (на входные, выходные, входные-выходные) дополнено механизмами реализации их функций, отраженными в правилах объявления формальных параметров. Сначала рассмотрим два вида объявлений параметров: параметры-значения и параметры-переменные.

Параметры-значения всегда являются только входными параметрами. Соответствующими фактическими параметрами могут быть выражения (в частности, константы и переменные). При вызове подпрограммы выражения вычисляются и полученные значения заносятся (а значения констант и переменных просто копируются) в ячейки памяти, представленные соответствующими формальными параметрами, значения которых внутри подпрограммы можно изменять. Механизм параметров-значений таков, что эти изменения никак не отразятся на значениях фактических параметров, т. е. после выхода из подпрограммы они останутся неизменными. Таким образом, формальный параметр-значение можно рассматривать как переменную, известную внутри подпрограммы, а механизм таких параметров — как защиту фактических параметров, если они переменные, от непреднамеренного изменения.

Объявление параметров-значений дается просто их именем (списком имен через запятую) и типом (без предшествующих слов **var**, **const**, **out**). В рассмотренной процедуре PutMod5 оба параметра представляют собой параметры-значения.

Параметры-переменные являются одновременно и входными, и выходными параметрами. Соответствующими фактическими параметрами могут быть только переменные. Механизм параметров-переменных таков, что изменения, выполняемые в алгоритме подпрограммы над соответствующими формальными параметрами, приводят к изменению значений фактических, и эти изменения сохраняются после выхода из подпрограммы. Достигается это за счет того, что в подпрограмму передается не сама переменная, а ссылка на нее, т. е. обеспечивается непосредственный доступ из подпрограммы к данным фактического параметра.

Объявление параметров-переменных дается с предшествующим их имени (списку имен) словом **var** и типом.

Пример процедуры, умножающей данные в первых n ячейках массива *Mas* на дробную часть числа *R* (все участвующие в обработке данные должны передаваться через параметры):

```
type tMas=array[1..8] of Real;
. . . . .
//Объявление процедуры
procedure MasMulR(R:Real; N:Integer; var Mas:tMas);
var
    i:Integer;
begin
    R:=Frac(R); //Получить дробную часть R
    for i:=1 to N do
        Mas[i]:=Mas[i]*R;
end; //MasMulR - конец текста процедуры
. . . . .
var
    X:tMas=(1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0);
    Z:Real=3.4;
. . . . .
begin // РАЗДЕЛ ОПЕРАТОРОВ ПРОГРАММЫ
. . . . .
    MasMulR(Z,5,X); //Вызов процедуры
. . . . .
end.
```

В этом примере *Mas* является и входным, и выходным, поэтому он объявлен как параметр-переменная (с предшествующим словом **var**). Параметр *R* — только входной. Так как он объявлен как параметр-значение, то его после использования в качестве входного данного можно менять внутри подпрограммы, т. е. использовать как дополнительную переменную, не опасаясь, что изменится значение соответствующего фактического параметра. При указанных в примере начальных значениях переменных после выполнения процедуры переменная *Z* сохранит свое значение 3.4, а в массиве будут следующие данные: 0.4, 0.8, 1.2, 1.6, 2.0, 6.0, 7.0, 8.0.

Параметры-значения обладают одним недостатком, который может оказаться очень существенным, если параметром является структурная переменная, занимающая большой объем памяти, например, массив. В этом случае подпрограмма, получив управле-

ние, должна выделить такой же объем памяти, как фактический параметр, и скопировать в нее данные из фактического параметра. Таким образом, недостаток заключается в том, что нерационально используется память и происходит потеря времени на копирование данных перед началом их обработки по алгоритму подпрограммы. Указанных недостатков лишен другой вид входного параметра, получивший название «параметр-константа».

Параметр-константа, как и параметр-значение, служит для представления только входных данных. При его использовании в подпрограмму передается ссылка на фактический параметр (как и для параметра-переменной), но для исключения непреднамеренного изменения во время работы программы уже на этапе компиляции выполняют соответствующие проверки. Исполняемая программа не создается, если подпрограмма содержит операторы, которые могут изменить данные фактического параметра.

Объявление параметров-констант дается с предшествующим их имени (списку имен) словом **const** и типом.

Последний вид параметра следует использовать для параметров, представляющих только результаты работы подпрограммы. По аналогии с параметрами других видов ему подошло бы название *параметр-результат* или по предшествующему его имени слову **out** в объявлении — *параметр-выходная переменная*. Этот параметр передается по ссылке, но компилятор не запрещает его использование в качестве источника исходных данных. За этим должен следить программист, а слово **out** перед именем в списке формальных параметров призвано облегчить понимание алгоритма подпрограммы, напоминая, что параметр не представляет входных данных.

Пример процедуры, в объявлении которой желательно использовать параметр-константу и параметр-результат. Подпрограмма должна вычислять сумму положительных элементов массива $X(100, 100)$ и их количество:

```
type
  tMas=array[1..100,1..100] of Extended;
. . . . .
//Объявление процедуры
procedure SumCol(const X:tMas,
                  out S: Extended;
                  out K:Integer);

var
  i,j: Integer;
begin
```

```

S:=0;
K:=0;
for i:=1 to 100 do
  for j:=1 to 100 do
    if X[i,j]>0 then
      begin
        S:=S+X[i,j];
        K:=K+1;
      end;
  end;
end;
. . . . .
var
  Y:tMas; Sum: Extended; Col: Integer;
. . . . .
begin // РАЗДЕЛ ОПЕРАТОРОВ ПРОГРАММЫ
. . . . .
  SumCol(Y,Sum,Col);
. . . . .
end.

```

Взглянув на заголовок подпрограммы, сразу можно сказать, что первый параметр является только входным, а два последних — только выходными. Первый параметр объявлен параметром-константой, а не параметром-значением, чтобы избежать выделения памяти в 100 000 байт и копирования в нее данных из фактического параметра — массива Y .

6.2. Пример выполнения задания

Составить процедуру, копирующую из матрицы $A(m, n)$, $m \leq 10$, $n \leq 14$ положительные элементы в массив Pol и подсчитывающую их количество kPol, а отрицательные элементы — в массив Otr и подсчитывающую их количество kOtr. Использовать эту процедуру для матрицы B в основной программе. Если окажется, что положительных и/или отрицательных элементов в матрице нет, то вывести соответствующие сообщения, иначе — скопированные в массивы данные:

```

program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
const
  mMax=10; nMax=14;

```

```
type
  tMatr=array[1.. mMax,1.. nMax] of Real;
  tMas=array[1.. mMax * nMax] of Real;

procedure PolOtr(const A:tMatr;

                  m,n:Integer;
                  out Pol, Otr:tMas;
                  out kPol, kOtr:Integer);

var
  i,j:Integer;
begin
  kPol:=0;
  kOtr:=0;
  for i:=1 to m do
    for j:=1 to n do
      begin
        if A[i,j]>0 then
          begin
            kPol:= kPol +1;
            Pol[kPol]:=A[i,j];
          end
        else if A[i,j]<0 then
          begin
            kOtr:= kOtr +1;
            Otr[kOtr]:=A[i,j];
          end;
        end;
      end;
    end;
  end;

var
  B:tMatr; P,O:tMas;
  m,n,i,j,kP,kO:Integer;

begin // РАЗДЕЛ ОПЕРАТОРОВ ПРОГРАММЫ
  WriteLn('Введите количество строк и столбцов');
  ReadLn(m,n);
  WriteLn('Введите матрицу по строкам');
  for i:=1 to m do
    begin
      for j:=1 to n do
        Read(B[i,j]);
```

```
    ReadLn;
end;
PolOtr(B,m,n,P,O,kP,kO);
if kP>0 then
begin
    WriteLn(' Массив положительных');
    for i:=1 to kP do
        Write(P[i]:6:1, ' ');
    end
else
    WriteLn('Положительных элементов нет');
WriteLn;
if kO>0 then
begin
    WriteLn(' Массив отрицательных');
    for i:=1 to kO do
        Write(O[i]:6:1, ' ');
    end
else
    WriteLn('Отрицательных элементов нет');
ReadLn;
end.
```

6.3. Задания для самостоятельной работы

1. Составить процедуру перемножения двух матриц. Процедура должна содержать проверку возможности умножения матриц и выдавать код ошибки, равный 1, если умножение невозможно, иначе 0. С помощью процедуры вычислить n -ю степень A^n квадратной матрицы $A(m, m)$, $m \leq 10$.

2. Составить процедуру, которая находит минимальный и максимальный элементы всей матрицы и переписывает все элементы, находящиеся между ними (при просмотре матрицы по строкам), в одномерный массив. Использовать составленную процедуру для формирования из матрицы $C(k, l)$, $k \leq 12$, $l \leq 14$ массива D , который упорядочить по возрастанию, если он не пуст, и выдать соответствующее сообщение в противном случае.

3. Составить процедуру, которая находит в матрице первый и пятый положительные элементы при просмотре ее по строкам и переписывает все отрицательные элементы, находящиеся между ними, в одномерный массив Z . Использовать составленную процедуру для получения Z из матрицы $V(l, m)$, $l \leq 15$, $m \leq 18$. Упорядочить

дочить полученный массив по убыванию, если он не пуст, и выдать сообщение в противном случае.

4. Составить процедуру, которая формирует матрицу $H(m, m-1)$ из матрицы $F(m, m)$, $m \leq 12$, путем вычеркивания минимального элемента каждой строки матрицы. Умножить исходную матрицу на полученную. В основной программе с помощью составленной процедуры получить на основе матрицы A две новые матрицы: B (без минимальных элементов) и C (результат умножения A на B).

5. Составить процедуру, которая находит в матрице B строки, содержащие минимальный и максимальный элементы. Из найденных строк сформировать матрицу D , рассматривая их как столбцы новой матрицы. Умножить исходную матрицу на D . Использовать составленную процедуру для получения из матрицы $H(l, m)$, $l \leq 9$, $m \leq 8$ матриц P и Q .

6. Составить процедуру, которая удаляет из матрицы строки, целиком состоящие из нулевых элементов, а затем определяет минимальный и максимальный по абсолютной величине элементы преобразованной матрицы. Использовать составленную процедуру для матрицы $B(l, m)$, $l \leq 13$, $m \leq 14$.

7. Составить процедуру, которая определяет точки, расстояние между которыми максимально и минимально. Координаты точек заданы в массивах $X(n)$, $Y(n)$, $n \leq 20$. Использовать составленную процедуру для массивов $X1$, $Y1$, вывести номера найденных точек, а также расстояние между ними.

8. Составить процедуру, которая в матрице переставляет элементы каждой строки так, чтобы в начале располагались положительные элементы, а затем — неположительные. Для строк, содержащих положительные элементы, вычислить средние геометрические положительных элементов и занести их в массив G . Для строк, не содержащих положительных элементов, вычислить средние арифметические значения и занести их в массив A . (Перестановку элементов строк провести без использования алгоритма упорядочения за один цикл просмотра элементов строки.) Использовать составленную процедуру для матрицы $W(m, n)$, $m \leq 12$, $n \leq 15$.

9. Составить процедуру, которая в матрице обменивает местами строки, содержащие соответственно минимальный и максимальный элементы всей матрицы. Если обмен строк местами невозможен (максимальный и минимальный элементы расположены в одной строке), то поменять местами столбцы, содержащие найден-

ные элементы. Если все элементы матрицы одинаковы, то никаких преобразований не делать и сформировать признак результата -1 , иначе сформировать признак результата 0 . Использовать составленную процедуру для матрицы $T(m, n)$, $m \leq 12$, $n \leq 15$.

10. Составить процедуру, которая первый отрицательный элемент каждой строки матрицы заменяет суммой положительных элементов. Если замена невозможна (нет отрицательных элементов в строке), то преобразований не производить, а занести номер этой строки в массив **NOM** без пропусков. Использовать составленную процедуру для матрицы $Q(k, l)$, $k \leq 14$, $l \leq 15$.

11. Составить процедуру, упорядочивающую строки матрицы по возрастанию сумм их элементов. Использовать составленную процедуру для матрицы $E(k, m)$, $k \leq 14$, $m \leq 12$.

12. Составить процедуру, которая в матрице находит строки (векторы) с наибольшей суммой и наибольшим произведением элементов. Сформировать массив, представляющий собой попарные произведения элементов двух массивов с одноименными индексами. Использовать составленную процедуру для матрицы $SP(k, n)$, $k \leq 12$, $n \leq 15$.

13. Составить процедуру, которая на основе коэффициентов двух многочленов, хранящихся в двух массивах по убыванию степеней, находит массив коэффициентов многочлена, являющегося произведением заданных. Вычислить значение полученного многочлена при заданном значении аргумента. Использовать составленную процедуру для умножения двух многочленов порядка не более 15.

14. Составить процедуру, которая последний положительный элемент каждой строки матрицы заменяет суммой остальных ее элементов. Если замена невозможна (нет положительных элементов), то преобразований в строке не производить, а занести ее номер в массив **NOM** без пропусков. Использовать составленную процедуру для матрицы $Q(k, l)$, $k \leq 14$, $l \leq 15$.

15. Составить процедуру, которая в каждой строке матрицы заменяет каждый нулевой элемент суммой элементов, стоящих между предыдущим и текущим нулевыми элементами. Первый нулевой элемент строки заменить суммой всех предшествующих. Если нулевой элемент стоит на первом месте в строке, то оставить его без изменений. Использовать составленную процедуру для матрицы $W(m, n)$, $m \leq 12$, $n \leq 15$.

16. Составить процедуру, которая в каждой строке матрицы заменяет каждый нулевой элемент максимальным из стоящих между предыдущим и текущим нулевыми элементами. Первый

нулевой элемент строки заменить максимальным из всех предшествующих. Если нулевой элемент стоит на первом месте в строке, то оставить его без изменений. Использовать составленную процедуру для матрицы $W(m, n)$, $m \leq 12$, $n \leq 15$.

17. Составить процедуру, которая в матрице во всех строках со второй по предпоследнюю находит максимальный и минимальный элементы. Максимальный помещается на место первого элемента строки, а минимальный — на место последнего элемента строки. Во всех столбцах матрицы со второго по предпоследний найти сумму и произведение элементов, сумму поместить на первое место в столбце, а произведение — на последнее. Элементам, стоящим в углах матрицы, присвоить нуль. Использовать составленную процедуру для матрицы $B(l, m)$, $l \leq 13$, $m \leq 14$.

18. В квадратной матрице в каждой строке найти минимальный элемент, стоящий под главной диагональю, и максимальный элемент, стоящий над главной диагональю. Из найденных элементов сформировать два вектора. Найти сумму попарных произведений элементов полученных векторов с одноименными индексами. Использовать составленную процедуру для матрицы $A(l, l)$, $l \leq 14$.

19. Составить процедуру, которая находит в каждой строке матрицы минимальный элемент и проверяет, является ли найденный элемент максимальным в том столбце, где он расположен. Если это условие выполняется, то запомнить в матрице $NOM(k, 2)$ индексы найденных элементов. Использовать составленную процедуру для матрицы $S(l, m)$, $l \leq 13$, $m \leq 11$.

20. Составить процедуру упорядочения строк матрицы по возрастанию количества элементов строк, нацело делящихся на заданное число m . Использовать составленную процедуру для матрицы $Z(k, l)$, $k \leq 12$, $l \leq 14$.

21. Составить процедуру, образующую из двух целочисленных одномерных массивов, где все элементы разные, третий, в который переписываются элементы, присутствующие в первом либо во втором массиве (объединение двух массивов). Использовать процедуру для массивов $E(l)$, $l \leq 20$, $F(m)$, $m \leq 30$.

22. Составить процедуру, образующую из двух целочисленных одномерных массивов, где все элементы разные, третий массив, в который переписываются элементы, присутствующие одновременно в первом и втором массивах (пересечение двух массивов). Использовать процедуру для массивов $E(l)$, $l \leq 20$, $F(m)$, $m \leq 30$.

23. Составить процедуру, образующую из двух целочисленных одномерных массивов, где все элементы разные, третий массив, в который переписываются элементы первого массива, отсутствующие во втором массиве (дополнение второго массива до первого). Использовать процедуру для массивов $E(l)$, $l \leq 20$, $F(m)$, $m \leq 30$.

24. Составить процедуру, которая транспонирует исходную матрицу и умножает первую матрицу на транспонированную. В полученной матрице найти максимальный элемент верхней треугольной матрицы и минимальный элемент нижней. Использовать составленную процедуру для матрицы $H(l, n)$, $l \leq 12$, $n \leq 14$.

25. Составить процедуру, которая в каждой строке квадратной матрицы среди элементов, стоящих над главной диагональю, находит максимальный, а в каждом столбце среди элементов, стоящих под главной диагональю, минимальный. (Элементы главной диагонали не рассматривать.) Найденные элементы, расположенные в одноименных строке и столбце, поменять местами. Использовать составленную процедуру для матрицы $H(n, n)$.

6.4. Функции

Объявление функции показано на рис. 6.4.

Заголовок функции приведен на рис. 6.5.

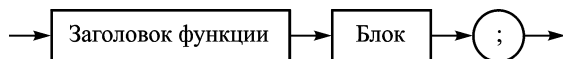


Рис. 6.4

Обращение к функции понятно из рис. 6.6.

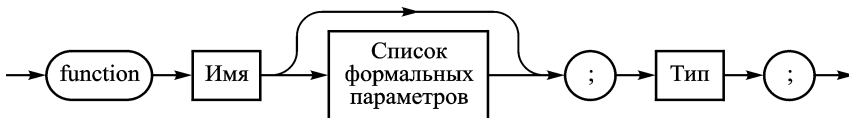


Рис. 6.5

Отличие функции от процедуры состоит в том, что:

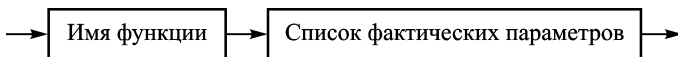


Рис. 6.6

— в заголовке после списка параметров необходимо указать *тип функции* (т. е. тип вычисляемого ею результата) — имя ранее объявленного или стандартного типа;

— в вызывающей программе обращение к функции можно записывать в правой части оператора присваивания и в выражениях, если тип результата простой (но присваивать значение имени функции в вызывающей программе запрещено);

— в объявлении функции ее имя (не обращение, делающее подпрограмму рекурсивной) не должно встречаться в правой части операторов присваивания или в выражениях;

— в объявлении функции должен быть хотя бы один оператор, присваивающий ее имени или объявленной по умолчанию локальной (известной только внутри функции) переменной *Result* того же типа, что и тип функции, результат вычислений;

— в отличие от имени переменную *Result* можно использовать в правых частях операторов присваивания и в выражениях как дополнительную переменную, представляющую результат вычислений.

Обращение к функции, как и к процедуре, можно записывать как отдельный оператор в режиме расширенного синтаксиса, используемом в Delphi по умолчанию. Обращение к функции отдельным оператором имеет смысл, когда интересующий нас результат представлен параметрами, а не именем функции. Отключить режим *расширенного синтаксиса* можно директивой `{X-}` или `{$EXTENDEDSTYNTAX OFF}`, но тогда не будет объявлена по умолчанию локальная переменная *Result* и при попытке ее использования компилятор сообщит об ошибке.

Пример 1. Составить и использовать функцию, возвращающую максимальный из первых N элементов массива $X(N)$, $N \leq 100$:

```
type
  tMas=array[1..100] of Real;
. . . . .
function MaxMas(N:Integer; const X:tMas):Real;
var
  i:Integer;
begin
  Result:=X[1];
  for i:=2 to N do
    if X[i]>Result then
      Result:=X[i];
end;
. . . . .
var
```

```

Y:tMas;
begin // РАЗДЕЛ ОПЕРАТОРОВ ПРОГРАММЫ
. . . . .
//Вызов функции в операторе вывода
WriteLn('Максимальный из сорока '
        , 'элементов массива Y = '
        , MaxMas(40, Y));
. . . . .
end.

```

Если отключить режим расширенного синтаксиса, то в объявлении функции пришлось бы отказаться от использования переменной **Result** и объявить дополнительную переменную для поиска максимального значения в массиве. В результате получили бы следующее объявление функции:

```

function MaxMas(N:Integer; const X:tMas):Real;
var
  i:Integer;
  R:Real; //Дополнительная переменная
begin
  R:=X[1];
  for i:=2 to N do
    if X[i]>R then
      R:=X[i];
  //Чтобы функция возвратила вычисленное значение,
  //оно должно быть присвоено ее имени
  MaxMas:=R;
end;

```

Использование функций позволяет не только сделать текст программы более удобным для понимания алгоритма, но и уменьшить его при необходимости повторных вычислений в разных частях программы.

Пример 2. Составить программу поиска приближенных значений корней уравнения

$$\frac{X \cos(3PX)}{12P} - \frac{\sin(3PX)}{36P^2} - \frac{3X \cos(PX)}{4P} + \frac{3 \sin(PX)}{4P^2} = 0$$

и их уточненных значений методом половинного деления:

```

program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
//Функция, вычисляющая выражение,

```

```
//входящее в уравнение
function F_PX(const P,X:Real):Real;
var
    PX:Real;
begin
    PX:=P*X;
    F_PX:=X*Cos(3*PX)/P/12-Sin(3*PX)/Sqr(P)/36
        -3*X*Cos(PX)/P/4 +3*Sin(PX)/Sqr(P)/4;
end;

//функция, уточняющая значение корня
function KorenF_PX(P,X0,X1,Eps:Real):Real;
var
    F0,F,F1,X:Real;
begin
    F0:=F_PX(P,X0);
    repeat
        X:=(X0+X1)/2;
        F:=F_PX(P,X);
        if F0*F>0 then
            X0:=X
        else
            X1:=X;
    until Abs(X1-X0)<Eps;
    KorenF_PX:=X;
end;

var
    P,X, A, B, dX, F0, F1,Eps,Koren:Real;

begin // РАЗДЕЛ ОПЕРАТОРОВ ПРОГРАММЫ
    Write('Введите границы интервала '
        , 'и шаг аргумента:');
    ReadLn(A,B,dX);
    Write('Введите параметр выражения P: ');
    ReadLn(P);
    Write('Введите требуемую точность корня: ');
    ReadLn(Eps);
    F0:=F_PX(P,A);
    X:=A+dX;
    while X<B+dX/2 do
    begin
        F1:=F_PX(P,X);
```

```

if F0*F1<0 then
begin
  //Найдены приближенные значения корня:
  //X-dX - приближение, меньшее корня,
  //X - приближение, большее корня.
  WriteLn(X-dX, ' = левое приближение корня');
  //Поиск приближения с точностью Eps
  Koren := KorenF_PX(P,X-dX,X,Eps);
  //Вывод результатов
  WriteLn(Koren, ' = корень с точностью ',Eps);
  WriteLn(F_PX(P,Koren)
    , ' = значение функции в корне');
  WriteLn;
end;
F0:=F1;
X:=X+dX;
end;
ReadLn;
end.

```

6.5. Пример выполнения задания

Составить функцию для вычисления среднего арифметического элементов одномерного массива. Использовать эту функцию для формирования массива средних арифметических значений элементов каждой строки матрицы $A(m, n)$, $m \leq 12$, $n \leq 11$:

```

program Fun1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
const
  mMax=12; nMax=11;
type
  tStrk=array[1.. nMax] of Real;
  tMatr=array[1.. mMax] of tStrk;
  tStlb=array[1.. mMax] of Real;

  //Функция вычисления среднего арифметического
  //элементов одномерного массива
function Sred(const B:tStrk; n:Integer):Real;
var
  i:Integer;

```

```
begin
  Result:=0;
  for i:=1 to n do
    Result:= Result+B[i];
  Result:= Result/n;
end; //конец функции Sred

var
  A: tMatr;           //Массив исходной матрицы
  StrkA: tStrk;       //Массив строки матрицы
  SredStrkA: tStlb;   //Массив средних значений
                     //строк матрицы
  m,n,i,j:Integer;

begin // РАЗДЕЛ ОПЕРАТОРОВ ПРОГРАММЫ
  WriteLn('Введите количество строк и столбцов');
  ReadLn(m,n);
  WriteLn('Введите матрицу по строкам');
  for i:=1 to m do
    begin
      for j:=1 to n do
        Read(a[i,j]);
      ReadLn;
    end;
  //Вычисление средних арифметических
  //значений в строках матрицы
  for i:=1 to m do
    //Обращение к функции
    SredStrkA[i]:=Sred(A[i],n);
  //Вывод матрицы по строкам и правее
  //каждой строки - ее среднего арифметического
  WriteLn('Исходная матрица '
    , 'и средние арифметические');
  for i:=1 to m do
    begin
      //Вывод строки матрицы
      for j:=1 to n do
        Write(a[i,j]:6:1, ' ');
      //Вывод ее среднего арифметического
      WriteLn(' ', SredStrkA[i]:6:2);
    end;
    ReadLn;
  end.
end.
```

Поставленную задачу можно решить и по-другому, если функция должна работать непосредственно с матрицей. Функция в этом случае возвращает сразу сформированный массив средних арифметических:

```
program Fun1a;
{$APPTYPE CONSOLE}
uses
  SysUtils;
const
  mMax=12; nMax=11;
type
  tStrk=array[1.. nMax] of Real;
  tMatr=array[1.. mMax] of tStrk;
  tStlb=array[1.. mMax] of Real;

//Функция вычисления среднего арифметического
//элементов строк матрицы
function Sred(const B: tMatr; m,n:Integer):tStlb;
var
  i,j:Integer;  S:Real;
begin
  for i:=1 to m do
    begin
      S:=0;
      for j:=1 to n do
        S:= S +B[i,j];
      Sred[i]:=S/n
    end;
end; //конец функции Sred
var
  //Массив исходной матрицы
  A: tMatr;
  //Массив средних значений строк матрицы
  SredStrkA: tStlb;
  m,n,i,j:Integer;

begin // РАЗДЕЛ ОПЕРАТОРОВ ПРОГРАММЫ
  WriteLn('Введите количество строк и столбцов');
  ReadLn(m,n);
  WriteLn('Введите матрицу по строкам');
  for i:=1 to m do
    begin
```

```
    for j:=1 to n do
        Read(a[i,j]);
    ReadLn;
end;
//Вычисление средних арифметических
//значений в строках матрицы
SredStrkA:=Sred(A,m,n);
//Вывод матрицы по строкам и правее
//каждой строки - ее среднее арифметическое
WriteLn('Исходная матрица '
        , 'и средние арифметические');
for i:=1 to m do
begin
    //Вывод строки матрицы
    for j:=1 to n do
        Write(a[i,j]:6:1, ' ');
    //Вывод ее среднего арифметического
    WriteLn(' ', SredStrkA[i]:6:2);
end;
ReadLn;
end.
```

6.6. Задания для самостоятельной работы

1. Составить функцию, которая для заданного целого числа C определяет, является ли оно простым или нет. Функция должна возвращать значение *истина*, если число простое, и *ложь*, если не простое. Используя составленную функцию, определить все простые числа в матрице $A(m, n)$, $m \leq 10$, $n \leq 12$ и переписать их без пропусков в массив B . Полученный массив упорядочить по возрастанию методом нахождения наибольшего. Вывести полученный массив, если он не пуст, и сообщение об отсутствии в матрице простых чисел в ином случае.

2. Составить функцию, которая определяет сумму цифр целого числа A . Для массива целых чисел $K(m)$, $m \leq 20$ сформировать массив L сумм цифр элементов исходного массива. Упорядочить элементы массива K по возрастанию сумм их цифр методом пузырька.

3. Составить функцию, позволяющую определить, является ли заданное число A числом Фибоначчи или нет. Функция должна возвращать значение *истина*, если заданное число является числом Фибоначчи, и *ложь* — в противном случае. Используя со-

ставленную функцию, определить в матрице $C(k, l)$, $k \leq 12$, $l \leq 9$ все числа Фибоначчи и переписать их без пропусков в массив F , который упорядочить по возрастанию методом нахождения минимального. Числа Фибоначчи образуются по правилу $f_1 = 1$, $f_2 = 1$, $f_i = f_{i-1} + f_{i-2}$.

4. Составить функцию, которая в одномерном массиве определяет первый от начала положительный элемент. В случае нахождения положительного числа функция возвращает это число, в противном случае возвращает -1 . С помощью функции определить в каждой строке матрицы $H(l, m)$, $l \leq 12$, $m \leq 14$ первые положительные элементы, которые записать без пропусков в массив P . Полученный массив P упорядочить по убыванию методом пузырька. Вывести полученный массив, если он не пуст, или сообщение об отсутствии в матрице положительных чисел в ином случае.

5. Составить функцию, которая в одномерном массиве D определяет первый от начала отрицательный элемент. В случае нахождения отрицательного числа функция возвращает найденное число, в противном случае возвращает 1 . С помощью функции определить в каждой строке матрицы $H(l, m)$, $l \leq 12$, $m \leq 14$ первые отрицательные элементы, которые записать без пропусков в массив P . Полученный массив P упорядочить по возрастанию методом пузырька. Вывести полученный массив, если он не пуст, или сообщение об отсутствии в матрице отрицательных чисел в ином случае.

6. Составить функцию, которая в одномерном массиве D определяет первый от конца положительный элемент. В случае нахождения положительного числа функция возвращает найденное число, в противном случае возвращает -1 . С помощью функции определить в каждой строке матрицы $H(l, m)$, $l \leq 12$, $m \leq 14$ первые от конца положительные элементы, которые записать без пропусков в массив P . Полученный массив P упорядочить по убыванию методом нахождения минимального. Вывести полученный массив, если он не пуст, или сообщение об отсутствии в матрице положительных чисел в ином случае.

7. Составить функцию, которая в одномерном массиве D определяет первый от конца отрицательный элемент. В случае нахождения отрицательного числа функция возвращает найденное число, в противном случае возвращает 1 . С помощью функции определить в каждой строке матрицы $H(l, m)$, $l \leq 12$, $m \leq 14$ первые от конца отрицательные элементы, которые записать без пропус-

ков в массив P . Полученный массив P упорядочить по возрастанию методом нахождения наибольшего. Вывести полученный массив, если он не пуст, или сообщение об отсутствии в матрице отрицательных чисел в ином случае.

8. Составить функцию, которая в одномерном массиве подсчитывает количество чисел, нацело делящихся на заданное число m . С помощью составленной функции определить по каждой строке матрицы $F(k, l)$, $k \leq 14$, $l \leq 15$ количество элементов, нацело делящихся на заданное число n , и занести их в массив KOL . Упорядочить строки матрицы по возрастанию найденных количеств.

9. Составить функцию, которая для одномерного массива P возвращает значение *истина*, если его элементы образуют арифметическую прогрессию, и *ложь* в противном случае. С помощью составленной функции определить в матрице $R(m, n)$, $m \leq 14$, $n \leq 18$ строки, где элементы образуют арифметическую прогрессию, и составить из них новую матрицу, строки которой записаны без пропусков. Определить сумму элементов каждой прогрессии.

10. Составить функцию, которая для одномерного массива P возвращает значение *истина*, если его элементы образуют геометрическую прогрессию, и *ложь* в противном случае. С помощью составленной функции определить в матрице $R(m, n)$, $m \leq 14$, $n \leq 18$ строки, где элементы образуют геометрическую прогрессию, и составить из них новую матрицу, строки которой записаны без пропусков. Определить сумму элементов каждой прогрессии.

11. Составить функцию, которая для одномерного массива целых чисел проверяет, является ли заданная последовательность чисел симметричной. В случае симметричности функция должна возвращать значение *истина*, в противном случае — *ложь*. С помощью составленной функции определить в матрице $X(k, m)$, $k \leq 12$, $m \leq 14$ все строки, являющиеся симметричными, и переписать их без пропусков в новую матрицу.

12. Составить функцию, которая для целого числа определяет его симметричность (цифры числа должны образовывать одну и ту же последовательность при рассмотрении слева направо и справа налево) и возвращает значение *истина*, если оно симметрично, и *ложь* — в противном случае. С помощью составленной функции проверить симметричность чисел матрицы $Y(l, m)$, $l \leq 11$, $m \leq 15$ и найти строки с наименьшим и наибольшим количеством симметричных чисел.

13. Составить функцию, возвращающую значение *истина*, если одномерный массив не содержит повторяющихся чисел, и *ложь* — в противном случае. Использовать составленную функцию для определения в матрице $V(l, m)$, $l \leq 12$, $m \leq 14$ столбцов, целиком состоящих из различных чисел, и переписать найденные столбцы в новую матрицу без пропусков.

14. Составить функцию для нахождения длины отрезка на основе координат его вершин. Составить функцию нахождения площади треугольника на основе длин трех его сторон. Используя составленные функции, вычислить площадь произвольного выпуклого m -угольника, координаты вершин которого заданы в двух массивах: $X(m)$, $Y(m)$, $m \leq 20$.

15. Составить функцию, которая вычисляет среднее геометрическое элементов одномерного массива, если все его элементы положительные, и возвращает нуль, если среди элементов есть хотя бы один отрицательный или нулевой. Использовать составленную функцию для вычисления средних геометрических значений элементов каждой строки матрицы $F(k, l)$, $k \leq 12$, $l \leq 20$ и занесения их в массив G .

16. Составить функцию, вычисляющую максимальный и минимальный элементы одномерного массива (результат функции — массив из двух элементов). Использовать составленную функцию для нахождения в каждой строке матрицы $R(m, n)$, $m \leq 14$, $n \leq 10$ минимального и максимального элементов. Вывести матрицу и полученный результат так, чтобы слева от элементов строки стоял минимальный элемент, а справа от элементов строки — максимальный.

17. Составить функцию, которая вычисляет сумму и произведение элементов одномерного массива (результат функции — массив из двух элементов). Использовать составленную функцию для нахождения в каждой строке матрицы $H(m, n)$, $m \leq 14$, $n \leq 10$ суммы и произведения ее элементов. Из найденных значений сформировать одномерный массив, разместив в нем сначала суммы элементов строк, а затем — произведения. Упорядочить в этом массиве суммы по возрастанию, а произведения — по убыванию.

18. Составить функцию, которая проверяет факт расположения элементов одномерного массива по неубыванию. Функция возвращает значение *истина*, если элементы расположены по неубыванию, а в противном случае — *ложь*. Использовать составленную функцию для определения в матрице $H(l, m)$, $l \leq 12$, $m \leq 14$

всех строк, элементы которых расположены по неубыванию и для формирования из них новой матрицы NU .

19. Составить функцию, проверяющую факт чередования положительных и отрицательных элементов одномерного массива. Использовать составленную функцию для определения в матрице $D(l, m)$, $l \leq 12$, $m \leq 14$ всех строк, в которых положительные и отрицательные элементы чередуются. Сформировать из найденных строк новую матрицу R , записав в нее нужные строки без пропусков.

20. Составить функцию, которая в одномерном массиве подсчитывает количество чисел, превышающих среднее арифметическое значение всех чисел этого массива. Использовать составленную функцию для определения в матрице $D(l, m)$, $l \leq 12$, $m \leq 14$ строк, в которых количество элементов, превышающих среднее арифметическое значение элементов строки, максимально и минимально.

21. Составить функцию, которая в одномерном массиве подсчитывает количество чисел, превышающих среднее геометрическое значение всех чисел массива. Если среди элементов есть нулевые или отрицательные элементы, среднее геометрическое считать равным нулю. Использовать составленную функцию для определения в матрице $D(l, m)$, $l \leq 12$, $m \leq 14$ строк, в которых количество элементов, превышающих среднее геометрическое значение элементов строки, максимально и минимально.

22. Составить функцию, которая проверяет, состоит ли массив из одинаковых элементов или нет. Функция возвращает значение *истина*, если все элементы одинаковые, в противном случае — *ложь*. Использовать составленную функцию для формирования из матрицы $IS(l, m)$, $l \leq 12$, $m \leq 11$ новой матрицы RES , в которую переписать все столбцы исходной, которые составлены из одинаковых чисел.

23. Составить функцию, которая в одномерном массиве подсчитывает количество различных чисел. Использовать составленную функцию для определения в матрице $RL(m, n)$, $m \leq 13$, $n \leq 15$ строк с наибольшим и наименьшим количеством различающихся чисел.

24. Составить функцию, вычисляющую среднее арифметическое положительных элементов одномерного массива. Использовать составленную функцию для нахождения в матрице $T(m, m)$, $m \leq 12$ среднего арифметического положительных элементов верхней треугольной матрицы, нижней треугольной матрицы, а также

треугольных матриц, расположенных над побочной диагональю и под ней. Среди найденных определить максимальное и минимальное значения. Перед обращением к функции составить из элементов треугольной матрицы одномерный массив.

25. Составить функцию, которая подсчитывает в одномерном массиве количество элементов, принадлежащих интервалу $[A, B]$. Использовать составленную функцию для формирования из матрицы $D(l, m)$, $l \leq 12$, $m \leq 14$ новой, в которую переписать без пропусков те строки исходной, где количество элементов, попадающих в интервал $[A, B]$, больше, чем количество элементов, попадающих в интервалы $(-\infty, A)$ и (B, ∞) .

6.7. Рекурсивные подпрограммы

Язык Object Pascal допускает создание и использование рекурсивных подпрограмм. Это означает, что внутри подпрограммы можно обращаться к ней самой. Чтобы такое обращение имело смысл, подпрограмма должна быть организована должным образом, т. е. реализовывать именно рекурсивный алгоритм. Например, составим функцию, вычисляющую сумму элементов одномерного массива. Если использовать известный прием накопления суммы, предусматривающий вычисление ее в цикле, то такая функция может иметь следующий вид:

```
program Fun2rec;
{$APPTYPE CONSOLE}
uses
  SysUtils;
const
  nn=20;
type
  mas=array[1..nn] of Real;
var
  a:mas;
  i,n:Integer;
  s:Real;

function Sum(a:mas;n:Integer):Real;
var i:Integer;
begin
  Result:=0;
  for i:=1 to n do
    Result:=Result+a[i];
  end;
```

```
begin // РАЗДЕЛ ОПЕРАТОРОВ ПРОГРАММЫ
  ReadLn(n);
  for i:=1 to n do
    Read(a[i]);
  ReadLn;
  s:=sum(a,n);
  WriteLn('s= ',s:6:1);
  ReadLn;
end.
```

Внутри функции для накопления суммы использовалась внутренняя переменная `Result`. Использование внутри цикла оператора присваивания `Sum:=Sum+a[i]` привело бы к ошибке ввиду того, что появление в правой части оператора присваивания имени функции означает обращение к этой функции. Так как после имени функции отсутствуют параметры, это означает синтаксическую ошибку. Но поскольку внутри функции реализован циклический алгоритм, а не рекурсивный, то оператор `Sum:=Sum(a,n)+a[i]` приводил бы также к ошибке (зацикливанию).

Для построения рекурсивного варианта подпрограммы следует сформулировать рекурсивное утверждение и условие окончания рекурсии. В нашем случае рекурсивное утверждение будет заключаться в том, что сумма из n элементов массива может быть вычислена как сумма $(n-1)$ -го начального элемента и последнего, т. е. $S_n = S_{n-1} + a_n$ ($n > 1$). Условие окончания рекурсии $S_1 = a_1$. Рекурсивный вариант функции приведен ниже:

```
function Sum(a:mas;n:Integer):Real;
begin
  if n=1 then
    Sum:=a[n]
  else
    Sum:=a[n]+Sum(a,n-1)
end;
```

Рекурсивное описание обычно более компактно и выглядит нагляднее, если природа самого алгоритма рекурсивна. Однако надо принимать во внимание и то, как реализуется рекурсивный алгоритм в ЭВМ. Например, при $n = 4$ (массив состоит из четырех элементов) производится первое обращение к функции со значениями параметров массива a и $n = 4$. При этих значениях выполняется оператор `Sum:=a[4]+Sum(a,3)`, затем обращение к

функции со значениями a и 3, т. е. $\text{Sum}(a, 3)$. Следом выполняется обращение к функции со значениями a и 2, т. е. $\text{Sum}(a, 2)$. При выполнении этого обращения к функции производится еще одно обращение: $\text{Sum}(a, 1)$. На последнем шаге при обращении к функции с этими параметрами будет выполнен оператор $\text{Sum}:=a[1]$. Далее процесс развивается в обратном направлении, т. е. выполняется оператор $\text{Sum}:=a[2]+\text{Sum}(a, 1)$, что даст результат $\text{Sum}:=a[2]+a[1]$, затем оператор $\text{Sum}:=a[3]+\text{Sum}(a, 2)$ с результатом $\text{Sum}:=a[3]+a[2]+a[1]$. На последнем шаге выполнится оператор $\text{Sum}:=a[4]+\text{Sum}(a, 3)$, дающий окончательный результат. Рассмотрев описанный процесс, можно прийти к выводу, что рекурсивная подпрограмма требует больших затрат машинного времени и памяти. Это объясняется необходимостью повторных обращений к подпрограмме и хранения (в стеке) фактических параметров при обращениях к подпрограмме.

Рекурсия может быть и косвенной. В этом случае программа обращается сама к себе опосредованно, т. е. первая программа обращается ко второй, которая в свою очередь содержит обращение к первой:

```
procedure Pr1(a:Real;var b:Real);
begin
  . . . . .
  Pr2(a,b);
  . . . . .
end;

procedure Pr2(c:Real;var d:Real);
begin
  . . . . .
  Pr1(c,d);
  . . . . .
end;
```

Согласно правилам языка Object Pascal каждый идентификатор должен быть предварительно объявлен, поэтому такая конструкция недопустима. Чтобы разрешить использование косвенной рекурсии, в состав языка введено опережающее объявление, реализуемое директивой **forward**. Сначала необходимо объявить заголовок одной из процедур, затем поместить указанную директиву. После этого следует расположить вторую процедуру и разместить первую процедуру, в заголовке которой можно опустить список формальных параметров. Таким образом, во второй процедуре можно

обращаться к первой, так как к этому моменту ее заголовок уже известен второй процедуре:

```

procedure Pr1(a:Real;var b:Real); forward;

procedure Pr2(c:Real;var d:Real);
begin
  . . . . .
  Pr1(c,d);
  . . . . .
end;

procedure Pr1(a:Real;var b:Real);
begin
  . . . . .
  Pr2(a,b);
  . . . . .
end;

```

6.8. Пример выполнения задания на составление рекурсивной подпрограммы

Вычислить значение определенного интеграла, если соответствующий неопределенный интеграл и первообразная функция имеют следующий вид [10]:

$$\int \frac{x^2 dx}{(p^2 - q^2 x^2)^m} = \begin{cases} \frac{x}{2(m-1)q^2(p^2 - q^2 x^2)^{m-1}} - \frac{1}{2(m-1)q^2} \int \frac{dx}{(p^2 - q^2 x^2)^{m-1}}, & \text{если } m \geq 2; \\ -\frac{x}{q^2} + \frac{p}{2q^3} \ln \left| \frac{p+qx}{p-qx} \right|, & \text{если } m = 1; \end{cases}$$

$$\int \frac{dx}{(p^2 - q^2 x^2)^m} = \begin{cases} \frac{x}{2(m-1)p^2(p^2 - q^2 x^2)^{m-1}} + \frac{2m-3}{2(m-1)p^2} \int \frac{dx}{(p^2 - q^2 x^2)^{m-1}}, & \text{если } m \geq 2; \\ \frac{1}{pq} \ln \left| \frac{p+qx}{p-qx} \right|, & \text{если } m = 1. \end{cases}$$

Расчет определенного интеграла произвести двумя способами:
 1) с использованием приведенных выражений, вычисляемых ре-

курсивными подпрограммами (точное), и 2) с помощью численных методов (приближенно).

В основной программе необходимо осуществить ввод исходных данных — параметров p , q , m , пределов интегрирования a , b и точности ε , с которой будет вычисляться интеграл численными методами. Сравнить полученные результаты.

В рассматриваемой задаче исходный интеграл вычисляется по рекуррентной формуле, причем в этой формуле присутствует другой интеграл, который также вычисляется по рекуррентной зависимости. Текст подпрограммы для вычисления второго интеграла необходимо разместить до подпрограммы, в которой рассчитывается исходный интеграл, так как в противном случае будет производиться обращение к еще не объявленной подпрограмме. Поскольку каждая из подпрограмм должна возвращать одно значение, то используют функции, а не процедуры.

Для приближенного вычисления интеграла использован метод парабол, причем в программе для сокращения количества операций производится запоминание сумм значений функции, найденных на предыдущем шаге. Текст программы имеет следующий вид:

```
program PrimerRecPodp;
{$APPTYPE CONSOLE}
uses
  SysUtils, Math;
var
  p, q, a, b, int1, int2, eps: Real;
  m, l, k: Integer;

//Рекурсивная функция вычисления второго интеграла
function Integ1(p, q: Real; m: Integer; x: Real): Real;
begin
  if m >= 2 then
    Integ1 := x / (2 * (m - 1) * p * p * Power((p * p - q * q * x * x), m - 1))
      + (2 * m - 3) / (2 * (m - 1) * p * p) * Integ1(p, q, m - 1, x)
  else if m = 1 then
    Integ1 := Ln(Abs((p + q * x) / (p - q * x))) / (2 * p * q);
end;

//Рекурсивная функция исходного интеграла
function Integ(p, q: Real; m: Integer; x: Real): Real;
begin
  if m >= 2 then
```



```

    Integ:=x/(2*(m-1)*q*q*Power((p*p-q*q*x*x),m-1))
        -1/(2*(m-1)*q*q)*Integ1(p,q,m-1,x)
else if m=1 then
    Integ:=-x/(q*q)+p/(2*q*q*q)
        *Ln(Abs((p+q*x)/(p-q*x)));
end;

```

```

//Функция вычисления интеграла
//по методу парабол (Симпсона)
function SimpsonMod(a,b,eps,p,q:Real;
                    m:Integer):Real;
var
    n,i:Integer;
    x,dx,sch,snch,s1,s2,i1,i2:Real;
begin
    n:=Trunc((b-a)/Sqrt(Sqrt(eps)))+1;
    if Odd(n) then n:=n+1;
    s1:=a*a/Power((p*p-q*q*a*a),m);
    s2:=b*b/Power((p*p-q*q*b*b),m);
    sch:=0;
    snch:=0;
    dx:=(b-a)/n;
    for i:=1 to n div 2 do
    begin
        x:=a+2*i*dx;
        sch:=sch+x*x/Power((p*p-q*q*x*x),m);
        x:=a+(2*i-1)*dx;
        snch:=snch+x*x/Power((p*p-q*q*x*x),m);
    end;
    sch:=sch-s2;
    i2:=(s1+s2+4*snch+2*sch)*dx/3;
    repeat
        i1:=i2;
        n:=n*2;
        dx:=(b-a)/n;
        sch:=snch+sch;
        snch:=0;
        for i:=1 to n div 2 do
        begin
            x:=a+(2*i-1)*dx;
            snch:=snch+x*x/Power((p*p-q*q*x*x),m);
        end;
        i2:=(s1+s2+4*snch+2*sch)*dx/3;
    until i2-i1<eps;
end;

```

```

until Abs(i2-i1)/3<eps;
Result:=i2;
end;

begin // РАЗДЕЛ ОПЕРАТОРОВ ПРОГРАММЫ
  Writeln('Введите значения параметров  p,q,m');
  ReadLn(p,q,m);
  Writeln('Введите пределы интегрирования a,b'
    +' и точность eps ');
  ReadLn(a,b,eps);
  int1:=Integ(p,q,m,b)-Integ(p,q,m,a);
  l:=Trunc(-Log10(eps))+2;
  k:=Trunc(Log10(Abs(int1)))+m+3;
  Writeln('Значение интеграла '
    , 'по рекурсивной функции ='
    , int1:k:l);
  int2:= SimpsonMod(a,b,eps,p,q,m);
  Writeln('Значение интеграла '
    , 'по методу Симпсона ='
    , int2:k:l);
  ReadLn;
end.

```

6.9. Задания для самостоятельной работы

Составить программу для вычисления определенного интеграла с помощью рекурсивной подпрограммы. В основной программе осуществить ввод исходных данных — пределов интегрирования, параметра n и при необходимости — параметров p (варианты 1–3, 5–8, 16, 17, 19, 24–26), a (варианты 13, 14, 16–18), b (вариант 18); вычислить интеграл с помощью подпрограммы; вывести исходные данные и полученный результат с поясняющим текстом. Вычислить интеграл приближенно с точностью ϵ и сравнить полученные результаты. В заданиях приведены выражения для неопределенных интегралов и первообразных.

$$1. \int \sin^n px \, dx = \begin{cases} -\frac{\sin^{n-1} px \cos px}{np} + \frac{n-1}{n} \int \sin^{n-2} px \, dx, & n > 2, \\ \frac{x}{2} - \frac{1}{4p} \sin 2px, & n = 2, \\ -\frac{\cos px}{p}, & n = 1. \end{cases}$$

$$2. \int x \sin^n px \, dx =$$

$$= \begin{cases} \frac{\sin^{n-1} px}{n^2 p^2} (\sin px - np x \cos px) + \frac{n-1}{n} \int x \sin^{n-2} px \, dx, & n > 2, \\ \frac{x^2}{4} - \frac{x}{4} \sin 2px - \frac{\cos 2px}{8p^2}, & n = 2, \\ \frac{1}{p^2} \sin px - \frac{x}{p} \cos px, & n = 1. \end{cases}$$

$$3. \int x^n \sin px \, dx =$$

$$= \begin{cases} -\frac{\cos px}{p}, & n = 0, \\ \frac{1}{p^2} \sin px - \frac{x}{p} \cos px, & n = 1, \\ \frac{2x \sin px}{p^2} - \frac{p^2 x^2 - 2}{p^3} \cos px, & n = 2, \\ -\frac{x^n \cos px}{p} + \frac{nx^{n-1}}{p^2} \sin px - \frac{n(n-1)}{p^2} \int x^{n-2} \sin px \, dx, & n > 2. \end{cases}$$

$$4. \int \frac{\cos 2x}{\cos^n x} dx =$$

$$= \begin{cases} \frac{\sin 2x}{2}, & n = 0, \\ 2 \sin x - \ln \left| \operatorname{tg} \left(\frac{\pi}{4} + \frac{x}{2} \right) \right|, & n = 1, \\ -\frac{\sin x}{(n-1) \cos^{n-1} x} + \frac{n}{n-1} \int \frac{dx}{\cos^{n-2} x}, & n \geq 2 \text{ (см. задание 5)}. \end{cases}$$

$$5. \int \frac{dx}{\cos^n px} = \begin{cases} x, & n = 0, \\ \frac{1}{p} \ln \left| \operatorname{tg} \left(\frac{\pi}{4} + \frac{px}{2} \right) \right|, & n = 1, \\ \frac{\sin px}{(n-1)p \cos^{n-1} px} + \frac{n-2}{n-1} \int \frac{dx}{\cos^{n-2} px}, & n \geq 2. \end{cases}$$

$$6. \int \frac{dx}{\sin^n px} = \begin{cases} x, & n=0, \\ \frac{1}{p} \ln \left| \operatorname{tg} \frac{x}{2} \right|, & n=1, \\ -\frac{\cos px}{(n-1)p \sin^{n-1} px} + \frac{n-2}{n-1} \int \frac{dx}{\sin^{n-2} px}, & n \geq 2. \end{cases}$$

$$7. \int \cos^n px dx = \begin{cases} x, & n=0, \\ \frac{\sin px}{p}, & n=1, \\ \frac{x}{2} + \frac{\sin 2px}{4p}, & n=2, \\ \frac{\cos^{n-1} px \sin px}{np} + \frac{n-1}{n} \int \cos^{n-2} px dx, & n > 2. \end{cases}$$

$$8. \int x \cos^n px dx =$$

$$= \begin{cases} \frac{\cos px}{p^2} + \frac{x}{p} \sin px, & n=1, \\ \frac{x^2}{4} + \frac{x}{4p} \sin 2px + \frac{\cos 2px}{8p^2}, & n=2, \\ \frac{\cos^{n-1} px}{n^2 p^2} (\cos px + np x \sin px) + \frac{n-1}{n} \int x \cos^{n-2} px dx, & n > 2. \end{cases}$$

$$9. \int \frac{dx}{\sin x \cos^n x} = \begin{cases} \ln \left| \operatorname{tg} \frac{x}{2} \right|, & n=0, \\ \ln |\operatorname{tg} x|, & n=1, \\ \frac{1}{(n-1) \cos^{n-1} x} + \int \frac{dx}{\sin x \cos^{n-2} x}, & n \geq 2. \end{cases}$$

$$10. \int \frac{dx}{\sin^n x \cos x} = \begin{cases} \ln \left| \operatorname{tg} \left(\frac{\pi}{4} + \frac{x}{2} \right) \right|, & n=0, \\ \ln |\operatorname{tg} x|, & n=1, \\ -\frac{1}{(n-1) \sin^{n-1} x} + \int \frac{dx}{\sin^{n-2} x \cos x}, & n \geq 2. \end{cases}$$

$$11. \int \operatorname{tg}^n x dx = \begin{cases} x, & n=0, \\ -\ln |\cos x|, & n=1, \\ \frac{\operatorname{tg}^{n-1} x}{n-1} - \int \operatorname{tg}^{n-2} x dx, & n \geq 2. \end{cases}$$

$$12. \int \operatorname{ctg}^n x dx = \begin{cases} x, & n=0, \\ \ln |\sin x|, & n=1, \\ -\frac{\operatorname{ctg}^{n-1} x}{n-1} - \int \operatorname{ctg}^{n-2} x dx, & n \geq 2. \end{cases}$$

$$13. \int x^n e^{ax} dx = \begin{cases} \frac{e^{ax}}{a}, & n=0, \\ \frac{e^{ax}(ax-1)}{a^2}, & n=1, \\ \frac{x^n e^{ax}}{a} - \frac{n}{a} \int x^{n-1} e^{ax} dx, & n > 1. \end{cases}$$

$$14. \int \frac{e^{ax}}{x^n} dx = \begin{cases} \frac{e^{ax}}{a}, & n=0, \\ \ln |x| + \sum_{n=1}^{\infty} \frac{a^n x^n}{n \cdot n!}, & n=1, \\ -\frac{e^{ax}}{(n-1)x^{n-1}} + \frac{a}{n-1} \int \frac{e^{ax}}{x^{n-1}} dx, & n \geq 2. \end{cases}$$

$$15. \int x^n e^{-x^2} dx = \begin{cases} \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1) \cdot n!}, & n=0, \\ -\frac{x^{n-1} e^{-x^2}}{2} + \frac{n-1}{2} \int x^{n-2} e^{-x^2} dx, & n \geq 1. \end{cases}$$

$$16. \int e^{ax} \cos^n px dx = \begin{cases} \frac{e^{ax}}{a}, & n=0, \\ \frac{e^{ax}(a \cos px + p \sin px)}{a^2 + p^2}, & n=1, \\ \frac{e^{ax} \cos^{n-1} px}{a^2 + n^2 p^2} (a \cos px + np \sin px) + \\ + \frac{n(n-1)p^2}{a^2 + n^2 p^2} \int e^{ax} \cos^{n-2} px dx, & n \geq 2. \end{cases}$$

$$17. \int e^{ax} \sin^n px \, dx = \begin{cases} \frac{e^{ax}}{a}, & n=0, \\ \frac{e^{ax}(a \sin px - p \cos px)}{a^2 + p^2}, & n=1, \\ \frac{e^{ax} \sin^{n-1} px}{a^2 + n^2 p^2} (a \sin px - np \cos px) + \\ + \frac{n(n-1)p^2}{a^2 + n^2 p^2} \int e^{ax} \sin^{n-2} px \, dx, & n \geq 2. \end{cases}$$

$$18. \int \ln^n(a+bx) \, dx =$$

$$= \begin{cases} \frac{(a+bx) \ln(a+bx)}{b} - x, & n=1, \\ \frac{(a+bx) \ln^n(a+bx)}{b} - n \int \ln^{n-1}(a+bx) \, dx, & n > 1. \end{cases}$$

$$19. \int x^n \operatorname{sh} px \, dx =$$

$$= \begin{cases} \frac{\operatorname{ch} px}{p}, & n=0, \\ \frac{x^n \operatorname{ch} px}{p} - \frac{n}{p} \int x^{n-1} \operatorname{ch} px \, dx, & n \geq 1 \text{ (см. задание 26)}. \end{cases}$$

$$20. \int \operatorname{th}^n x \, dx = \begin{cases} x, & n=0, \\ \ln |\operatorname{ch} x|, & n=1, \\ -\frac{\operatorname{th}^{n-1} x}{n-1} + \int \operatorname{th}^{n-2} x \, dx, & n \geq 2. \end{cases}$$

$$21. \int \operatorname{cth}^n x \, dx = \begin{cases} x, & n=0, \\ \ln |\operatorname{sh} x|, & n=1, \\ -\frac{\operatorname{cth}^{n-1} x}{n-1} + \int \operatorname{cth}^{n-2} x \, dx, & n \geq 2. \end{cases}$$

$$22. \int \frac{dx}{\operatorname{sh} x \operatorname{ch}^n x} = \begin{cases} \ln \left| \operatorname{th} \frac{x}{2} \right|, & n=0, \\ \ln |\operatorname{th} x|, & n=1, \\ \frac{1}{(n-1)\operatorname{ch}^{n-1} x} + \int \frac{dx}{\operatorname{sh} x \operatorname{ch}^{n-2} x}, & n \geq 2. \end{cases}$$

$$23. \int \frac{dx}{\operatorname{sh}^n x \operatorname{ch} x} = \begin{cases} \operatorname{arctg}(\operatorname{sh} x), & n=0, \\ \ln |\operatorname{th} x|, & n=1, \\ -\frac{1}{(n-1)\operatorname{sh}^{n-1} x} - \int \frac{dx}{\operatorname{sh}^{n-2} x \operatorname{ch} x}, & n \geq 2. \end{cases}$$

$$24. \int x^n \cos px \, dx =$$

$$= \begin{cases} \frac{\sin px}{p}, & n=0, \\ \frac{1}{p^2} \cos px + \frac{x}{p} \sin px, & n=1, \\ \frac{2x \cos px}{p^2} + \frac{p^2 x^2 - 2}{p^2} \sin px, & n=2, \\ \frac{x^n \sin px}{p} + \frac{nx^{n-1}}{p^2} \cos px - \frac{n(n-1)}{p^2} \int x^{n-2} \cos px \, dx, & n > 2. \end{cases}$$

$$25. \int \frac{dx}{\operatorname{ch}^n px} = \begin{cases} x, & n=0, \\ \frac{1}{p} \operatorname{arctg}(\operatorname{sh} px), & n=1, \\ \frac{\operatorname{sh}(px)}{p(n-1)\operatorname{ch}^{n-1} px} + \frac{n-2}{n-1} \int \frac{dx}{\operatorname{ch}^{n-2} px}, & n \geq 2. \end{cases}$$

$$26. \int x^n \operatorname{ch} px \, dx =$$

$$= \begin{cases} \frac{\operatorname{sh} px}{p}, & n=0, \\ \frac{x^n \operatorname{sh} px}{p} - \frac{n}{p} \int x^{n-1} \operatorname{sh} px \, dx, & n \geq 1 \text{ (см. задание 19)}. \end{cases}$$

6.10. Дополнительные сведения о подпрограммах и массивах

До сих пор мы использовали для объявления формальных параметров-массивов имена ранее объявленных типов. В этом случае для объявления массивов, применяемых в качестве фактических параметров, должны использоваться те же имена типов массивов (иначе компилятор выдаст сообщение об ошибке), что в ряде случаев может привести к нерациональному расходу оперативной памяти. Например, если требуется процедура, объединяющая элементы массивов $X(n)$, $n \leq 100$ и $Y(m)$, $m \leq 100$ в массиве Z , то придется использовать следующие объявления имен типов и параметров процедуры:

```
type
  tMXY=array[1..100] of Real;
  tMZ=array[1..200] of real;
procedure Objedinenie(const X,Y:tMXY;
                      n,m:Integer;
                      out Z:tZ);
var
  i:Integer;
begin
  for i:=1 to n do
    Z[i]:=X[i];
  for i:=1 to m do
    Z[i+n]:=Y[i];
end;
```

и объявление фактических массивов, например

```
var
  A,B:tMXY;
  C:tZ;
```

даже когда реальные значения n и m значительно меньше 100, как в вызове процедуры

```
Objedinenie(A,B,3,5,C);
```

Исключить указанные ограничения и недостатки позволит использование открытых массивов при объявлении формальных параметров и динамических массивов в блоках программ и подпрограмм.

Параметры — открытые массивы

Для передачи одномерных массивов в подпрограмму можно использовать так называемые открытые массивы. Открытый массив представляет собой формальный параметр подпрограммы, описывающий базовый тип элементов, но не диапазон его индексов. Применительно к рассмотренному примеру процедура *Objedinenie* при использовании открытого массива будет иметь следующий вид:

```
procedure Objedinenie(const X,Y:array of Real;
                      n,m:Integer;
                      out Z: array of Real);

var
  i:Integer;
begin
  for i:=0 to n-1 do
    Z[i]:=X[i];
  for i:=0 to m-1 do
    Z[i+n]:=Y[i];
end;
```

У открытого массива минимальный индекс всегда равен нулю, что и нашло отражение в тексте подпрограммы при организации циклов.

При использовании открытых массивов подпрограмма становится более универсальной, так как соответствующими фактическими параметрами могут быть массивы любого типа с тем же базовым типом, что и базовый тип формального параметра. Однако это не приводит к уменьшению расхода памяти, если в подпрограмме обрабатывается лишь часть элементов массивов, объявленных в блоке вызывающей программы, как в последней реализации процедуры *Objedinenie*. Максимальный эффект использования памяти будет достигнут, когда в блоке вызывающей программы массив имеет размер, равный количеству обрабатываемых данных. В этом случае можно сократить список параметров, так как размер обрабатываемого массива определяют в подпрограмме с помощью стандартной функции *Length*, а индекс последнего элемента — с помощью стандартной функции *High*. Возвращаемое функцией *High* значение будет на единицу меньше размера массива, поскольку индексация элементов открытого массива начинается с нуля.

Таким образом, используя открытые массивы, можно передавать в подпрограмму и обрабатывать внутри нее одномерные

массивы с произвольным количеством элементов. Это демонстрирует следующее объявление процедуры *Objedinenie* для работы с открытыми массивами:

```
procedure Objedinenie(const X,Y: array of Real;
                      out Z: array of Real);
var
  i,n:Integer;
begin
  n:= High(X);
  for i:=0 to n do
    Z[i]:=X[i];
  for i:=0 to High(Y) do
    Z[i+n]:=Y[i];
end;
```

При обращении к подпрограмме, имеющей в качестве формального параметра открытый массив, для передачи фактического параметра можно использовать *конструктор массива*. Конструктор массива представляет собой список значений элементов массива, разделенных запятыми и заключенных в квадратные скобки. Например, используя конструктор массива, можно записать обращение к процедуре *Objedinenie* следующим образом:

```
Objedinenie([3.5, 2.2, 1, 8], [1.3, 3.2, 8.5, 7.4, 2.5], Z),
```

в результате чего элементам массива *Z* с индексами от 1 до 8 будут присвоены значения

3.5, 2.2, 1, 8, 1.3, 3.2, 8.5, 7.4, 2.5.

При передаче константного массива соответствующий формальный параметр в подпрограмме должен быть параметром-значением или параметром-константой, но не может быть параметром-переменной или выходным параметром-результатом.

Базовым типом открытого массива может быть сложный тип, объявленный ранее, например, статического массива или записи. Соответствующим фактическим параметром должен быть массив с тем же базовым типом.

Пример. Составить процедуру для вычисления суммы элементов матриц с произвольным числом строк и числом столбцов, равным 3. Использовать эту процедуру для матриц *A*(2, 3) и *B*(4, 3). Матрицы задать начальными значениями. Для контроля за изменением индексов суммируемых элементов матриц, значений элементов и накоплением суммы использовать в процедуре оператор вывода:

```
program Project2;
{$APPTYPE CONSOLE}
uses
  SysUtils;
type
  tn=array[-1..1] of Integer;
  tmn2=array[1..2] of tn;
  tmn4=array[-5..-2] of tn;

procedure Pp1(mn:array of tn; out s:Integer);
var i,j:Integer;
begin
  s:=0;
  for i:=Low(mn) to High(mn) do
    for j:=Low(tn) to High(tn) do
// или для второго измерения статического массива
// for j:=Low(mn[0]) to High(mn[0]) do
    begin
      s:=s+mn[i,j];
      WriteLn('      i=',i,' j=',j:2
        , ' mn[i,j]=' ,mn[i,j], ' s=',s);
    end;
end;

var
  A:tmn2=((5,3,7),
    (7,9,2));
  B:tmn4=((2,4,1),
    (7,6,2),
    (0,1,8),
    (3,7,5));
  r:Integer;

begin // РАЗДЕЛ ОПЕРАТОРОВ ПРОГРАММЫ
  WriteLn('Вызов процедуры '
    , 'для обработки матрицы A');
  Pp1(A,R);
  WriteLn('Сумма элементов массива B = ',R);
  WriteLn;
  WriteLn('Вызов процедуры '
    , 'для обработки матрицы B');
  Pp1(B,R);
```

```

WriteLn('Сумма элементов массива B = ',R);
ReadLn;
end.

```

Результат работы программы представлен на рис. 6.7.

```

D:\Temp\Project1.exe
Вызов процедуры для обработки матрицы A
i=0 j=-1 mn[i,j]=5 s=5
i=0 j= 0 mn[i,j]=3 s=8
i=0 j= 1 mn[i,j]=7 s=15
i=1 j=-1 mn[i,j]=7 s=22
i=1 j= 0 mn[i,j]=9 s=31
i=1 j= 1 mn[i,j]=2 s=33
Сумма элементов массива B = 33

Вызов процедуры для обработки матрицы B
i=0 j=-1 mn[i,j]=2 s=2
i=0 j= 0 mn[i,j]=4 s=6
i=0 j= 1 mn[i,j]=1 s=7
i=1 j=-1 mn[i,j]=7 s=14
i=1 j= 0 mn[i,j]=6 s=20
i=1 j= 1 mn[i,j]=2 s=22
i=2 j=-1 mn[i,j]=0 s=22
i=2 j= 0 mn[i,j]=1 s=23
i=2 j= 1 mn[i,j]=8 s=31
i=3 j=-1 mn[i,j]=3 s=34
i=3 j= 0 mn[i,j]=7 s=41
i=3 j= 1 mn[i,j]=5 s=46
Сумма элементов массива B = 46

```

Рис. 6.7

Следует обратить внимание на граничные значения индексов, представляющих номера строк и столбцов. Номера строк изменяются от 0 до значения, на единицу меньшего числа строк в фактическом массиве, так как формальный параметр является открытым массивом (по количеству содержащихся в нем одномерных массивов типа *tn*), а номера столбцов изменяются от -1, как в массивах — фактических параметрах, до 1, так как для типа статического массива стандартные функции *Low* и *High* возвращают минимальное и максимальное значения индексов из объявления типа статического массива.

Динамические массивы

Наиболее удобным способом рационального использования оперативной памяти для хранения массивов данных, размеры которых не определены или могут меняться в широком диапазоне, является применение динамических массивов. При объявлении таких массивов не указывают границы индексов. Объявив тип

```
tmas=array of Real;
```

можно использовать его для объявления динамических массивов, например,

```
var  
  A,B,C:tmas;
```

размеры которых можно задавать и изменять динамически (по мере необходимости) с помощью стандартной процедуры `SetLength`. Например, при выполнении `SetLength(A, 3)` размер массива *A* станет равным 3. В дальнейшем с помощью процедуры `SetLength` можно увеличить или уменьшить размер массива. В динамических массивах для индексации используют только целые значения, минимальное значение индекса всегда 0. Минимальное значение индекса массива определяется стандартной функцией `Low` (всегда 0), максимальное значение индекса — стандартной функцией `High`, а длина (размер) — стандартной функцией `Length` (как для открытых массивов в подпрограммах). При увеличении размера прежние элементы сохраняют свои значения, а новые со значениями 0 добавляются в конец массива. При уменьшении размера пропадают элементы с наибольшими индексами, а остальные сохраняют свои значения. При задании нуля в качестве размера массива занимаемая им память освобождается полностью.

Объявив двумерные динамические массивы, например,

```
type tmatr=array of array of Real;
```

в подпрограмму можно передавать матрицу (двумерный массив) с произвольным количеством строк и столбцов. В общем случае динамический массив с любым числом измерений и размерами может использоваться в качестве фактического параметра, соответствующего формальному параметру «открытый массив», причем в теле подпрограммы для любого его измерения можно будет найти длину и максимальное значение индекса (минимальное всегда равно 0), используя имя формального параметра и стандартные функции `Length` и `High` (для статических массивов при числе измерений, больше 2, для определения в подпрограмме диапазонов младших индексов потребуется использовать имена типов, что снижает универсальность подпрограммы). Например, для вычисления суммы элементов трехмерного динамического массива *X*, имеющего тип `tkmn`:

```
type  
  tmn=array of array of Integer;  
  tkmn=array of tmn;
```

подпрограмму можно оформить так:

```

procedure Sum1(const a:array of tnm;
               out r:Integer);
var i,j,k:integer;
begin
  r:=0;
  for k:=Low(a) to High(a) do
    for i:=Low(a[0]) to High(a[0]) do
      for j:=Low(a[0,0]) to High(a[0,0]) do
        r:=r+a[k,i,j];
      end;
    end;
  end;
end;

```

Выделение памяти и указание пределов изменения индексов по каждому измерению динамического массива производится в процессе выполнения программы процедурой `SetLength(a,n)`. Нижняя граница индексов по любому измерению динамического массива всегда равна нулю. Наибольший индекс примет тогда значение $n - 1$, где n — количество элементов массива, задаваемое при обращении к процедуре `SetLength`, а первый параметр — имя массива. При выделении памяти под матрицу можно выполнить обращение `SetLength(b,m,n)`, если матрица имеет прямоугольную форму, т. е. количество элементов во всех строках одинаково и равно n (m — количество строк матрицы).

Использование динамических массивов позволяет работать с матрицами, у которых в каждой строке разное количество элементов. При выделении памяти для многомерных массивов (в частности, для хранения двумерных матриц) сначала устанавливается длина первого измерения, затем второго, третьего и т. д. Например, выделим память для хранения элементов треугольной матрицы:

```

. . . . .
var
  a:array of array of Real;
  n,l,j:Integer;

begin // РАЗДЕЛ ОПЕРАТОРОВ ПРОГРАММЫ
  ReadLn(n);
  SetLength(a,n);
  for i:=0 to n-1 do
    Setlength(a[i],i+1);
  . . . . .
end.

```

При изменении длины уже созданного динамического массива сначала резервируется память для размещения нового мас-

сива, затем элементы старого копируются в новый, после чего освобождается память, выделенная под первоначальный массив.

Для освобождения памяти помимо процедуры `SetLength` можно использовать процедуру `Finalize` или идентификатору массива присвоить значение **nil**. Например, для освобождения памяти в предыдущем примере можно записать оператор `a:=nil` или `Finalize(a)`.

Перегружаемые подпрограммы

В рамках одной программы можно объявить несколько подпрограмм (процедур или функций) с одинаковыми именами, но различающихся по типу или числу параметров. Если при объявлении заголовка подпрограммы поставить ключевое слово **overload**, то компилятор после анализа фактических параметров в вызове подпрограммы выберет подходящий вариант из числа одноименных подпрограмм. Например, при делении целых чисел их надо делить нацело, а при делении действительных чисел требуется получать результат действительного типа. В этом случае следует объявить две одноименных функции деления, у которых будет различный тип аргументов, а также различаться тип результата:

```
program Funperegr;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var
  a,b,c:Real;

  k,l,m:Integer;
function Divide(a,b:Real):Real; overload;
begin
  Result:=a/b;
end;

function Divide(a,b:Integer):Integer; overload;
begin
  Result:=a div b;
end;

begin // РАЗДЕЛ ОПЕРАТОРОВ ПРОГРАММЫ
  ReadLn(a,b);
  c:= Divide(a,b);
```

```

ReadLn(k,n);
m:= Divide(k,l);
WriteLn('c =' ,c:6:2, ' m=',m);
ReadLn;
end.

```

При вводе $a = 3$ и $b = 2$ будет вызвана функция, выполняющая деление вещественных чисел, так как a и b — вещественные переменные, и выведено $c = 1.5$. При вводе $k = 3$ и $n = 2$ будет вызвана функция, выполняющая деление целых чисел, так как k и n — переменные целого типа, и будет выведено $m = 1$.

Параметры со значениями по умолчанию

Параметры в объявлении подпрограмм можно задавать по умолчанию. Значение по умолчанию — это значение, используемое при вызове подпрограммы, если в нее не передано фактическое. Использование параметров по умолчанию равносильно разрешению указывать в вызове подпрограммы не все необходимые параметры.

Значения по умолчанию задают добавлением после объявления типа формального параметра знака равенства, после которого записывается константное выражение. Рассмотрим функцию, которая рассчитывает силу тяжести бруска в форме прямоугольного параллелепипеда в зависимости от его размеров, материала (плотности) и ускорения свободного падения:

```

program Funumpar;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var
  a,b,c,ro,g:Real;
  p1,p2,p3,p4:Real;

function St(a:Real=4; b:Real=5; c:Real=6;
            ro:Real=7.8; g:Real=9.81):Real;
begin
  st:=a*b*c*ro*g;
end;

begin // РАЗДЕЛ ОПЕРАТОРОВ ПРОГРАММЫ
  ReadLn(a,b,c,ro,g);

```



```
p1:=St(a,b,c,ro,g);  
p2:=St(); //или p2:=st;  
p3:=St(a,b,c,ro);  
p4:=St(a,b,c);  
WriteLn('p1=',p1:7:2,' p2=',p2:7:2  
, ' p3=',p3:7:2,' p4=',p4:7:2);  
ReadLn;  
end.
```

При первом обращении к функции вычисляется сила тяжести бруска при значениях всех параметров, введенных в основной программе с клавиатуры, во втором случае, наоборот, все параметры берутся по умолчанию, т. е. вычисления ведутся с теми значениями, которые заданы в заголовке функции. В третьем случае по умолчанию берется только ускорение свободного падения, а в четвертом случае — плотность вещества и ускорение свободного падения.

Аргументы по умолчанию должны быть самыми правыми (последними) в списке параметров подпрограммы, при передаче параметров действует правило: если пропущенный параметр не является последним, то и все параметры, стоящие справа от него, тоже пропускаются. Принимая это правило во внимание, следует сделать вывод о том, что последними в списке параметров по умолчанию надо указывать те, которые чаще всего остаются равными заданным по умолчанию значениям. Пропускать при вызове можно только несколько последних параметров по умолчанию. Нельзя вызвать функцию следующим образом: $p1:=St(a, c, ro, g)$. Значения по умолчанию могут задаваться не всем, а только некоторым параметрам подпрограммы. При этом они должны располагаться в конце списка параметров, так как действует правило: если какой-то параметр имеет значение по умолчанию, то и все последующие должны иметь значения по умолчанию.

Примеры организации программ с подпрограммами

Пример 1. Определить в матрице все элементы, которые являются числами Фибоначчи, и переписать их в одномерный массив. Полученный массив упорядочить по возрастанию.

Числа Фибоначчи образуются по следующему правилу: $f_0 = 1$, $f_1 = 1$, $f_i = f_{i-2} + f_{i-1}$ ($i \geq 2$).

Программирование проведем с использованием подпрограмм. Составим функцию, которая определяет, является ли число, переданное в качестве аргумента, числом Фибоначчи. Кроме того, со-

ставим процедуру упорядочения элементов одномерного массива по возрастанию. Для проверки правильности выполнения отдельных этапов алгоритма выведем получаемый массив чисел Фибоначчи до его упорядочения и после. В этом случае целесообразно составить процедуру вывода элементов одномерного массива, так как это действие приходится осуществлять дважды. Для придания программе большей универсальности будем использовать динамические массивы.

Проверку заданного числа на число Фибоначчи будем проводить путем его последовательного сравнения со всеми числами Фибоначчи, начиная с единицы. Процесс сравнения заканчивается, если заданное число оказалось числом Фибоначчи или очередное число Фибоначчи превысило заданное число.

Для упорядочения чисел используется алгоритм пузырьковой сортировки:

```
program Procfib;
{$APPTYPE CONSOLE}
uses
  SysUtils;
type
  matr=array of array of Integer;
  mas=array of Integer;
var
  a:matr; b:mas;
  m,n,i,j,k:Integer;

  //Функция, определяющая, является ли
  //заданное число числом Фибоначчи
  function Fib(a:Integer):Boolean;
  var
    f0,f1,f2:Integer;
  begin
    Result:=False;
    f1:=0;f2:=1;
    repeat
      f0:=f1;
      f1:=f2;
      f2:=f0+f1;
      if a=f2 then Result:=True
    until Result or (f2>=a);
  end; //конец функции Fib
```

```
//Процедура упорядочения элементов
//одномерного массива по возрастанию
procedure Upor(var b:mas);
var
    i,k,c,n:Integer;
    pr:boolean;
begin
    n:=High(b);
    k:=0;
    repeat
        k:=k+1;
        pr:=True;
        for i:=0 to n-k do
            if b[i]>b[i+1] then
                begin
                    c:=b[i];
                    b[i]:=b[i+1];
                    b[i+1]:=c;
                    pr:=False;
                end;
        until pr;
    end; //конец процедуры Upor

//Процедура вывода элементов одномерного массива
procedure Wiwod(const b:mas);
var
    i:Integer;
begin
    for i:=0 to High(b) do
        Write(b[i]:4);
        WriteLn;
    end; //конец процедуры Wiwod

begin //РАЗДЕЛ ОПЕРАТОРОВ
    WriteLn('Введите количество строк и столбцов');
    ReadLn(m,n);
    SetLength(a,m,n);
    Writeln('Введите матрицу по строкам');
    for i:=0 to m-1 do
        begin
            for j:=0 to n-1 do
                Read(a[i,j]);
```

```
    ReadLn;
end;
WriteLn('Исходная матрица');
for i:=0 to m-1 do
begin
    for j:=0 to n-1 do
        Write(a[i,j]:4);
        WriteLn;
    end;
    k:=0;
    for i:=0 to m-1 do
        for j:=0 to n-1 do
            if Fib(a[i,j]) then
                begin
                    k:=k+1;
                    SetLength(b,k);
                    b[k-1]:=a[i,j];
                end;
            if k=0 then
                WriteLn('В матрице нет чисел Фибоначчи')
            else
                begin
                    WriteLn('Массив чисел Фибоначчи');
                    Wiwod(b);
                    Upor(b);
                    WriteLn('Упорядоченный массив '
                        , 'чисел Фибоначчи');
                    Wiwod(b);
                end;
            ReadLn;
        end;
    end.
```

Пример 2. Определить в каждой строке матрицы $A(m, n)$, $m \leq 10$, $n \leq 12$ первое по порядку простое число и занести его в одномерный массив. Если в строке нет простых чисел, то для этой строки занести в массив нулевой элемент.

Поскольку максимальные размеры матрицы заданы в условии, в программе целесообразно использовать статические массивы. Определение вида числа (простое или нет) проведем в подпрограмме-функции, которая возвращает значение *истина*, если число простое, и *ложь* — в противном случае.

Для определения вида числа следует определить остатки от деления этого числа на все целые, начиная с двух и заканчивая

числом, которое является квадратным корнем из этого числа. Если при делении на все эти числа остатки будут отличны от нуля, то число является простым. Если же при делении на очередное число остаток окажется равным нулю, то это число не является простым, и дальнейшие проверки можно не производить. Эти действия выполняются в функции с помощью цикла с заранее неизвестным числом повторений.

Поиск первого простого числа в каждой строке матрицы также реализован в основной программе с помощью цикла с заранее неизвестным числом повторений. Цикл выполняется, пока не найдено в этой строке простое число и не исчерпаны элементы строки. Если простое число найдено или все элементы строки проанализированы, то цикл следует завершить:

```
program funprost;
{$APPTYPE CONSOLE}
uses
  SysUtils;
const
  mm=10; nn=12;
type
  matr=array[1..mm,1..nn] of Integer;
  mas=array[1..mm] of Integer;
var
  a:matr; i,j,m,n:Integer;
  b:mas; pr:boolean;

function Pros(a:Integer):Boolean;
var
  i:Integer;
begin
  i:=2;
  Result:=True;
  while Result and (i<=Sqrt(a)) do
    if a mod i=0 then
      Result:=False
    else
      i:=i+1;
  end; //конец функции Pros

begin //РАЗДЕЛ ОПЕРАТОРОВ ПРОГРАММЫ
  WriteLn('Введите количество строк '
    , 'и столбцов матрицы');
```

```
ReadLn(m,n);
  SetLength(a,m,n);
WriteLn('Введите матрицу по строкам');
for i:=1 to m do
begin
  for j:=1 to n do
    Read(a[i,j]);
  ReadLn;
end;
for i:=1 to m do
begin
  j:=1;
  pr:=False;
  while(j<=n) and not pr do
  begin
    pr:= Pros(a[i,j]);
    if not pr then j:=j+1;
  end;
  if pr then b[i]:=a[i,j]
  else b[i]:=0;
end;
WriteLn('Исходная матрица');
for i:=1 to m do
begin
  for j:=1 to n do
    Write(a[i,j]:4);
  WriteLn;
end;
WriteLn('Полученный массив');
for i:=1 to m do
  Write(b[i]:4);
ReadLn;
end.
```

Пример 3. Составить процедуру, которая в каждой строке матрицы заменяет каждый положительный элемент минимальным отрицательным элементом из стоящих между предыдущим и текущим положительными. Если положительный элемент стоит на первом месте в строке, то его не изменять. Если отрицательных элементов нет между положительными, то замену не производить.

Использовать составленную процедуру для матрицы $W(m, n)$, $m \leq 12$, $n \leq 15$. В основной программе произведем ввод исходных

данных (количества строк и столбцов матрицы и самой матрицы), с помощью процедуры преобразуем матрицу и осуществим вывод исходной и преобразованной матриц. Поскольку вывод необходимо выполнить два раза, то целесообразно организовать для этого процедуру.

Преобразование матрицы осуществляется в процедуре. Суть выполняемых действий сводится к тому, что каждый элемент строки матрицы необходимо анализировать на положительность. При появлении положительного элемента надо найти минимальный элемент, стоящий между предыдущим и текущим положительными. Если найденный минимальный элемент окажется отрицательным, то его следует поместить на место текущего положительного элемента. После нахождения положительного элемента необходимо запомнить индекс столбца, в котором он расположен:

```
program Proc_sam_pol;
{$APPTYPE CONSOLE}
uses
  SysUtils;
const
  mm=12;nn=15;
type
  matr=array[1..mm,1..nn] of Real;
var
  w:matr;
  i,j,m,n:Integer;

procedure Wiwod(a:matr;m,n:Integer);
var
  i,j:Integer;
begin
  for i:=1 to m do
    begin
      for j:=1 to n do
        Write(a[i,j]:6:1,' ');
      WriteLn;
    end;
  end;
end;

procedure Samen(var a:matr;m,n:Integer);
var
  i,j,jpol,l:Integer;
  Min:Real;
```

```
begin
  for i:=1 to m do
    begin
      jpol:=0;
      for j:=1 to n do
        begin
          if a[i,j]>0 then
            begin
              Min:=a[i,jpol+1];
              for l:=jpol+1 to j-1 do
                if a[i,l]<Min then Min:=a[i,l];
              if Min<0 then a[i,j]:=Min;
              jpol:=j;
            end;
          end;
        end;
      end;
    end;
  end;

begin //РАЗДЕЛ ОПЕРАТОРОВ ПРОГРАММЫ
  WriteLn(Rus('Введите количество строк '))
    ,Rus('и столбцов матрицы'));
  ReadLn(m,n);
  WriteLn(Rus('Введите матрицу по строкам'));
  for i:=1 to m do
    begin
      for j:=1 to n do
        Read(w[i,j]);
      ReadLn;
    end;
  WriteLn(Rus('Исходная матрица'));
  Wiwod(w,m,n);
  Samen(w,m,n);
  WriteLn(Rus('Преобразованная матрица'));
  Wiwod(w,m,n);
  ReadLn;
end.
```


7. Модули пользователей

7.1. Создание и использование модулей

Приложение Delphi помимо стандартных модулей может использовать модули, создаваемые пользователями. Каждый модуль пользователя является отдельно подготовленной и хранящейся в отдельном файле с расширением .pas программной единицей, которая может быть использована любой программой.

В Delphi реализован модульный принцип программирования, причем модули играют роль наборов заранее подготовленных и отлаженных подпрограмм, именованных констант, типов, переменных, которые могут использоваться в тех частях программы (в основной программе и модулях), куда они подключены с помощью *предложения использования*, показанного на рис. 7.1.

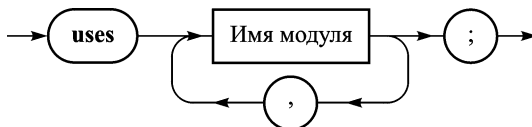


Рис. 7.1

Модули также могут содержать код, выполняемый до передачи управления в указанные части программы, и код, выполняемый после возврата управления из них.

Исходный текст модуля имеет следующую структуру:

unit <имя модуля>;	{заголовок модуля}
interface	{интерфейсная часть}
<предложение использования>;	
<объявление именованных констант>	
<объявление типов>	
<объявление переменных>	
<объявление заголовков подпрограмм>	
implementation	{часть реализации}
<предложение использования>;	
<объявление меток>	
<объявление именованных констант>	
<объявление типов>	

<объявление переменных>
<объявление подпрограмм>

initialization {часть инициализации}

<операторы>

finalization {часть финализации}

<операторы>

end.

Любая часть модуля может быть пустой, однако ключевые слова **interface** и **implementation** опускать нельзя. Если в модуле не требуются части инициализации и финализации, то не нужно записывать слова **initialization** и **finalization**, но если часть финализации необходима, то должна присутствовать и часть инициализации, даже если в ней нет ни одного оператора.

В *интерфейсной части* объявляют только те имена, которые могут использоваться в основной программе или в модуле, к которому подключен данный модуль, причем именованные константы, типы и переменные объявляют как обычно, а процедуры и функции — своими заголовками. Эти имена будут известны и в остальных частях модуля.

Полное объявление подпрограмм, заголовки которых представлены в интерфейсной части, должно располагаться в *части реализации*. Кроме них в части реализации могут объявляться другие подпрограммы, метки, а также, в дополнение к объявлениям в интерфейсной части, именованные константы, типы и переменные, недоступные вне модуля. Они могут иметь вспомогательное значение при реализации подпрограмм интерфейсной части или использоваться в частях инициализации и финализации.

Часть инициализации предназначена для размещения операторов, выполнение которых предшествует выполнению операторов основной программы (той части программы, которая использует данный модуль), а *часть финализации* — для операторов, выполняемых после окончания основной (указанной части) программы. Например, в части инициализации можно создать временные файлы для хранения данных только во время выполнения программы, а в части финализации удалить эти файлы. Такое решение позволит упростить разработку программы и избежать засорения дискового пространства ненужными файлами при возникновении исключений в основной программе, так как часть финализации будет выполнена в любом случае. Например, если модуль Unit1, в котором создается файл 'TempFile.txt':

```
unit Unit1;
interface
var
    //Файловая переменная f будет представлять
    //временный файл в программе,
    //использующей данный модуль
    f: TextFile;
implementation
initialization
    //Связать файловую переменную f
    //с внешним именем файла 'TempFile.txt'
    Assign(f, 'TempFile.txt');
    Rewrite(f); //создать файл
finalization
    if FileExists('TempFile.txt') then
        //Если файл не был уничтожен,
        begin //то
            try
                CloseFile(f); //Закреть файл
            except
                //Сообщения не будет,
                //если файл был закрыт в основной программе
            end;
            Erase(f); //Уничтожить закрытый файл
        end
    end.
end.
```

— присоединить к программе Project1:

```
program Project1;
{$APPTYPE CONSOLE}
uses
    SysUtils,
    Unit1; //Использовать модуль Unit1 в этой программе
. . . . .
```

то файл 'TempFile.txt' будет создан до передачи управления в основную программу и с ним можно будет работать, используя файловую переменную *f* (открывать, закрывать, открывать для добавления текста, записывать текст в файл, читать из файла, уничтожать файл), а после выхода из основной программы при любом ее завершении (нормальном или аварийном) управление будет передано операторам части финализации модуля Unit1, и файл 'TempFile.txt' будет уничтожен.

Объявления в отдельных частях модуля могут располагаться в любой последовательности и чередоваться при соблюдении правила использования имен: при объявлении нового имени (константы, типа, переменной, процедуры, функции) могут использоваться только ранее объявленные имена или имена из подключенных модулей.

Компилятор узнает о подключенных модулях, анализируя предложения использования в основной программе или в самих модулях. *Предложение использования* строится из ключевого слова **uses** и следующего за ним списка имен подключаемых модулей. В основной программе и в интерфейсной части модуля формы, создаваемых в среде Delphi, предложения использования вставляются автоматически — имена новых модулей следует просто добавить в уже имеющийся список. При создании нового модуля пользователь при необходимости должен сам добавить предложения использования. Например, в модуль, создаваемый для решения вычислительных задач, следует в интерфейсную часть или в часть реализации включить предложение использования со стандартным модулем **Math**:

```
uses Math;
```

Присутствие имени модуля в предложении использования основной программы (или другого модуля) означает, что объявленные в его интерфейсной части константы, типы, переменные и подпрограммы доступны для использования.

Существует отличие в назначении предложений использования в разных частях модуля. Предложения использования в интерфейсных частях должны строиться так, чтобы не возникало взаимных ссылок модулей непосредственно или через другие модули. Только в этом случае компилятор сможет определить порядок использования объявлений из интерфейсных частей.

Для части реализации такого ограничения нет. Это позволяет строить взаимно рекурсивные подпрограммы, принадлежащие разным модулям. Можно использовать взаимную рекурсию и внутри модуля без применения директивы **forward**, так как оно действует в модулях по умолчанию для всех подпрограмм, заголовки которых размещены в интерфейсной части.

При разработке программы в среде Delphi вновь создаваемый модуль (по команде *File/New/Unit*) добавляется в предложение использования основной программы автоматически. Также автоматически добавляется в предложение использования основной программы ссылка на готовый модуль. Для этого следует ввести команду *Project/Add to Project...*, в диалоге выбрать папку с

модулем и в ней — сам модуль с расширением .pas (при разработке приложения с формой текст основной программы можно отобразить командой *Project/View Source*). После добавления в проект модулей программист уже в них должен прописать в предложениях использования взаимные ссылки. В приложениях с формой убедиться, что модуль подключен, можно, отобразив текст основной программы командой *Project/View Source*, где должна быть ссылка на модуль в предложении **uses**, или открыв окно диалога Project Manager командой *View/Project Manager...*

Пример 1. Составить консольное приложение, выполняющее обработку матриц по формуле $(A + B) \cdot C + D$. Приложение должно использовать модули Unit1 и Unit2, подготовленные в проекте и хранящиеся в одной папке с основной программой. Модуль Unit1 предназначен для объявления типа массивов, которые будут хранить матрицы, участвующие в вычислениях, и две процедуры: ReadMatr ввода матрицы и WriteMatr вывода. Модуль Unit2 также должен содержать процедуры AddMatr сложения матриц и MulMatr умножения.

Требования к модулю Unit1. Для хранения матриц следует использовать двумерные динамические массивы. Первый параметр процедуры ReadMatr должен представлять матрицу, размеры которой заданы вторым и третьим параметрами, а четвертый параметр целого типа с начальным значением 0 указывать режим работы процедуры. Процедура ReadMatr должна обеспечить ввод матрицы с клавиатуры в виде матрицы по строкам, если четвертый параметр при ее вызове опущен. Этот параметр предназначен для отладки программ, использующих модуль Unit1. При задании в вызове процедуры ReadMatr четвертого параметра, не равного 0, процедура должна генерировать матрицу случайных чисел от нуля до абсолютного значения этого параметра включительно, причем если он меньше нуля, то при многократных запусках программы генерироваться должны разные данные, иначе — одни и те же. Сгенерированные матрицы следует выводить в виде матрицы по строкам. Процедура WriteMatr должна иметь один параметр, представляющий матрицу, и обеспечивать ее вывод в виде матрицы по строкам с пробелом между числами не менее одного.

Требования к модулю Unit2. Для хранения матриц следует использовать двумерные динамические массивы типа, объявленного в модуле Unit1. В процедурах AddMatr и MulMatr два первых параметра должны представлять только входные данные (матрицы, представляющие только исходные данные), а третий — только выходные данные (результатирующую матрицу).

Требования к основной программе. Использовать условную компиляцию, обеспечивающую при объявлении имени Debug директивой `{ $IFDEF Debug }` генерацию случайных чисел для матриц, представляющих исходные данные, иначе — ввод матриц с клавиатуры.

```
//Модуль Unit1 объявления типа двумерного
//динамического массива
//и процедур ввода и вывода матриц.
unit Unit1;
interface
    uses
        Math;
    type
        tm1=array of Integer;
        //тип двумерного динамического массива
        tm2=array of tm1;
    procedure ReadMatr(out x:tm2;
                       m,n:Integer;
                       r:Integer=0);
    procedure WriteMatr(const x:tm2);
implementation
    procedure ReadMatr(out x:tm2;
                       m,n:Integer;
                       r:Integer=0);
    {Ввод матрицы m*n по строкам в динамический
    массив типа tm2: при r=0 - ввод с клавиатуры,
    иначе - от датчика случайных чисел,
    причем при r>0 - без Randomize,
    иначе - с Randomize}
    var i,j:Integer;
    begin
        //Установить размеры массива x
        //равными размерам вводимой матрицы
        SetLength(x,m,n);
        if r=0 then begin
            //Ввод матрицы с клавиатуры
            for i:=0 to m-1 do begin
                for j:=0 to n-1 do
                    Read (x[i,j]);
                ReadLn
            end;
        end
    end
```

```
else
begin
  //Генерация матрицы случайных целых чисел,
  //из интервала 0..r+1
  if r<0 then //При r<0 будет создан
    //новый набор случайных чисел для матрицы
    Randomize;
  r:=Abs(r)+1;
  for i:=0 to m-1 do
  for j:=0 to n-1 do
    x[i,j]:=Random(r);
  WriteMatr(x);
end;
end; //ReadMatr
procedure WriteMatr(const x:tm2);
{Вывод матрицы m*n по строкам
из динамического массива}
var i,j,m,n,xmax:Integer;
begin
m:=High(x);
n:=High(x[0]);
xmax:=Abs(x[1,1]);
for i:=0 to m do
  for j:=0 to n do
    if Abs(x[i,j])>xmax then
      xmax:=Abs(x[i,j]);
for i:=0 to m do begin
  for j:=0 to n do
    Write (x[i,j]:Trunc(Log10(xmax))+2);
  WriteLn;
end;
end; //WriteMatr
end.
```

```
//Модуль Unit2 объявления процедур
//сложения и умножения матриц
unit Unit2;
interface
uses
  Unit1;
{Фактические параметры обеих процедур не могут
быть одновременно и входными, и выходными}
procedure AddMatr(const x,y:tm2; out z:tm2);
```

```

    procedure MulMatr(const x,y:tm2; out z:tm2);
implementation
    procedure AddMatr(const x,y:tm2; out z:tm2);
    {Процедура сложения матриц Z=X+Y}
    var
        i,j,m,n:Integer;
    begin
        m:=High(x);    //m+1 - число строк в X, Y и Z
        n:=High(x[0]); //n+1 - число столбцов в X, Y и Z
        //Установить размеры массива z
        //равными размерам матрицы Z
        SetLength(z,m+1,n+1);
        for i:=0 to m do
            for j:=0 to n do
                z[i,j]:=y[i,j]+x[i,j];
            end; //AddMatr
        procedure MulMatr(const x,y:tm2; out z:tm2);
        {Процедура умножения матриц Z=X*Y}
        var
            i,j,m,n,l,k:Integer;
        begin
            // m+1 - число строк в X и Z столбцов в Y
            m:=High(x);
            // n+1 - число столбцов в X
            n:=High(x[0]);
            // l+1 - число столбцов в Y
            l:=High(y[0]);
            //Установить размеры массива z
            //равными размерам матрицы Z
            SetLength(z,m+1,l+1);
            for i:=0 to m do
                for j:=0 to l do begin
                    z[i,j]:=0;
                    for k:=0 to n do
                        z[i,j]:=z[i,j]+x[i,k]*y[k,j];
                    end;
                end; //MulMatr
            end.

```

```

//Основная программа вычисления матрицы E=(A+B)*C+D
program Project1;
{$APPTYPE CONSOLE}

```



```
uses
    SysUtils,
    Unit1 in 'Unit1.pas',
    Unit2 in 'Unit2.pas';
//Объявление имени Debug,
//используемого при условной компиляции.
{$DEFINE Debug}
//Отменить объявление имени Debug можно,
//удалив строку с директивой {$DEFINE Debug}
//или превратив ее в комментарий: //{$DEFINE Debug}.
var
    a,b,c,d,e,f,g:tm2;
begin
    {$IFDEF Debug}
    //Если имя Debug объявлено директивой
    //{$DEFINE Debug}, то в исполняемую программу
    //будут включены следующие операторы, обеспечивающие
    //генерацию матриц случайных чисел и их вывод
    WriteLn(' Матрица A 2x3 целых случайных чисел '
        , 'в диапазоне 0..7');
    ReadMatr(a,2,3,8);
    WriteLn(' Матрица B 2x3 целых случайных чисел '
        , 'в диапазоне 0..5');
    ReadMatr(b,2,3,6);
    WriteLn(' Матрица C 3x4 целых случайных чисел '
        , 'в диапазоне 0..8');
    ReadMatr(c,3,4,-9);
    WriteLn(' Матрица D 2x4 целых случайных чисел '
        , 'в диапазоне 0..8');
    ReadMatr(d,2,4,9);
    {$ELSE} //иначе, то есть если имя Debug
        //не объявлено, то в исполняемую программу
        //будут включены следующие операторы,
        //обеспечивающие ввод матриц с клавиатуры.
    WriteLn(' Введите матрицу A 2x3');
    ReadMatr(a,2,3);
    WriteLn(' Введите матрицу B 2x3');
    ReadMatr(b,2,3);
    WriteLn(' Введите матрицу C 3x4');
    ReadMatr(c,3,4);
    WriteLn(' Введите матрицу D 2x4');
    ReadMatr(d,2,4);
    {$ENDIF}
```

```
//Вычисление и вывод матриц
AddMatr(a,b,f);
WriteLn(' Матрица F=(A+B) ');
WriteMatr(f);
MulMatr(f,c,g);
WriteLn(' Матрица G=(A+B)xC');
WriteMatr(g);
AddMatr(g,d,e);
WriteLn(' Матрица E=(A+B)xC+D');
WriteMatr(e);
ReadLn;
end.
```

При запуске программы с директивой, объявляющей имя Debug, в окно программы сразу будут выведены результаты построения матриц случайных чисел с поясняющими текстами и вычисления $(A + B) \cdot C + D$ по шагам, как показано на рис. 7.2.

```
с:\z_D\!\vt-i-!\t2005-6 yr\Практикум 1 2006 весна\!Весь практикум\!!!Вес...
Матрица A 2x3 целых случайных чисел в диапазоне 0..7
0 0 7
1 2 6
Матрица B 2x3 целых случайных чисел в диапазоне 0..5
2 1 2
2 0 3
Матрица C 3x4 целых случайных чисел в диапазоне 0..8
1 0 1 2
4 7 5 0
6 4 6 6
Матрица D 2x4 целых случайных чисел в диапазоне 0..8
8 5 1 6
4 6 8 1
Матрица F=(A+B)
2 1 9
3 2 9
Матрица G=(A+B)xC
60 43 61 58
65 50 67 60
Матрица E=(A+B)xC+D
68 48 62 64
69 56 75 61
```

Рис. 7.2

При повторных запусках программы матрицы A и B не изменяются, а C и D будут создаваться всякий раз новые, так как генерация новых случайных чисел стандартной функцией Random начинается с вызова ReadMatr(c, 3, 4, -9) для построения матрицы C , где отрицательное значение последнего параметра требует выполнения процедуры Randomize. После отладки программы строку с директивой { \$DEFINE Debug } следует превратить в комментарий.

К Unit1 должен подключаться стандартный модуль Math, так как в процедуре WriteMatr используется стандартная процедура Log10, но поскольку объявления интерфейсной части Unit1 не требуют подключения Math, то предложение использования `uses Math;` размещено в его части реализации.

Модуль Unit1 должен быть подключен к Unit2 в интерфейсной части, так как объявленный в Unit1 тип `tm2` двумерного динамического массива используется при объявлении заголовков процедур AddMatr и MulMatr.

Следует обратить внимание на одну особенность динамических массивов при работе с матрицами, использованную в процедурах WriteMatr, AddMatr и MulMatr, — размеры массивов всегда совпадают с размерами матриц, что исключает необходимость указания их в качестве параметров подпрограмм.

В соответствии с условиями примера 1 входные параметры объявлены как **const**-параметры, а выходные — как **out**-параметры. С одной стороны, это позволяет избежать выделения памяти под параметры, представляющие матрицы, и копирования в нее значений соответствующих входных параметров, а с другой — ограничивает одновременное использование одних и тех же имен массивов в качестве входных и выходных фактических параметров в вызове процедуры, что в ряде случаев позволило бы уменьшить число операторов и дополнительных массивов (в программе это массивы *f* и *g*). Для расширения возможностей процедур AddMatr и MulMatr можно создать новый модуль, заменяющий Unit2, или использовать его подпрограммы в новом модуле. Рассмотрим второй вариант, так как он интересен еще и использованием одинаковых глобальных имен, объявленных в разных модулях.

Пример 2. Составить консольное приложение, выполняющее обработку матриц по формуле $ABC + C + DA$. Приложение должно использовать подготовленные ранее (см. пример 1) модули Unit1 и Unit2, подключаемые к проекту из другой папки, и новый модуль Unit3, хранящийся в одной папке с основной программой. Модуль Unit3 также должен содержать объявления процедур AddMatr и MulMatr с тем же назначением, что и одноименные процедуры модуля Unit2. Указанные процедуры модуля Unit3 должны допускать применение одного и того же массива при обращении к ним в качестве разных параметров одновременно и по возможности использовать одноименные процедуры модуля Unit2.

```

unit Unit3;
interface
  uses Unit1, Unit2;
  procedure AddMatr(var x, y, z: tm2);
  procedure MulMatr(var x, y, z: tm2);
implementation
  procedure AddMatr(var x, y, z: tm2);
  var
    i, j, m, n: Integer;
  begin
    m := High(x);
    n := High(x[0]);
    if (x <> z) and (y <> z) then
      //Если адрес z отличен от адресов x и y,
      //то дать массиву z те же размеры, что у x и y
      SetLength(z, m+1, n+1);
    for i := 0 to m do
      for j := 0 to n do
        z[i, j] := y[i, j] + x[i, j];
      end; //AddMatr

  procedure MulMatr(var x, y, z: tm2);
  var
    w: tm2;
  begin
    if (x = z) or (y = z) then //Если адрес z равен
      //адресу x или y,
    begin
      //то вычислить
      Unit2.MulMatr(x, y, w); //произведение X*Y
      //и поместить в массив w,
      z := w; //а затем скопировать
      //в массив z,
    end
    else //иначе вычислить
      Unit2.MulMatr(x, y, z); //произведение X*Y
      //и поместить в массив z
    end; //MulMatr
  end.

program Project1;
//Основная программа вычисления матрицы A•B•C+C•D•A
{$APPTYPE CONSOLE}
uses
  SysUtils,

```

```
Unit1 in '..\Пример 1\Unit1.pas',
Unit2 in '..\Пример 1\Unit2.pas',
Unit3 in 'Unit3.pas';
//Объявление имени Debug,
//используемого при условной компиляции
{$DEFINE Debug}
//Отменить объявление имени Debug можно,
//удалив строку с директивой {$DEFINE Debug} или
//превратив ее в комментарий: //{ $DEFINE Debug}.
var
    a,b,c,d:tm2;
begin
    WriteLn;
    {$IFDEF Debug}
    //Если имя Debug объявлено
    //директивой {$DEFINE Debug}, то в исполняемую
    //программу будут включены следующие операторы,
    //обеспечивающие генерацию матриц случайных
    //чисел и их вывод,
    WriteLn(' Матрица A 2x3 целых случайных чисел'
        , ' в диапазоне 0..10');
    ReadMatr(a,2,3,10);
    WriteLn(' Матрица B 3x3 целых случайных чисел'
        , ' в диапазоне 0..10');
    ReadMatr(b,3,3,10);
    WriteLn(' Матрица C 3x2 целых случайных чисел'
        , ' в диапазоне 0..10');
    ReadMatr(c,3,2,10);
    WriteLn(' Матрица D 3x2 целых случайных чисел'
        , ' в диапазоне 0..10');
    ReadMatr(d,3,2,-10);
    {$ELSE} //иначе, то есть если имя Debug
    //не объявлено, то в исполняемую программу будут
    //включены следующие операторы,
    //обеспечивающие ввод матриц с клавиатуры.
    WriteLn(' Введите матрицу A 2x3');
    ReadMatr(a,2,3);
    WriteLn(' Введите матрицу B 3x3');
    ReadMatr(b,3,3);
    WriteLn(' Введите матрицу C 3x2');
    ReadMatr(c,3,2);
    WriteLn(' Введите матрицу D 3x2');
    ReadMatr(d,3,2);
    {$ENDIF}
```

```
//Вычисление и вывод матриц.  
MulMatr(a,b,b);  
WriteLn(' Матрица AxB');  
WriteMatr(b);  
MulMatr(b,c,b);  
WriteLn(' Матрица AxBxC');  
WriteMatr(b);  
AddMatr(b,c,b);  
WriteLn(' Матрица AxBxC+C');  
WriteMatr(b);  
Unit2.MulMatr(a,d,c);  
WriteLn(' Матрица AxD');  
WriteMatr(c);  
Unit2.AddMatr(b,c,a);  
WriteLn(' Матрица AxBxC+C+AxD');  
WriteMatr(a);  
ReadLn;  
end.
```

Чтобы использовать при вызове процедуры одно и то же имя массива в качестве параметра, можно сделать входные параметры параметрами-значениями. Но тогда при вызове процедуры с любыми параметрами выделялась бы память и выполнялось копирование входных матриц. Для минимизации указанных недостатков в процедурах модуля Unit3 все параметры сделаны параметрами-переменными (**var**-параметрами), а решение о выделении памяти и копировании массивов принимается только в одном случае и только в процедуре MulMatr. Сделано это так. Если в процедуре MulMatr модуля Unit3 выясняется, что адрес выходного фактического массива совпадает с адресом хотя бы одного входного фактического массива, то вызывается процедура MulMatr модуля Unit2 с указанием в качестве выходного параметра нового, объявленного в модуле Unit3, массива, из которого после возврата управления данные копируются в выходной фактический параметр процедуры MulMatr модуля Unit3. Если же в процедуре MulMatr модуля Unit3 выясняется, что адрес выходного фактического массива не совпадает с адресами входных фактических массивов, то процедура MulMatr модуля Unit2 вызывается с набором параметров, который был передан в процедуру MulMatr модуля Unit3.

Проверка задания одного и того же массива в качестве выходного фактического параметра и одного или обоих входных фактических параметров при вызове процедуры MulMatr моду-

ля Unit2 выполняется сравнением их адресов (при проверке на равенство динамических массивов происходит сравнение их адресов):

```
if      //Если адрес выходного параметра
(x=z)   //равен адресу первого
or      //или
(y=z)   //адресу второго входного параметра,
then //значит, выходной параметр совпадает
      //с одним или обоими входными
```

При наличии в разных модулях одинаковых имен, объявленных в их интерфейсных частях, использование их подчиняется следующему правилу: для избежания ошибок перед именем в качестве префикса следует использовать имя модуля, если оно использовано без префикса, то будет отнесено к модулю, указанному в предложении использования последним, где это имя было объявлено. Так, в основной программе последним в предложении использования был модуль Unit3, поэтому вызовы его процедуры MulMatr записаны без префиксов (например, MulMatr(a,b,b)), а вызовы процедур модуля Unit2 — с префиксом (например, Unit2.MulMatr(a,d,c)). Результат работы программы при вводе данных с клавиатуры представлен на рис. 7.3.

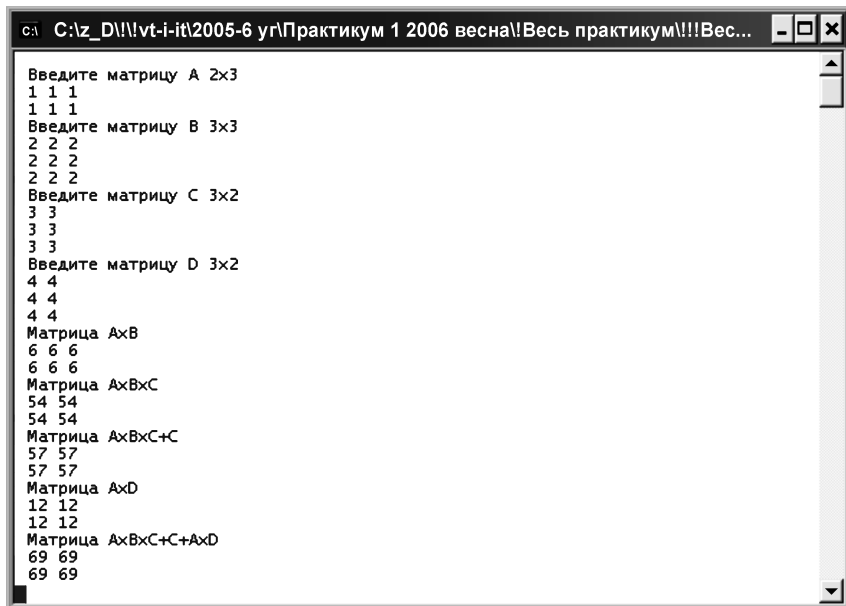


Рис. 7.3

При рассмотрении предметной области заданий на составление модулей авторы учитывали уровень знаний первокурсников, а также тот факт, что это задание не должно существенно превышать по трудоемкости другие. Поэтому первая группа включает задания на матрицы, а вторая связана с решением геометрических задач. При решении задач первой группы студенты могут использовать фрагменты ранее написанных программ, модифицировав их в соответствии с требованиями оформления модулей и придав более универсальный характер. Модуль, содержащий ряд часто используемых подпрограмм, может использоваться в дальнейшем при изучении курса «Информатика», а также при выполнении заданий и по другим дисциплинам.

7.2. Пример выполнения задания

В качестве примера 1 выполнения задания на составление модуля рассмотрим решение следующей задачи. Удалить из матрицы $B(M, N)$ действительных чисел строки, содержащие максимальный и минимальный элементы. Если минимальный и максимальный элементы находятся в одной строке, то никакие строки не удалять. Строки преобразованной матрицы упорядочить методом пузырька по возрастанию сумм цифр целой части последних элементов строк матрицы.

Приступая к составлению модуля, необходимо определить частные задачи, которые позволят решить общую задачу. Решение каждой частной задачи будет оформлено в виде подпрограммы, входящей в состав модуля. Таким образом, будем придерживаться основного принципа разбиения программы на подпрограммы — каждая подпрограмма реализует одну, но законченную функцию.

В соответствии с поставленным заданием можно перечислить основные этапы его выполнения:

1) ввод исходных данных — количества строк и столбцов, а также самих элементов матрицы (подпрограмма ввода исходных данных);

2) вывод исходных данных для последующего контроля правильности решения задачи (подпрограмма вывода матрицы);

3) определение номера строки матрицы, содержащей максимальный элемент, т. е. поиск максимального элемента всей матрицы и номера строки, в которой он расположен (подпрограмма поиска максимального элемента);

4) определение номера строки матрицы, содержащей минимальный элемент, т. е. поиск минимального элемента всей матрицы и номера строки, в которой он расположен (подпрограмма поиска минимального элемента);

5) удаление из матрицы строки с заданным номером;

6) определение цифр целого числа;

7) вычисление суммы элементов одномерного массива;

8) сортировка строк матрицы;

9) вывод полученных результатов:

```
unit ModMatr; //Вариант со статическими массивами
interface
  const mm=20; nn=15; kk=10;
  type
    mas=array[1..nn] of Real;
    matr=array[1..mm] of mas;
    masbyt=array[1..kk] of Byte;
    masint=array[1..mm] of Integer;
  function Rus(s:String):String;
  procedure Input(out m,n:Integer;
                  out b:matr);
  procedure Output(const m,n:Integer;
                   const b:matr);
  function MaxStr(const m,n:Integer;
                  const b:matr):Integer;
  function MinStr(const m,n:Integer;
                  const b:matr):Integer;
  procedure Udalenie(const m,n,nom:Integer;
                     var b:matr);
  procedure Zifra(a:LongInt;
                  out c:masbyt;
                  out kol:Byte);
  procedure Sortirov(const m:Integer;
                     var b:matr;
                     var s:masint);
implementation
  function Rus(s:String):String;
  var
    i:Byte;
  begin
    Result:='';
    for i:=1 to Length(s) do
      case s[i] of
```

```
'A'..'п': Result:=Result+Chr(Ord(s[i])-64);
'p'..'я': Result:=Result+Chr(Ord(s[i])-16);
'Ё': Result:=Result+Chr(240);
'ё': Result:=Result+Chr(241);
    else
        Result:=Result+s[i];
    end;
end;

procedure Input(out m,n:Integer; out b:matr);
var
    i,j:Integer;
begin
    WriteLn(Rus('Введите количество строк '))
        ,Rus('и столбцов матрицы'));
    ReadLn(m,n);
    WriteLn(Rus('Введите элементы матрицы '))
        ,Rus('по строкам '));
    for i:=1 to m do
        begin
            for j:=1 to n do
                Read(b[i,j]);
            ReadLn;
        end;
    end;
end;

procedure Output(const m,n:Integer;
                  const b:matr);
var
    i,j:Integer;
begin
    for i:=1 to m do
        begin
            for j:=1 to n do
                Write(b[i,j]:6:1, ' ');
            WriteLn;
        end;
    end;
end;

function MaxStr(const m,n:Integer;
                 const b:matr):Integer;
var
    i,j:Integer;
```

```
bmax:Real;
begin
  bmax:=b[1,1];
  Result:=1;
  for i:=1 to m do
    for j:=1 to n do
      if b[i,j]>bmax then
        begin
          bmax:=b[i,j];
          Result:=i;
        end;
    end;
  end;

function MinStr(const m,n:Integer;
                 const b:matr):Integer;
var
  i,j:Integer;
  bmin:Real;
begin
  bmin:=b[1,1];
  Result:=1;
  for i:=1 to m do
    for j:=1 to n do
      if b[i,j]<bmin then
        begin
          bmin:=b[i,j];
          Result:=i;
        end;
    end;
  end;

procedure Udalenie(const m,n,nom:Integer;
                   var b:matr);
var
  i:Integer;
begin
  for i:=nom to m-1 do
    b[i]:=b[i+1];
  end;

procedure Zifra(a:LongInt;
                out c:masbyt;
                out kol:Byte);
var k:Byte;
```

```
begin
  kol:=0;
  repeat
    k:=a mod 10;
    kol:=kol+1;
    c[kol]:=k;
    a:=a div 10;
  until a=0;
end;

procedure Sortirov(const m:Integer;
                   var b:matr;
                   var s:masint);

var
  i,k,sp:Integer;
  pr:Boolean;
  f:mas;
begin
  k:=0;
  repeat
    k:=k+1;
    pr:=True;
    for i:=1 to m-k do
      if s[i]>s[i+1] then
        begin
          pr:=False;
          sp:=s[i];
          s[i]:=s[i+1];
          s[i+1]:=sp;
          f:=b[i];
          b[i]:=b[i+1];
          b[i+1]:=f;
        end;
    until pr;
  end;
end.

program Modul1;
//Основная программа, использующая модуль ModMatr
{$APPTYPE CONSOLE}
uses
  SysUtils,ModMatr;
```

```
var
  b:matr;
  m,n,imax,imin,i,j:Integer;
  kol:Byte;
  c:masbyt;
  s:masint;
begin
  Input(m,n,b);
  Output(m,n,b);
  imax:=MaxStr(m,n,b);
  imin:=MinStr(m,n,b);
  WriteLn('imax=',imax:3,' imin=',imin:3);
  if imax>imin then
    begin
      Udalenie(m,n,imax,b);
      Udalenie(m,n,imin,b);
      m:=m-2;
    end
  else if imin>imax then
    begin
      Udalenie(m,n,imin,b);
      Udalenie(m,n,imax,b);
      m:=m-2;
    end;
  Output(m,n,b);
  for j:=1 to m do
    begin
      Zifra(Trunc(Abs(b[j,n])), c, kol);
      s[j]:=0;
      for i:=1 to kol do
        begin
          Write(c[i]:2);
          s[j]:=s[j]+c[i];
        end;
      WriteLn(s[j]:5);
    end;
  Sortirov(m,b,s);
  Output(m,n,b);
  for i:=1 to m do
    Write(s[i]:5);
  ReadLn;
end.
```

```
//Вариант модуля с динамическими массивами
unit ModMatrDinMas;
interface
    type
        mas=array of Real;
        matr=array of mas;
        masbyt=array of Byte;
        masint=array of Integer;
    function Rus(s:String):String;
    procedure Input(out m,n:Integer;
                   out b:matr);
    procedure Output(const m,n:Integer;
                    const b:matr);
    function MaxStr(const m,n:Integer;
                  const b:matr):Integer;
    function MinStr(const m,n:Integer;
                  const b:matr):Integer;
    procedure Udalenie(const m,nom:Integer;
                      var b:matr);
    procedure Zifra(a:LongInt;
                   out c:masbyt;
                   out kol:Byte);
    procedure Sortirov(const m:Integer;
                     var b:matr;
                     var s:masint);
implementation
    function Rus(s:String):String;
    var
        i:Byte;
    begin
        Result:='';
        for i:=1 to Length(s) do
            case s[i] of
                'A'..'П': Result:=Result+Chr(Ord(s[i])-64);
                'p'..'я': Result:=Result+Chr(Ord(s[i])-16);
                'Ё': Result:=Result+Chr(240);
                'ё': Result:=Result+Chr(241);
                else
                    Result:=Result+s[i];
            end;
        end;
    end;
//Процедура ввода исходных данных:
//количества строк и столбцов матрицы,
```

```
//и элементов матрицы
procedure Input(out m,n:Integer; out b:matr);
var
  i,j:Integer;
begin
  WriteLn(Rus('Введите количество строк')
    ,Rus(' и столбцов матрицы'));
  ReadLn(m,n);
  WriteLn(Rus('Введите матрицу по строкам'));
  SetLength(b,m,n);
  for i:=0 to m-1 do
    begin
      for j:=0 to n-1 do
        Read(b[i,j]);
      ReadLn;
    end;
  end;
  //Процедура вывода исходных данных - матрицы,
  procedure Output(const m,n:Integer;
    const b:matr);
  var
    i,j:Integer;
  begin
    for i:=0 to m-1 do
      begin
        for j:=0 to n-1 do
          Write(b[i,j]:6:1, ' ');
        WriteLn;
      end;
    end;
  //Функция определения строки, содержащей
  //максимальный элемент матрицы
  function MaxStr(const m,n:Integer;
    const b:matr):Integer;
  var
    i,j:Integer;
    bmax:Real;
  begin
    bmax:=b[0,0];
    Result:=0;
    for i:=0 to m-1 do
      for j:=0 to n-1 do
        if b[i,j]>bmax then
```

```
begin
    bmax:=b[i,j];
    Result:=i;
end;
end;
//Функция определения строки,
//содержащей минимальный элемент матрицы
function MinStr(const m,n:Integer;
                const b:matr):Integer;
var
    i,j:Integer;
    bmin:Real;
begin
    bmin:=b[0,0];
    Result:=0;
    for i:=0 to m-1 do
        for j:=0 to n-1 do
            if b[i,j]<bmin then
                begin
                    bmin:=b[i,j];
                    Result:=i;
                end;
            end;
        end;
    end;
//Функция удаления строки матрицы
//с заданным номером
procedure Udalenie(const m,nom:Integer;
                   var b:matr);
var
    i:Integer;
begin
    for i:=nom to m-2 do
        b[i]:=b[i+1];
    end;
//Процедура определения цифр целого числа
//и формирования массива цифр
procedure Zifra(a:LongInt;
               out c:masbyt;
               out kol:Byte);
var
    k:Byte;
begin
    kol:=0;
    repeat
```



```
k:=a mod 10;
kol:=kol+1;
SetLength(c,kol);
c[kol-1]:=k;
a:=a div 10;
until a=0;
end;
//Процедура сортировки строк матрицы
//по возрастанию суммы цифр целой части
//последнего элемента строки
procedure Sortirov(const m:Integer;
                    var b:matr;
                    var s:masint);
var
    i,k,sp:Integer;
    pr:Boolean;
    f:mas;
begin
    k:=0;
    repeat
        k:=k+1;
        pr:=True;
        for i:=0 to m-k-1 do
            if s[i]>s[i+1] then
                begin
                    pr:=False;
                    sp:=s[i];
                    s[i]:=s[i+1];
                    s[i+1]:=sp;
                    f:=b[i];
                    b[i]:=b[i+1];
                    b[i+1]:=f;
                end;
        until pr;
    end;
end.

program Modul1Din;
{$APPTYPE CONSOLE}
uses
    SysUtils,ModMatrdinmas;
var
    b:matr;
```

```
m,n,imax,imin,i,j:Integer;
kol:Byte;
c:masbyt;
s:masint;
begin
  Input(m,n,b);
  WriteLn(Rus('Исходная матрица'));
  Output(m,n,b);
  imax:=MaxStr(m,n,b);
  imin:=MinStr(m,n,b);
  WriteLn('imax=',imax+1:3,' imin=',imin+1:3);
  if imax>imin then
    begin
      Udalenie(m,imax,b);
      Udalenie(m,imin,b);
      m:=m-2;
    end
  else if imin>imax then
    begin
      Udalenie(m,imin,b);
      Udalenie(m,imax,b);
      m:=m-2;
    end;
  WriteLn(Rus('Полученная матрица'));
  Output(m,n,b);
  SetLength(s,m);
  for j:=0 to m-1 do
    begin
      Zifra(Trunc(Abs(b[j,n-1])), c, kol);
      s[j]:=0;
      for i:=0 to kol-1 do
        s[j]:=s[j]+c[i];
      WriteLn(Rus('Сумма цифр последнего '
        ,Rus('элемента строки: '),s[j]:5);
    end;
    Sortirov(m,b,s);
    WriteLn(
      Rus('Полученная отсортированная матрица '));
    Output(m,n,b);
    for i:=0 to m-1 do
      Write(s[i]:5);
    ReadLn;
  end.
```

7.3. Задания для самостоятельной работы

В заданиях требуется составить модуль, содержащий ряд подпрограмм. Разбиение всей программы на подпрограммы должно выполняться по функциональному принципу: каждая подпрограмма должна реализовывать одну, но законченную функцию. Следует предусмотреть подпрограммы: 1) ввода исходных данных (параметров, задающих фактическую размерность массивов и самих элементов массивов) или формирования элементов массивов по заданному в условии закону (где требуется сформировать массив); 2) вывода исходных данных (массивов); 3) две-три подпрограммы, реализующие алгоритм решения поставленной задачи; 4) вывода полученных результатов.

1. Сформировать матрицу $Y(N, N)$ по формуле $Y_{ij} = A_i B_j$, где A , B — массивы по N элементов каждый. Вывести полученную матрицу в виде матрицы. Вычислить $Z = \sqrt[3]{e^{SR} + \sqrt{|Y_{\max} Y_{\min}|} + 2^{SP}}$, где SR — сумма элементов той строки матрицы, среднее арифметическое элементов которой имеет наибольшее значение, Y_{\max} , Y_{\min} — максимальный и минимальный элементы матрицы, SP — произведение элементов той строки матрицы, среднее геометрическое модулей элементов которой имеет наибольшее значение. (Предусмотреть подпрограммы нахождения суммы элементов строки матрицы, поиска строки с наибольшим значением среднего арифметического, поиска минимального, максимального элементов, строки с наибольшим средним геометрическим элементов, произведения элементов строки.)

2. Сформировать матрицу $W(M, N)$, элементы которой вычисляются по формуле $W_{ij} = \cos d_i \sqrt[3]{X_j}$, где d_i — элементы массива D , а X_j изменяются как простая переменная от значения $X_{\text{нач}}$ с шагом ΔX . Вывести полученную матрицу в виде матрицы. Сформировать вектор B , записав в его начало столбец матрицы W с минимальным произведением элементов, а затем столбец с максимальной суммой элементов. (Предусмотреть подпрограммы: 1) нахождения столбца с минимальным произведением, 2) нахождения столбца с максимальной суммой, 3) формирования вектора.)

3. В матрице $V(L, L)$ поменять местами первый и последний столбцы, второй и предпоследний и т. д. В преобразованной матрице определить среднее арифметическое отрицательных элементов, лежащих ниже побочной диагонали, и среднее геометрическое

кое положительных элементов, лежащих выше. Если требуемые элементы отсутствуют, то выдать соответствующее сообщение. (Предусмотреть подпрограммы: 1) переформирования матрицы, 2) нахождения среднего арифметического, 3) нахождения среднего геометрического.)

4. Вычислить сумму ряда $S = \sum_{i=1}^{\infty} \frac{ax^i}{i!}$ с точностью ε . Опреде-

лить в матрице $B(L, M)$ все элементы, превосходящие значение полученной суммы, и разместить найденные элементы в одномерном массиве C (просмотр матрицы вести по строкам). Отсортировать элементы массива C по возрастанию. (Предусмотреть подпрограммы: 1) вычисления суммы ряда, 2) формирования одномерного массива, 3) сортировки элементов массива.)

5. В матрице $A(M, N)$ определить в столбцах, полностью состоящих из положительных элементов, максимальные значения, а в столбцах, состоящих полностью из отрицательных элементов, — минимальные. Для столбцов, содержащих и отрицательные, и положительные элементы, вычислить средние арифметические значения всех элементов. Найденные значения сохранить в одномерном массиве F , элементы которого упорядочить по убыванию. (Предусмотреть подпрограммы: 1) определения типа одномерного массива — состоит целиком из положительных, целиком из отрицательных элементов или содержит оба вида элементов, 2) нахождения среднего арифметического элементов одномерного массива, 3) нахождения максимального элемента одномерного массива, 4) нахождения минимального элемента одномерного массива, 5) сортировки элементов одномерного массива.)

6. В целочисленной матрице $C(M, K)$ удалить строки, целиком состоящие из нулей. Найти в каждой строке матрицы наибольший элемент, нацело делящийся на заданную величину L . Заменить найденный элемент произведением элементов, расположенных после него. Если элемент является последним в строке, то заменить его суммой предшествующих элементов. (Предусмотреть подпрограммы: 1) определения типа одномерного массива — состоит целиком из нулевых элементов или содержит ненулевые, 2) удаления из матрицы строк с заданным номером, 3) нахождения наибольшего элемента одномерного массива, нацело делящегося на заданное число, 4) нахождения произведения элементов одномерного массива, расположенных после заданного элемента, 5) нахождения суммы элементов одномерного массива, расположенных до заданного элемента.)

7. В матрице $D(N, N)$ найти все простые числа, лежащие выше главной диагонали, и запомнить их в одномерном массиве P . В полученном массиве повторяющиеся элементы оставить в одном экземпляре, удалив их дубликаты. (Предусмотреть подпрограммы: 1) определения типа числа — простое или нет, 2) формирования массива простых чисел, 3) удаления повторяющихся элементов в массиве.)

8. В матрице $T(M, N)$ найти все строки, в которых минимальный элемент строки предшествует максимальному. Заменить минимальные элементы суммой предшествующих, а максимальные — произведением последующих. Если минимальный элемент первый в строке, а максимальный — последний, то заменить их нулями. (Предусмотреть подпрограммы: 1) определения типа строки матрицы по порядку расположения минимального и максимального элементов, 2) вычисления суммы элементов массива, 3) вычисления произведения элементов массива.)

9. В матрице $H(L, M)$ найти среднее арифметическое X всех ее элементов. Далее вычислить

$$S = \begin{cases} \sum_{k=1}^L \prod_{i=1}^k \frac{k}{ix}, & \text{если } X > 1, \\ \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)(2n+1)!}, & \text{если } -1 \leq X \leq 1, \\ \sum_{i=1}^L \sum_{j=1}^M H_{ij}, & \text{если } H_{ij} < 0, X < -1. \end{cases}$$

Сумму ряда вычислить с точностью ϵ . (Предусмотреть подпрограммы: 1) определения среднего арифметического всех элементов матрицы, 2) вычисления суммы отрицательных элементов матрицы, 3) нахождения суммы ряда, 4) нахождения суммы произведений.)

10. Вычислить в каждой строке матрицы $Y(M, N)$ максимальный элемент. Далее сформировать одномерный массив Z по следующему правилу: если максимальный элемент i -й строки матрицы отрицательный, то соответствующий элемент массива Z равен сумме отрицательных элементов строки матрицы, если же максимальный элемент положительный, то соответствующий элемент массива Z равен произведению положительных элементов строки матрицы, если максимальный элемент равен нулю, то и элемент массива Z равен нулю. (Предусмотреть подпрограммы 1) определения максимального элемента одномерного массива,

2) нахождения суммы отрицательных элементов массива, 3) нахождения произведения положительных элементов массива.)

11. Определить в матрице $F(K, L)$ минимальный F_{\min} и максимальный F_{\max} элементы. Вычислить $\int_{F_{\min}}^{F_{\max}} X^2 \log_3(X^4 + 2) dX$ с точностью ϵ методом парабол. (Предусмотреть подпрограммы: 1) определения максимального элемента матрицы, 2) определения минимального элемента матрицы, 3) вычисления интеграла методом парабол.)

12. Вычеркнуть в матрице $R(L, M)$ строку, содержащую наибольшее количество нулевых элементов. Определить в каждой строке преобразованной матрицы количество элементов, попадающих в δ -окрестность среднего арифметического элементов этой же строки. Сохранить найденные количества в массиве. (Предусмотреть подпрограммы: 1) определения количества нулевых элементов одномерного массива, 2) определения среднего арифметического элементов одномерного массива, 3) определения количества элементов одномерного массива, попадающих в δ -окрестность среднего арифметического элементов этого массива, 4) удаления из матрицы строки с заданным номером.)

13. Из матрицы $A(L, L)$ получить матрицу B путем вычеркивания элементов побочной диагонали. Умножить матрицу A на матрицу B и сохранить результат в матрице C . В полученной матрице сравнить количество отрицательных и положительных элементов. (Предусмотреть подпрограммы: 1) получения матрицы с удаленными элементами побочной диагонали, 2) умножения двух матриц, 3) определения количества отрицательных элементов матрицы, 4) определения количества положительных элементов матрицы.)

14. Из двух одномерных массивов $P(M)$ и $Q(N)$, где все элементы положительные, сформировать матрицу $R(M, N)$, элементы которой вычисляются по правилу $R_{ij} = P_i \cdot Q_j$. В каждой строке полученной матрицы определить элементы, лежащие в интервале от среднего геометрического до среднего арифметического элементов той же строки, и запомнить их в массиве T , который упорядочить по убыванию методом пузырька. (Предусмотреть подпрограммы: 1) формирования матрицы на основе двух одномерных массивов, 2) вычисления среднего арифметического элементов одномерного массива, 3) вычисления среднего геометрического элементов одномерного массива, 4) определения элементов

одномерного массива, попадающих в заданный интервал, 5) сортировки элементов массива методом пузырька.)

15. В матрице $D(L, M)$ в столбцах, содержащих только положительные элементы, вычислить средние геометрические значения, а в столбцах, содержащих неположительные элементы, вычислить средние арифметические значения элементов каждого столбца. Вычисленные средние сохранить в массиве V . В полученном массиве подсчитать количество положительных, нулевых и отрицательных элементов. (Предусмотреть подпрограммы: 1) определения наличия неположительных элементов в одномерном массиве, 2) вычисления среднего арифметического элементов одномерного массива, 3) вычисления среднего геометрического элементов одномерного массива, 4) определения количества положительных элементов одномерного массива, 5) определения количества нулевых элементов одномерного массива, 6) определения количества отрицательных элементов одномерного массива.)

16. В каждой строке матрицы $T(M, N)$ найти первый положительный элемент и сохранить в массиве A (если в строке нет положительных элементов, то соответствующий элемент массива A равен нулю). Рассматривая элементы полученного массива как коэффициенты многочлена $(M - 1)$ -го порядка, вычислить значение многочлена при аргументе, равном максимальному элементу матрицы. (Предусмотреть подпрограммы: 1) нахождения первого положительного элемента в одномерном массиве, 2) нахождения максимального элемента матрицы, 3) вычисления значения многочлена.)

17. Вычислить выражение $OTN = \frac{nS}{(A_{\min} + A_{\max}) \frac{i_{\min} j_{\min}}{i_{\max} j_{\max}}}$, где

S — сумма ряда $\sum_{k=1}^{\infty} \frac{\log_5^k X}{(k-1)!}$, вычисленная с точностью ε , n — количество просуммированных слагаемых, A_{\min}, A_{\max} — минимальный и максимальный элементы матрицы $A(L, M)$, а $i_{\min}, j_{\min}, i_{\max}, j_{\max}$ — индексы этих элементов. (Предусмотреть подпрограммы: 1) вычисления суммы ряда, 2) нахождения максимального элемента матрицы и его индексов, 3) нахождения минимального элемента матрицы и его индексов.)

18. Сформировать матрицу $W(L+2, L)$ по формуле $W_{ij} = \log_2 |A_i| \sqrt[3]{e^{B_j}}$, где A_i — элементы массива $A(L+2)$, а B_j изменя-

ются от 0 с шагом ΔB . В полученной матрице удалить строки, содержащие минимальный и максимальный элементы. В преобразованной матрице вычислить средние арифметические значения элементов, расположенных выше и ниже главной диагонали, и их отношение. (Предусмотреть подпрограммы: 1) вычисления значений элементов матрицы по заданной формуле, 2) нахождения номера строки матрицы, содержащей максимальный элемент матрицы, 3) нахождения номера строки матрицы, содержащей минимальный элемент матрицы, 4) удаления из матрицы строки с заданным номером.)

19. В матрицу $A(M-1, M)$ после строки, содержащей максимальный элемент всей матрицы, в качестве строки вставить одномерный массив $B(M)$. Вычислить A^M и найти в полученной матрице наименьший и наибольший элементы. (Предусмотреть подпрограммы: 1) нахождения номера строки матрицы, содержащей максимальный элемент матрицы, 2) добавления в матрицу новой строки после строки с заданным номером, 3) умножения двух матриц, 4) нахождения в матрице наибольшего элемента, 5) нахождения в матрице наименьшего элемента.)

20. В матрице $A(L, M)$ найти минимальный A_{\min} и максимальный A_{\max} элементы и их индексы $i_{\min}, j_{\min}, i_{\max}, j_{\max}$. Далее вычислить

$$S = \begin{cases} X \left(A_{\min} \frac{i_{\min}}{j_{\min}} + A_{\max} \frac{i_{\max}}{j_{\max}} \right), & \text{если } X \geq 1, \\ \sum_{n=1}^{\infty} \frac{nx}{(x+1)^n}, & \text{если } -1 < X < 1, \\ \sum_{i=1}^L \sum_{j=1}^M H_{ij}, & \text{если } H_{ij} < 0, X \leq -1. \end{cases}$$

Сумму ряда вычислить с точностью ϵ . (Предусмотреть подпрограммы: 1) вычисления суммы ряда, 2) нахождения максимального элемента матрицы и его индексов, 3) нахождения минимального элемента матрицы и его индексов, 4) вычисления заданного произведения сумм.)

21. В матрице $M(L, N)$ определить в каждой строке количество элементов, нацело делящихся на заданное число K , а сами элементы переписать в массив P . Упорядочить строки матрицы по возрастанию найденных количеств методом нахождения наибольшего. Подсчитать в массиве P количество двузначных чи-

сел. (Предусмотреть подпрограммы: 1) определения количества элементов одномерного массива, нацело делящихся на заданное число, и переписи найденных элементов в одномерный массив, начиная с заданного номера, 2) упорядочения строк матрицы, 3) определения количества двузначных чисел в массиве.)

22. В матрице $P(K, L)$ определить все простые числа и переписать их в массив Q . Удалить из массива Q копии элементов и упорядочить преобразованный массив по возрастанию методом пузырька. (Предусмотреть подпрограммы: 1) определения типа числа — простое или нет, 2) удаления копий элементов в одномерном массиве, 3) упорядочения элементов массива методом пузырька.)

23. В матрице $D(M, N)$ найти максимальный элемент D_{\max} той строки матрицы, сумма элементов которой минимальна, и минимальный элемент D_{\min} той строки, сумма элементов кото-

рой максимальна. Вычислить затем
$$\text{INT} = \int_{D_{\min}}^{D_{\max}} \sqrt{x} \ln(x^2 + 1) dx$$

методом трапеций с точностью ϵ . (Предусмотреть подпрограммы: 1) вычисления суммы элементов одномерного массива, 2) определения минимального элемента одномерного массива, 3) определения максимального элемента одномерного массива, 4) вычисления интеграла с точностью методом трапеций.)

24. В матрице $V(L, M)$ найти сумму элементов каждой строки. Упорядочить строки матрицы по возрастанию второй цифры целой части найденных сумм методом нахождения минимального. (Предусмотреть подпрограммы: 1) вычисления суммы элементов одномерного массива, 2) определения заданной цифры целого числа, 3) упорядочения строк матрицы по возрастанию найденных цифр.)

25. В каждой строке матрицы $R(L, M)$ найти элемент, для которого модуль разности и среднего арифметического его соседей имеет минимальное значение. Для первого элемента строки соседом слева считать последний элемент, а для последнего элемента соседом справа считать первый элемент. Удалить из матрицы строки, для которых найденные значения модуля имеют минимальное и максимальное значения. (Предусмотреть подпрограммы: 1) определения элемента одномерного массива, модуль разности которого и среднего арифметического его соседей минимален, 2) определения индекса минимального элемента одномерного массива, 3) оп-

ределения индекса максимального элемента одномерного массива, 4) удаления из матрицы строки с заданным номером.)

7.4. Пример выполнения задания

В качестве примера 2 рассмотрим решение задачи с геометрическим содержанием, предусматривающей составление модуля. В массивах $X1(N)$, $Y1(N)$ хранятся координаты точек на плоскости, на которых строятся окружности первого множества. В массивах $X2(M)$, $Y2(M)$ хранятся координаты точек на плоскости, на которых строятся окружности второго множества. Найти две такие окружности (из разных множеств), для которых разность площадей четырехугольников, образованных центрами окружностей, точками касания общих внутренних касательных и точкой пересечения этих касательных, максимальна. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество). Проверить также возможность построения окружностей на каждом множестве точек.

Выделим основные частные задачи, которые необходимо решить в ходе выполнения общей задачи:

- 1) ввод исходных данных;
- 2) проверку возможности построения окружности на трех заданных точках;
- 3) вычисление координат центра и радиуса окружности на основе координат трех точек, ей принадлежащих;
- 4) проверку возможности построения внутренней касательной к паре окружностей;
- 5) вычисление длин сторон четырехугольника;
- 6) вычисление площади четырехугольника.

Окружность можно провести через любые три точки, не лежащие на одной прямой. Проверка возможности построения окружности на очередной тройке точек будет сводиться к проверке принадлежности этих точек одной прямой. Если три точки лежат на одной прямой, то наблюдается пропорциональность их координат, т. е. выполняется равенство

$$\frac{X2 - X1}{X3 - X2} = \frac{Y2 - Y1}{Y3 - Y2},$$

где $(X1, Y1)$, $(X2, Y2)$, $(X3, Y3)$ — координаты рассматриваемых точек или $(X2 - X1)(Y3 - Y2) = (X3 - X2)(Y2 - Y1)$.

Координаты центра окружности и радиус можно определить, решив систему трех уравнений с тремя неизвестными:

$$(XC - X1)^2 + (YC - Y1)^2 = R^2,$$

$$(XC - X2)^2 + (YC - Y2)^2 = R^2,$$

$$(XC - X3)^2 + (YC - Y3)^2 = R^2.$$

Приравняем левые части первых двух уравнений системы, откуда получим

$$2XC(X2 - X1) = 2YC(Y1 - Y2) + Y2^2 - Y1^2 + X2^2 - X1^2.$$

Аналогично приравняв, например, левые части первого и третьего уравнений, получим

$$2XC(X3 - X1) = 2YC(Y1 - Y3) + Y3^2 - Y1^2 + X3^2 - X1^2.$$

Обозначив

$$A = 2(X2 - X1), \quad B = 2(Y1 - Y2), \quad C = Y2^2 - Y1^2 + X2^2 - X1^2,$$

$$D = 2(X3 - X1), \quad E = 2(Y1 - Y3), \quad F = Y3^2 - Y1^2 + X3^2 - X1^2,$$

запишем полученную систему двух уравнений с двумя неизвестными в следующем виде:

$$A \cdot XC = B \cdot YC + C,$$

$$D \cdot XC = E \cdot YC + F.$$

Решив эту систему, получим

$$XC = \frac{BF - CE}{BD - AE}, \quad YC = \frac{AF - CD}{BD - AE}.$$

Зная координаты центра окружности, ее радиус можно определить из любого уравнения исходной системы, подставив в него полученные значения координат центра.

Внутренние касательные к паре окружностей можно построить, если расстояние между их центрами превышает сумму их радиусов, т. е. $L > R1 + R2$, где $XC1, YC1, R1$ — координаты центра и радиус первой окружности, $XC2, YC2, R2$ — координаты центра

и радиус второй окружности, $L = \sqrt{(XC1 - XC2)^2 + (YC1 - YC2)^2}$ — расстояние между центрами окружностей.

Для нахождения площади четырехугольника, образованного точками касания касательных к окружности, ее центром и точкой пересечения касательных, достаточно вычислить площадь треугольника, образованного точкой касания одной из касательных, центром окружности и точкой пересечения касательных. Площадь треугольника будет определяться половиной произведения катетов. Рассматриваемый треугольник прямоугольный, так как касательная перпендикулярна радиусу окружности, проведенному в точку ка-

сания. Один из катетов является радиусом окружности, другой — отрезок касательной, заключенный между точкой касания и точкой пересечения касательных.

Для вычисления второго катета b следует сначала найти длину общей внутренней касательной согласно следующему выражению: $l_{\text{кас}} = \sqrt{l^2 - (R1 + R2)^2}$, а затем, воспользовавшись подобием прямоугольных треугольников, из пропорциональности сторон

$\frac{b}{l_{\text{кас}}} = \frac{R1}{R1 + R2}$ получить длину катета $b = \frac{l_{\text{кас}} R1}{R1 + R2}$. У второго

треугольника, образованного центром второй окружности, точкой касания внутренней касательной и точкой пересечения касательных, катеты будут равны $R2$ и $l_{\text{кас}} - b$.

Текст модуля приведен ниже:

```
unit ModulGeom;
interface
  type
    mas=array of Real;
  function Rus(s:String):String;
  procedure Input(out n:Integer; out x,y:mas);
  procedure Output(const n:Integer;const x,y:mas);
  function WozmPostr(const x1,y1,x2,y2
                    ,x3,y3:Real):Boolean;
  procedure ParamOkr(const x1,y1,x2,y2,x3,y3:Real;
                    out xc,yc,r:Real);
  function Dlina(const x1,y1,x2,y2:Real):Real;
  function RasnPl(const x1,y1,r1
                ,x2,y2,r2:Real):Real;
implementation
  function Rus(s:String):String;
  var i:Byte;
  begin
    Result:='';
    for i:=1 to Length(s) do
      case s[i] of
        'A'..'n': Result:=Result+Chr(Ord(s[i])-64);
        'p'..'я': Result:=Result+Chr(Ord(s[i])-16);
        'Ё': Result:=Result+Chr(240);
        'ё': Result:=Result+Chr(241);
        else
          Result:=Result+s[i];
        end;
    end;
end;
```

```
//Процедура ввода данных
procedure Input(out n:Integer; out x,y:mas);
var i:Integer;
begin
  WriteLn(Rus('Введите количество точек'));
  ReadLn(n);
  if n<0 then
    WriteLn(Rus('Количество точек '
                ,Rus('не может быть отрицательным')
  else if n<3 then
    WriteLn(Rus('Количество точек недостаточно'))
  else
    begin
      WriteLn(Rus('Введите координаты точек x,y'));
      SetLength(x,n);
      SetLength(y,n);
      for i:=0 to n-1 do
        Read(x[i],y[i]);
      ReadLn;
    end;
  end;
//Процедура вывода исходных данных
procedure Output(const n:Integer;const x,y:mas);
var i:Integer;
begin
  for i:=0 to n-1 do
    Write(x[i]:6:1,' ',y[i]:6:1);
  WriteLn;
end;
//Функция проверки
//возможности построения окружности
function WozmPostr(const x1,y1,x2
                    ,y2,x3,y3:Real):Boolean;
begin
  if (x2-x1)*(y3-y2)=(x3-x2)*(y2-y1) then
    Result:=False
  else
    Result:=True;
end;
//Процедура определения параметров окружности
//по координатам трех точек
procedure ParamOkr(const x1,y1,x2,y2,x3,y3:Real;
                    out xc,yc,r:Real);
```

```

var a,b,c,d,e,f:Real;
begin
  a:=2*(x2-x1); b:=2*(y1-y2);
  c:=y2*y2-y1*y1+x2*x2-x1*x1;
  d:=2*(x3-x1); e:=2*(y1-y3);
  f:=y3*y3-y1*y1+x3*x3-x1*x1;
  xc:=(b*f-c*e)/(b*d-a*e); //координаты центра
  yc:=(a*f-c*d)/(b*d-a*e); //окружности
  r:=dlina(xc,yc,x1,y1); //радиус окружности
end;
//Функция вычисления
//расстояния между двумя точками
function Dlina(const x1,y1,x2,y2:Real):Real;
begin
  Result:=Sqrt(Sqr(x2-x1)+Sqr(y2-y1));
end;
//Функция вычисления
//разности площадей четырехугольников
function RasnPl(const x1,y1,r1
                ,x2,y2,r2:Real):Real;
var lc,rs,lk,lk1,lk2,s1,s2:Real;
begin
  //Расстояние между центрами окружностей
  lc:=Dlina(x1,y1,x2,y2);
  //сумма радиусов окружностей
  rs:=r1+r2;
  //Проверка существования внутренней касательной
  if lc>rs then
  begin
    //Длина внутренней касательной
    lk:=Sqrt(Sqr(lc)-Sqr(rs));
    //Катет первого треугольника
    lk1:=lk*r1/rs;
    //Катет второго треугольника
    lk2:=lk-lk1;
    //Площадь первого четырехугольника
    s2:=r2*lk2;
    //Площадь второго четырехугольника
    Result:=Abs(s1-s2);
  end
  else
    Result:=-1;
  end;
end.

```

В основной программе с помощью подпрограмм рассмотренного модуля осуществляется ввод исходных данных: при количестве точек в множестве, меньшем трех, программа завершает выполнение. При достаточном количестве точек в обоих множествах поставленная задача решается методом полного перебора. При этом берутся три очередные точки первого множества, проверяется возможность построения окружности на них (не должны лежать на одной прямой). Если окружность существует, то рассматривают очередные три точки второго множества. Если существует окружность, построенная на очередных точках второго множества, то вычисляют параметры окружностей и разность площадей четырехугольников. Если внутренняя касательная не существует, то функция вычисления разности площадей возвращает -1 . Поскольку вычисляется модуль разности площадей, то при поиске максимальной разности подобный вариант взаимного расположения окружностей на результат не влияет.

Для исключения повторного рассмотрения вариантов точек номер второй должен превышать номер первой точки, а номер третьей должен превышать номер второй точки. Поэтому индекс внешнего цикла должен меняться от номера первой точки до номера третьей от конца точки, индекс второго цикла должен меняться от значения, на единицу большего индекса первого цикла, до номера предпоследней точки, а индекс внутреннего цикла — от значения, на единицу большего значения индекса второго цикла, до номера последней точки.

Текст основной программы выглядит следующим образом:

```
program Modul2;
{$APPTYPE CONSOLE}
uses
  SysUtils, ModulGeom;
var
  x1,y1,x2,y2:mas;
  n1,n2,i1,i2,i3,j1,j2,j3:Integer;
  i1m,i2m,i3m,j1m,j2m,j3m,kol:Integer;
  xc1,yc1,r1,xc2,yc2,r2,rasn,maxrasn:Real;
begin
  Input(n1,x1,y1);
  kol:=0;
  maxrasn:=-1;
  if n1>=3 then
    begin
```

```

WriteLn(
    Rus('Координаты точек первого множества'));
Output(n1,x1,y1);
ReadLn;
Input(n2,x2,y2);
if n2>=3 then
begin
    WriteLn(
        Rus('Координаты точек второго множества));
    Output(n2,x2,y2);
    ReadLn;
    for i1:=0 to n1-3 do
        for i2:=i1+1 to n1-2 do
            for i3:=i2+1 to n1-1 do
//Возможность построения окружности на
//очередных трех точках первого множества
                if WozmPostr(x1[i1],y1[i1]
                    ,x1[i2],y1[i2]
                    ,x1[i3],y1[i3]) then
                    begin
                        for j1:=0 to n2-3 do
                            for j2:=j1+1 to n2-2 do
                                for j3:=j2+1 to n2-1 do
//Возможность построения окружности на
//очередных трех точках второго множества
                                    if WozmPostr(x2[j1],y2[j1]
                                        ,x2[j2],y2[j2]
                                        ,x2[j3],y2[j3]) then
                                        begin
//Определение параметров окружности,
//построенной на трех точках первого множества
                                            ParamOkr(x1[i1],y1[i1],x1[i2]
                                                ,y1[i2],x1[i3],y1[i3]
                                                ,xc1,yc1,r1);
//Определение параметров окружности,
//построенной на трех точках второго множества
                                            ParamOkr(x2[j1],y2[j1],x2[j2]
                                                ,y2[j2],x2[j3],y2[j3]
                                                ,xc2,yc2,r2);
                                            rasn:=RasnPl(xc1,yc1,r1
                                                ,xc2,yc2,r2);
//Определение разности площадей
//четыреугольников, построенных на паре

```



```
//окружностей двух множеств
        if rasn>maxrasn then
            begin
//Количество рассмотренных пар окружностей
//с общей касательной
                kol:=kol+1;
                maxrasn:=rasn;
//Номера точек первого множества
                i1m:=i1;
                i2m:=i2;
                i3m:=i3;
//Номера точек второго множества
                j1m:=j1;
                j2m:=j2;
                j3m:=j3;
            end;
        end;
end;
if kol>0 then
    WriteLn(Rus('Максимальная разность =')
        ,maxrasn:8:2
        ,Rus(' номера точек первого множества ')
        ,i1m:3,i2m:3,i3m:3
        ,Rus(' номера точек второго множества')
        ,j1m:3,j2m:3,j3m:3)
else
    WriteLn(Rus('Нет ни одной пары окружностей'));
ReadLn;
end;
end.
```

7.5. Задания для самостоятельной работы

Основные расчетные формулы для решения предлагаемых задач приведены в приложении 3.

1. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины, найти треугольник с максимальной медианой. Вывести номера точек результирующего треугольника, координаты его вершин и медианы, ее длину. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить кор-

ректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

2. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины, найти треугольник с максимальной высотой. Вывести номера точек результирующего треугольника, координаты его вершин, координаты высоты и ее длину. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

3. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины, найти треугольник с максимальной биссектрисой. Вывести номера точек результирующего треугольника, координаты его вершин, координаты биссектрисы и ее длину. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

4. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины треугольников, найти треугольник с максимальным углом между медианой и высотой. Вывести номера точек результирующего треугольника, координаты его вершин, высоты, медианы, значение угла. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

5. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины треугольников, найти треугольник с максимальным углом между медианой и биссектрисой. Вывести номера точек результирующего треугольника, координаты его вершин, биссектрисы, медианы, значение угла. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

6. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины треугольников, найти треугольник с максимальным углом между высотой и биссектрисой. Вывести номера точек результирующего треугольника, координаты его вершин, биссектрисы, высоты, значение угла. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

7. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины треугольников, найти треугольник с максимальной площадью вписанного круга. Вывести номера точек результирующего треугольника, координаты его вершин, центра круга, его радиус и площадь. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

8. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины треугольников, найти треугольник с максимальной площадью описанного круга. Вывести номера точек результирующего треугольника, координаты его вершин, центра круга, его радиус и площадь. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

9. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины треугольников, найти треугольник с максимальной разностью площадей описанного и вписанного кругов. Вывести номера точек результирующего треугольника, координаты его вершин, центров кругов, их радиусы, площади и разность площадей. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

10. Переменные XC , YC , R хранят соответственно координаты центра и радиус окружности. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины треугольников, найти такой треугольник, у которого сторона является касательной к окружности, а площадь треугольника, образованного его центром (точка пересечения медиан), центром окружности и точкой касания, максимальна. Вывести номера точек результирующего треугольника, координаты его вершин, координаты вершин треугольника, площадь которого вычислялась, саму его площадь. Если все треугольники вырожденные или нет искомого треугольника, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

11. Переменные XC , YC , R хранят соответственно координаты центра и радиус окружности. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины треугольников, найти такой треугольник, для которого прямая, соединяющая точку пересечения его медиан с центром окружности, образует максимальный угол с осью абсцисс. Вывести номера точек результирующего треугольника, координаты его вершин. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

12. Переменные XC , YC , R хранят соответственно координаты центра и радиус окружности. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины треугольников, найти такой треугольник, для которого прямая, соединяющая точку пересечения его высот с центром окружности, образует максимальный угол с осью абсцисс. Вывести номера точек результирующего треугольника, координаты его вершин. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

13. Переменные XC , YC , R хранят соответственно координаты центра и радиус окружности. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранят-

ся координаты точек на плоскости. Рассматривая эти точки как вершины треугольников, найти такой треугольник, для которого прямая, соединяющая точку пересечения его биссектрис с центром окружности, образует максимальный угол с осью абсцисс. Вывести номера точек результирующего треугольника, координаты его вершин. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

14. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины треугольников, найти такой треугольник, внутри которого расположено максимальное количество заданных в массивах X , Y точек. Вывести номера точек результирующего треугольника, координаты его вершин, номера точек, лежащих внутри, и их количество. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

15. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины треугольников, найти треугольник с максимальной разностью его площади и площади вписанного круга. Вывести номера точек результирующего треугольника, координаты его вершин, координаты центра круга, радиус, площади треугольника и круга и их разность. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

16. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины треугольников, найти треугольник с максимальной разностью площади описанного круга и площади треугольника. Вывести номера точек результирующего треугольника, координаты его вершин, координаты центра круга, радиус, площади круга и треугольника и их разность. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

17. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины треугольников, найти треугольник с минимальной площадью вписанного круга. Вывести номера точек результирующего треугольника, координаты его вершин, координаты центра круга, радиус и площадь. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

18. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины треугольников, найти треугольник с минимальной площадью описанного круга. Вывести номера точек результирующего треугольника, координаты его вершин, координаты центра круга, радиус и площадь. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

19. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины треугольников, найти треугольник с минимальной разностью площадей описанного и вписанного кругов. Вывести номера точек результирующего треугольника, координаты его вершин, координаты центров кругов, их радиусы, площади и разность площадей. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

20. Переменные XC , YC , R хранят соответственно координаты центра и радиус окружности. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая точки как вершины треугольников, найти треугольник, у которого сторона является касательной к окружности, а площадь треугольника, образованного его центром (точка пересечения медиан), центром окружности и точкой касания, минимальна. Вывести номера точек результирующего треугольника, координаты его вершин, координаты вершин треугольника, площадь которого вычислялась, саму его площадь. Если все треугольники вырожденные или нет искомого треугольника, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количе-

ства точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

21. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины треугольников, найти треугольник, у которого медиана образует минимальный угол с осью абсцисс. Вывести номера точек результирующего треугольника, координаты его вершин, медианы и угол, который она образует с осью абсцисс. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

22. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины треугольников, найти треугольник, у которого высота образует максимальный угол с осью ординат. Вывести номера точек результирующего треугольника, координаты его вершин, высоты и угол, который она образует с осью ординат. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

23. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины треугольников, найти треугольник, у которого биссектриса образует минимальный угол с осью ординат. Вывести номера точек результирующего треугольника, координаты его вершин, биссектрисы и угол, который она образует с осью ординат. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

24. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Рассматривая эти точки как вершины треугольников, найти треугольник максимальной площади, внутри которого расположено минимальное количество заданных в массивах X , Y точек. Вывести номера точек результирующего треугольника, координаты его вершин, номера точек, лежащих внутри, и их количество. Если все треугольники вырожденные, вывести об этом сообщение и вычислений не производить. Проверить корректность вводимого количества точек, в случае некорректности выве-

сти соответствующее сообщение (отрицательное количество, недостаточное количество).

25. Переменные X , Y , R хранят соответственно координаты центра и радиус окружности. В массивах $X(n)$, $Y(n)$, $n \leq 15$ хранятся координаты точек на плоскости. Среди этих точек найти такую, для которой угол между двумя касательными к заданной окружности, проведенными из этой точки, максимален. Вывести номер полученной точки, ее координаты и значение найденного угла. Проверить корректность вводимого количества точек, в случае некорректности вывести соответствующее сообщение (отрицательное количество, недостаточное количество).

Функция перекодировки кириллицы

При выводе на экран русскоязычных текстов можно использовать следующую функцию перекодировки символов:

```
function Rus(S:String):String;
var i:byte;
begin
    Result:='';
    for i:=1 to Length(S) do
        case S[i] of
            'А'..'п': Result:=Result+Chr(Ord(S[i])-64);
            'р'..'я': Result:=Result+Chr(Ord(S[i])-16);
            'Ё': Result:=Result+Chr(240);
            'ё': Result:=Result+Chr(241);
            else
                Result:=Result+S[i];
        end;
    end;
end;
```

Представленное выше объявление функции следует поместить в любом месте перед разделом операторов, а в операторах вывода обращаться к ней, указывая выводимый текст в качестве параметра. Например, чтобы в окне программы отобразилась строка, изображенная на рис. П.1.1, следует использовать следующий оператор:



Рис. П.1.1

```
WriteLn(Rus('При i = '), i
        ,Rus(' градусов sin(i) = ')
        ,sin(30*Pi/180)).
```

Дополнительные сведения по обработке исключений

Delphi предоставляет программные средства выявления ошибок, связанных с техническими ограничениями и особенностями выполнения некоторых операций. Такие ошибки могут приводить к неверным результатам или аварийному завершению программы (генерации *исключения* и стандартной его обработке) без выдачи каких-либо данных об их причине. В результате программа теряет *информативность*, оставляя пользователя в недоумении, что же произошло и как избежать подобных ситуаций.

Одним из способов *обработки исключений* и устранения указанного недостатка программы является использование оператора **try except**, имеющего синтаксическую диаграмму, показанную на рис. П.2.1, где блок **try** должен содержать операторы, выполнение которых может вызвать исключение, блок **except** — либо операторы **on** (из них будет выполнен только один, соответствующий возникшему исключению), либо другие, получающие управление в случае возникновения исключения. Если операторы блока **try** не вызывают исключения, то операторы блока **except** не выполняются. Если же исключение возникло, то действия, выполненные операторами блока **try**, отменяются.

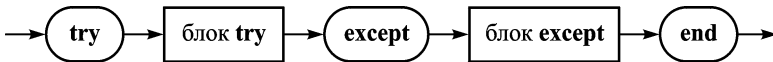


Рис. П.2.1

Чтобы узнать причину исключения, следует выявить операторы, вызывающие исключение (например, выполняя трассировку программы), заключить их в блок **try**, а в блок **except** вставить оператор **on Exception do**. Например, если исключение возникает при выполнении оператора

```
T:=Sqrt(S*2/g),
```

то его следует заменить оператором

```
try
  T:=Sqrt(S*2/g)
except
  on Exception do
end
```

В этом примере исключение возникнет при вычислении выражения $\text{Sqrt}(S^2/g)$, если ввести отрицательное значение переменной S .

При запуске программы из среды Delphi, если ввести отрицательное значение переменной S , работа программы будет остановлена, строка программы с оператором

```
T:=Sqrt(S*2/g)
```

будет выделена, и появится сообщение вида (рис. П.2.2) (окно с сообщением не появится, если запустить исполняемую программу из папки).



Рис. П.2.2

В сообщении текст, выделенный апострофами, поясняет причину возникновения исключения.

После закрытия окна сообщения можно продолжить работу с программой в режиме отладки: узнать текущие значения переменных и выражений (просматривая в окне наблюдения или подводя к ним курсор мыши), установить точки останова, использовать любую из команд выполнения программы (выполнить, выполнить до курсора, выполнить шаг трассировки и другие), подготовить программу к новому запуску (выполнить команду Run/ Program Reset).

В приведенном примере сообщения текст 'Invalid floating point operation' указывает на недопустимую операцию над вещественными данными. Причинами появления такого сообщения могут быть попытка вычислить функцию для аргумента, значение которого не принадлежит области ее определения (как в приведенном примере программы — извлечение корня квадратного из отрицательного числа), попытка разделить 0 на 0 и др. Соответствующий класс исключения — `EInvalidOp`. Зная это, заменим прежний пустой оператор **on** обработки исключения, т. е. **on** Exception **do**, оператором, выводящим сообщение о причине ошибки и завершающим работу программы:

```
try  
  T:=Sqrt(S*2/g)
```

```
except
  on EInvalidOp do
  begin
    WriteLn(Rus('Значение выражения S*2/g '))
      ,Rus('должно быть >= 0'));
    WriteLn(Rus('Для завершения работы программы')
      ,Rus(' нажмите клавишу Enter'));
    ReadLn;
    Halt
  end.
```

При запуске измененной исполняемой программы из окна папки стандартное сообщение (см. рис. П.2.2) появляться не будет, а программа выведет свой текст о причине ошибки:

Вводимое значение должно быть >= 0

Для завершения работы программы нажмите клавишу Enter

Примером «неправильного» вычисления может служить сложение двух целых чисел 2000000000, как показано в следующей программе:

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var
  I:Integer=2000000000;
begin
  I:=I+I;
  WriteLn(I);
  ReadLn
end.
```

Результатом будет вывод отрицательного числа –294967296. Объясняется это тем, что данные целого типа Integer обрабатываются на 32-разрядных регистрах, и все старшие двоичные цифры результата просто отбрасываются, а знак оставшейся части числа определяется двоичной цифрой в старшем разряде (0 — положительное число, 1 — отрицательное число).

Чтобы вызвать исключение при возникновении подобных ситуаций, следует ввести в программу директиву компилятора {\$Q+} (например, перед строкой с оператором `I:=I+I`). Стандартная обработка такого исключения приведет к выводу сообщения с текстом 'Integer overflow' — переполнение целого, и к выделе-

нию соответствующей строки программы. После закрытия окна сообщения можно просмотреть значения переменных, но продолжение работы программы невозможно — она будет снята с выполнения. Для обеспечения информативности программы и возможности продолжения работы наряду с директивой `{SQ+}` необходимо использовать оператор **try except** обработки исключения.

Рассмотрим некоторые общие положения, связанные с исключениями и их обработкой. *Исключения* — это объекты классов исключения, которые создаются автоматически при наступлении соответствующего события и уничтожаются сразу после их обработки (рассмотрены выше).

Единым предком всех классов исключений является класс `Exception`. Использование его имени в блоке **except** позволяет обработать исключение любого из его наследников, а тип исключения, которое реально возникло, вывести, используя следующую конструкцию:

```
try
  <ОП>;
except
  on E:Exception do
    begin
      .....
      WriteLn('Возникло исключение ', E.Message
              , ' при выполнении оператора <ОП>');
      .....
    end
end,
```

где `<ОП>` — оператор, вызвавший исключение, а `E` — имя переменной, используемое для вывода сообщения. Аналогично имя любого класса исключения может использоваться в блоке **try except** для обработки исключений его наследников.

В дальнейшем нас будут интересовать некоторые исключения, связанные с обработкой числовых данных, поэтому ограничимся рассмотрением классов `EDivByZero`, `ERangeError` и `EIntOverflow`, представляющих данные целых типов и имеющие общего родителя — класс `EIntError`, и классов `EInvalidOp`, `EOverflow` и `EZeroDivide`, представляющих вещественные данные и имеющие общего родителя — класс `EMathError`. Ниже даны пояснения к причинам появления исключений указанных классов.

— **EDivByZero** — деление данного целого типа (знак операции **div**) на выражение целого типа со значением 0;

— **ERangeError** — попытка присвоить переменной целого типа значение, выходящее за пределы допустимого диапазона, или выход значения выражения, используемого в качестве индекса, за допустимые пределы. Исключение возникает только в том случае, если соответствующий участок программы скомпилирован с директивой «проверка диапазона» **{R+}**. Конец участка отмечается директивой **{R-}**. Таких участков может быть несколько. Размещение одной директивы **{R+}** в начале программы обеспечит указанные проверки во всей программе;

— **EIntOverflow** — попытка вычислить значение выражения целого типа с использованием операций сложения, вычитания, умножения и стандартных функций **Abs**, **Sqr**, **Succ**, **Pred**, **Inc**, **Dec**, выходящее за пределы диапазона $-2147483648 \dots 2147483647$. Исключение возникает только в том случае, если соответствующий участок программы скомпилирован с директивой «проверка целочисленного переполнения» **{Q+}**. Конец участка отмечается директивой **{Q-}**. Таких участков может быть несколько. Размещение одной директивы **{Q+}** в начале программы обеспечит указанные проверки во всей программе;

— **EInvalidOp** — пояснения были даны ранее;

— **EOverflow** — попытка присвоить переменной вещественного типа значение, большее или меньшее допустимого для ее типа;

— **EZeroDivide** — попытка вычислить значение вещественного выражения, содержащее деление на 0.

В случае возникновения исключения в блоке **try** переменные программы возвращаются в состояние, бывшее перед входом в этот блок, что позволяет в цикле, не завершая работу программы, предпринять повторные попытки выполнения операторов этого блока при задании новых значений переменных.

В общем случае блок **except** может содержать несколько операторов **on**, допустимо также вложение блоков один в другой. Например,

```
try //Начало внешнего блока try except
...//Проверяемые операторы
...//внешнего блока try except
try //Начало внутреннего блока try except
.....//Проверяемые операторы
.....//внутреннего блока try except
except
```

```
.....//Операторы вида
.....// "on <Имя исключения> do <Оператор>"
.....//внутреннего блока try except
end; //Конец внутреннего блока try except
except
...//Операторы вида
...// "on <Имя исключения> do <Оператор> "
...//внешнего блока try except
end; //Конец внешнего блока try except
```

Если исключение, возникшее во внутреннем блоке **try**, обработано в нем не будет, то оно может быть обработано в ближайшем охватывающем внешнем блоке. Если же в программе не предусмотрена обработка исключения, то оно будет обработано стандартным способом.

Завершить работу программы можно вызовом не только процедуры **Halt**, но и **Exit** или **Abort**. Различие в их выполнении состоит в следующем. Процедура **Halt**, даже при ее вызове в подпрограмме, приводит к выходу из блока программы, а процедура **Exit** — только из блока программы или подпрограммы (из подпрограммы управление передается в охватывающий блок на продолжение выполнения операторов). Процедура **Abort** порождает исключение **EAbort**, стандартная обработка которого приводит к завершению программы без выдачи сообщения, но может быть обработано в программе в охватывающем блоке **try except**.

Основные формулы, используемые при решении геометрических задач

1. Уравнение прямой, проходящей через две точки с координатами (X_1, Y_1) , (X_2, Y_2) :

$$AX + BY + C = 0,$$

где $A = Y_1 - Y_2$; $B = X_2 - X_1$; $C = (X_1 - X_2)Y_1 + (Y_2 - Y_1)X_1$.

2. Координаты точки пересечения $(X_{\text{п}}, Y_{\text{п}})$ двух прямых, описываемых уравнениями $A_1X + B_1Y + C_1 = 0$, $A_2X + B_2Y + C_2 = 0$:

$$X_{\text{п}} = \frac{B_1C_2 - B_2C_1}{A_1B_2 - A_2B_1}, \quad Y_{\text{п}} = \frac{C_1A_2 - C_2A_1}{A_1B_2 - A_2B_1} \quad \text{при } A_1B_2 - A_2B_1 \neq 0.$$

Если $A_1B_2 - A_2B_1 = 0$, то прямые параллельны и не пересекаются.

3. Угол φ между двумя пересекающимися прямыми, заданными уравнениями $A_1X + B_1Y + C_1 = 0$, $A_2X + B_2Y + C_2 = 0$, определяется выражением

$$\varphi = \arctg \frac{A_1B_2 - A_2B_1}{A_1A_2 + B_1B_2}.$$

4. Условие параллельности двух прямых $A_1B_2 - A_2B_1 = 0$.

5. Условие перпендикулярности двух прямых $A_1A_2 + B_1B_2 = 0$.

6. Условие принадлежности трех точек с координатами (X_1, Y_1) , (X_2, Y_2) , (X_3, Y_3) одной прямой $(X_2 - X_1)(Y_3 - Y_1) - (X_3 - X_1)(Y_2 - Y_1) = 0$.

7. Расстояние от точки с координатами (X_1, Y_1) до прямой, заданной уравнением $AX + BY + C = 0$,

$$\delta = \frac{AX_1 + BY_1 + C}{\sqrt{A^2 + B^2}}.$$

8. Определение координат центра окружности (X_C, Y_C) и радиуса R на основе координат трех точек (X_1, Y_1) , (X_2, Y_2) , (X_3, Y_3) , ей принадлежащих:

$$X_C = \frac{BF - CE}{BD - AE}, \quad Y_C = \frac{AF - CD}{BD - AE},$$

где $A = 2(X_2 - X_1)$, $B = 2(Y_1 - Y_2)$, $C = Y_2^2 - Y_1^2 + X_2^2 - X_1^2$,
 $D = 2(X_3 - X_1)$, $E = 2(Y_1 - Y_3)$, $F = Y_3^2 - Y_1^2 + X_3^2 - X_1^2$.

Радиус окружности определяется из уравнения окружности $(X - XC)^2 + (Y - YC)^2 = R^2$ после подстановки в него координат любой из трех заданных точек: $\sqrt{(X_1 - XC)^2 + (Y_1 - YC)^2}$.

Список литературы

1. *Алексеев В.Е., Алексеев Ю.Е., Ваулин А.С.* и др. Практикум по программированию: Учеб. пособие по курсу «Вычислительная техника и информационная технология» / Под ред. Б.Г. Трусова. М.: Изд-во МГТУ им. Н.Э. Баумана, 1989. 132 с.
2. *Алексеев В.Е., Ваулин А.С., Петрова Г.Б.* Вычислительная техника и программирование. Практикум по программированию: Практ. пособие / Под ред. А.В. Петрова. М.: Высш. шк., 1991. 400 с.
3. *Архангельский А.Я.* Object Pascal в Delphi. М.: БИНОМ, 2002. 384 с.
4. *Блюмин А.Г., Гусев Е.В., Федотов А.А.* Численные методы. Метод. указания к лабораторным работам / Под ред. Г.А. Колотушкина. М.: Изд-во МГТУ им. Н.Э. Баумана, 2002. 48 с.
5. *Выгодский М.Я.* Справочник по высшей математике. М.: Наука, 1976. 872 с.
6. *Иванова Г.С.* Основы программирования: Учебник. М.: Изд-во МГТУ им. Н.Э. Баумана, 2007. 415 с.
7. *Йенсен К., Вирт Н.* Паскаль: руководство для пользователя / Пер. с англ. А.Я. Архангельского. М.: БИНОМ, 2002. 384 с.
8. *Кнут Д.* Искусство программирования для ЭВМ. В 3 т. Т. 3. Сортировка и поиск: Пер с англ. М.: Мир, 1978. 844 с.
9. *Мак-Кракен Д., Дорн У.* Численные методы и программирование на Фортране: Пер. с англ. М.: Мир, 1977. 584 с.
10. *Смолянский М.Л.* Таблицы неопределенных интегралов. М.: Физматгиз, 1961. 108 с.
11. *Фаронов В.В.* Система программирования Delphi. СПб.: БХВ-Петербург, 2005. 912 с.

Алфавитный указатель

Б

Бесконечный ряд 99
Блок 13
Булево выражение 45, 47

В

Вектор 80
Вложенные циклы 129
Внешний цикл 129
Внутренний цикл 129
Входной параметр 174, 175
Вызов подпрограммы 172
Выходной параметр 175

Д

Двумерный массив 146
Дизъюнкция 46
Динамические массивы 212
Директива 11
Дополнительный параметр
цикла 59

З

Заголовок модуля 225
Заголовок программы 11
Заголовок процедуры 173
Заголовок функции 184
Заголовок цикла 57

И

Именованные константы 13
Индексная переменная 78
Интерфейсная часть 225
Исключения 274

К

Ключевые слова 11
Комментарий 11
Конъюнкция 46
Курсор 9

М

Массив 78
Массив одномерный 78
Матрицы 146
Метод касательных 116
Метод парабол 121
Метод половинного деления 115
Метод простых итераций 111
Метод прямого включения 164
Метод прямого выбора 166
Метод прямого обмена 165
Метод прямоугольников 121
Метод пузырька 165
Метод Симпсона 121
Метод трапеций 121
Модули пользователей 12, 225

Н

Накопление произведения 86
Накопление суммы 86
Начальные значения
переменных 16
Неравнозначность 46

О

Обработка исключений 274
Обращение к подпрограмме 174
Обращение к функции 184

Общая формула члена ряда 100
 Объявление функции 184
 Объявление подпрограммы 173
 Объявление типа двумерного массива 146
 Окно наблюдения 19
 Окно программы 9
 Окно редактирования программы 11
 Оператор выбора 44
 Операторные скобки 46
 Операторы 11
 Операторы циклов 57
 Описание подпрограммы 172
 Ординальный тип 57, 78
 Отладка 16
 Отрицание 46

П

Параметр — открытый массив 209
 Параметр цикла 57
 Параметр-константа 177
 Параметр-результат 177
 Параметры со значениями по умолчанию 216
 Параметры-значения 175
 Параметры-переменные 175
 Перегружаемые подпрограммы 215
 Переменная 13
 Поиск наибольшего 87
 Поиск наименьшего 87
 Порядковый тип 57, 78
 Последовательность 80
 Предложение использования 11, 225
 Программы линейной структуры 21
 Программы разветвляющейся структуры 44
 Программы циклической структуры 57
 Проект 12
 Пустой оператор 45

Р

Раздел операторов 11
 Расширенный синтаксис 185
 Рекуррентная формула члена ряда 100
 Рекурсивные подпрограммы 196

С

Синтаксические ошибки 15
 Смешанный способ вычисления члена ряда 101
 Сортировки 163
 Список фактических параметров 173, 174
 Список формальных параметров 173
 Стандартные типы 21
 Стандартные подпрограммы 12
 Стандартный модуль 12
 Статический массив 78
 Сумма ряда 99

Т

Тело цикла 57
 Тип данных 12
 Тип функции 185
 Трассировка 18

У

Упорядочения 163
 Условная компиляция 17
 Условные операторы 44

Ф

Фактические параметры 173, 174
 Формальные параметры 173

Ц

Цикл 57
 Цикл с заданным числом повторений 57
 Цикл с параметром 57
 Цикл с постусловием 57
 Цикл с предусловием 57

Ч

Часть инициализации 226

Часть реализации 225

Часть финализации 226

Член ряда 99

Ш

Шаблон консольной программы 11

Э

Элемент массива 78

Учебное издание

**Алексеев Юрий Евтихович
Ваулин Анатолий Сергеевич
Куров Андрей Владимирович**

ПРАКТИКУМ ПО ПРОГРАММИРОВАНИЮ

Редактор *В.М. Царев*
Техн. редактор *Э.А. Кулакова*
Корректор *И.М. Мартынова*
Художник *Н.Г. Столярова*
Компьютерная графика *Т.Н. Аверчивой*
Компьютерная верстка *Н.Ф. Бердацевой*

Оригинал-макет подготовлен в издательстве МГТУ им. Н.Э. Баумана

Подписано в печать 21.04.08. Формат 60×90/16. Бумага офсетная.
Печать офсетная. Усл. печ. л. 18,0. Уч.-изд. л. 16,75.

Тираж 2000 экз. Заказ

Издательство МГТУ им. Н.Э. Баумана
105005, Москва, 2-я Бауманская, д. 5

ISBN 978-5-7038-3159-5



9 785703 831595

Для заметок

Для заметок