С. М. Кашаев, Л. В. Шерстнева

bhv

ПАСКАЛЬ ДЛЯ ШКОЛЬНИКОВ подготовка к ЕГЭ

2-е издание

В теоретической части не требует обращения к другим источникам

Дает возможность получить практические навыки выполнения типовых заданий, в том числе части С

Делает акценты на разделах программирования, выносимых на ЕГЭ

Содержит примеры экзаменов 2010 года и заданий ЕГЭ прошлых лет

ИНФОРМАТИКА И ИНФОРМАЦИОННО-КОММУНИКАЦИОННЫЕ ТЕХНОЛОГИИ

+CD

С. М. Кашаев
Л. В. Шерстнева

ПАСКАЛЬ
ДЛЯ ШКОЛЬНИКОВ
подготовка к ЕГЭ

2-е издание

Санкт-Петербург
«БХВ-Петербург»
2011

УДК 681.3.068+800.92Pascal
ББК 32.973.26-018.1
К31

Кашаев, С. М.

К31 Паскаль для школьников. Подготовка к ЕГЭ / С. М. Кашаев, Л. В. Шерстнева. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2011. — 336 с.: ил. + CD-ROM — (ИиИКТ)

ISBN 978-5-9775-0702-8

Подробно описаны приемы программирования на Паскале и технология разработки различных алгоритмов программ с акцентом на темы, выносимые на Единый государственный экзамен по информатике и информационно-коммуникационным технологиям. Рассматриваются: описание языка Паскаль, конструкции алгоритмов и блок-схемы, одномерные и двумерные массивы, строки и записи, файлы, численное интегрирование и анализ функций, подпрограммы и функции, работа с данными. По каждому разделу приводится теоретическая информация и типовые задания с подробными пояснениями. По темам, выносимым на ЕГЭ прошлых лет, в том числе прошедшего 2010 года, что отличает второе издание, в соответствующих главах приводятся примеры заданий этих ЕГЭ. Книга может использоваться как при подготовке к ЕГЭ, так и в текущем учебном процессе учащимися и учителями школ и колледжей, а также для самостоятельного изучения языка программирования Паскаль.

Прилагаемый компакт-диск содержит рассматриваемые в книге программы.

Для образовательных учреждений

УДК 681.3.068+800.92Pascal
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Людмила Еремеевская*
Зав. редакцией	*Григорий Добин*
Редактор	*Юрий Рожко*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.01.11.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 27,09.
Тираж 2000 экз. Заказ № 733
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов в ГУП "Типография "Наука" 199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0702-8

Оглавление

Введение

Основным практическим результатом школьного курса информатики является формирование навыков программирования на одном из языков высокого уровня — Бейсик, Паскаль, Си. Анализ учебного процесса показывает, что это составляет наиболее трудную часть курса информатики. Об этом говорят и результаты Единого государственного экзамена (ЕГЭ) по информатике. Так, статистика результатов за 2009 и 2010 годы относит дисциплину "Информатика и информационно-коммуникационные технологии" к категории наиболее трудных для учащихся (среднестатистические показатели по информатике оказались хуже статистики большинства дисциплин). Фактически эта трудность заложена в алгоритмизации задач и программировании. От учащихся в экзаменационных билетах требуется анализ, разработка и последующее программирование алгоритмов. Эти вопросы занимают большое место в Едином государственном экзамене и формируют категорию наиболее сложных заданий. Задача нашей книги сводится к последовательному и поэтапному формированию навыков программирования, а также к подготовке читателей к решению заданий ЕГЭ по данной теме.

В настоящее время используются различные языки программирования. При этом в школьной программе традиционными являются Бейсик и Паскаль. В представленной книге сделан выбор в пользу языка Паскаль, который можно считать наиболее распространенным в школьной программе (его можно назвать базовым языком программирования для школьников).

Паскаль был разработан Николасом Виртом в шестидесятые годы прошлого века в качестве учебного языка для студентов. В настоящее время существует несколько программных продуктов, которые предназначены для написания и выполнения программ на Паскале. Наибольшей популярностью среди учащихся и педагогов пользуется Turbo Pascal, и при рассмотрении примеров в книге мы будем использовать именно среду Turbo Pascal. Эта система программирования была разработана фирмой Borland, и на протяжении последних двух десятков лет именно Turbo Pascal остается наиболее удобной платформой для изучения программирования.

Книга построена таким образом, чтобы учащиеся школ (а также техникумов и колледжей) смогли самостоятельно рассмотреть как необходимую теоретическую информацию, так и на разнообразных примерах получить навыки практического про-

граммирования. Здесь рассмотрено большое количество заданий ЕГЭ по дисциплине "Информатика и информационно-коммуникационные технологии". И если вы собираетесь сдавать ЕГЭ по этой дисциплине, то мы поможем познакомиться с типовыми задачами, которые вас ожидают.

В целом книга состоит из девяти глав, содержание которых охватывает все основные разделы программирования на языке Паскаль. Можно сказать, что главы книги включают все темы, выносимые на Единый государственный экзамен, касающиеся практического программирования. Структура каждой главы построена по схожему сценарию. Так, вместе с необходимыми теоретическими сведениями, которые излагаются в понятном для учащихся стиле, приводятся типовые примеры программных разработок. В большинстве глав (1—4, 8, 9) приводится разбор заданий, которые *встречались в билетах Единого государственного экзамена в прошлые годы*.

Краткое содержание книги

В *главе 1* читатели познакомятся с основными операторами языка Паскаль и технологией выполнения программ в среде Turbo Pascal. Мы разберем типы данных, существующие в Паскале, организацию ввода и вывода информации.

В любой, даже несложной, программе используются операторы циклов и проверки условий. Циклы позволяют организовать повторяющиеся вычисления. Что касается условий, то в зависимости от выполнения либо невыполнения условия могут выполняться (или не выполняться) определенные фрагменты программы. Рассмотрению циклов и условий посвящена *глава 2* книги.

В *главе 3* подробно разбирается работа с одномерными массивами, которые являются одной из наиболее используемых на практике структур данных. Рассматриваются задачи поиска элементов, вычисления суммы и среднего значения в массиве.

Двумерные массивы встречаются практически в каждом билете Единого государственного экзамена, и работе с такими структурами посвящена *глава 4*. Аналогично содержанию третьей главы мы разберем задачи поиска и суммирования элементов по условию.

В *главе 5* рассматриваются строки и записи. Это наиболее популярные структуры при работе с текстовой информацией. Записи позволяют компактно в одном элементе хранить различную информацию (текст, числовые и логические данные).

Любая система программирования включает ресурсы для работы с файлами. *Глава 6* книги касается рассмотрения вопросов, связанных с созданием и чтением файлов. Мы познакомимся со стандартными программными процедурами и функциями, предназначенными для открытия, чтения и записи информации в файлы.

Из *главы 7* вы узнаете, каким образом можно разделить программу на отдельные блоки — пользовательские процедуры и функции. Практически все программные разработки строятся в виде совокупности подобных блоков.

Математические вычисления составляют доминирующую составляющую программных разработок. И в *главе 8* мы разберем примеры решений уравнений и неравенств, численное интегрирование и метод Монте-Карло.

Глава 9 практически целиком построена на рассмотрении примеров заданий Единого государственного экзамена. Все эти примеры отличаются достаточной сложностью и связаны с различными алгоритмами обработки данных.

От авторов книги

Нам бы хотелось выразить благодарность всем читателям, которые познакомились с нашей книгой. Что касается методики изложения информации, то она связана с работой на курсах по подготовке к Единому государственному экзамену в Нижегородском государственном техническом университете. Много работы в этом плане проводит кафедра "Прикладная математика". Хотим выразить свою благодарность заведующему этой кафедрой Митякову Сергею Николаевичу. Также большую помощь в работе нам оказал заместитель директора Института дистанционного обучения Воронков Юрий Васильевич, и ему мы тоже весьма признательны.

Необходимо также упомянуть издательство "БХВ-Петербург" за сотрудничество в плане подготовки к изданию наших книг на протяжении последних лет. Особенно хотим поблагодарить заместителей главного редактора издательства Людмилу Юрьевну Еремеевскую и Рыбакова Евгения Евгеньевича за поддержку, ценные советы и внимание.

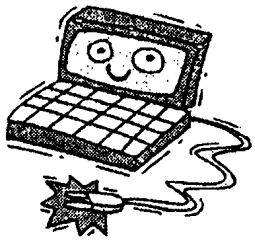
Первое издание книги у читателей пользовалось несомненным интересом. Это явилось причиной выпуска второго издания, которое вы и держите сейчас в своих руках. Основное дополнение касается включения в эту книгу заданий из ЕГЭ по дисциплине "Информатика и информационно-коммуникационные технологии" за 2010 год.

Книга может быть использована школьниками в качестве вспомогательного материала по информатике в процессе учебных занятий, а также при подготовке к Единому государственному экзамену. Думаем, что книга может быть полезна и учителям информатики в плане организации занятий и подбора примеров. В заключение подчеркнем, что материал, изложенный в книге, предназначается для самостоятельного изучения технологии программирования в системе Turbo Pascal.

Разумеется, в книге возможны некоторые неточности, за которые заранее приносим читателям свои извинения и о которых (если они будут обнаружены) просим присылать сообщения.

Связаться с авторами книги можно по адресу электронной почты **mail@bhv.ru** или с помощью сайта **www.bhv.ru** издательства "БХВ-Петербург".

ГЛАВА 1



Знакомство с языком Паскаль

Из языков программирования среди учащихся и педагогов школ наибольшей популярностью пользуется Паскаль. При этом в качестве среды программирования чаще всего используется система Turbo Pascal, которая была разработана фирмой Borland и которая на протяжении последних двух десятков лет остается одним из наиболее удобных средств для изучения программирования. Все рассмотренные в книге теоретические разделы и примеры программирования алгоритмов приводятся именно в этой среде.

Основой функциональности среды Turbo Pascal является возможность создания и выполнения программ на Паскале. При этом для набора текста программ в этой среде имеется собственный текстовый редактор, хотя сами тексты можно создавать и в другом текстовом редакторе (например, в известном приложении Windows — Блокнот). Кроме редактора, основными компонентами среды также являются: *компилятор*, *компоновщик* и *отладчик*.

Turbo Pascal представляет собой технологическое средство для разработки программ, где каждая программа реализует тот или иной алгоритм. Понятие алгоритма занимает ключевое место в процессе программирования. *Алгоритм* представляет собой строгую систему правил, определяющую последовательность действий для решения конкретной задачи. Следуя такой системе, в разных ситуациях нужно действовать совершенно одинаково (по единой схеме). Можно сказать, что алгоритмом является такая последовательность арифметических и логических действий, которая позволяет решить поставленную задачу за конечное число шагов. В качестве решаемой задачи может быть, например, нахождение решения уравнения или поиск необходимой информации в файле с данными. Этапы разработки вычислительной программы выглядят следующим образом:

1. *Постановка задачи*, которая выполняется специалистом в предметной области (математика, физика, строительство и т. д.). На этом этапе определяется общий подход к решению задачи. Среди задач, рассматриваемых в книге, мы встретимся с поиском максимального значения в массиве данных, сортировкой чисел, расчетом средних показателей и т. д.

2. *Анализ задачи и моделирование*, которые приводят к построению (или выбору) математической модели. Этот этап очень важен, т. к. в ряде случаев анализ показывает, что поставленную задачу можно решить аналитическими методами.
3. *Построение алгоритма* решения задачи, выполняемое на основании математической модели. В большинстве ситуаций существуют несколько способов построения работающего алгоритма. От разработчика всегда требуется найти оптимальную систему правил для решения поставленной задачи.
4. Реализация алгоритма в виде *работающей программы* на одном из языков программирования. Этот этап часто называют *кодированием* — записью алгоритма в виде набора синтаксических конструкций выбранного языка программирования.
5. *Отладка* и *тестирование* программы. Отладка заключается в устранении программных ошибок. Для поиска ошибок разработчик должен предложить набор тестов, представляющих собой контрольные примеры с характерными параметрами, для которых решение задачи известно. Тестирование позволяет ответить на вопрос — является ли разработанная программа наилучшим решением данной задачи.

Описанная последовательность шагов достаточно понятна, но ее осуществление зависит от сложности поставленной задачи (для простых задач реализация перечисленных шагов достаточно очевидна, а для сложных ситуаций часто требуется длительное время и большие усилия). Фактически все программные разработки, которые вы знаете и с которыми познакомитесь в книге, также вписываются в данную схему. Мы будем выполнять перечисленные этапы, начиная с простых задач. Затем после приобретения определенного опыта в последующих главах разберем разделы, которые являются ключевыми, как для школьного курса информатики, так и для ЕГЭ по дисциплине "Информатика и информационно-коммуникационные технологии".

Среда Turbo Pascal

Для запуска Turbo Pascal используется один из исполняемых файлов среды, в качестве которых могут выступать turbo.exe или bp.exe. После выполнения любого из них на экране вы увидите элементы интегрированной среды:

- *главное меню* (строка сверху);
- *окно редактирования* ("полотно" синего цвета в центре);
- *описание функциональных клавиш* (строка в нижней части окна).

Несмотря на обширный набор элементов меню, тем не менее основных пользовательских действий не так уж много. Так, в разделе меню **File** (Файл) присутствуют команды **New** (Новый) и **Open** (Открыть). С помощью первой из них можно приступить к созданию нового файла, который будет содержать программу на языке Паскаль, а посредством второй открыть уже имеющийся файл. И в том, и в другом случае мы переходим к работе с текстовым редактором, входящим в систему Turbo Pascal.

Важно, что расширение файла, содержащего программу на языке Паскаль, должно быть PAS, и когда вы приступаете к созданию нового файла, то среда автоматически открывает новое окно с заголовком NONAME00.PAS. Это и есть имя файла, которое система Turbo Pascal определила автоматически. Далее от нас требуется написать программу, решающую ту или иную задачу.

Примечание

Для языка Паскаль не имеет значения, какие буквы вы используете при наборе текста программы (строчные либо заглавные). Обычно каждый программист сам выбирает свой стиль.

После набора текста программы ее необходимо сохранить на жестком диске. Для этого в меню **File** (Файл) имеется команда **Save File As** (Сохранить как). И от вас далее потребуется указать только имя для сохраняемого файла (практически всегда его выбирают самостоятельно вместо NONAME00.PAS).

Написанную программу на Паскале следует скомпилировать, скомпоновать и запустить на выполнение. В этом случае текст программы на языке Паскаль будет преобразован в последовательность команд, понятных процессору компьютера. Для выполнения всех этих перечисленных шагов за один прием предназначена команда **Run** (Запустить), которая находится в меню **Run** (Пуск).

Как правило, любая программа в процессе своей работы выводит информацию на экран (это могут быть промежуточные и итоговые результаты), после чего управление передается в среду (на экране опять отображается окно текстового редактора с исходной программой). Данную информацию можно увидеть на *экране пользователя*, для перехода в который предназначена комбинация клавиш <Alt>+<F5>. Чтобы вернуться обратно в окно редактора программного кода, необходимо нажать любую клавишу. В принципе, перечисленных сведений вполне достаточно для того, чтобы приступить к практической работе в среде Turbo Pascal и начать непосредственное знакомство с языком программирования. Но предварительно полезно пояснить *процесс компиляции* текста программы (написанной на языке Паскаль или на каком-либо другом языке программирования).

Исходный текст программы на Паскале не может непосредственно исполняться компьютером — вычислительная система не понимает синтаксических конструкций языка программирования. Для получения выполняемой программы исходный текст требуется обязательно *откомпилировать*, т. е. перевести в машинный код. *Машинный код* — это последовательность команд микропроцессора, состоящая из нулей и единиц. Так, первая команда представляет собой один набор двоичных разрядов, следующая команда — другой набор и т. д. Важно заметить, что каждый такой набор нулей и единиц вычислительная система однозначно понимает (в принципе, она *только это* и понимает).

Работу по переводу текста на языке Паскаль в машинный код выполняет специальная программа, называемая *компилятором*. Для компиляции программы в меню **Compile** (Компиляция) следует воспользоваться командой **Compile** (Компилировать). Если компилятор не обнаружит ошибок, то на экране появится окно с сообщением: *Compilation successful: press any key* (Компиляция прошла успешно:

нажмите любую клавишу). Данное окно остается на экране до тех пор, пока вы не нажмете какую-нибудь клавишу. Если обнаружена ошибка, то курсор устанавливается на ошибку в окне редактирования и выдается сообщение об ошибке.

Примечание

На практике удобнее пользоваться командой **Run** (Запустить), т. к. в этом случае ваша программа после компиляции будет автоматически запущена на выполнение (при условии, что в ней нет ошибок).

Программа, которую удалось откомпилировать, не обязательно работает правильно. Она может содержать ошибки, для выявления которых предназначен этап *отладки*. В целом программные ошибки могут быть трех видов:

- *синтаксические* (ошибки в правилах языка, которые обнаруживает компилятор);
- *алгоритмические* (ошибки в логике программы, которые при внешне безошибочной работе программы приводят к неверным результатам);
- *исполнения* (ошибки, возникающие в процессе работы запущенной программы).

Компилятор способен найти только синтаксические ошибки, для выявления же алгоритмических ошибок служит этап *тестирования* программы. Ошибки исполнения часто возникают как результат некорректных действий пользователя, недопустимых операций над данными (например, попытка извлечь квадратный корень из отрицательного числа или деление на ноль).

Программа вывода сообщения на экран

Практическую работу с языком программирования Паскаль начнем с простого примера. Так, наша задача заключается в создании программы, которая выводит на экран сообщение: "Пример первой программы". Подобные программы традиционно рассматриваются в качестве первых при изучении той или иной системы программирования. Необходимый текст программы (точнее один из вариантов, т. к. обеспечить вывод сообщения на экран можно и по-другому), который потребуется набрать в редакторе, представлен в листинге 1.1.

Листинг 1.1. Вывод сообщения на экран

```
program listing_1_1;
begin
  write ('Пример');
  write (' первой программы')
end.
```

Разберем содержание листинга 1.1. Так, первая строка является заголовком программы и начинается со слова `program`, после которого располагается имя программы (`listing_1_1`). Имена программ следует выбирать самостоятельно. Слово `program` является *ключевым* (зарезервированным) в Паскале и используется для определения начала программы.

Примечание

Ключевых слов в Паскале довольно много, и в дальнейшем мы познакомимся и с другими.

Важно заметить, что имя программы не должно содержать пробелов. В ситуациях же, когда пробел напрашивается, используют символ подчеркивания. В приведенном примере мы как раз этим и воспользовались.

Рассматриваемую заголовочную строку можно не включать в программу, т. к. она не является обязательной. Однако для придания программным разработкам определенного стиля заголовки программ с информативными (для программиста) именами рекомендуется включать.

Далее в тексте листинга 1.1 размещается исполняемая часть программы, которая начинается с ключевого слово `begin`. После него следуют выполняемые *операторы* (инструкции) *программы*. Каждый оператор в Паскале должен завершаться точкой с запятой. Это фактически является символом отделения одного оператора от другого. И поэтому операторы можно записывать в одной строке (один оператор может следовать за другим через точку с запятой), что делает текст программы существенно компактнее.

Примечание

Имеются ситуации, когда точку с запятой в конце строки ставить не надо, и они нам встретятся в книге далее.

После слова `begin` в приведенной программе листинга 1.1 располагаются два оператора `write`. Данный оператор позволяет обеспечить вывод информации на экран. Варианты его использования могут быть достаточно сложными. Это зависит от того, что мы выводим (в последующих разделах об этом будет сказано более подробно). В рассматриваемом примере осуществляется вывод текстового сообщения на экран. Важно отметить, что выводимый текст в операторе `write` необходимо заключать в одинарные кавычки — *апострофы*.

Примечание

Оператор `write` фактически представляет собой *стандартную процедуру языка* Паскаль. По-другому такие процедуры называются *встроенными*.

Последняя строка листинга 1.1 является *директивой окончания* программы (`end.`). Таким образом, теперь текст программы стал понятен, и следующий этап заключается в ее выполнении в среде Turbo Pascal.

Для выполнения программы следует воспользоваться одним из перечисленных вариантов:

- воспользоваться командой **Run** (Запустить) в меню **Run** (Пуск);
- использовать сочетание клавиш <Ctrl>+<F9>.

Для просмотра результата работы необходимо нажать комбинацию клавиш <Alt>+<F5>. В этом случае на экране мы увидим строку с текстовым сообщением: "Пример первой программы".

В данной программе мы могли бы обойтись одним оператором вывода, в качестве параметра которого следовало включить все сообщение целиком:

```
write ('Пример первой программы')
```

Такая редакция не требует завершения единственного оператора программы точкой с запятой (разделять нечего). Однако в содержимом листинга 1.1 мы разбили вывод сообщения на два оператора только для того, чтобы пояснить необходимость включения символа разделения операторов.

Таким образом, на рассмотренном примере мы познакомились с первой программой на Паскале и технологией ее выполнения в среде программирования Turbo Pascal.

Вычисления в программе

Как правило, большинство программ связано с организацией вычислительных действий. В качестве простого примера на данную тему рассмотрим реализацию программы, которая должна обеспечить возведение в квадрат значения целого числа, вводимого с клавиатуры. После этого результат вычисления необходимо вывести на экран. Подобная цель достаточно понятна, а набор правил для ее реализации выглядит так: ввод целого числа с клавиатуры, возведение этого числа в квадрат и вывод полученного результата на экран. Для описания алгоритмов, кроме такого словесного описания, используются *блок-схемы*, и с ними мы познакомимся далее в этой главе. Здесь же в связи с максимальной простотой решаемой задачи, кроме словесного перечисления шагов построения необходимого алгоритма для его пояснения больше ничего не требуется.

Примечание

Блок-схема — это графическое изображение алгоритма в виде плоских фигур, соединенных линиями. Блок-схема представляет собой стандартный способ записи алгоритма и существует государственный стандарт (ГОСТ), содержащий перечень правил построения блок-схем.

После формулировки задачи и определенности с алгоритмом перейдем к кодированию. Для этого в текстовом редакторе среды Turbo Pascal следует набрать текст программы (листинг 1.2), которая реализует поставленную задачу.

Листинг 1.2. Вычисление квадрата числа, вводимого с клавиатуры

```
program listing_1_2;
var
  N:integer;
begin
  read(N);
  N:=N*N;
  write (' Квадрат введенного числа равен ',N)
end.
```

Новое ключевое слово, которое нам здесь встретилось, это **var**. Оно означает начало области объявления переменных в программе.

Примечание

Сокращение **var** образовано от английского "variable" (переменная).

В тексте данной программы нам также встретился оператор *присваивания* (обозначается парой следующих друг за другом символов: двоеточие и знак равенства). Это наиболее часто используемый оператор, который работает следующим образом: сначала вычисляется выражение, стоящее *справа* от оператора присваивания :=, а затем результат записывается в переменную, стоящую *слева* от данного знака. Например, после выполнения оператора

```
K:=K+2;
```

текущее значение переменной K увеличится на 2.

Важно отметить, что справа от оператора присваивания может располагаться любое число либо выражение, а слева обязательно должно находиться имя переменной (мы должны указать системе — куда следует записать результат вычисления или просто данное). При этом тип результата выражения справа должен быть таким, чтобы он помещался в переменную слева от оператора присваивания.

Продолжим рассмотрение содержания листинга 1.2. После начала программы располагается строка, где определяется переменная N. Дело в том, что данные, с которыми мы работаем, необходимо где-то хранить. И для этого предназначается основная (оперативная) память компьютера. Такая память состоит из набора однотипных ячеек, при этом каждая ячейка имеет свой номер, который называется *адресом*. Чтобы не работать с адресами ячеек напрямую (не запоминать длинные числовые адреса ячеек), предусмотрена возможность присвоения ячейкам символического имени (идентификатора). Во второй строке текста, приведенного в листинге 1.2, мы использовали ключевое слово **var**, которое говорит о том, что после него определяются переменные программы. В нашем случае это переменная N, и для нее мы указали тип **integer**, который определяет целочисленное данное. Для переменных указанного типа в памяти отводится 16 бит (16 двоичных разрядов или 2 байта). При этом числа, хранящиеся в переменных типа **integer**, являются знаковыми, и их диапазон составляет от -2^{15} до $2^{15}-1$ [1, 2].

Примечание

Поскольку любые данные в памяти компьютера хранятся в двоичной системе счисления, то кроме имени переменной обязательно следует присвоить и *тип*, определяющий *диапазон значений*, принимаемых переменной, и *способ ее обработки* вычислительной системой.

Без дополнительной информации о *типе* данных, хранящихся в некоторой ячейке памяти, компьютеру было бы невозможно решить, что именно представляют собой эти данные. Например, 65 может быть просто числом в десятичной системе счисления, а может являться кодом буквы английского алфавита 'A' — все символы в

компьютере представлены *ASCII-кодами* (American Standard Code for Information Interchange, американский стандартный код обмена информацией) [1].

Имена (идентификаторы) переменных следует выбирать самостоятельно, и в рассматриваемой программе мы выбрали в качестве имени единственной переменной идентификатор `N`. Любые имена переменных в языке Паскаль строятся по следующим правилам:

- имена могут включать латинские буквы, цифры и знак подчеркивания (буквы русского алфавита не должны входить в состав идентификатора);
- имя состоит из одного слова, а если требуется пробел в имени, то он заменяется на подчеркивание (`St_1` будет правильным вариантом для имени, а `St 1` приведет к ошибке на этапе компиляции текста программы);
- имя всегда начинается с буквы либо со знака подчеркивания;
- между двумя именами должен располагаться, как минимум, один пробел;
- прописные и строчные буквы в именах не различаются компилятором языка Паскаль (`Z1` и `z1` — это одно и то же имя);
- имена переменных не могут совпадать с зарезервированными в языке *ключевыми (служебными) словами* (например, нельзя назвать **`begin`** ни одну переменную в программе);
- максимальное число символов, распознаваемых системой Turbo Pascal, в имени равняется 63, однако на практике практически всегда используются имена, не превышающие 10—20 символов.

Примечание

В программах, приводимых в книге, мы будем использовать следующий стиль выбора имен переменных: первая буква будет прописной, а последующие — строчными.

Вернемся к листингу 1.2, где после описания переменной располагаются три оператора. Первый из них представляет собой стандартную команду (**`read`**) ввода данных с клавиатуры. В скобках указано, что введенные данные фиксируются в переменной `N`. После этого располагается оператор (обозначается знаком `*`), который реализует умножение — значение переменной умножается на само себя. Фактически это является возведением исходного значения в квадрат. Результат перемножения заносится обратно в переменную `N` (в ячейку памяти, которую мы отвели для переменной).

Заметим, что введенное с клавиатуры значение должно представлять собой целое число (тип **`integer`**), и в случае ввода набора букв вместо целого числа при выполнении программы произойдет ошибка. Это связано с тем, что операция умножения для строк и символов отсутствует.

После выполнения умножения в программе располагается команда вывода информации на экран. При этом через запятую необходимо перечислить данные, которые следует вывести. В данном примере мы выводим на экран текст, который информирует пользователя (текст в команде **`write`** должен заключаться в одинарные кавычки), а также значение переменной `N`.

Важно заметить, что точку после ключевого слова **end** в конце программы ставить нужно обязательно.

Структура программы

После двух рассмотренных простых практических примеров приведем теперь общую структуру программы на языке Паскаль:

```
program name_prog
{Раздел описаний}
begin
{Исполнительный раздел}
end.
```

В начале программы, как мы уже знаем, располагается необязательный заголовок, состоящий из ключевого слова **program** и имени программы. В заголовке могут также присутствовать параметры, с помощью которых программа взаимодействует с внешним миром. Имена программ позволяют программисту их быстро идентифицировать, поэтому ими желательно не пренебрегать.

В программе можно использовать *комментарии* — пояснительный текст. Текст комментария должен быть окружен либо фигурными скобками ({...}), либо следующими парами символов: круглая скобка и звездочка. Например:

```
(*    текст комментария      *).
```

Комментарий можно записать в любом месте программы, где разрешен пробел. Символами комментария удобно пользоваться при отладке программ для временного исключения группы операторов. В этом случае фрагмент, заключенный в символы комментария, воспринимается компилятором как комментарий, и следовательно, не будет выполнен.

В разделе описаний необходимо объявить все встречающиеся в программе данные. Это касается переменных, констант и т. д. Важно, чтобы все описания объектов программы были сделаны до того, как они будут использоваться. В разделе описаний можно выделить несколько подразделов:

- *раздел описания констант* (в отличие от переменных константы не изменяются в операторах исполнительного раздела);
- *раздел описания переменных*;
- *раздел описания меток* (перед любым оператором можно поставить метку, которая позволит выполнить переход на него с помощью оператора `goto` из любого места программы);
- *раздел объявления типов* (кроме уже имеющихся типов данных в системе можно определить и свои пользовательские типы);
- *раздел для подключения стандартных и пользовательских библиотечных модулей*;

- *раздел описания процедур и функций* (это фрагменты программного кода, которые вызываются как единое целое).

В исполнительном разделе расположены инструкции (операторы), которые определяют алгоритм обработки данных. Важно, что этот раздел обязательно должен начинаться с ключевого слова `begin`, а завершаться ключевым словом `end`. (с точкой в конце).

Операторы в программах на Паскале не привязаны к конкретной позиции строки. Разрешается в одной строке размещать несколько операторов, разделяя их точкой с запятой.

Типы данных

Для решения любой задачи с использованием программирования производятся действия над определенными данными. Эти данные могут иметь разные типы. *Тип данных* определяет множество значений, которые могут принимать объекты программы (константы и переменные). Также тип определяет и совокупность действий, которые можно выполнять над данными определенного типа. Например, с числами можно выполнять большой набор действий (сложение, вычитание, умножение, деление и т. д.), а строки можно только складывать.

Все типы данных, которые используются в системе Turbo Pascal, можно разделить на группы:

- скалярные (или простые);
- структурированные (составные).

Скалярные типы в свою очередь подразделяются на *стандартные* и *пользовательские*. Стандартные типы заложены в систему Turbo Pascal и к ним относятся: целочисленные, вещественные, символьные, логические (булевы) и указатели. Категория структурированных типов (строки, массивы, множества, записи и файлы) рассмотрена в последующих главах.

Тип данных очень важен при выделении памяти под переменные. Это связано с тем, что каждому типу данных соответствует определенное число байтов памяти компьютера.

Рассмотрим, что представляют собой стандартные типы данных в системе Turbo Pascal. В примере листинга 1.2 мы уже встретились с одним из типов данных.

При организации вычислений программисты наиболее часто используют целочисленные типы данных, перечень которых представлен в табл. 1.1. Здесь указаны минимальные и максимальные значения для каждого типа данных, а также количество отводимых для них байтов в памяти компьютера. Действия над целыми числами указанных типов определены лишь тогда, когда исходные данные (операнды) и результат лежат в заданном интервале. В противном случае возникает *переполнение* [1]. Приведем пример описания нескольких переменных целочисленного типа:

```
var
    A: integer;
    B: longint;
    C: byte;
```

Как известно [1, 2], числа в вычислительной системе представлены в двоичной системе счисления и на каждое число в оперативной памяти отводится определенное количество разрядов (битов). В этом плане прокомментируем используемые в Паскале целые числовые типы данных. Так, наименьшее число разрядов (8) имеют типы **shortint** и **byte**. В случае использования **shortint** минимальное число равно -128, а максимальное 127. Такой тип данных позволяет работать только с короткими знаковыми числами. Если же воспользоваться типом **byte**, то нам предоставляется возможность работы только с беззнаковыми числами (но максимальное значение в этом случае будет больше).

Для вычислений наиболее часто используется тип **integer** (под данное отводится 16 двоичных разрядов), в котором минимальное число равно -2^{15}, а самое большое положительное равняется $2^{15}-1$. Если требуется отразить бóльшие целые числа, то следует воспользоваться типом **longint**. В этом случае числа могут принимать значения в интервале от -2^{31} до $2^{31}-1$.

Не все числа, с которыми приходится работать, являются целыми. В большинстве вычислений используются дробные числа. С точки зрения математики таких чисел бесконечно много. Что же касается вычислительной системы, то их также много, но общее количество в этом случае конечно.

Дробные числа, кроме целой части, включают еще и дробную составляющую, отделяемую от целой разделителем (в качестве разделителя используется точка). Такие числа принято называть *вещественными*, в Паскале существует несколько вещественных типов для числовых данных.

Примечание

Вещественные типы данных занимают от 4-х до 10-ти байтов.

Для записи вещественных чисел в языке Паскаль можно использовать *обычный* и *экспоненциальный* формат записи. Приведем в качестве примера несколько вещественных чисел в знакомом обычном формате: 4.5; -9.778; 90.678.

При использовании экспоненциального формата в записи числа выделяются две компоненты: мантисса и порядок. В этом случае число представляется в виде мантиссы, умноженной на 10 в определенной степени (степень соответствует порядку). Приведем пример нескольких чисел в таком формате:

- $4700 = 4.7 \cdot 10^3$, что в указанном формате выглядит 4.7000000E+03.
- $-25000 = -2.5 \cdot 10^4$ или в указанном формате -2.5000000E+04.
- $-0.0005 = -5 \cdot 10^{-4}$ или -5.0000000E-04.

Подчеркнем, что в языке Паскаль при записи вещественных чисел основание 10 заменяется на букву E, после которой размещается значение порядка. Если вы уви-

дите подобные числа в результате выполненных вычислений, то не удивляйтесь. В табл. 1.2 приведены варианты вещественных типов данных в системе Turbo Pascal.

Таблица 1.1. *Целочисленные типы данных в Turbo Pascal*

Тип	Диапазон значений	Размер в байтах
byte	0...255	1
word	0...65535	2
integer	-32768...32767	2
shortint	-128...127	1
longint	-2147483648...2147483647	4

Таблица 1.2. *Вещественные типы данных в Turbo Pascal*

Тип	Диапазон значений	Мантисса	Размер в байтах
real	2.9E-39...1.7E38	11—12	6
single	1.5E-45...3.4E38	7—8	4
double	5.0E-324...1.7E308	15—16	8
extented	3.4E-49321...1E4932	19—20	10

Приведем пример описания переменных вещественного типа в программе на языке Паскаль:

```
var K: real;
        W: double;
```

После такого определения уже можно в области исполнительных операторов программы работать с переменными K и W. Например, выполнить следующее присвоение значений:

```
K:=99.5;   W:=9.157;
```

Поскольку размер памяти, отводимой под мантиссу и порядок, ограничен, то вещественные числа всегда представляются в памяти компьютера с некоторой погрешностью. Например, простейшая вещественная дробь 2/3 дает в десятичном представлении 0,666666... и независимо от размера памяти, выделяемой для числа, невозможно хранить все цифры его дробной части.

Вещественные числа в вычислительной системе представлены приближенно и арифметические действия над ними также выполняются приближенно.

Для представления одиночных символов (букв, цифр и ряда других символов) в Паскале предназначен *символьный* тип данных. Для каждой переменной данного

типа в памяти отводится 1 байт, а для описания используется ключевое слово **char**. Приведем пример описания набора переменных символьного типа:

```
var
   X1,X2,X3:char;
```

После этого с ними можно работать, в частности, присвоить созданным переменным значения. Для этого следует использовать одинарные кавычки (необходимо поместить символ в одинарные кавычки):

```
X1:= 'A';
X2:= '1';
X3:= '+';
```

В результате в переменную X1 будет занесен код английской буквы A. Как уже говорилось, каждому символу соответствует свой код, который занимает 8 бит или 1 байт. И вместо одинарных кавычек при обозначении символа можно использовать знак #, за которым следует поместить код символа. Например, можно написать #65 вместо английской буквы A. Однако использование числового кода вместо указания самого символа разумно использовать только для специальных (служебных) символов, которые не имеют экранного представления. Например:

- #07 — подача короткого звукового сигнала;
- #08 — смещение курсора на одну позицию назад;
- #10 — горизонтальная табуляция;
- #13 — возврат каретки или перевод строки (соответствует нажатию клавиши <Enter>);
- #26 — конец файла (соответствует комбинации клавиш <Ctrl>+<Z>);
- #32 — пробел.

Кроме уже упомянутых типов данных, в Паскале используется *логический* (другое название — булевый тип). Ключевое слово для указания данного типа — **boolean**. Данные этого типа могут принимать только два возможных значения: **true** (истина) и **false** (ложь). Приведем пример описания переменных такого типа в программе на языке Паскаль:

```
var
   A1,A2,A3:boolean;
```

Дополнительно к стандартным типам в языке Паскаль можно использовать пользовательские типы данных. К ним относятся:

- интервальный (ограниченный) тип или диапазон;
- перечислимый тип.

В целом можно сказать, что данные типы используются для ограничения количества значений, принимаемых переменными. Это весьма полезно для исключения возможных ошибок при вводе или присвоении значений.

Интервальный тип данных позволяет задать две константы, которые определяют границы изменения переменных данного типа. Значение первой константы должно

быть меньше значения второй. Сами они должны быть целочисленными или символьными. В записи указания типа константы должны быть разделены следующими друг за другом точками. Например, описание нескольких переменных такого типа в программе на языке Паскаль может выглядеть так:

```
var
   B1,B2,B3: -7..4;
```

Перечислимый тип данных задается просто набором значений, которые переменная данного типа может принимать. Приведем пример описания переменных такого типа в программе на языке Паскаль:

```
var
   B1,B2,B3: (one,two,three);
```

В данном случае каждая из трех указанных переменных может принимать только одно из трех возможных значений, которые указаны в скобках.

Система Turbo Pascal, в отличие от ряда других систем программирования, перед использованием тех или иных объектов требует их обязательного описания с указанием типа данных. В итоге это повышает качество программ и застраховывает программиста от возможных ошибок. В противном случае пришлось бы находить и исправлять ошибки в процессе выполнения программы.

В Turbo Pascal кроме упомянутых типов существует механизм создания новых типов данных. После этого можно отводить память под переменные, имеющие этот новый тип данных. Раздел описания типов начинается с ключевого слова **type** и выглядит следующим образом:

```
type
  ИмяТипа = ОписаниеТипа;
```

Приведем пример описания переменных с предварительным определением перечислимого типа:

```
type
   People=(Nike,Pete,Oleg);
var
   B1,B2,B3: People;
```

Здесь создали новый тип данных — `People`.

Операции и выражения

Выражение определяет порядок выполнения действий над данными и состоит из операндов, круглых скобок и знаков операций (также в выражении могут присутствовать функции, но об этом далее). Например, выражение может быть построено так:

```
X1+X2*(Y1+Y2)
```

Здесь мы использовали имена переменных (констант), знаки сложения, умножения и круглые скобки для указания приоритетности вычислений. Разумеется, при вы-

числении выражения все объекты, которые составляют выражение, должны быть определены.

В Паскале имеются три типа выражений:

- *арифметические*, которые предназначены для арифметических действий над числами;
- *логические* — для сравнения данных;
- *символьные* — для работы с текстом.

Существуют две категории, на которые делятся все операции:

- *унарные* (действие над одним операндом), которых в Паскале немного;
- *бинарные* (действие над двумя операндами), которые составляют большинство.

Арифметические операции

В программных вычислениях наиболее широко представлены арифметические операции, которые приведены в табл. 1.3. В этой таблице большинство операций являются бинарными, за исключением трех последних унарных операций.

Таблица 1.3. *Арифметические операции в Turbo Pascal*

Операция	Действие	Тип операндов	Тип результата
+	Сложение	Целый, вещественный	Целый, вещественный
-	Вычитание	Целый, вещественный	Целый, вещественный
*	Умножение	Целый, вещественный	Целый, вещественный
/	Деление	Целый, вещественный	Целый, вещественный
div	Целочисленное деление	Целый	Целый
mod	Остаток от деления	Целый	Целый
and	Арифметическое "И"	Целый	Целый
shl	Побитовый сдвиг влево	Целый	Целый
shr	Побитовый сдвиг вправо	Целый	Целый
or	Арифметическое "ИЛИ"	Целый	Целый
xor	Арифметическое "Исключающее ИЛИ"	Целый	Целый
+	Сохранение знака (унарная операция)	Целый, вещественный	Целый, вещественный
-	Отрицание знака (унарная операция)	Целый, вещественный	Целый, вещественный
not	Арифметическое отрицание (унарная операция)	Целый	Целый

Кроме арифметических операций (табл. 1.3), в системе Turbo Pascal имеется множество стандартных арифметических процедур и функций. Они позволяют производить тригонометрические вычисления, извлекать квадратный корень из числа и т. д. О них мы еще поговорим далее в этой главе.

Некоторые из указанных в табл. 1.3 операций являются *побитовыми*, т. е. действие выполняется побитно (поразрядно) над каждым битом (разрядом) операнда (операндов).

Арифметическое "И" (and) производит логическое побитовое умножение операндов в соответствии со следующими правилами:

- `1 and 1 = 1;`
- `1 and 0 = 0;`
- `0 and 1 = 0;`
- `0 and 0 = 0.`

Здесь определены действия над каждой парой соответствующих битов операндов. Например, если мы собираемся вычислить выражение с числами в десятичной системе счисления (`5 and 11`), то на уровне компьютерных вычислений это выглядит так:

```
0000 0101  and  0000 1011 = 0000 0001.
```

Здесь исходные числа (5 и 11) записаны в десятичной системе счисления, но в компьютере они хранятся в двоичной системе [1]. И над каждой парой соответствующих битов чисел производится операция логического умножения **and**. В результате вычисления получим ответ:

```
5 and 11 = 1.
```

Арифметическое "ИЛИ" (or) производит логическое побитовое сложение операндов в соответствии со следующими правилами:

- `1 or 1 = 1;`
- `1 or 0 = 1;`
- `0 or 1 = 1;`
- `0 or 0 = 0.`

Например, если мы собираемся вычислить выражение: `17 or 3`, то на уровне компьютерных вычислений это выглядит так:

```
0001 0001  or  0000 0011 = 0001 0011.
```

В результате получим:

```
17 or 3 = 19.
```

Побитовый сдвиг **shl** приводит к сдвигу битов числа A влево на N битов:

```
A shl N.
```

Например:

```
2 shl 3 = 16,
```

в чем легко убедиться, если записать двоичное представление числа 2 и сдвинуть его разряды влево.

Примечание

В операции **shl** биты, которые выдвигаются влево из старшего разряда, пропадают.

Побитовый сдвиг вправо **shr** выполняется аналогично, но в противоположном направлении (в направлении младших разрядов). Например:

```
24 shr 2 = 6,
```

в чем легко убедиться.

Примечание

В операции **shr** биты, которые выдвигаются вправо из младшего разряда, пропадают.

Исключающее "ИЛИ" (xor) производит логическую операцию в соответствии со следующими правилами:

- `1 xor 1 = 0`;
- `1 xor 0 = 1`;
- `0 xor 1 = 1`;
- `0 xor 0 = 0`.

Например, если мы собираемся вычислить выражение: `15 xor 3` над десятичными числами, то на уровне компьютерных вычислений это выглядит так:

```
0000 1111  xor  0000 0011 = 0000 1100.
```

В результате вычисления получим: `15 xor 3 = 12`.

Арифметическое отрицание (not) выполняет побитовую инверсию (над одним операндом). Двоичные нули заменяются единицами, а единицы нулями.

Операции отношения

Кроме арифметических операций, можно выделить категорию операций *отношения* или иначе *логических операций*, которые выполняют сравнение двух операндов и определяют: истинно (true) выражение или ложно (false). Результат выражения с использованием таких операций имеет логический тип (о нем мы уже говорили). Определены следующие операции отношения:

- < — меньше;
- > — больше;
- = — равно;

- >= — больше или равно;
- <= — меньше или равно;
- <> — не равно.

Результат каждого логического выражения с использованием одной из перечисленных операций отношения может принимать одно из двух значений: `false` и `true`. Операции отношения определены над числовыми переменными, над символьными переменными и строками. Приведем несколько примеров использования операций отношения:

- 6 >= 3, результат которой равен `true`;
- 2.25 > 7.9, с результатом `false`;
- 'Pete' = 'Pets', что дает результат `false`.

Используя знаки логических операций (табл. 1.4) можно получать более сложные логические выражения. Здесь рассматривается построение логических выражений над булевыми данными, которые условно обозначены `X1` и `X2`.

Таблица 1.4. *Логические операции в Turbo Pascal*

X1	X2	Not X1	X1 and X2	X1 or X2
true	true	false	true	true
true	false	false	false	true
false	true	true	false	true
false	false	true	false	false

Приоритет операций

В выражениях, как правило, используется несколько знаков операций и здесь важно знать приоритеты операций. Если об этом не задумываться, то можно вычислить совершенно другое выражение, отличное от того, которое планировали. Перечень операций в порядке уменьшения приоритета выглядит так:

- унарные операции;
- *, /, `div`, `mod`, `and`, `shl`, `shr`;
- +, -, `or`, `xor`;
- =, <>, >, <, >=, <=.

Важно подчеркнуть, что использование скобок в выражениях позволяет менять порядок вычислений.

Ввод данных

В приведенных ранее программах (см. листинги 1.1 и 1.2) рассматривались ситуации, связанные с вводом и выводом данных. Однако в функциональном плане эти примеры были весьма ограниченны. Здесь мы дадим более полный комментарий по операторам ввода. В целом *ввод данных* представляет собой передачу исходной информации в оперативную память компьютера. В качестве устройств ввода, как правило, рассматриваются клавиатура или файл с данными.

Примечание

Использование файла с данными при вводе информации будет рассмотрено в последующих главах книги.

Для ввода информации используется оператор **read**, формат которого имеет следующий вид:

```
read (Z1,Z2,...).
```

Здесь через запятую перечисляется список переменных, которым присваиваются вводимые с клавиатуры значения. Когда в выполняемой программе управление передается на данный оператор, то производится ожидание вводимых данных (выполнение последующих операторов приостанавливается). Вводимые данные должны отделяться друг от друга пробелом, а завершить ввод следует нажатием клавиши <Enter>. После этого выполнение последующих операторов продолжится.

В Паскале имеется еще один оператор для ввода данных — **readln**. Его синтаксис такой же, как и у рассмотренного чуть ранее:

```
readln (Z1,Z2,...).
```

В функциональном плане оператор **readln** аналогичен оператору **read** и отличается тем, что после считывания последнего значения в списке вводимых данных для одного оператора **readln** данные для следующего оператора **readln** будут считываться с начала новой строки.

Часто используют вариант оператора ввода **readln** без указания параметров. Это применяется для создания паузы в программе, которая продолжается до нажатия пользователем клавиши <Enter>. Данный вариант обычно используется, когда необходимо обратить внимание пользователя на содержание экрана с информацией.

Важно отметить, что тип данных значения, которое вводится во время работы программы, должен соответствовать типу переменной, указанной в инструкции ввода. В случае несоответствия типа введенных данных типу переменной, указанной в операторе ввода, программа завершает работу и на экран выводится сообщение об ошибке ввода/вывода.

Примечание

На самом деле операторы **read** и **readln** являются *процедурами*, однако в большинстве книг по программированию на Паскале их принято называть *операторами*.

Вывод данных

Вывод данных представляет собой передачу информации на экран (либо в файл). Для вывода данных используются операторы **write** и **writeln**, формат использования которых выглядит так:

- **write**(Z1,Z2,...);
- **writeln**(Z1,Z2,...).

Здесь через запятую перечисляются выводимые данные. Это могут быть константы, переменные, выражения.

Отличие этих двух операторов друг от друга заключается только в том, что **writeln** после вывода информации производит перевод курсора на следующую строку.

Примечание

Выводимые данные не должны иметь перечислимый тип данных.

Следует сделать небольшое пояснение по формату вывода вещественных чисел. Вещественные числа автоматически выводятся в экспоненциальном формате (об этом мы уже говорили). Однако удобнее для восприятия организовать вывод информации в обычном формате (целая часть, разделитель дробной и целой части, а затем дробная часть). Для этого предусмотрен формат вывода вещественного числа:

```
write(Z:m:n),
```

где:

- Z — переменная вещественного типа;
- m — ширина поля вывода;
- n — количество знаков в дробной части числа.

Если число не помещается в m позиций, то поле вывода будет автоматически расширено до минимально необходимого.

В качестве выводимых данных могут быть выражения, включающие стандартные и пользовательские функции Turbo Pascal. Например, можно использовать такой вариант оператора вывода:

```
write(sin(3):5:3),
```

где мы воспользовались стандартной функцией **sin**(X) для вычисления синуса числа, а также отвели три позиции после точки, разделяющей целую и дробную части.

Примеры вычислений в программе

В этом разделе мы рассмотрим несколько примеров программ, которые демонстрируют использование уже рассмотренных типов данных, операции, ряд стандартных функций, а также операторы ввода/вывода. В качестве первого примера рассмот-

рим работу с целыми числами. В листинге 1.3 приведен пример программы, демонстрирующей действия, которые могут выполняться над целыми числами.

Листинг 1.3. Пример действий над целыми числами

```
program listing_1_3;
var N,M,W:integer;
begin
  N:= 25;
  M:= 6;
  W:= N*M;
  writeln('25 умножить на 6 равняется ',W);
  W:= N div M;
  writeln('Результат целого деления 25 на 6 равняется ',W);
  W:= N mod M;
  writeln('Остаток от деления 25 на 6 равняется ',W);
  W:= N + M;
  writeln('Сумма 25 и 6 равняется ',W);
  W:= N - M;
  writeln('Если от 25-ти отнять 6, то получится ',W)
end.
```

Приведенные здесь манипуляции над числами весьма несложные. Следует только обратить внимание на операцию **mod** (вычисление остатка от деления). Так, после деления одного целого числа на другое получается целая часть результата и остаток, который и представляет собой результат данной операции. Например, остаток от деления 25 на 6 равен 1 (целая часть равна 4).

Рассмотрим теперь пример использования в вычислениях вещественных чисел (листинг 1.4).

Листинг 1.4. Пример сложения вещественных чисел

```
program listing_1_4;
var Z,W,X:real;
begin
  W:=2.5;
  Z:=2.2;
  X:=W+Z;
  write (' Summa =',X)
end.
```

После запуска данной программы мы увидим на экране число: 4.7000000E+00. Это число записано в экспоненциальном формате. Для вывода значения переменной Summa в обычной форме можно предложить такую конструкцию:

```
write (' Summa =',X:6:3).
```

Здесь мы указали, что при выводе числа x необходимо отвести три десятичных разряда под его дробную часть, а под все число следует выделить шесть разрядов.

Использование блок-схем в разработках

Рассмотрим еще один пример, с которого начинается наше знакомство с построением блок-схем алгоритмов. В данном случае нам требуется написать программу, в которой производится обмен значений двух переменных. Однако если просто записать в переменную (в ячейку памяти) новое данное, то предыдущая информация, содержащаяся в переменной, пропадет. Поэтому для обмена значений переменных необходимо предварительно определить еще и третью (вспомогательную) переменную. Она понадобится для промежуточного запоминания значения одной из двух исходных переменных. В листинге 1.5 приведен пример программы, которая реализует этот обмен.

Листинг 1.5. Обмен значений двух переменных

```
program listing_1_5;
var N,M,W:integer;
begin
  N:=25;    M:=6;
  W:=N;
  N:=M;
  M:=W;
  writeln('N равняется',N);
  writeln('M равняется',M)
end.
```

Данная программа несложная, однако здесь уже уместно познакомиться с технологией описания алгоритмов с помощью блок-схем. На рис. 1.1 приведена блок-схема, иллюстрирующая алгоритм решения данной задачи.

Элементы блок-схемы, которые мы здесь использовали, составляют только часть всего ассортимента имеющихся элементов, применяемых для схематичного представления алгоритмов с помощью блок-схем. В данном случае наши действия в соответствии с рассматриваемым алгоритмом заключаются в простых последовательных шагах.

Первый и последний блок под названием "терминатор" используется для обозначения начала и завершения любой блок-схемы или подпрограммы. Прямоугольный блок с названием "процесс" предназначен для отражения вычислительного действия или присвоения значений (например, вычисления арифметического выражения). Действия, связанные с общением с внешним миром (ввод/вывод данных), в блок-схеме отражаются с помощью параллелограмма — блока "ввода/вывода". В данном случае мы используем этот элемент для организации вывода информации

на экран (значений переменных после обмена). Используемыми в этом примере блоками ассортимент элементов блок-схем не ограничивается. И в последующих разделах мы познакомимся со всеми имеющимися "строительными кирпичиками" блок-схем.

Хотя пока сделаны только первые шаги в плане алгоритмизации задач, но тем не менее у нас уже сформирована необходимая база для создания несложных разработок.

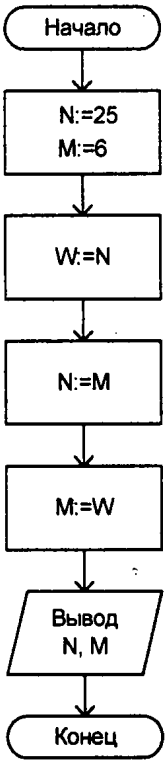


Рис. 1.1. Блок-схема к программе листинга 1.5

Использование стандартных функций

Ряд действий над числами являются стандартными, и они оформлены в виде стандартных функций системы Turbo Pascal. Например, часто используется вычисление квадрата исходного числа и получение абсолютного значения. Рассмотрим пример программы (листинг 1.6), в которой применяются функции **sqr** (для возведения в квадрат) и **abs** (для получения модуля числа).

Листинг 1.6. Использование функций возведения в квадрат и вычисления модуля

```
program listing_1_6;
var
  N,M:integer;
begin
  N:=5;    M:= -6;
  N:= sqr(N);
  M:= abs(M);
  writeln(' 5 в квадрате равняется',N);
  writeln(' Модуль числа -6  равняется',M)
end.
```

В табл. 1.5 представлены некоторые стандартные функции Turbo Pascal, которые наиболее часто используются на практике.

Таблица 1.5. *Стандартные функции Turbo Pascal*

Формат	Тип аргумента	Тип результата	Комментарий
`sin(X)`	Вещественный, целый	Вещественный	Функция синус
`cos(X)`	Вещественный, целый	Вещественный	Функция косинус
`arctan(X)`	Вещественный, целый	Вещественный	Функция арктангенс
`abs(X)`	Вещественный, целый	Вещественный, целый	Модуль числа
`exp(X)`	Вещественный, целый	Вещественный	e^x
`ln(X)`	Вещественный, целый	Вещественный	Натуральный логарифм
`sqr(X)`	Вещественный, целый	Вещественный, целый	x^2
`sqrt(X)`	Вещественный, целый	Вещественный	$\sqrt{x}$
`random`		Вещественный	Случайное число от 0 до 1
`random(n)`	Целый	Целый	Случайное число от 0 до `n-1`

В развитие темы приведем еще один пример программной разработки (листинг 1.7) с использованием вещественных чисел и обращения к стандартным функциям.

Листинг 1.7. Пример вычислительных действий над вещественными числами

```
program listing_1_7;
var
 X,Y:real;
begin
  X:=5.0;
  Y:=sqr(X);
  writeln('5.0 в квадрате равняется',Y);
  Y:=sqrt(X);
  writeln('квадратный корень из 5.0 равняется',Y);
  Y:=sin(X);
  writeln('Синус от 5.0 равняется',Y);
  Y:=exp(X);
  writeln('Экспонента от 5.0 равняется',Y)
end.
```

Следует обратить внимание на то, что названия функций `sqr` и `sqrt` достаточно похожи в плане написания. При этом первая из них обеспечивает возведение в квад-

рат, а вторая извлечение квадратного корня из числа. В рассматриваемом примере (листинг 1.7) вывод на экран не очень нагляден, удобнее для восприятия информации представить числа в более привычной для нас форме (листинг 1.8).

Листинг 1.8. Использование обычного формата при выводе вещественных чисел

```
program listing_1_8;
var
   X,Y:real;
begin
  X:=5.0;
  Y:=sqr(X);
  writeln('5.0 в квадрате равняется',Y:6:3);
  Y:=sqrt(X);
  writeln('Квадратный корень из 5.0 равняется',Y:6:3);
  Y:=sin(X);
  writeln('Синус от 5.0 равняется',Y:6:3);
  Y:=exp(X);
  writeln('Экспонента от 5.0 равняется',Y:6:3)
end.
```

Приведем еще несколько примеров арифметических выражений. Например, если необходимо возвести переменную X в пятую степень, то программная конструкция может быть записана как `X*X*X*X*X` или `sqr(X)*sqr(X)*X`. Можно также использовать еще и такое выражение:

```
sqr(sqr(X))*X.
```

Последний вариант показывает, что результаты, выдаваемые одними функциями, могут быть аргументами других.

Если же необходимо вычислить выражение $\sin^2 X$, то на Паскале оно выглядит так: `Z:=sqr(sin(X))`. При вычислении $\mathrm{tg}^2 X$ трудность заключается в том, что функции тангенса в Паскале нет. Но в этом случае можно воспользоваться представлением тангенса в виде отношения синуса к косинусу `Z:=sqr(sin(X)/cos(X))`.

Как правило, в программах встречаются сложные выражения с многочисленными действиями и использованием разнообразных функций. В этом случае важно знание определенных правил и приоритета операций, которые предполагают правильную очередность выполнения всех операций в выражении, т. е. то, что нужно выполнить сначала, а что потом. В системе Turbo Pascal имеются такие правила:

- действия над переменными, расположенными в скобках, выполняются в первую очередь;
- после вычисления значений внутри всех скобок вычисляются все функции;
- после функций выполняются умножение и деление (они имеют одинаковый приоритет);

- далее выполняются сложение и вычитание;
- операции с одинаковым приоритетом выполняются слева направо.

Примечание

Практическая рекомендация: если есть какие-то сомнения в приоритетах операций, то рекомендуется пользоваться скобками.

Действия с данными разных типов

В практических ситуациях часто требуется присваивать переменным одного типа значения другого типа. Например, применительно к числовым данным можно переменной вещественного типа (`w`) присвоить целочисленное значение (здесь считаем, что переменная `K` является целой):

```
W:= K+5;
```

В этом случае преобразование вещественного типа в целочисленный производится автоматически. Обратное преобразование в Паскале невозможно: переменной целочисленного типа нельзя присвоить вещественное значение. Для преобразования в подобном случае следует применять стандартные функции:

- **`trunc`** — отбрасывает дробную часть;
- **`round`** — производит округление до ближайшего целого.

В листинге 1.9 приведена программа, которая реализует ввод с клавиатуры вещественного числа и демонстрирует использование данных функций для преобразования в целое число.

Листинг 1.9. Преобразование вещественного числа в целое

```
program listing_1_9;
var
    N:integer;
    S:real;
begin
  writeln (' Введите число с дробной частью');
  readln(S);
  N:= trunc(S);
  writeln (' Преобразование в целое с отбрасыванием дробной части=',N);
  N:= round(S);
  writeln (' Преобразование в целое с округлением =',N)
end.
```

Часто в выражениях могут совместно встречаться переменные как целого, так и вещественного типа. В этом случае возникает проблема их совмещения. Если при выполнении сложения, деления или умножения хотя бы один аргумент выражения имеет тип **`real`**, то и результат выражения также имеет тип **`real`**. В листинге 1.10

приведен пример выполнения суммирования целого и вещественного числа. В этом случае результат получается правильный (система самостоятельно переводит целое число к вещественному типу).

Листинг 1.10. Использование в вычислениях переменных разных типов

```
program listing_1_10;
var
    N:real;
    L:integer;
begin
  N:=5.9;
  L:=7;
  N:=N+L;
  writeln(' N =',N);
end.
```

Если же мы попробуем результат вычисления рассматриваемого выражения присвоить целочисленной переменной, то система проинформирует нас об ошибке. Так, программа в листинге 1.11 не будет выполнена, а система выдаст на экран диагностическое сообщение об ошибке.

Листинг 1.11. Ошибочное присвоение вещественного значения целой переменной

```
program listing_1_11;
var
   N:real;
   L:integer;
begin
  N:=5.9;
  L:=7;
  L:=N+L;
  writeln(' L =',L);
end.
```

Если необходимо вещественный результат записать в переменную целого типа, то следует воспользоваться знакомыми функциями **trunc** или **round**. В применении к программе, приведенной в листинге 1.11, первая функция

```
L:=trunc(N+L);
```

вернет значение 12, а вторая

```
L:=round(N+L);
```

вернет значение 13.

Константы

Иногда в программе необходимо использовать данные, которые в ходе ее выполнения не изменяются. Для работы с подобной информацией в Паскале предусмотрена такая категория, как *константы*. Часто они определяются (имя и значение) в начале программы и далее просто используются. Пример, приведенный в листинге 1.12, дает представление о том, как в плане синтаксиса реализуется определение констант в тексте программы. Здесь мы ввели константу, соответствующую числу π, и далее использовали ее при вычислении длины окружности.

Листинг 1.12. Пример использования константы

```
program listing_1_12;
const
    Chislo_PI=3.14159265;
var
    S,R:real;
begin
  writeln(' Введите значение радиуса ');
  readln(R);
  S:=2 * Chislo_PI * R;
  writeln(' S =',S)
end.
```

Примечание

В системе Turbo Pascal имеется функция `pi`, которая выдает в качестве своего значения число π.

Кроме числовых констант существуют и другие их виды:

- *логические* константы служат для проверки истинности или ложности некоторых условий в программе и могут принимать только одно из двух значений: `true` или `false`;
- *символьные* константы могут принимать значение любого печатаемого символа и записываются как символ, заключенный в *апострофы* (одинарные кавычки).

Если требуется записать сам символ апострофа как символьную константу, то внутри внешних апострофов он удваивается: `''''`.

К символьным также относятся константы вида `#X`, где `X` — числовое значение от 0 до 255 включительно, представляющее собой десятичный *ASCII-код* символа.

Кроме того, можно использовать и *строковые константы* — это любые последовательности символов, заключенные в апострофы. Как правило, строковые константы служат для записи приглашений к вводу данных, выдаваемых программой, вывода диагностических сообщений и т. д. Например, строковые константы можно записать так:

```
'Введите значение X:'
'Ответ='
```

Именованные константы перечисляются в разделе описаний программы оператором следующего вида:

```
const
 Имя1=Значение1;
 Имя2=Значение2;
 ...
ИмяN=ЗначениеN;
```

В программе листинга 1.12 мы встретились с подобными константами. Ключевое слово **const** определяет начало раздела описаний именованных констант.

Работа с символами

Кроме чисел, другим распространенным видом данных являются символы и наборы символов. В данном разделе мы об этом и поговорим. В Паскале для переменных, предназначенных для размещения одиночных символов, существует тип **char**.

За каждым символом закреплен определенный код из восьми двоичных битов, который называется *ASCII-кодом* (американский стандартный код обмена информацией). Разумеется, цифры также имеют представление в виде кода. Например, цифра 0 кодируется двоичной последовательностью 00110000, а цифра 9 — двоичной последовательностью 00111001. Учитывая, что общее количество различных комбинаций из 8-ми битов равно 256, то, соответственно, существует 256 ASCII-кодов. В листинге 1.13 представлен пример, который демонстрирует использование двух функций, предназначенных для работы с символами.

Листинг 1.13. Пример действий над символами

```
program listing_1_13;
var
    Z,W: char;
    X:integer;
begin
  W:='L';
  X:=ord(W);
  writeln ('ASCII-код символа L равен ',X);
  Z:=chr(X);
  write ('Simvol =',Z)
end.
```

Здесь мы использовали ключевое слово **char** для описания переменных, предназначенных для размещения в них символов. Далее, используя уже знакомые символы кавычек ('), мы поместили в переменную W код символа L.

Для получения кода определенного символа используется стандартная функция **ord**, которая возвращает числовое значение ASCII-кода символа, задаваемого в ка-

честве параметра. Другая функция **chr** выполняет обратное преобразование (формирует символ по его ASCII-коду).

Рассмотрим еще один пример, связанный с вводом символов с клавиатуры (листинг 1.14). В этом случае реакция программы на ввод набора символов (ввод всего набора символов должен завершаться нажатием клавиши <Enter>) заключается в выводе на экран только первого символа из введенной последовательности.

Листинг 1.14. Вывод только первого символа из введенной последовательности

```
program listing_1_14;
var
   Z: char;
begin
  write ('Ввести набор символов ');
  readln(Z);
  write ('Simvol =', Z)
end.
```

Еще один пример приведен в листинге 1.15. Здесь при вводе строки данных мы выделяем третий, пятый и седьмой символы. Эти символы в итоге отображаются на экране.

Листинг 1.15. Выделение нескольких символов из строки

```
program listing_1_15;
var
   Z1,Z2,Z3,Z: char;
begin
  write ('Ввести набор символов ');
  readln(Z, Z, Z1, Z, Z2, Z, Z3);
  writeln ('Nabor =',Z1,Z2,Z3)
end.
```

Использование логического типа данных

В начале этой главы мы уже говорили о булевых переменных. Для их описания в языке Паскаль введен тип данных **boolean**. Переменные типа **boolean** используются в логических условиях. Пример на эту тему приведен в листинге 1.16. Здесь переменной w присваивается результат операции: z<5. Фактически эта операция заключается в проверке: истинно указанное условие или ложно? Если введенное с клавиатуры число оказывается меньше пяти, то результат операции — **true**, в противном случае результат — **false**.

Листинг 1.16. Пример работы с булевыми переменными

```
program listing_1_16;
var
   Z:integer;
   W:boolean;
begin
  write ('Ввести целое число ');
  readln(Z);
  W:=Z<5;
  writeln(W)
end.
```

Часто условие является комбинацией нескольких простых условий. Здесь следует использовать операции конъюнкции и дизъюнкции. В Паскале конъюнкция обозначается **and**, а дизъюнкция — **or**. Другие функции используются реже:

- **not** — отрицание;
- **xor** — сложение по модулю два.

В листинге 1.17 приведен пример комбинации двух условий (проверка одновременного выполнения обоих условий).

Листинг 1.17. Пример использования конъюнкции при формировании условия

```
program listing_1_17;
var
   Z,X:integer;
   W:boolean;
begin
  write ('Ввести первое число ');
  readln(Z);
  write ('Ввести второе число ');
  readln(X);
  W:=(Z<5) and (X>2) ;
  writeln(W)
end.
```

При объединении логических операций нужно помнить, что они подчинены правилам приоритета. Самый высокий приоритет у операции **not** (отрицание), следующий у операции **and**, а самый низший у операций **or** и **xor**. Последние две операции имеют одинаковый приоритет, а значит, при отсутствии скобок выполняются слева направо.

Примечание

Приоритет любой операции сравнения ниже, чем любой логической операции. Это означает, что при объединении сравнений при помощи логических операций каждое

сравнение необходимо взять в скобки. Например, условие нахождения числа Z в интервале от 2 до 5 должно быть записано так: `(2<=Z) and (Z<=5)`.

Примеры

ПРИМЕР 1.1

Необходимо вычислить сумму длин сторон прямоугольника (периметр прямоугольника). С клавиатуры вводятся значения двух соседних сторон рассматриваемого прямоугольника. Результат вычисления периметра следует вывести на экран. Текст необходимой программной разработки для решения данной задачи приведен в листинге 1.18. Здесь используются два оператора ввода, в диалоге с которыми мы должны ввести два числа, обозначающие соседние стороны прямоугольника (для них в программе отводятся переменные A и B). После этого вычисляется удвоенная сумма сторон, которая и выводится на экран. Для того чтобы после вычисления результат оставался на экране, мы воспользовались дополнительно функцией **readln**.

Листинг 1.18. Вычисление периметра прямоугольника

```
program listing_1_18;
var
   A,B,S:real;
begin
  write(' Ввод A =');
  readln(A);
  writeln(' Ввод B =');
  readln(B);
  S:=2*(A+B);
  write(' S =',S);
  readln
end.
```

Учитывая, что в представленной программе используются вещественные числа, вывод удобнее выполнить не в экспоненциальной, а в обычной форме:

```
write(' S =',S:4:1).
```

ПРИМЕР 1.2

Необходимо вычислить синус и косинус угла, введенного с клавиатуры. Будем считать, что вводимое с клавиатуры значение угла измеряется в градусах. Однако стандартные тригонометрические функции системы Turbo Pascal требуют указания аргумента в радианах. В листинге 1.19 представлена необходимая программа, где мы воспользовались уже знакомой функцией, которая в качестве своего значения выдает константу — число π. Далее производится пересчет угла из градусов в радианы и выполняются вычисления тригонометрических функций.

Листинг 1.19. Вычисление тригонометрических функций

```
program listing_1_19;
var
   Y,C,S:real;
begin
  write(' Ввод Y =');
  read(Y);
  Y:=Y*pi/180;
  C:=cos(Y);
  writeln(' Косинус равен =',C);
  S:=sin(Y);
  writeln(' Синус равен =',S);
end.
```

ПРИМЕР 1.3

Требуется написать программу, которая будет вычислять стоимость покупки в магазине канцтоваров. Для определенности будем считать, что в покупаемый набор входят тетради и ручки (и тетради и ручки одного вида). С клавиатуры вводятся:

- стоимость одной тетради;
- стоимость одной ручки;
- количество тетрадей и количество ручек.

После ввода необходимой информации в программе (листинг 1.20) выполняются арифметические действия, а результат отображается на экране.

Листинг 1.20. Вычисление стоимости покупки канцтоваров

```
program listing_1_20;
var
    Ct,Cr,Sum:real;
    Nt,Nr:integer;
begin
  write(' Ввод цены тетради =');
  readln(Ct);
  write(' Ввод цены ручки =');
  readln(Cr);
  write(' Ввод числа тетрадей =');
  readln(Nt);
  write (' Ввод числа ручек =');
  readln(Nr);
  Sum:=Nt*Ct+Nr*Cr;
  writeln(' Сумма покупки =',Sum:5:2);
end.
```

ПРИМЕР 1.4

Необходимо выполнить вычисление площади прямоугольного треугольника. При этом два катета треугольника вводятся с клавиатуры.

Здесь мы должны вспомнить формулу для вычисления площади прямоугольного треугольника:

$$S = \frac{a \cdot b}{2}, \tag{1.1}$$

где a и b обозначают длины катетов. В листинге 1.21 представлена необходимая программа, в которой после ввода двух параметров вычисляется площадь треугольника.

Листинг 1.21. Вычисление площади прямоугольного треугольника

```
program listing_1_21;
var
  A,B,S:real;
begin
  write(' Ввод значения 1-го катета ');
  readln(A);
  write(' Ввод значения 2-го катета ');
  read(B);
  S:=A*B/2;
  write(' Площадь равна =',S:5:2);
end.
```

ПРИМЕР 1.5

Требуется написать программу, которая должна рассчитать время поездки из пункта A в пункт C. При этом необходимо сначала доехать по шоссе из A в B, а затем по грунтовой дороге из B в пункт C. Длины путей (в километрах) из A в B и из B в C вводятся с клавиатуры. Также с клавиатуры вводятся скорости по шоссе и по грунтовой дороге (единица измерения для скоростей — км/час). Время в пути необходимо вычислить с точностью до часа (следует округлить вычисленное значение).

Программа должна выполнить арифметические действия и отобразить результат (время в пути из A в C) на экране. В листинге 1.22 представлена необходимая программа.

Листинг 1.22. Вычисление времени в пути

```
program listing_1_22;
var
    Dab,Dbc,Vab,Vbc:real;
    T:integer;
```

```
 begin
  write(' Ввод пути из A в B ');
  readln(Dab);
  write(' Ввод пути из B в C ');
  readln(Dbc);
  write(' Ввод скорости по шоссе ');
  readln(Vab);
  write(' Ввод скорости по грунтовой дороге ');
  readln(Vbc);
  T:=round(Dab/Vab+Dbc/Vbc);
  writeln(' Время в пути =',T);
end.
```

ПРИМЕР 1.6

Необходимо написать программу, которая должна выполнить преобразование времени, выраженного в секундах, в соответствующее значение, представленное в минутах и секундах. После выполнения необходимых преобразований результат следует отобразить на экране. В листинге 1.23 представлена программа, решающая поставленную задачу. Для получения количества минут мы воспользовались целочисленным делением исходного значения на 60. В данном случае остаток от целочисленного деления является числом секунд, добавляемым к вычисленным минутам.

Листинг 1.23. Преобразование значения времени в минуты и секунды

```
program listing_1_23;
var
   Vremya,Min,Sec:integer;
begin
  write(' Ввод значения времени в секундах =');
  readln(Vremya);
  Min:=Vremya div 60;
  Sec:=Vremya mod 60;
  write(Min,' мин  ');
  write(Sec,' сек');
end.
```

ПРИМЕР 1.7

Требуется написать программу, которая должна выполнить преобразование времени, выраженного в секундах, в соответствующее значение, только представленное в часах, минутах и секундах. Программа (листинг 1.24) получается лишь немного сложнее предыдущего примера.

Листинг 1.24. Преобразование значения времени в часы, минуты и секунды

```
program listing_1_24;
var
   Vremya,Chas,Min,Sec:integer;
begin
  write(' Ввод значения времени в секундах =');
  readln(Vremya);
  Chas:=Vremya div 3600;
  Min:=(Vremya mod 3600)div 60;
  Sec:= Vremya - Chas*3600 - Min*60;
  write(Chas,' час  ');
  write(Min,' мин  ');
  write(Sec,' сек ');
end.
```

ПРИМЕР 1.8

С клавиатуры вводится двузначное число. Необходимо сформировать другое двузначное число, в котором цифры исходного числа будут переставлены. Например, если исходное число 57, то число с переставленными цифрами равно 75. В листинге 1.25 приведена соответствующая программа. Здесь мы воспользовались уже знакомыми операциями **div** и **mod** для выделения первой и второй цифры в числе. После этого цифра, присутствующая в разряде единиц, умножается на 10 (становится в новом числе цифрой в разряде десятков) и складывается с другой.

Листинг 1.25. Перестановка цифр в двузначном числе

```
program listing_1_25;
var
   A:integer;
begin
  write(' Введите двузначное число ');
  readln(A);
  A:=(A div 10) + (A mod 10)*10;
  write(' Число с перестановкой цифр равно',A);
end.
```

ПРИМЕР 1.9

С клавиатуры вводится трехзначное число. Необходимо подсчитать сумму цифр этого числа. Например, если исходное число 257, то число с просуммированными цифрами равно 14. В листинге 1.26 приведена программная разработка, решающая данную задачу. Переменные B, C и D мы отвели, соответственно, для первой, третьей и второй цифры исходного числа.

Листинг 1.26. Суммирование цифр в трехзначном числе

```
program listing_1_26;
var
   A,B,C,D:integer;
 begin
  write(' Введите трехзначное число ');
  readln(A);
  B:=A mod 10;
  C:=A div 100;
  D:= (A mod 100) div 10;
  write(' Сумма цифр в числе равна ',B+C+D);
end.
```

ПРИМЕР 1.10

С клавиатуры вводится четырехзначное число. Необходимо подсчитать сумму цифр этого числа. Например, если исходное число 1257, то число с просуммированными цифрами равно 15. В листинге 1.27 приведена программа, которая является фактически чуть более усложненным вариантом предыдущей разработки. Назначение переменных выглядит так:

- B — для цифры в разряде единиц;
- C — для цифры в разряде тысяч;
- D — для цифры в разряде десятков;
- E — для цифры в разряде сотен.

Листинг 1.27. Суммирование цифр в четырехзначном числе

```
program listing_1_27;
var
   A,B,C,D,E:integer;
begin
  write(' Введите четырехзначное число ');
  readln(A);
  B:= A mod 10;
  C:= A div 1000;
  D:= (A mod 100) div 10;
  E:= (A mod 1000) div 100;
  write(' Сумма цифр в числе равна ',B+C+D+E);
end.
```

ПРИМЕР 1.11

С клавиатуры вводится вещественное число. Необходимо определить первую цифру дробной части этого числа. Например, если исходное число 1257.237, то такая

цифра равна 2. В листинге 1.28 приведена программа, которая реализует поставленную задачу. Мы воспользовались умножением исходного числа на 10 для перевода первой цифры из дробной части в разряд единиц целой части. После этого с помощью стандартной функции `trunc` отбрасываем дробную часть. Далее осталось сформировать только остаток от целочисленного деления полученного результата на 10.

Листинг 1.28. Выделение первой цифры дробной части числа

```
program listing_1_28;
var
    A:integer;
    B:real;
 begin
  write(' Введите вещественное число ');
  readln(B);
  A:=trunc(B*10) mod 10;
  write(' Первая цифра дробной части равна ', A);
end.
```

ПРИМЕР 1.12

С клавиатуры вводится вещественное число. Необходимо найти сумму первых двух (старших) цифр дробной части числа. Например, если исходное число 1257.237, то сумма двух первых цифр дробной части равна 5. В листинге 1.29 приведена необходимая программа. Аналогично шагам предыдущего примера мы сначала переносим первые две цифры из дробной части в целую. После этого отбрасываем оставшуюся дробную часть и формируем двузначное число. На завершающем этапе следует выделение цифр целого числа.

Листинг 1.29. Выделение двух первых цифр дробной части числа

```
program listing_1_29;
var
    A,C,D:integer;
    B:real;
 begin
  write(' Введите вещественное число ');
  readln(B);
  A:=trunc(B*100) mod 100;
  C:=A mod 10;
  D:= A div 10;
  write(' Сумма первых двух цифр дробной части числа ', C+D);
end.
```

ПРИМЕР 1.13

С клавиатуры вводится четырехзначное число. Необходимо в этом числе переставить первую и вторую цифру (поменять цифру в разряде единиц и цифру в разряде десятков). В листинге 1.30 приведена необходимая программа, в которой основные используемые приемы нам уже знакомы. Так, для выделения двух младших разрядов в числе A мы воспользовались двумя операторами:

```
B:= A mod 10;
C:=(A mod 100) div 10;
```

После этого в исходном числе необходимо оставить цифры только в разрядах сотен и тысяч (получить исходное число без цифр в разрядах десятков и единиц). Таким образом, в результате мы получили составляющие искомого числа. На завершающем этапе эти составляющие осталось объединить в единое целое:

```
E:= D + C + B*10.
```

Листинг 1.30. Перестановка цифр в числе

```
program listing_1_30;
var
   A,B,C,D,E:integer;
begin
  write(' Введите четырехзначное число ');
  readln(A);
  B:=A mod 10;
  C:=(A mod 100)div 10;
  D:= (A div 100)*100;
  E:=  D + C +B*10;
  write(' Число с переставленными цифрами равно ',E);
end.
```

ПРИМЕР 1.14

С клавиатуры вводятся координаты точки (X, Y), расположенной на плоскости. Необходимо программно определить, попадает ли точка в область, определяемую следующими соотношениями:

X>5 и Y>7.

Если точка попадает, то на экран необходимо выдать информирующее сообщение. Соответственно, если точка не попала в указанную область на экране, то это также должно быть отражено соответствующим сообщением.

В разработке (листинг 1.31) мы воспользовались переменной логического (булевого) типа.

Листинг 1.31. Проверка принадлежности точки определенной области

```
program listing_1_31;
var
    X,Y:real;
    W:boolean;
begin
  write(' Введите координаты точки ');
  readln(X,Y);
  W:=(X>5) and (Y>7) ;
  writeln(W)
end.
```

ПРИМЕР 1.15

С клавиатуры вводятся координаты точки (X, Y), расположенной на плоскости. Необходимо программно определить: попадает ли точка в круг с центром в начале координат и радиусом 1?

Здесь мы должны вычислить расстояние от начала координат до точки на плоскости. При этом нам понадобится функция возведения в квадрат. Разработка, реализующая поставленную задачу, приведена в листинге 1.32.

Листинг 1.32. Проверка принадлежности точки кругу

```
program listing_1_32;
var
    X,Y,Z:real;
    W:boolean;
begin
  write(' Введите координаты точки ');
  readln(X,Y);
  Z:=Sqr(X)+Sqr(Y);
  W:=(Z<=1);
  writeln(W)
end.
```

ПРИМЕР 1.16

С клавиатуры вводятся координаты точки (X, Y), расположенной на плоскости. Необходимо программно определить: попадает ли точка в круг радиуса 5 с центром в точке X0 и Y0. Предполагается, что координаты центра круга также вводятся с клавиатуры. Разработка, реализующая поставленную задачу, приведена в листинге 1.33.

Листинг 1.33. Проверка принадлежности точки кругу

```
program listing_1_33;
var
    X,Y,X0,Y0,Z:real;
    W:boolean;
begin
   write(' Введите координаты точки ');
  readln(X,Y);
   write(' Введите координаты центра круга ');
  readln(X0,Y0);
  Z:=sqrt(sqr(X-X0)+sqr(Y-Y0));
  W:=(Z<=5);
  writeln(W)
end.
```

ПРИМЕР 1.17

На плоскости расположен круг радиуса R с центром в точке X0 и Y0. Указанные параметры вводятся с клавиатуры. Необходимо программно определить: попадает ли центр координат в данный круг. Программа, решающая поставленную задачу, приведена в листинге 1.34.

Листинг 1.34. Проверка принадлежности центра координат кругу

```
program listing_1_34;
var
   X0,Y0,R,Z:real;
   W:boolean;
begin
  write(' Введите координаты центра круга ');
  readln(X0,Y0);
  write(' Введите радиус круга ');
  readln(R);
  Z:=sqrt(sqr(X0)+sqr(Y0));
  W:=(Z<=R);
  writeln(W)
end.
```

ПРИМЕР 1.18

В плоскости расположены две точки. Координаты точек вводятся с клавиатуры. Необходимо программно определить расстояние между точками. Для решения задачи здесь необходимо воспользоваться теоремой Пифагора. Программа, решающая поставленную задачу, приведена в листинге 1.35.

Листинг 1.35. Вычисление расстояния между точками

```
program listing_1_35;
var
   X1,Y1,X2,Y2,Z:real;
begin
  write(' Введите координаты первой точки ');
  readln(X1,Y1);
  write(' Введите координаты второй точки ');
  readln(X2,Y2);
  Z:=sqrt(sqr(X1-X2)+sqr(Y1-Y2));
  writeln(Z:7:2)
end.
```

ПРИМЕР 1.19

В плоскости построен треугольник. Координаты вершин треугольника вводятся с клавиатуры. Необходимо программно определить периметр треугольника. Текст программной разработки приведен в листинге 1.36.

Листинг 1.36. Вычисление периметра треугольника

```
program listing_1_36;
var
  X1,Y1,X2,Y2,X3,Y3,Z:real;
begin
   write(' Введите координаты первой вершины ');
  readln(X1,Y1);
   write(' Введите координаты второй вершины ');
  readln(X2,Y2);
  write(' Введите координаты третьей вершины ');
   readln(X3,Y3);
   Z:=sqrt(sqr(X1-X2)+sqr(Y1-Y2));
   Z:=Z+sqrt(sqr(X2-X3)+sqr(Y2-Y3));
   Z:=Z+sqrt(sqr(X1-X3)+sqr(Y1-Y3));
   writeln(Z:7:2)
end.
```

ПРИМЕР 1.20

С клавиатуры вводится трехзначное число. Необходимо определить, есть ли в этом числе одинаковые цифры. Например, если исходное число 525, то программа должна выдать ответ "true". В листинге 1.37 приведена программа, решающая поставленную задачу. Назначение переменных таково:

- B — для цифры в разряде единиц;
- C — для цифры в разряде десятков;
- D — для цифры в разряде сотен.

Листинг 1.37. Анализ наличия одинаковых цифр в трехзначном числе

```
program listing_1_37;
var
    A,B,C,D:integer;
    W:boolean;
 begin
  write('Введите трехзначное число ');
  readln(A);
  B:= A mod 10;
  C:= (A mod 100) div 10;
  D:= A div 100;
  W:= (B=C) or (B=D) or (C=D);
  writeln(W)
end.
```

ПРИМЕР 1.21

С клавиатуры вводятся три различных числа. Необходимо сделать вывод о том, какое из чисел является максимальным. В листинге 1.38 приведена программа, решающая поставленную задачу.

Листинг 1.38. Поиск максимального числа

```
program listing_1_38;
var
    A,B,C:integer;
    W1,W2,W3:boolean;
 begin
  write(' Введите три числа ');
  readln(A,B,C);
   W1:= (A>B) and  (A>C);
   W2:= (B>A) and  (B>C);
   W3:= (C>A) and  (C>B);
  writeln(' A максимальное - ',W1);
  writeln(' B максимальное - ',W2);
  writeln(' C максимальное - ',W3);
end.
```

Если не включать условие единственности максимального числа, то фрагмент листинга 1.38 изменится на следующий:

```
   W1:= (A>=B) and  (A>=C);
   W2:= (B>=A) and  (B>=C);
   W3:= (C>=A) and  (C>=B);
```

ПРИМЕР 1.22

С клавиатуры вводятся три числа. Необходимо сделать вывод о том, все ли они равны между собой или нет. В листинге 1.39 приведена программа, решающая поставленную задачу.

Листинг 1.39. Проверка чисел на равенство

```
program listing_1_39;
var
    A,B,C:integer;
    W:boolean;
 begin
   write(' Введите три числа ');
   readln(A,B,C);
   W:= (A=B) and  (A=C);
   writeln(' Все числа одинаковые — ',W);
end.
```

Типовые задачи и задания из ЕГЭ за 2008 — 2010 годы

В этом разделе приведены готовые листинги программ, которые вам необходимо проанализировать. Создания каких-либо программных разработок здесь не требуется. В задачах желательно найти правильные ответы без использования компьютера. В последнем разделе главы приведены правильные ответы.

ЗАДАЧА 1.1

В листинге 1.40 приведена программа с использованием действий над целыми числами. Необходимо определить значения переменных A и B, которые будут выведены на экран.

Листинг 1.40. Программа к задаче 1.1

```
program listing_1_40;
var
   A,B:integer;
begin
A:= 3 + 8*4;
B:= (A mod 10) + 14;
A:= (B mod 10) + 2;
writeln(' A =',A);
writeln(' B =',B);
end.
```

ЗАДАЧА 1.2

В листинге 1.41 приведена программа с использованием действий над целыми числами. Необходимо определить значения переменных A и B.

Листинг 1.41. Программа к задаче 1.2

```
program listing_1_41;
var
  A,B:integer;
begin
A:= 1819;
B:= (A div 100)*10 + 9;
A:= (10*B - A) mod 100;
writeln(' A =',A);
writeln(' B =',B);
end.
```

ЗАДАЧА 1.3

В листинге 1.42 приведена программа с использованием действий над целыми числами. Необходимо без выполнения ее на компьютере определить значение переменной A.

Листинг 1.42. Программа к задаче 1.3

```
program listing_1_42;
var A,B:integer;
begin
A:= 1;
B:= 9;
A:= -A + B*2;
A:= 4*A mod 10;
writeln(' A =',A);
end.
```

ЗАДАЧА 1.4

В листинге 1.43 приведен фрагмент программы. Необходимо определить, что появится на экране после ее выполнения.

Листинг 1.43. Программа к задаче 1.4

```
program listing_1_43;
var A:integer;
begin
A:= 1;
A:= A*200;
```

```
writeln(' A');
writeln(A,A,A);
end.
```

ЗАДАЧА 1.5

В листинге 1.44 приведен фрагмент программы. Необходимо определить, что появится на экране после ее выполнения.

Листинг 1.44. Программа к задаче 1.5

```
program listing_1_44;
var A:integer;
begin
A:= 1;
A:= A*200;
writeln(' A',' ','A');
writeln(A:5,A:7,A:1);
end.
```

ЗАДАЧА 1.6

В листинге 1.45 приведен фрагмент программы. Необходимо без выполнения ее на компьютере определить, что появится на экране после ее выполнения.

В данной программе используется стандартная процедура Turbo Pascal `inc`, которая увеличивает значение целочисленной переменной, задаваемой в качестве первого параметра. Это увеличение производится на единицу, если второй параметр отсутствует, или на значение, указанное в качестве второго параметра.

Листинг 1.45. Программа к задаче 1.6

```
program listing_1_45;
var
    A:integer;
begin
   A:= 100;
   inc(A);
   inc(A,4);
   writeln(A);
end.
```

ЗАДАЧА 1.7

В листинге 1.46 приведен фрагмент программы. Необходимо определить, что появится на экране после ее выполнения.

Процедура **dec** уменьшает значение целочисленной переменной, задаваемой в качестве первого параметра. Это уменьшение производится на единицу, если второй параметр отсутствует, или на значение, указанное в качестве второго параметра.

Листинг 1.46. Программа к задаче 1.7

```
program listing_1_46;
var
   A:integer;
begin
A:= 100;
dec(A,A);
inc(A,5);
writeln(A);
end.
```

ЗАДАЧА 1.8

В листинге 1.47 приведена программа. Необходимо определить, что появится на экране после ее выполнения.

Листинг 1.47. Программа к задаче 1.8

```
program listing_1_47;
var
  N:integer;
begin
  N:=2;
  N:=N*N*N;
  N:= Sqr(N)*Sqr(N);
  write (' N = ',N)
end.
```

ЗАДАЧА 1.9

В листинге 1.48 приведен фрагмент программы с использованием вещественных чисел. Необходимо определить, что появится на экране после ее выполнения.

Листинг 1.48. Программа к задаче 1.9

```
program listing_1_48;
var
   Z,W,X:real;
begin
  W:=2*2.54;
```

```
  Z:=1.15;
  X:=W-2*Z;
  write (' Summa =',X:4:1)
end.
```

ЗАДАЧА 1.10

В листинге 1.49 приведен пример программы, которая касается вычислительных действий над целыми числами. Необходимо определить результат, который мы увидим на экране после выполнения программы.

Листинг 1.49. Программа к задаче 1.10

```
program listing_1_49;
var
   N,M,W:integer;
begin
  N:=7;
  M:=12;
  inc(N);
  W:=N;
  N:=M;
  M:=W;
  writeln('N равняется',N);
  writeln('M равняется',M)
end.
```

ЗАДАЧА 1.11

В листинге 1.50 приведен пример программы, которая касается использования стандартных функций системы Turbo Pascal. Необходимо определить число, которое мы увидим на экране в результате выполнения программы.

Листинг 1.50. Программа к задаче 1.11

```
program listing_1_50;
var
  M:integer;
begin
  M:= -6;
  M:= Sqr(-abs(M)+round(pi));
  writeln(M)
end.
```

ЗАДАЧА 1.12

В листинге 1.51 приведена программа, которая реализует использование уже знакомых функций для преобразования вещественных чисел в целые. Необходимо определить, что будет выведено на экран.

Листинг 1.51. Программа к задаче 1.12

```
program listing_1_51;
var
    N:integer;
    S:real;
begin
  S:=-2.4;
  N:= trunc(2*S)+ round(abs(S));
  writeln (N)
end.
```

ЗАДАЧА 1.13

В листинге 1.52 приведена программа, которая реализует использование знакомой функции, предназначенной для работы с символами. Необходимо определить, что будет выведено на экран.

Листинг 1.52. Программа к задаче 1.13

```
program listing_1_52;
var
    W: char;
    X:integer;
begin
  W:='5';
  X:=ord(W)-ord('0');
  W:='2';
  X:=10*X+ord(W)-ord('0');
  write(X)
end.
```

ЗАДАЧА 1.14

В листинге 1.53 приведена программа, в которой используется уже знакомая функция для работы с символами. Необходимо определить, что будет выведено на экран.

Листинг 1.53. Программа к задаче 1.14

```
program listing_1_53;
var W: char;
    X:integer;
```

```
begin
  W:='4';
  X:=ord(W)-ord('9');
  W:='5';
  X:=10*X+ord(W)-ord('7');
  write (X)
end.
```

ЗАДАЧА 1.15

В листинге 1.54 приведена программа, которая использует две уже знакомые функции для работы с символами. Необходимо определить, что будет выведено на экран в конце программы.

Листинг 1.54. Программа к задаче 1.15

```
program listing_1_54;
var
    W:char;
    X:integer;
begin
  W:= '5';
  X:= ord(W)-1;
  W:= chr(X+1);
  X:=10*(ord(W)-ord('4'));
  writeln (W,' ',X)
end.
```

ЗАДАЧА 1.16

В листинге 1.55 приведена программа с использованием действий над целыми числами. Необходимо определить значение переменной C, которое будет выведено на экран в конце программы.

Листинг 1.55. Программа к задаче 1.16

```
program listing_1_55;
var
  A,B,C:integer;
begin
A:= 7;
A:= A - 4;
B:= - A;
C:=-A+2*B;
writeln(' C =',C);
end.
```

ЗАДАЧА 1.17

В листинге 1.56 приведена программа с использованием действий над целыми числами. Необходимо определить значения переменных X и Y, которые в конце программы будут выведены на экран.

Листинг 1.56. Программа к задаче 1.17

```
program listing_1_56;
var
  X,Y:integer;
begin
X:= 8+2*5;
Y:= (X mod 10) + 14;
X:= (Y div 10) + 3;
writeln(' X =',X, ' Y =',X  );
end.
```

ЗАДАЧА 1.18

В листинге 1.57 приведена программа с использованием действий над целыми числами. Необходимо определить значения переменных X и Y, которые в конце программы будут выведены на экран.

Листинг 1.57. Программа к задаче 1.18

```
program listing_1_57;
var
  X,Y:integer;
begin
X:= 4+8*3;
Y:= (X mod 10) + 15;
X:= (Y div 10) + 3;
writeln(' X =',X, ' Y =',X  );
end.
```

ЗАДАЧА 1.19

В листинге 1.58 приведена программа с использованием действий над целыми числами. Необходимо определить значение переменной C, которое будет выведено на экран.

Листинг 1.58. Программа к задаче 1.19

```
program listing_1_58;
var
  A,B,C:integer;
```

```
begin
A:= 3;
B:= A + 3;
B:= 1 - B;
C:=-B+3*A;
writeln(' C =',C);
end.
```

Ответы к задачам и заданиям из ЕГЭ

Задача 1.1

Первый вычислительный оператор в программе листинга 1.40 приводит к результату: A:=35. Остаток от деления 35 на 10 равен 5. Далее это число складывается с 14 и результат (19) записывается в переменную B. Таким образом, ответ на один вопрос — значение переменной B равно 19. Остаток от деления 19 на 10 равен 9. Далее это число складывается с числом 2 и результат (11) записывается в переменную A. В результате значение переменной A равняется 11.

Ответ: 11 и 19.

Задача 1.2

Результат целочисленного деления 1819 на 100 равен 18. Это значение умножается на 10, а результат складывается с числом 9 (в итоге получается 189). Таким образом, значение B после выполнения второго оператора равно 189. В третьем операторе выражение в скобках дает результат 71 (189 умножается на 10, а из результата вычитается 1819). Остаток от деления 71 на 100 равен 71. Таким образом, значение переменной A равняется 71, а переменной B равняется 189.

Задача 1.3

Первый вычислительный оператор в программе листинга 1.3 приводит к занесению в переменную A числа 17 (из 18 вычитается 1). Далее в следующем операторе порядок приоритетности операций говорит о том, что сначала будет выполнена операция умножения, а затем вычисление остатка от деления 10 (точнее, приоритет операций одинаков, но первым в выражении стоит знак умножения). В результате в процессе вычисления мы получим число 8.

Задача 1.4

В программе листинга 1.43 приведены два оператора вывода информации на экран. При этом первый оператор выводит пробел и букву A, после чего выполняется перевод строки. В следующей строке три раза подряд выводится значение переменной A. В результате мы увидим:

```
 A
200200200
```

Задача 1.5

В программе листинга 1.44 приведены два операторы вывода информации на экран. При этом первый оператор выводит набор символов, после чего выполняется перевод строки. В следующей строке три раза подряд выводится значение переменной A с определенным форматированием. В результате мы увидим:

```
A A
 200     200200
```

Задача 1.6

В связи с приведенным комментарием при постановке задачи значение переменной A сначала увеличится на 1, а затем еще на 4. В результате на экране мы увидим:

```
105.
```

Задача 1.7

Учитывая комментарий, приведенный при постановке задачи, значение переменной A сначала уменьшается на 100, а затем увеличивается на 5. В результате на экране мы увидим число 5.

Задача 1.8

Здесь сначала 2 возводится в куб (результат равен 8). После этого выполняется вычисление квадрата числа 8, который умножается сам на себя. В результате получим 4096.

Задача 1.9

Здесь выполняются действия с вещественными числами. Вывод результата представлен в обычном формате с ограничением числа разрядов дробной части.

Вычисления вручную дают:

```
5.08 - 2.3 = 2.78.
```

Округление до одного разряда дробной части приводит к числу 2.8.

Задача 1.10

Фактически в тексте программы листинга 1.49 осуществляется обмен значений. При этом предварительно исходное значение переменной N увеличивается на единицу. В результате мы увидим на экране, что N равняется 12, а M примет значение 8.

Задача 1.11

Здесь выполняется вычисление модуля числа M, а затем округление числа π. Далее сумма этих составляющих возводится в квадрат. В результате мы получим число 9.

Задача 1.12

В программе осуществляется вычисление выражения из двух слагаемых. Первое слагаемое представляет целую часть от произведения 2 на -2.4, что дает -4. В ре-

зультате вычисления второго слагаемого получим ответ: 2. И в конечном итоге ответ равен –2.

Задача 1.13

Первое вычисление X представляет вычитание кодов двух символов. Учитывая, что коды цифр расположены плотно, вычисление приводит к ответу: X=5. Второе действие формирует ответ, равный 52.

Задача 1.14

Здесь первая вычислительная операция дает ответ: –5. Второе действие формирует ответ: -52.

Задача 1.15

Вычисление X во второй строке исполнительной части программы приводит к коду числа 4 (из ASCII-кода числа 5 вычитается единица). В следующей строке получаем символ 5. Это значение будет первым выведено на экран. Второй результат — это число 10. Таким образом, на экране мы увидим:

```
5 10
```

Задача 1.16

Первый вычислительный оператор (после присвоения переменной A значения 7) в программе листинга 1.55 приводит к результату: A:=3. Далее вычисляется значение переменной B, что приводит к значению -3. После этого вычисляется значение переменной C, что дает значение -9. Это результат сложения -3 и -6.

Ответ: -9.

Задача 1.17

В первой строке с вычислением рассчитывается значение переменной X, которое равно 18. Далее в следующей строке число 14 складывается с остатком от деления 18 на 10. В результате этого действия переменная Y принимает значение 22. В последней вычислительной строке производится целочисленное деление 22 на 10, что дает 2. К этому значению добавляется 3 и результат заносится в переменную X. Таким образом, X=5.

Ответ: X=5 Y=22.

Задача 1.18

В первой строке с вычислением рассчитывается значение переменной X, которое равно 16. Далее в следующей строке число 15 складывается с остатком от деления 16 на 10. В результате этого действия переменная Y принимает значение 21. В последней вычислительной строке производится целочисленное деление 21 на 10, что дает 2. К этому значению добавляется 3 и результат заносится в переменную X.

Ответ: X=5 Y=21.

Задача 1.19

Первый вычислительный оператор (после присвоения переменной A значения 3) в программе листинга 1.58 приводит к результату: B:=6. Далее вычисляется новое значение переменной B, что приводит к значению -5. После этого вычисляется значение переменной C, что дает значение 14.

Ответ: 14.

ГЛАВА 2

Условия, выбор и циклы

Программы, которые мы рассмотрели в первой главе, были достаточно простыми и заключались лишь в последовательном выполнении инструкций. Исходные данные на эту последовательность не влияли, и шаги алгоритма были одни и те же независимо от действий пользователя. В реальных ситуациях необходимо выполнять те или иные действия в зависимости от определенных входных значений (часто не буквально входных значений, а результатов текущих вычислений). Например, из курса математики вы знаете, что соотношения для нахождения решения квадратного уравнения различаются в зависимости от параметров (коэффициентов) этого уравнения. В языке Паскаль, как и в любом другом языке программирования, имеются средства для реализации подобных ситуаций — это несколько вариантов оператора условия.

В большом числе случаев при построении алгоритмов требуется выполнять периодически повторяющиеся действия. Для этого в языках программирования существуют операторы циклов. *Цикл* или циклический вычислительный процесс характеризуется повторением одних и тех же действий над различными данными. Числом повторений цикла управляет специальная переменная, называемая *счетчиком цикла*. На счетчик накладывается условие, определяющее, до каких пор следует выполнять цикл. Повторяемый блок вычислений называют *телом цикла*. В цикле должно быть обеспечено изменение значения счетчика, чтобы он мог завершиться через конечное число шагов. Однократное выполнение тела цикла называют его *шагом*.

В организации цикла часто подразумевается, что число шагов цикла (повторений) известно заранее. Однако встречаются и ситуации, когда количество прохождений цикла заранее неизвестно. Оно определяется лишь постепенно после некоторого количества повторений. В этом случае применяется другая разновидность цикла — это *цикл с условием*. В языке Паскаль предусмотрено два варианта цикла с условием:

- условие проверяется перед телом цикла;
- условие проверяется после тела цикла.

Далее в этой главе мы подробно рассмотрим как необходимые теоретические сведения по данной теме, так и практические примеры, которые помогут вам сформировать необходимые навыки в плане самостоятельного программирования.

Оператор условия

Это один из ключевых операторов в любом языке программирования. *Оператор условия* позволяет организовать дальнейшее выполнение алгоритма по одному из двух направлений. Эта возможность с учетом уже рассмотренных в первой главе базовых сведений по языку Паскаль позволяет реализовывать очень сложные алгоритмы.

Познакомимся с оператором условия на простом практическом примере программной разработки. Так, задача заключается во вводе целого числа с клавиатуры и его последующем анализе. Сам анализ достаточно простой: если число меньше 5, то его квадрат выводится на экран. В противном случае (если данное условие не выполняется) на экран ничего не выводится. В листинге 2.1 приведена программа, которая решает данную задачу.

Листинг 2.1. Пример использования неполной формы оператора условия

```
program listing_2_1;
var
  Z:integer;
begin
  writeln('Введите целое число');
  readln(Z);
  if Z < 5 then
       writeln(sqr(Z))
end.
```

Данную разработку дополним блок-схемой (рис. 2.1). Здесь использован новый элемент в виде ромба (блок с названием "решение"), который имеет один вход и два выхода. В зависимости от выполнения указанного внутри ромба условия последующее действие организуется по одному из двух направлений. Технически данный элемент блок-схемы реализуется с помощью оператора `if`, формат которого выглядит следующим образом:

```
if  условие then
          оператор,
```

где под условием понимается любое выражение, результат которого имеет тип `boolean`. Приведенный пример демонстрирует неполную форму оператора условия (не указан оператор, который должен выполняться, если условие ложно). Однако существует и полная форма данного оператора:

```
if  условие then
  оператор
else
  оператор
```

В этом случае при выполнении условия производится выполнение оператора (или нескольких операторов, что будет показано далее) между ключевыми словами **then** и **else**. Если же условие не выполняется, то выполняется оператор (операторы), расположенный после **else**. Таким образом, в этом случае всегда выполняется одна из двух групп операторов.

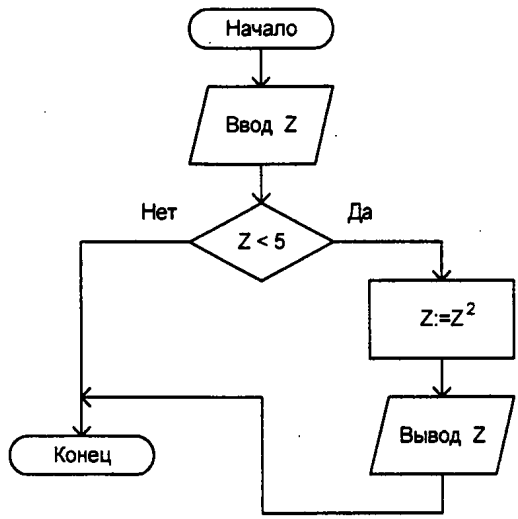


Рис. 2.1. Блок-схема к программе листинга 2.1

Для иллюстрации полной формы оператора **if** в листинге 2.2 приведен пример, базирующийся на предыдущей программе. Здесь в случае, если введенное число меньше пяти, то его квадрат выводится на экран, в противном же случае на экран выводится корень квадратный из введенного целочисленного значения. На рис. 2.2 представлена блок-схема данного алгоритма.

Листинг 2.2. Пример использования полной формы оператора условия

```
program listing_2_2;
var
  Z:integer;
begin
  writeln('Введите целое число');
  readln(Z);
  if Z < 5 then
       writeln(sqr(Z))
  else
       writeln(sqrt(Z))
end.
```

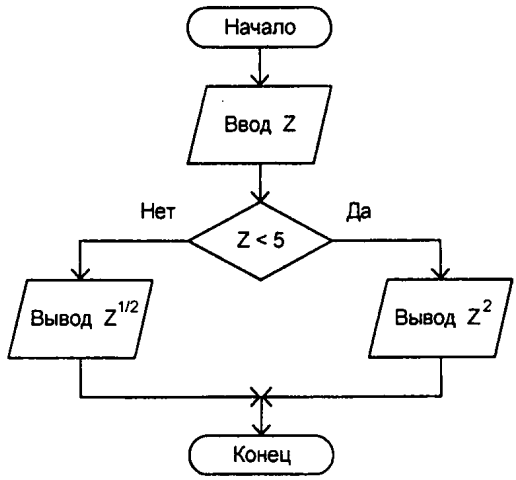


Рис. 2.2. Блок-схема к программе листинга 2.2

Примечание

Важно отметить, что перед **else** нельзя ставить точку с запятой.

Разумеется, приведенные примеры являются ограниченными в том плане, что после служебных слов **then** и **else** мы использовали только одну строку (один оператор). Если же требуется выполнить несколько операторов подряд, то их заключают в блок, образуемый служебными словами **begin** и **end**. Модернизация (в большей степени искусственная) предыдущей программы с целью демонстрации включения таких блоков приведена в листинге 2.3.

Листинг 2.3. Пример оператора условия с использованием блоков

```
program listing_2_3;
var
   Z:integer;
begin
  writeln('Введите целое число');
  readln(Z);
  if Z < 5 then
    begin
       Z:=sqr(Z);
       writeln(Z);
    end
  else
    writeln(sqrt(Z));
end.
```

Приведем еще один пример, в котором производится ввод с клавиатуры трех чисел. Далее в случае, если они все равны, на экран выводится соответствующее сообще-

ние. В случаях, если равна только одна пара чисел или если все введенные числа различны, также формируются соответствующие сообщения. Листинг 2.4 содержит текст программы, а на рис. 2.3 приведена блок-схема алгоритма.

Листинг 2.4. Сопоставление трех введенных с клавиатуры чисел

```
program listing_2_4;
var
   Z,X,W:integer;
begin
  writeln('Введите числа Z,X,W :');
  read(Z,X,W);
  if (X = Z) and (X = W) then
     writeln('Все числа равны')
   else
        if (X = Z) or (W = Z) or (X = W) then
           begin
              write('Только');
              writeln('два числа равны');
           end
        else
           writeln('Все числа разные');
end.
```

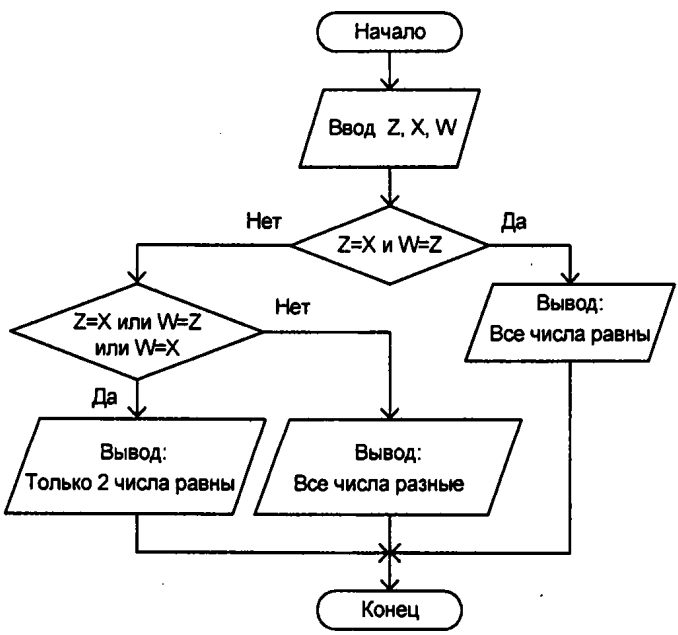


Рис. 2.3. Блок-схема к программе листинга 2.4

В подобных ситуациях можно использовать форму составного условного оператора. Этот вариант удобно применять, когда имеется более двух вариантов расчета. Составной оператор может включать произвольное число условий и ветвей расчета. Его общий вид выглядит следующим образом:

```
if условие1 then
     оператор1
else if условие2 then
     оператор2
   ...
else if условиеN then
     операторN
else
     оператор0;
```

При использовании такого оператора последовательно проверяются логические выражения (от первого до N-го). Если одно из этих выражений истинно, то выполняется соответствующий оператор и управление передается на оператор, следующий за данным условным оператором. Если все условия ложны, то выполняется оператор0 (если он задан). При этом число ветвей N неограниченно, а последней ветви (**else** оператор0;) может и не быть.

В качестве примера на данную тему рассмотрим такую задачу: необходимо по трем введенным с клавиатуры числам принять решение:

- если среди данных чисел имеется одно, которое превосходит два других, то его необходимо вывести на экран;
- если же нет одного превосходящего два других, то необходимо сообщение об этом вывести на экран.

Листинг 2.5 содержит текст программы, а на рис. 2.4 приведена блок-схема алгоритма.

Листинг 2.5. Анализ трех введенных с клавиатуры чисел

```
program listing_2_5;
var
   Z,X,W:integer;
begin
  writeln('Введите числа Z,X,W :');
  readln(Z,X,W);
  if (X > Z) and (X > W) then
     writeln('X больше двух других чисел')
   else if (Z > X ) and (Z > W) then
     writeln('Z больше двух других чисел')
   else if (W > X ) and (W > Z) then
     writeln('W больше двух других чисел')
   else
     writeln('Среди чисел имеются одинаковые');
end.
```

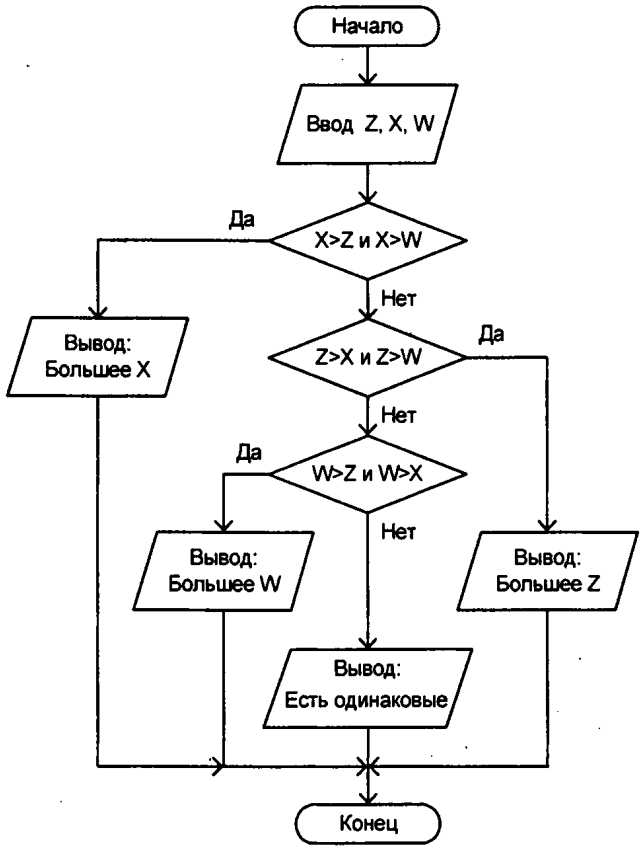


Рис. 2.4. Блок-схема к программе листинга 2.5

К примерам, связанным с оператором условия, мы вернемся в этой главе при рассмотрении многочисленных примеров программных разработок.

Оператор выбора

Для случаев, когда требуется осуществить выбор одного значения из конечного набора вариантов, оператор **if** удобнее заменить оператором *выбора (переключателем)* **case**:

```
case выражение of
  список1: оператор1;
  список2: оператор2;
  . . .
  списокN: операторN
  else оператор0;
end;
```

В целом оператор **case** выполняется так же, как и составной условный оператор. Выражение (после ключевого слова **case**) должно иметь целый или символьный

тип. Значение этого выражения последовательно сравнивается со значениями в расположенных ниже списках. Если в одном из списков находится совпадающее значение, то выполняется соответствующий оператор.

Элементы списка перечисляются через запятую (если их несколько), ими могут быть константы и диапазоны значений того же типа, что тип выражения. В листинге 2.6 приведен простой пример использования оператора выбора, с помощью которого организуется вывод текстового названия месяца года в зависимости от числового обозначения месяца. Здесь в каждом списке структуры **case** используется только одно значение (списки формируются из одного элемента).

Листинг 2.6. Формирование названия месяца по номеру

```
program listing_2_6;
var
  Z:integer;
begin
  writeln('Введите номер месяца :');
  readln(Z);
  case Z of
  1: writeln('Январь');
  2: writeln('Февраль');
  3: writeln('Март');
  4: writeln('Апрель');
  5: writeln('Май');
  6: writeln('Июнь');
  7: writeln('Июль');
  8: writeln('Август');
  9: writeln('Сентябрь');
  10: writeln('Октябрь');
  11: writeln('Ноябрь');
  12: writeln('Декабрь')
  else writeln ('Номер месяца указан неправильно');
  end;
end.
```

Организация списков может быть построена в виде диапазонов. Это удобнее, чем перечислять через запятую возможные значения, если таких достаточно много. Диапазоны указываются в виде:

```
Минимальное значение .. Максимальное значение
```

Как уже говорилось, оператор диапазона записывается как два рядом стоящих символа точки. В диапазон входят все значения от минимального до максимального включительно.

В качестве примера приведем программу формирования названия времени года в зависимости от числового номера месяца (листинг 2.7).

Листинг 2.7. Формирование времени года по номеру месяца

```
program listing_2_7;
var
  Z:integer;
begin
  writeln('Введите номер месяца :');
  readln(Z);
  case Z of
  12,1,2: writeln('Зима');
  3..5: writeln('Весна');
  6..8: writeln('Лето');
  9..11: writeln('Осень')
  else writeln ('Номер месяца указан неправильно');
  end;
end.
```

Следующий пример на рассматриваемую тему приведен в листинге 2.8. Здесь по введенному с клавиатуры символу определяется: к какой группе символов он относится.

Листинг 2.8. Классификация введенного символа

```
program listing_2_8;
var
  Z:char;
begin
  writeln('Введите символ :');
  readln(Z);
  case Z of
 'A'..'Z','a'..'z':
       writeln ('Латинская буква');
 '0'..'9':
       writeln ('Цифра')
 else
    writeln ('Другой символ');
  end;
end.
```

Если по ветви оператора **case** нужно выполнить несколько операторов, то действует то же правило, что и для оператора **if**, т. е. ветвь алгоритма заключается в конструкцию **begin ... end**.

Рассмотрим еще один пример на тему оператора выбора. Необходимо по введенному с клавиатуры номеру тарифа и количеству оплачиваемых месяцев выдать значение суммы для оплаты. Листинг 2.9 содержит текст программы, а на рис. 2.5 приведена блок-схема алгоритма.

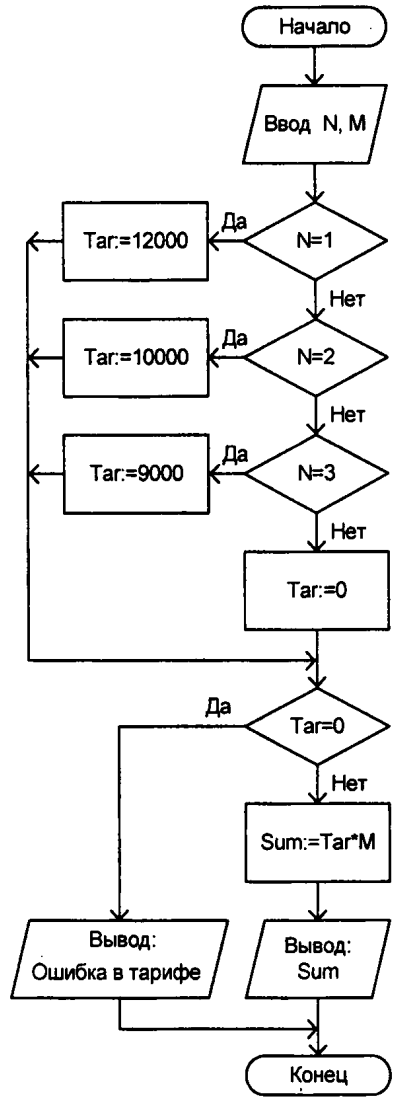


Рис. 2.5. Блок-схема к программе листинга 2.9

Листинг 2.9. Анализ тарифа и вычисление суммы для оплаты

```
program listing_2_9;
var
    M,N:integer;
    Sum,Tar:real;
begin
   writeln('Ввести число оплачиваемых месяцев');
  readln(M);
  writeln('Ввести номер тарифа');
  readln(N);
```

```
  case N of
  1: Tar:=12000;
  2: Tar:=10000;
  3: Tar:=9000
  else Tar:=0;
  end;
 if Tar=0 then
   write('Ошибка в указании номера тарифа')
 else
  begin
    Sum:=Tar*M;
    writeln('К оплате ',Sum);
  end;
end.
```

В листинге 2.10 приведен более усложненный вариант предыдущей разработки: в случае если указан первый тариф и оплачивается число месяцев не менее трех, то для человека предоставляется скидка в виде 5 % от суммы.

В программе скидка в 5 % реализуется с помощью умножения суммы, представленной к оплате, на 0.95.

Листинг 2.10. Вычисление суммы с учетом скидки

```
program listing_2_10;
var
     M,N:integer;
     Sum,Tar,Skidka:real;
begin
  Skidka:=1;
  writeln('Ввести число оплачиваемых месяцев');
  readln(M);
  writeln('Ввести номер тарифа');
  readln(N);
  case N of
  1:
     begin
       if M >= 3 then
             Skidka:=0.95;
       Tar:=12000;
    end;
  2: Tar:=10000;
  3: Tar:=9000
  else Tar:=0;
  end;
  if Tar=0 then
     write('Ошибка в указании номера тарифа')
```

```
 else
  begin
    Sum:=Tar*M*Skidka;
    writeln('К оплате ',Sum);
  end;
end.
```

Оператор цикла *for*

В программе часто требуется повторять определенные действия. Такое повторение называется *циклом*. А сама последовательность выполняемых команд представляет собой тело цикла. Наиболее простая и в то же время часто встречающаяся ситуация связана со случаем, когда число повторений цикла известно заранее. На языке Паскаль для реализации такого алгоритма действий используется оператор `for`.

Подсчет количества выполняемых действий осуществляется при помощи специальной переменной — *счетчика*. Существуют две формы цикла `for`. Первая форма реализует последовательное увеличение (на единицу) значения счетчика:

```
for счетчик:=A to B do
   оператор,
```

где `A` — начальное значение счетчика, а `B` — конечное значение.

Важно отметить, что счетчик обязательно должен иметь целочисленный тип. В операторе `for` указываются его начальное и конечное значение. После очередного выполнения тела цикла значение счетчика увеличивается на единицу. Далее это новое значение сравнивается с предельным (конечным) значением. Если счетчик не вышел за конечное значение, то цикл повторяется опять (выполняется оператор после конструкции `for`). Таких проходов цикла может быть очень много. Но как только счетчик превысит конечное значение, выполнение цикла прекращается.

Вторая форма данного оператора позволяет последовательно уменьшать значение счетчика на единицу. Для этого используется следующая конструкция:

```
for счетчик:=A downto B do
   оператор.
```

Рассмотрим сначала пример, связанный с последовательным увеличением счетчика. Наша задача заключается в том, чтобы обеспечить ввод с экрана десяти оценок студентов по определенной дисциплине. После этого программа должна подсчитать средний балл по введенным оценкам и вывести его на экран. Текст разработки приведен в листинге 2.11.

Листинг 2.11. Расчет среднего балла по дисциплине

```
program listing_2_11;
var
    A,J:integer;
    Srednee:real;
```

```
begin
    Srednee:=0.0;
    for J:=1 to 10 do
     begin
         readln(A);
         Srednee:=Srednee+A;
     end;
    Srednee:=Srednee/10;
    writeln(Srednee);
end.
```

Здесь переменная J является счетчиком цикла. Для отражения описанных действий в блок-схеме имеется необходимый элемент. Его функциональность мы поясним на примере блок-схемы только что рассмотренного алгоритма расчета среднего балла (рис. 2.6). Новый элемент (под названием "подготовка") внешне представляет собой шестиугольник, который описывает организацию цикла. Внутри шестиугольника указывается конструкция, включающая счетчик и три числа. Первым после счетчика цикла располагается число, представляющее для него начальное значение, следующее число — это конечное значение для счетчика, а последнее — шаг цикла (с каким шагом изменяется счетчик цикла). Левый вход в этот блок определяет возврат к началу после очередного прохождения тела цикла, а правый выход определяет переход после выхода из цикла (когда значение счетчика вышло за пределы своего конечного значения).

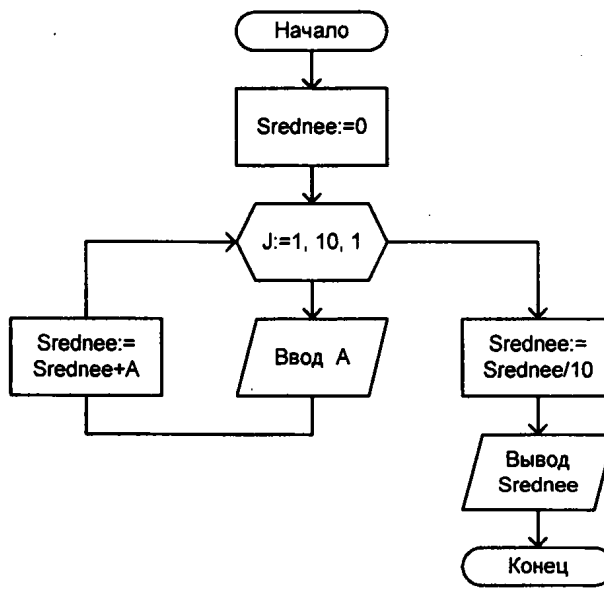


Рис. 2.6. Блок-схема к программе листинга 2.11

Рассмотрим пример программы, которая вычисляет отдельно сумму четных чисел и сумму нечетных чисел в исходной последовательности целых чисел от 1 до N. Зна-

чение N вводится с клавиатуры, а вычисленные суммы чисел должны выводиться на экран. В листинге 2.12 приведен текст программы, а на рис. 2.7 приведена блок-схема алгоритма решения данной задачи.

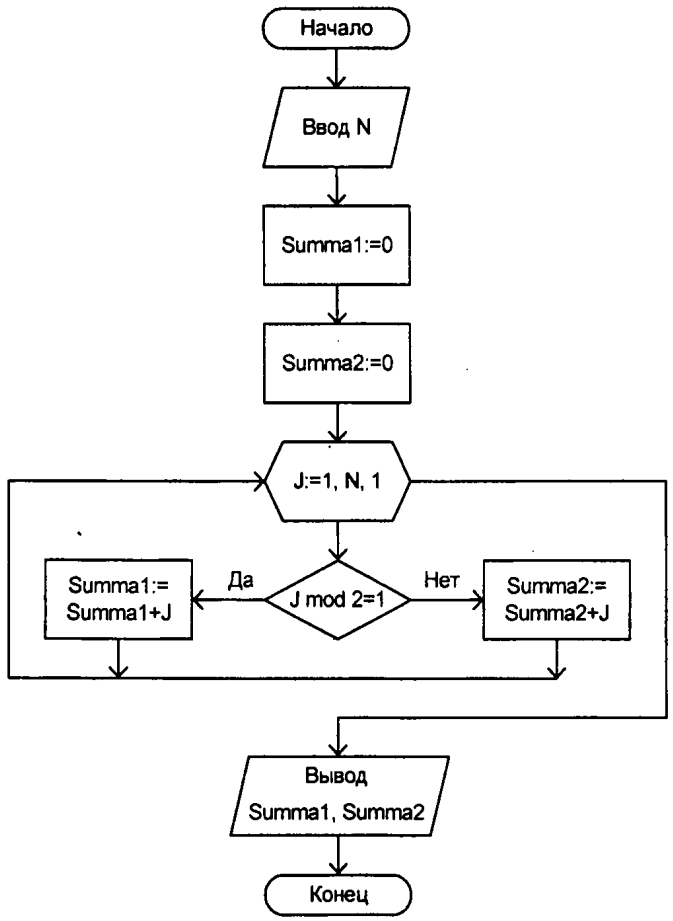


Рис. 2.7. Блок-схема к программе листинга 2.12

Листинг 2.12. Расчет сумм четных и нечетных чисел

```
program listing_2_12;
var
    N,J:integer;
    Summa1,Summa2:integer;
begin
   writeln('Введите целое положительное число');
   readln(N);
   Summa1:=0;
   Summa2:=0;
```

```
   for J:= 1 to N do
    begin
       if (J mod 2) = 1 then
           Summa1:= Summa1+J
       else
           Summa2:= Summa2+J;
    end;
    writeln('Сумма нечетных чисел равна',Summa1);
    write('Сумма четных чисел равна',Summa2);
end.
```

Рассмотрим теперь построение следующего алгоритма: требуется по введенному с клавиатуры целому положительному числу выдать заключение — является ли оно простым.

Примечание

Простыми считаются целые положительные числа, которые делятся без остатка только на себя и на единицу.

Идея алгоритма заключается в анализе ситуаций, когда остаток от деления исходного числа N на числа в интервале от 1 до N равен 0. Если таких случаев будет больше двух, то число не простое. В противном случае число является простым. Блок-схема алгоритма приведена на рис. 2.8, а текст программы представлен в листинге 2.13.

Листинг 2.13. Проверка, является ли введенное число простым

```
program listing_2_13;
var
   N,J:integer;
   Summa:integer;
begin
  writeln('Введите число');
  readln(N);
  Summa:=0;
  for J:= 1 to N do
    begin
       if (N mod J) = 0 then
           Summa:= Summa+1;
    end;
  if Summa > 2 then
    writeln('Число не простое')
  else
    writeln('Число простое')
end.
```

Однако рассмотренный алгоритм не является оптимальным с точки зрения вычислительных действий. Очевидно, что проверку "является ли число простым" можно организовать более эффективным способом. А именно: если мы найдем в интервале от 2 до N-1 хотя бы одно число, на которое N делится без остатка, то из этого уже следует, что N не является простым.

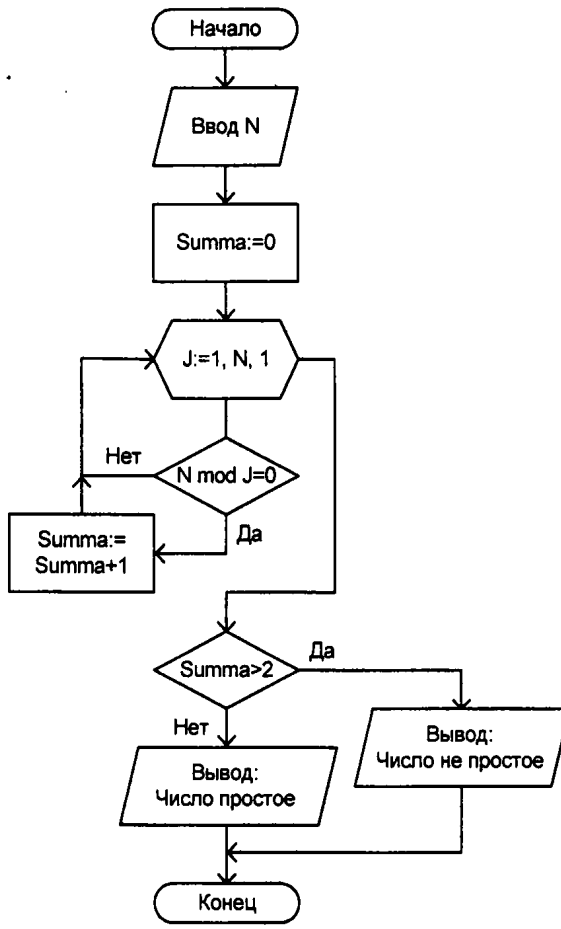


Рис. 2.8. Блок-схема к программе листинга 2.13

Для реализации этой идеи в виде программной разработки нам потребуется новый оператор. Это — **break**, прерывающий выполнение цикла и передающий управление оператору, который должен выполняться после окончания цикла. И в листинге 2.14 показана программа, которая делает рассмотренную проверку чисел более эффективной. Алгоритм поясняет блок-схема, которая представлена на рис. 2.9. Мы ввели переменную `Flag`, которая играет роль индикатора. В начале программы эта переменная устанавливается в ноль. Если в процессе анализа находится значение, на которое исходное число делится без остатка, то происходит установка переменной `Flag` в единицу и цикл завершается. В заключительной части программы в зависимости от значения `Flag` производится вывод соответствующего сообщения.

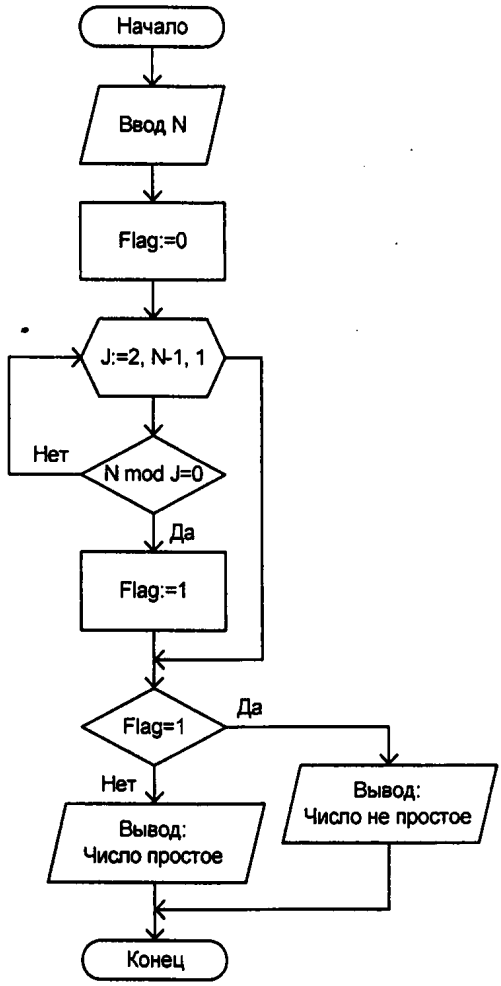


Рис. 2.9. Блок-схема к программе листинга 2.14

Листинг 2.14. Более эффективный алгоритм анализа простых чисел

```
program listing_2_14;
var
    N,J:integer;
    Flag:integer;
begin
 writeln('Введите число');
 readln(N);
 Flag:=0;
 for J:= 2 to N-1 do
    begin
       if (N mod J) = 0 then
         begin
           Flag:= 1;
```

```
          break;
        end;
  end;
  if Flag = 1 then
    writeln('Число не простое')
  else
    writeln('Число простое')
end.
```

Все рассмотренные примеры касались использования счетчика цикла на увеличение. Однако, как мы уже говорили, в языке Паскаль имеется вариант оператора **for** с организацией уменьшения счетчика. Рассмотрим пример на эту тему.

Задана числовая последовательность Exp(N)*N, где N изменяется от 1 до 50. Необходимо найти такое максимальное N, при котором значение элемента этой последовательности меньше 10000. В данном случае мы организуем цикл в направлении уменьшения N от 50 до 1. При нахождении искомого числа цикл завершается.

В листинге 2.15 приведена программа, которая решает поставленную задачу. Алгоритм поясняет блок-схема, представленная на рис. 2.10.

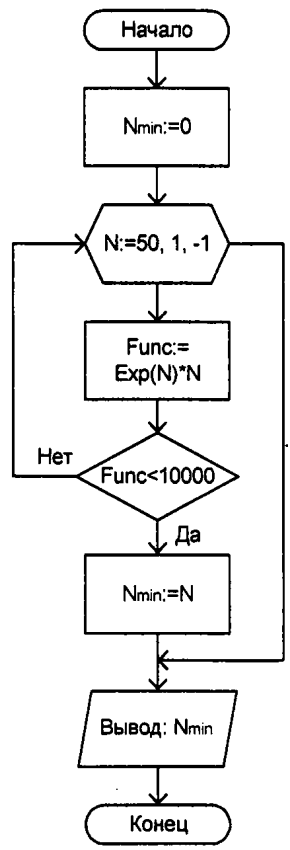


Рис. 2.10. Блок-схема к программе листинга 2.15

Листинг 2.15. Пример организации счетчика на уменьшение

```
program listing_2_15;
var
    N,Nmin:integer;
    Func:real;
begin
Nmin:=0;
   for N:= 50 downto 1 do
    begin
       Func:=Exp(N)*N;
       if  Func < 10000 then
          begin
            Nmin:=N;
            break;
          end;
    end;
  writeln('N=', Nmin);
end.
```

Цикл с предусловием

Рассмотренный цикл **for** выполняет необходимую функциональность, когда число повторений тела цикла известно к моменту его начала. Однако часто приходится решать задачи, когда число повторений цикла заранее неизвестно. В ряде ситуаций это значение определяется по мере выполнения вычислительных действий. И тогда применяют другую разновидность цикла — цикл с условием. В языке Паскаль предусмотрено два цикла с условием:

- условие цикла проверяется перед циклом (цикл с предусловием);
- условие цикла проверяется после цикла (цикл с постусловием).

На языке блок-схем цикл с предусловием показан на рис. 2.11. Здесь все организуется с помощью уже известных элементов. На языке Паскаль такая схема реализуется с использованием следующей синтаксической конструкции:

```
while логическое условие do
тело цикла
```

Так же, как и при использовании цикла **for**, после служебного слова **do** предполагается только один оператор. В случае если в теле цикла нужно выполнить несколько операторов, то следует использовать блок **begin** ... **end**.

Примечание

Точка с запятой не ставится ни перед служебными словами **then**, **else**, **do**, ни после них.

Рассмотрим практический пример. Используя цикл с предусловием, решим задачу, которая связана с вычислением факториала.

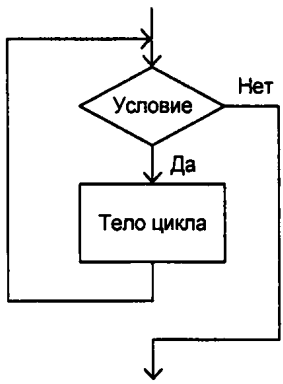


Рис. 2.11. Блок-схема цикла с предусловием

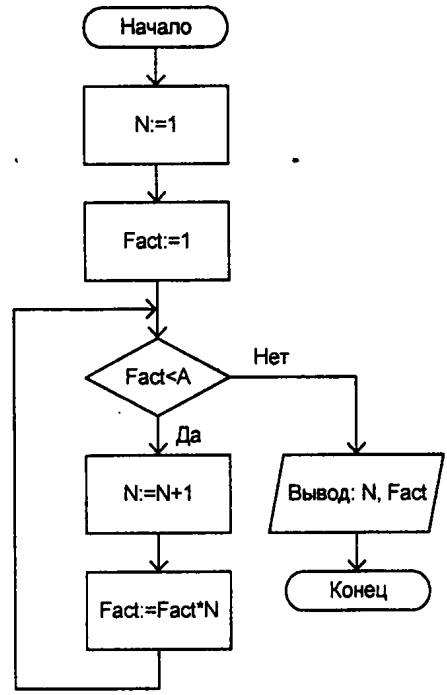


Рис. 2.12. Блок-схема к программе листинга 2.16

Примечание

Факториалом целого положительного числа N называется произведение всех целых чисел от 1 до N включительно.

Условие задания таково: необходимо найти наименьшее целое положительное число, факториал которого не меньше 10^{15}. Программное решение данной задачи приведено в листинге 2.16, а соответствующая блок-схема показана на рис. 2.12.

Листинг 2.16. Пример использования цикла с предусловием

```
program listing_2_16;
const
    A = 1.0E+15;
var N:integer;
    Fact:real;
begin
    N:=1;
    Fact:=1;
    while Fact < A do
     begin
       N:=N+1;
       Fact:= Fact*N;
     end;
   writeln('N=',N,' Fact=',Fact);
end.
```

Цикл с постусловием

В этом случае условие проверяется после цикла. Цикл повторяется до тех пор, пока проверка указанного условия будет давать результат "ложь", т. е. пока условие не выполнено. Важно отметить, что если условие сразу окажется истинным, то цикл все равно выполняется один раз. Блок-схема цикла с постусловием показана на рис. 2.13.

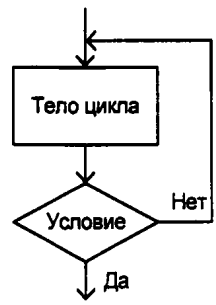


Рис. 2.13. Блок-схема с постусловием

Для цикла с постусловием сначала выполняется тело цикла, а затем управление передается на проверку условия. В зависимости от истинности или ложности условия тело цикла выполняется повторно или же происходит переход к оператору, следующему за телом цикла.

Основное различие двух рассмотренных вариантов циклов с условием: цикл с постусловием гарантированно *выполняется хотя бы раз*, а цикл с предусловием может быть не выполнен ни разу, если условие сразу же окажется ложным.

Выполнение цикла с постусловием продолжается, если проверка логического условия дает ложный результат. Если же логическое условие выполняется, то происходит выход из цикла. На языке Паскаль данный тип цикла реализуется с помощью следующих синтаксических конструкций:

```
repeat
тело цикла
until логическое выражение
```

Важно отметить, что операторы **begin** и **end** здесь не требуются.

Рассмотрим пример, в котором пользователю предоставляется возможность ввода с клавиатуры целых чисел и их суммирования. При вводе нуля суммирование заканчивается и на экране отображается результат. Блок-схема алгоритма показана на рис. 2.14, а текст программы приведен в листинге 2.17.

Листинг 2.17. Пример использования цикла с постусловием

```
program listing_2_17;
var
   A,Summa:real;
```

```
begin
  Summa:=0;
  repeat
    writeln('Введите число');
    read(A);
    Summa:=Summa+A;
  until A=0;
  write('Сумма чисел',Summa);
end.
```

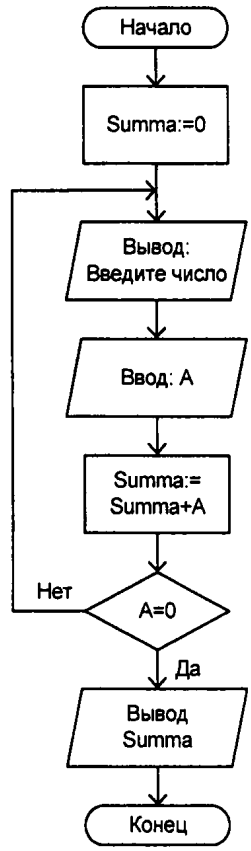


Рис. 2.14. Блок-схема к программе листинга 2.17

Метки

В большинстве случаев алгоритмы требуют организации программных переходов. Такие переходы можно реализовать и с использованием уже рассмотренных ранее ресурсов. Однако в Паскале имеется *оператор безусловного перехода*, который весьма полезен на начальном этапе изучения программирования. Синтаксис оператора выглядит так:

```
goto метка;
```

В этом операторе ключевую роль играет *метка* — произвольный идентификатор, позволяющий именовать определенный оператор в программе. После метки необходимо поставить двоеточие.

В листинге 2.18 приведен текст программы, которая позволяет вычислить произведение чисел, вводимых с клавиатуры. При вводе нуля в качестве очередного числа программа завершает работу, а результат выводится на экран.

Листинг 2.18. Пример использования меток в программе

```
program listing_2_18;
var
    A,Mul:real;
label
    M1,M2;
begin
  Mul:=1;
  M1:  writeln('Введите число');
       read(A);
       if A=0 then
          goto M2;
       Mul:= Mul*A;
       goto M1;
  M2: write('Произведение чисел',Mul);
end.
```

Работа с символьными строками

Для работы с цепочками символов в Паскале имеется специальный тип данных — **string**. Для работы с переменной типа **string** ее необходимо предварительно описать в разделе **var**. Например, это может выглядеть так:

```
var: S:string.
```

В этом случае под строку S выделяется 255 символов, а в памяти, соответственно, для переменной S отводится место. В Turbo Pascal возможны следующие действия над строками:

- ввод/вывод;
- сложение;
- присваивание;
- сравнение.

В листинге 2.19 приведен пример использования сложения, ввода/вывода и присваивания при работе со строковыми переменными.

Листинг 2.19. Пример работы со строками

```
program listing_2_19;
var
  A,B,C,D: string;
 begin
  readln(B);
  A:= 'Добрый день, ';
  C:= '!';
  D:=A+B+C;
  writeln(D);
end.
```

Довольно распространенное действие связано со сравнением строк. Здесь никакой новизны по сравнению с ранее рассмотренными примерами с использованием сравнения данных, которое встречалось нам в конструкции `if`, нет. Например, для сравнения двух переменных строкового типа достаточно использовать следующий фрагмент программного кода:

```
If Str1 = Str2 then.
```

Рассмотрим пример на данную тему. Так, с клавиатуры вводятся логин и пароль — набор символов, представленных в виде строки. Необходимо проверить, правильно ли они указаны. Программа, решающая данную задачу, приведена в листинге 2.20. В данном тексте мы разместили две переменные (`Aist` и `Bist`) для истинных пароля и логина.

Листинг 2.20. Проверка логина и пароля

```
program listing_2_20;
var
  A,B,Aist,Bist: string;
 begin
  Aist:='ZZZ';
  Bist:='123';
  writeln('Введите логин ');
  readln(A);
  writeln('Введите пароль ');
  readln(B);
  if ( A <> Aist ) or ( B <> Bist) then
    writeln('Пароль или логин введены с ошибкой!')
  else
     writeln('Пароль и логин введены правильно!');
end.
```

Рассмотрим пример использования функции работы со строками `pos(Str1,Str2)`, которая возвращает номер позиции символа в строке `Str2`, с которого начинается

первое вхождение подстроки Str1. Если же подстрока Str1 в строке Str2 отсутствует, то функция выдает 0. В листинге 2.21 приведен пример, в котором анализируется введенная строка на предмет того, имеется ли в ней слово "Pete".

Листинг 2.21. Анализ введенной строки на предмет наличия подстроки

```
program listing_2_21;
var
  Str1,Str2: string;
  N: integer;
 begin
  readln(Str2);
  Str1:= 'Pete';
  N:=Pos(Str1,Str2);
  if N = 0 then
     writeln('В строке нет искомого слова.')
  else
    writeln('В строке есть искомое слово.')
end.
```

Типовые примеры и задания из ЕГЭ

Далее мы разберем ряд практических примеров, которые можно отнести к использованию уже рассмотренных конструкций. Некоторые из приведенных примеров встречались в билетах Единого государственного экзамена в прошлые годы.

Подсчет суммы цифр в числе

Требуется разработать алгоритм подсчета суммы цифр в целом положительном числе, представленном в десятичной системе счисления. Например, в числе 3512_{10} сумма цифр равна 11_{10}, что определяется с помощью простого суммирования. Нам необходимо построить алгоритм, который будет автоматически выполнять указанное действие.

Примечание

В *главе 1* мы рассмотрели подобные задачи, но в них каждый раз ограничивалась разрядность числа. Здесь же мы не будем использовать это ограничение.

Вместе с разработкой алгоритма требуется написать программу, которая должна выполнять данную процедуру с целым числом, вводимым с клавиатуры.

Идея решения задачи заключается в использовании операции целочисленного деления и вычисления остатка от целочисленного деления. Так, если мы возьмем остаток деления исходного числа на 10, то получим самую младшую цифру исходного числа. Далее следует исходное число разделить на 10 и опять вычислить оста-

ток от деления полученного результата на 10. В итоге мы получим цифру, расположенную в разряде десятков исходного числа. Этот процесс следует продолжать до тех пор, пока результатом деления на 10 не окажется ноль. Для лучшей иллюстрации сказанного рассмотрим рис. 2.15, где представлена блок-схема алгоритма. Она соответствует словесному описанию механизма получения суммы цифр. Теперь алгоритм можно реализовать в виде программы (листинг 2.22). Здесь, учитывая блок-схему, мы воспользовались знакомым циклом с предусловием.

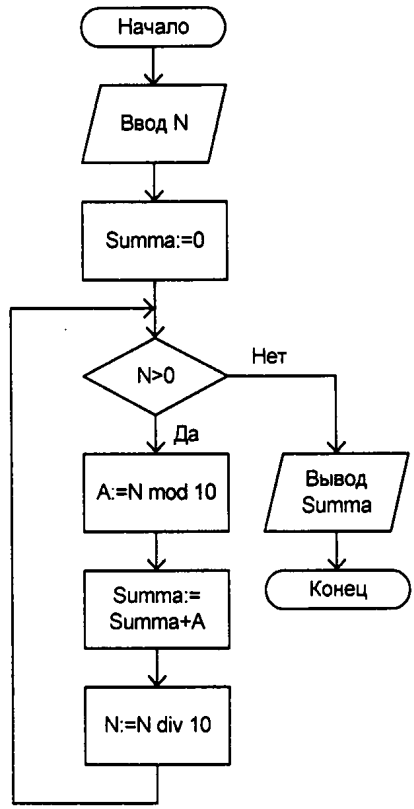


Рис. 2.15. Блок-схема к программе листинга 2.22

Листинг 2.22. Программа подсчета суммы цифр в десятичном числе

```
program listing_2_22;
var N,A,Summa:integer;
begin
  write('Введите целое число ');
  readln(N);
  Summa:=0;
  while (N > 0) do
     begin
       A:= N mod 10;
```

```
            Summa:= Summa+A;
        N:= N div 10;
      end;
   write('Summa=', Summa);
end.
```

Анализ четности пары чисел

Требуется написать программу, которая определяет, имеется ли среди введенных с клавиатуры целых чисел A и B хотя бы одно четное. Вариант реализации требуемого алгоритма представлен в листинге 2.23. Здесь мы сначала вычисляем остатки от деления двух исходных чисел на 2. Если оба этих остатка равны единице, то, следовательно, среди введенных чисел четных нет. В противном случае имеется хотя бы одно четное число.

Листинг 2.23. Проверка наличия четных чисел

```
program listing_2_23;
var
   A,B:integer;
begin
  write('Введите два числа');
  readln(A, B);
  A:= A mod 2;
  B:= B mod 2;
  A:= A + B;
  if A = 2 then
      write('Четных чисел нет')
  else
      write('Четное число есть');
end.
```

Реализуем еще один вариант решения, который связан с разработкой программы, использующей операцию or. В этом случае (листинг 2.24) вместо арифметического сложения остатков от деления используется логическая операция — *дизъюнкция*. А именно: вычисляется дизъюнкция условий равенства нулю остатков от деления исходных чисел на 2. Если хотя бы один остаток от деления равен нулю, то на экран будет выведено соответствующее сообщение о наличии четных чисел.

Листинг 2.24. Вариант алгоритма с использованием дизъюнкции

```
program listing_2_24;
var
  A,B:integer;
begin
  write('Введите два числа');
```

```
  readln(A, B);
  A:= A mod 2;
  B:= B mod 2;
   if (A = 0) or (B = 0) then
      write('Четное число есть')
  else
      write('Четных чисел нет');
end.
```

Построение треугольников из отрезков

Требуется написать программу, которая определяет, можно ли построить треугольник из отрезков с длинами X, Y, Z. Программа должна выводить соответствующее текстовое сообщение.

Идея алгоритма заключается в том, что в треугольнике сумма каждой пары сторон должна быть больше третьей стороны. И программно мы должны сравнить сумму каждой пары имеющихся отрезков с третьим отрезком. Если в каком-то случае (из трех) длина одного отрезка будет превышать сумму длин двух других, то, следовательно, и треугольник построить нельзя.

Существуют несколько вариантов написания программы в соответствии с данными условиями. Один из примеров правильной программы представлен в листинге 2.25. Здесь мы ввели переменную Flag, в которую, в случае возможности построения треугольника, заносится значение 1. При этом в начале программы значение этой переменной устанавливается равным нулю. На рис. 2.16 представлена блок-схема алгоритма.

Листинг 2.25. Проверка построения треугольника из отрезков

```
program listing_2_25;
var
   X,Y,Z:real;
   Flag:integer;
begin
  write('Введите три числа');
  readln(X, Y, Z);
  Flag:=0;
  if (X + Y) > Z then
       if (X + Z) > Y then
            if (Y + Z) > X then
                    Flag:=1;
  if Flag=1 then
       write('треугольник построить можно')
  else
       write('треугольник построить нельзя');
end.
```

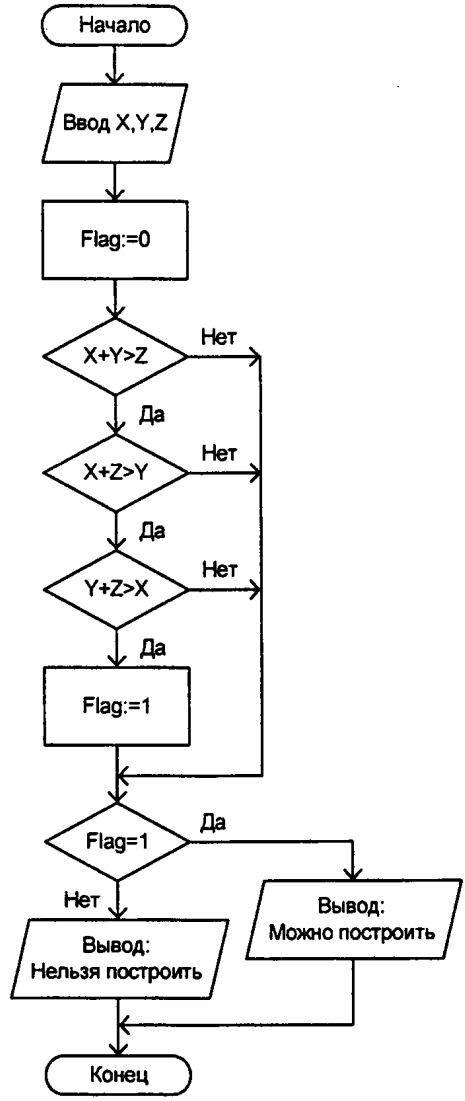


Рис. 2.16. Блок-схема к программе листинга 2.25

Можно привести еще один вариант алгоритма, в котором используется операция **and**. Листинг 2.26 содержит текст программы в новом варианте. В этом случае программа получается проще, и фактически все сводится к проверке сложного условия.

Листинг 2.26. Проверка построения треугольника из отрезков (с использованием операции *and*)

```
program listing_2_26;
var
   X,Y,Z:real;
```

```
begin
  readln(X, Y, Z);
  if ((X + Y) > Z)  and ((X + Z) > Y) and ((Y + Z) > X) then
        write('Треугольник построить можно')
  else
        write('Треугольник построить нельзя');
end.
```

Перевод числа в шестнадцатеричную систему

Необходимо разработать программу, которая позволит после ввода с клавиатуры целого числа в интервале от 0 до 15 вывести на экран его эквивалент в шестнадцатеричной системе счисления. Один из примеров программы, решающей данную задачу, представлен в листинге 2.27. Здесь учитывается, что в шестнадцатеричной системе счисления знаки, обозначающие цифры, плотно расположены в двух группах:

- от 0 до 9;
- от A до F.

При этом коды символов в каждой из этих групп тоже расположены "плотно". Поэтому после ввода числа его следует сначала проверить на принадлежность к интервалу от 0 до 15. Далее, если число меньше 10, то его символ формируется из кода нуля и добавления самого числа. Если же введенное число больше 9, то в качестве "базы" следует взять букву A. И тогда сумма кода буквы A и разности между числом и 10 представляет код символа шестнадцатеричной цифры.

Листинг 2.27. Перевод числа в шестнадцатеричную систему

```
program listing_2_27;
var
    Ch:char;
    N:integer;
begin
   write('Введите число ');
   readln(N);
   if (N>=0) and (N<16) then
     begin
       if N < 10 then
           Ch:=chr(ord('0')+N)
       else
           Ch:=chr(ord('A')+N-10);
       writeln('Число в шестнадцатеричной системе равно ',Ch);
    end
  else
     write ('Ошибка во входных данных');
end.
```

Подсчет по условию

Требуется написать программу, которая бы позволяла вводить оценки учащихся с клавиатуры и подсчитывала отдельно количество сдавших дисциплину (сдавшими считаются те ученики, которые получили оценки не менее 3) и не сдавших дисциплину (тех, кто получил 2).

Программа должна обеспечить последовательный ввод оценок. Признаком завершения данных считается ввод очередного числа вне интервала от двух до пяти. Один из вариантов программы, решающей поставленную задачу, представлен в листинге 2.28. Блок-схема алгоритма отражена на рис. 2.17.

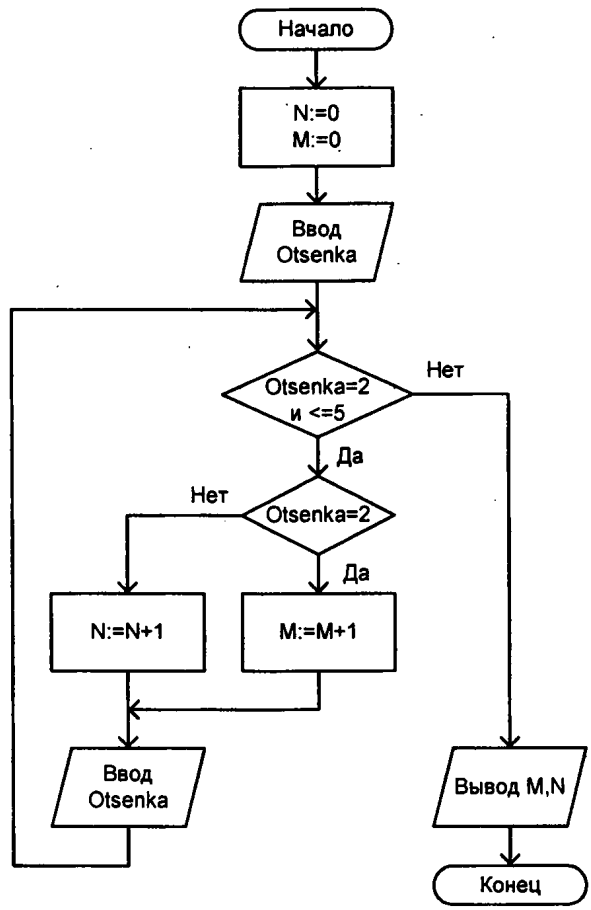


Рис. 2.17. Блок-схема к программе листинга 2.28

Листинг 2.28. Подсчет числа сдавших экзамен и числа не сдавших экзамен

```
program listing_2_28;
var
   N,M,Otsenka:integer;
```

```
begin
   N:=0;
   M:=0;
   writeln('Введите оценку');
   readln(Otsenka);
   while (Otsenka >= 2) and (Otsenka <= 5) do
   begin
     if Otsenka = 2 then
        M:=M+1
     else
        N:=N+1;
     readln(Otsenka);
   end;
    write (' Число сдавших ', N ,' Число не сдавших', M);
end.
```

Возможность построения прямоугольного треугольника

Необходимо проверить, может ли быть построен прямоугольный треугольник по длинам сторон A, B, C при условии, что A является гипотенузой. Мы воспользуемся теоремой Пифагора, при этом необходимо задать разумную точность сопоставления сторон треугольника. Программа, решающая поставленную задачу, представлена в листинге 2.29.

Листинг 2.29. Проверка возможности построения треугольника

```
program listing_2_29;
var
   A,B,C:real;
begin
  write ('Введите значение гипотенузы:');
  readln(A);
  write ('Введите значение 1-го катета:');
  readln(B);
  write ('Введите значение 2-го катета:');
  readln(C);
  if abs(sqrt(C*C+B*B)-A)<0.1 then
     writeln ('Прямоугольный треугольник может быть построен!')
  else
    writeln('Прямоугольный треугольник не может быть построен!')
end.
```

Представление слова с учетом падежа

С клавиатуры вводится денежная сумма в рублях в числовом формате. Программа должна напечатать введенную сумму с правильной формой падежа слова "рубль". Например, "23 рубля" или "51 рубль".

Понятно, что окончание, используемое для слова "рубль", зависит от последних цифр суммы. Самую последнюю цифру можно получить, если взять остаток от деления на 10. Однако если в сумме последние две цифры расположены в интервале от 11 до 14 включительно, то окончание меняется. Используя эту информацию, составим следующую программу (листинг 2.30).

Листинг 2.30. Написание слова "рубль" с учетом падежа

```
program listing_2_30;
var
   R,A,B:integer;
begin
 writeln ('Введите число рублей');
 readln (R);
 A:=R mod 10;
 B:=R mod 100;
 write (R,' ');
  if (B = 11) or (B = 12) or (B = 13) or (B = 14) or (A>4) or (A=0) then
      write ('рублей')
  else if (A = 1) then
      write ('рубль')
 else
      write ('рубля');
end.
```

Формирование таблицы стоимости товаров

Известна стоимость единицы товара. Необходимо составить таблицу стоимости 1, 2, 3, ..., K единиц товара, при этом значение K вводится с клавиатуры. Так как число единиц товара целое, то для него выберем тип **integer**. Для цены, соответственно, определим вещественный тип данных. Необходимая программа приведена в листинге 2.31.

Листинг 2.31. Формирование таблицы стоимости товаров

```
program listing_2_31;
var
     Tsena:real;
    I,K:integer;
begin
 writeln ('Стоимость единицы товара:');
 readln (Tsena);
 writeln ('Количество единиц товара:');
 readln (K);
 for I:=1 to K do
    writeln (i:5,(I*Tsena):10:2);
end.
```

Поиск чисел

Найти (программно) все двузначные числа, в записи которых имеется цифра 3 и одновременно само число делится на 3. Здесь мы воспользуемся операторами цикла и условия (листинг 2.32).

Листинг 2.32. Поиск чисел по условию

```
program listing_2_32;
var
    A,I:integer;
begin
    for I:=10 to 99 do
     If (I Mod 3 = 0) and ((I mod 10 = 3) or (I div 10 = 3)) then
            writeln(I);
end.
```

Еще один похожий пример. Требуется найти все целые числа в интервале от 100 до 999, сумма цифр которых больше произведения этих цифр. Программа приведена в листинге 2.33, а сам алгоритм программного поиска таких чисел представлен на рис. 2.18.

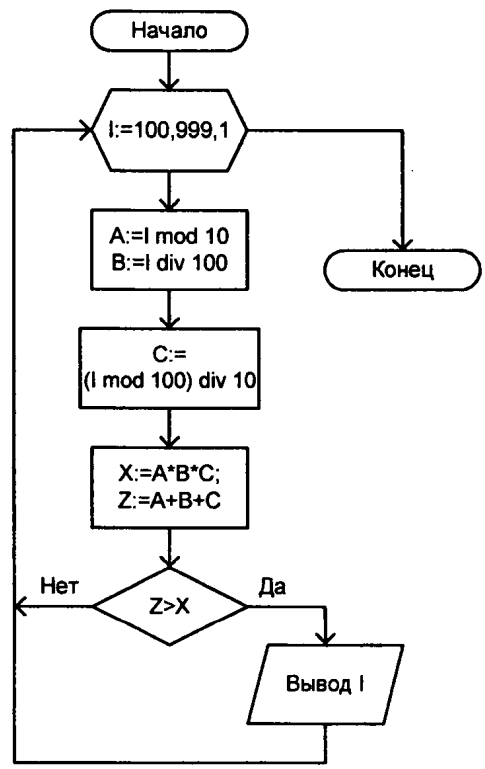


Рис. 2.18. Блок-схема к программе листинга 2.33

Листинг 2.33. Поиск чисел по условию

```
program listing_2_33;
var
   A,B,C,X,Z,I:longint;
begin
    for I:=100 to 999 do
     begin
       A:= I mod 10;
       B:= I div 100;
       C:= (I mod 100) div 10;
       X:=A*B*C;
       Z:=A+B+C;
       If Z > X then
         write(I,' ');
     end;
end.
```

Анализ чисел

С клавиатуры вводится четырехзначное число. Необходимо программно с использованием оператора цикла определить, имеются ли в нем четные цифры. Разработка представлена в листинге 2.34, а блок-схема алгоритма приведена на рис. 2.19.

Листинг 2.34. Поиск чисел по условию

```
program listing_2_34;
var
  A,B,Flag,I:integer;
begin
 writeln ('Введите четырехзначное число:');
 readln (A);
 Flag:=0;
 for I:=1 to 4 do
   begin
       B:= A mod 10;
       if  B  mod 2 = 0 then
           begin
             Flag:=1;
             break;
           end;
        A:=A div 10;
   end;
   if Flag = 1  then
       writeln('В числе есть четные цифры')
```

```
   else
      writeln('В числе нет четных цифр');
end.
```

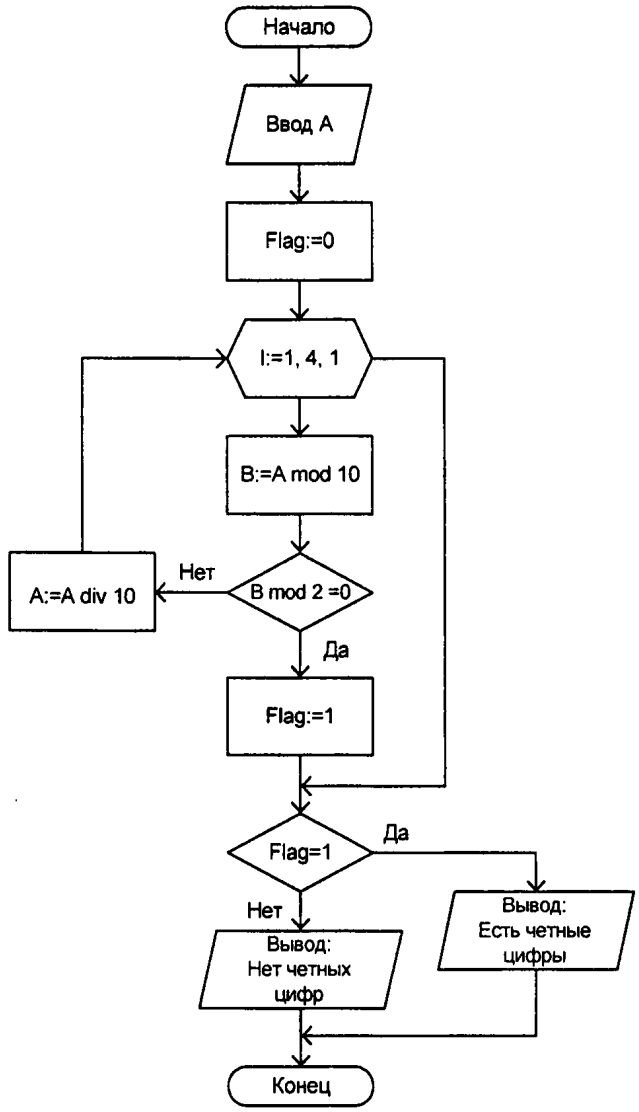


Рис. 2.19. Блок-схема к программе листинга 2.34

Рассмотрим еще один пример на тему анализа чисел. Как и в предыдущей разработке, с клавиатуры вводится четырехзначное число. Необходимо программно с использованием оператора цикла определить, имеются ли в нем соседние одинаковые цифры (одинаковые цифры в соседних разрядах). Разработка представлена в листинге 2.35, а блок-схема алгоритма на рис. 2.20.

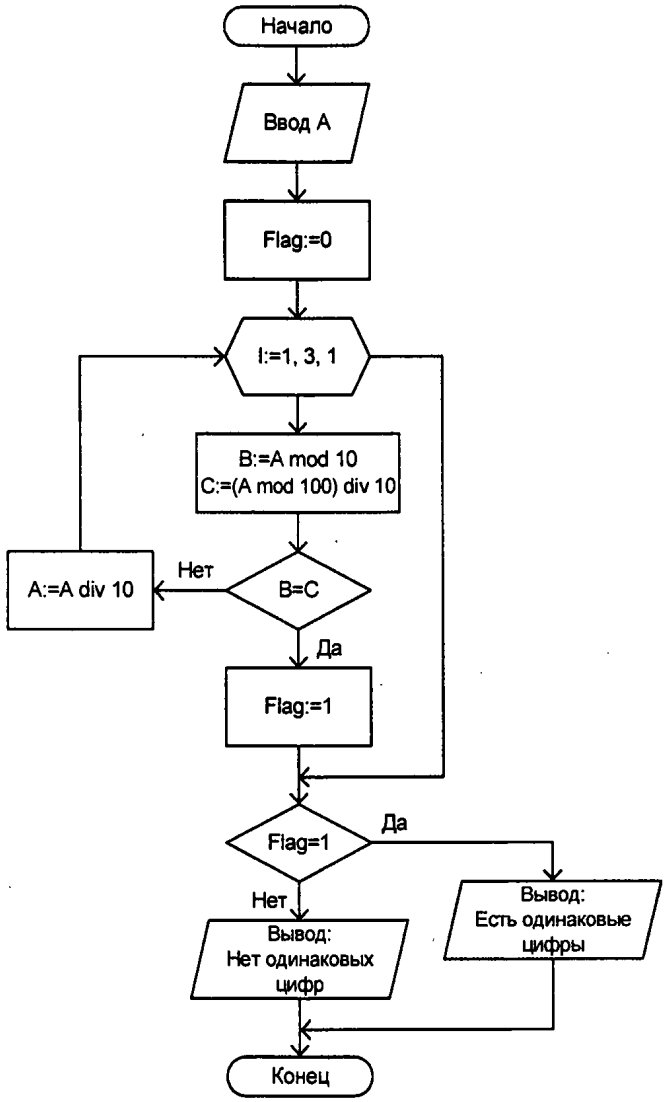


Рис. 2.20. Блок-схема к программе листинга 2.35

Листинг 2.35. Анализ на наличие одинаковых цифр

```
program listing_2_35;
var
   A,B,C,Flag,I:integer;
begin
 writeln ('Введите четырехзначное число:');
 read (A);
 Flag:=0;
 for I:=1 to 3 do
   begin
```

```
        B:= A mod 10;
        C:= (A mod 100) div 10 ;
        If  B  = C then
          begin
            Flag:=1;
            break;
          end;
       A:=A div 10;
    end;
  if Flag = 1  then
     writeln('В числе есть соседние одинаковые цифры')
 else
      writeln('В числе нет соседних одинаковых цифр');
end.
```

В следующем примере с клавиатуры вводится шестизначное число. Необходимо определить, является ли оно "счастливым". Напомним, что "счастливыми" считаются числа, у которых сумма первых трех цифр равна сумме вторых трех цифр. Разработка представлена в листинге 2.36, а блок-схема алгоритма на рис. 2.21.

Листинг 2.36. Анализ на счастливое число

```
program listing_2_36;
var
    A:longint;
    B,C,I:integer;
begin
  writeln ('Введите шестизначное число:');
 read (A);
 B:=0;
 for I:=1 to 3 do
     begin
         B:= B + (A mod 10);
         A:= A div 10;
     end;
  C:=0;
 for I:=1 to 3 do
     begin
         C:= C+ (A mod 10);
         A:=A div 10;
     end;
if B = C  then
      writeln('Число счастливое')
else
      writeln(' Число не является счастливым');
end.
```

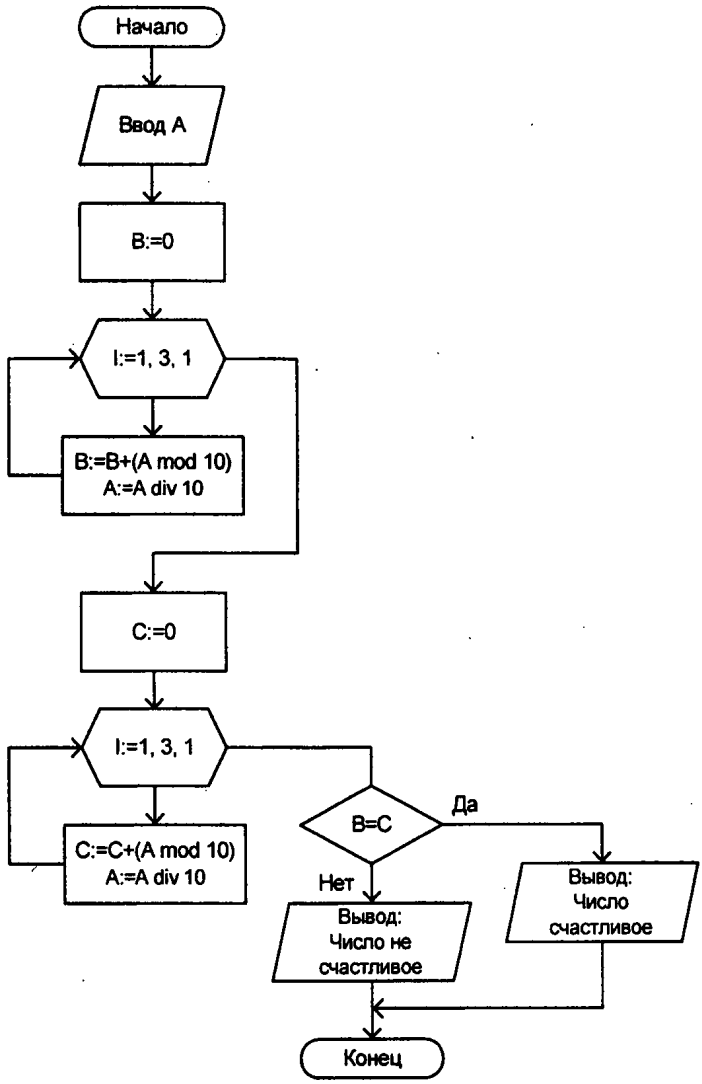


Рис. 2.21. Блок-схема к программе листинга 2.36

В следующем примере с клавиатуры вводится целое число (для определенности содержащее не более 7 знаков). Необходимо определить максимальную цифру в этом числе. Разработка представлена в листинге 2.37, а блок-схема алгоритма на рис. 2.22.

Листинг 2.37. Поиск максимальной цифры в числе

```
program listing_2_37;
var
    A,I:longint;
    B:integer;
```

```
begin
  writeln ('Введите число:');
 read (A);
 B:=0;
 I:=A;
 while I > 0 do
     begin
         if (I mod 10) > B then
            B:= I mod 10;
        I:= I div 10;
     end;
    writeln('Максимальная цифра в числе',B)
end.
```

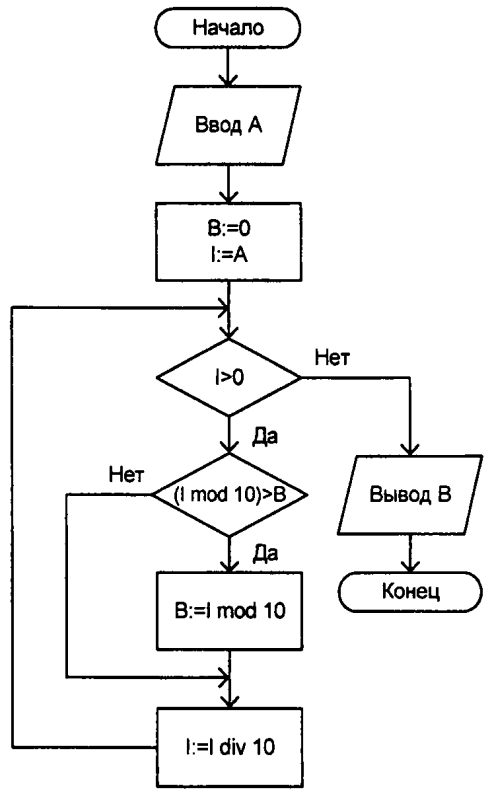


Рис. 2.22. Блок-схема к программе листинга 2.37

С клавиатуры вводится целое число (для определенности не более 7 знаков и не менее двух). Необходимо определить две максимальные (различные) цифры в этом числе. Например, если введено число 567374, то две максимальные цифры — это 7 и 6. Разработка представлена в листинге 2.38, а блок-схема на рис. 2.23.

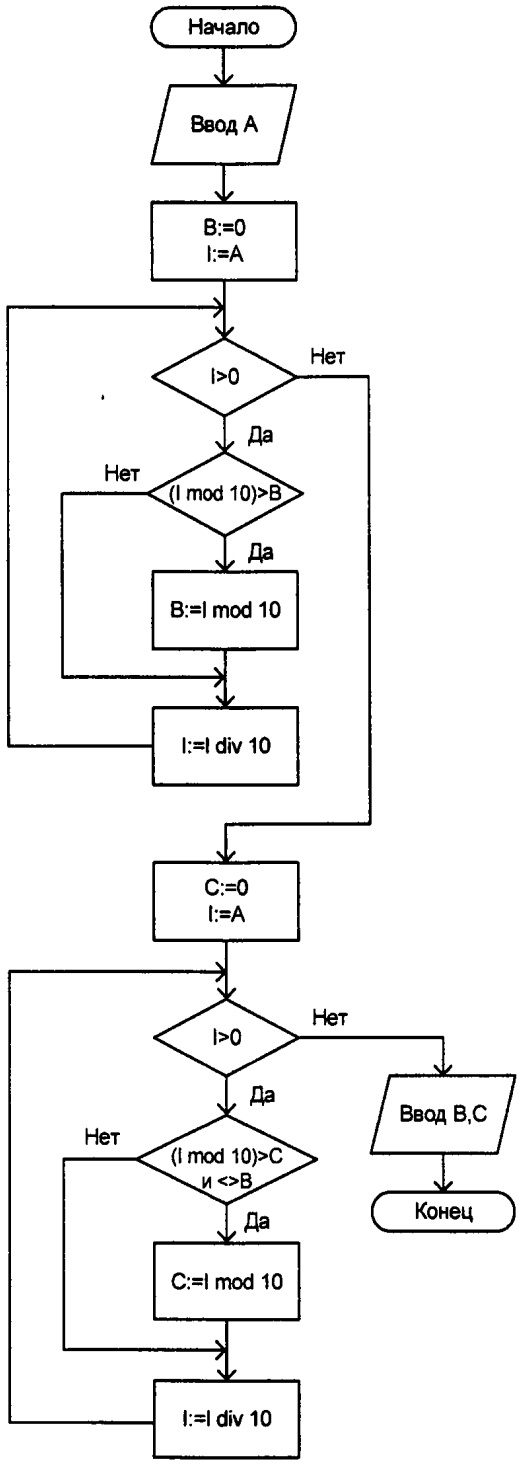


Рис. 2.23. Блок-схема к программе листинга 2.38

Листинг 2.38. Поиск двух максимальных цифр в числе

```
program listing_2_38;
var
    A,I:longint;
    B,C:integer;
begin
  writeln ('Введите число:');
 read (A);
 B:=0;
 I:=A;
 while I>0 do
     begin
         if (I mod 10) > B then
            B:= I mod 10;
        I:= I div 10;
     end;
 C:=0;
 I:=A;
 while I>0 do
     begin
         if ((I mod 10) > C) and ((I mod 10) <> B) then
            C:= I mod 10;
        I:= I div 10;
     end;
    writeln('Максимальные цифры в числе',B,C)
end.
```

С клавиатуры вводится целое число (не более 7 знаков и не менее двух). Необходимо определить минимальную цифру в этом числе и подсчитать, сколько раз она встречается. Например, если введено число 527272, то в нем минимальная цифра 2 встречается три раза. Разработка представлена в листинге 2.39. Для поиска цифры мы отвели переменную B, а количество повторений этой цифры будет фиксироваться в переменной C.

Листинг 2.39. Поиск минимальной цифры

```
program listing_2_39;
var
    A,I:longint;
    B,C:integer;
begin
  writeln ('Введите число:');
 read (A);
 B:=9;
 I:=A;
```

```
 while I>0 do
    begin
        if (I mod 10) < B then
              B:= I mod 10;
        I:= I div 10;
    end;
 C:=0;
 I:=A;
 while I>0 do
    begin
        if (I mod 10) = B then
            C:= C+1;
        I:= I div 10;
   end;
  writeln('Минимальная цифра в числе',B,' встречается ',C,' раз')
end.
```

Графики зависимостей

Необходимо построить график функции `y(N) = sin(0.2*N)`, где `N` представляет собой последовательность целых чисел в интервале от 0 до 20. Разработка представлена в листинге 2.40. Одно деление по оси ординат (одно знакоместо) соответствует 0.1, а взаимное расположение экрана и осей координат представлено на рис. 2.24. Ось ординат проходит параллельно левой границы экрана на расстоянии 20 символов. Для символа графика мы выбрали звездочку.

Листинг 2.40. График функции

```
program listing_2_40;
var
    N,M,I:integer;
    Y:real;
begin
  for N:=0 to 20  do
     begin
         Y:=sin(0.2*N);
         M:= 20+round(Y*10);
         for I:=0 to  M-1 do
               write(' ');
         writeln('*');
     end;
end.
```

В следующем задании нам необходимо построить функции `y(N) = sin(0.2*N)` в виде гистограммы, где `N` представляет собой последовательность целых чисел в интервале от 0 до 20. Разработка представлена в листинге 2.41. Здесь вместо точки

на графике мы выводим столбец звездочек, и чем больше значение функции, тем больше столбец, ему соответствующий.

Рис. 2.24. Взаимное расположение экрана и осей координат

Листинг 2.41. График функции в виде гистограммы

```
program listing_2_41;
var
    N,M,I:integer;
    Y:real;
begin
for N:=0 to 20  do
     begin
         Y:=sin(0.2*N);
         M:= 20+round(Y*10);
         for I:=0 to  M do
               write('*');
         writeln;
     end;
end.
```

Изменение чисел по условию

Необходимо возвести в квадрат трехзначное число, которое вводится с клавиатуры, при условии содержания в нем цифры 7. Результат вывести на экран. Если условие не выполняется, то следует вывести число в исходном виде. Разработка представлена в листинге 2.42.

Листинг 2.42. Возведение числа в квадрат при условии наличия в нем цифры 7

```
program listing_2_42;
var
    N:longint;
    A,B,C:integer;
```

```
begin
  writeln ('Введите число:');
  readln(N);
  A:= N div 100;
  B:= (N div 10) mod 10;
  C:= N mod 10;
  if (A=7) or (B=7) or (C=7) then
     N:= N*N;
  write(N);
end.
```

В следующем задании необходимо возвести в квадрат трехзначное число, которое вводится с клавиатуры, при условии, что все цифры в нем различные. В противном случае следует вывести число в исходном виде. Разработка представлена в листинге 2.43.

Листинг 2.43. Возведение числа в квадрат при условии различных цифр

```
program listing_2_43;
var
    N:longint;
    A,B,C:integer;
begin
  writeln ('Введите трехзначное число:');
  readln(N);
  A:= N div 100;
  B:= (N mod 100) div 10;
  C:= N mod 10;
  if (A<>B) and (B<>C) then
     N:= N*N;
  write(N);
end.
```

Типовые задачи и задания из ЕГЭ за 2008 — 2010 годы

В этом разделе приведены готовые листинги программ, которые вам требуется проанализировать. Создания каких-либо программных разработок здесь не требуется. Желательно находить правильные ответы без использования компьютера. В последнем разделе главы приведены правильные ответы.

ЗАДАЧА 2.1

В листинге 2.44 приведена программа, в начале которой с клавиатуры вводятся значения переменных A и B. Что будет выведено на экран, если в начале программы введено: A:=7 и B:=5?

Листинг 2.44. Программа к задаче 2.1

```
program listing_2_44;
var
    A,B,C:integer;
begin
  writeln ('Введите числа:');
  readln(A,B);
  C:= A div 3;
  C:= C + B mod 10;
  A:= C mod  B;
 if C < 3 then
   write(C*C)
 else
   write(C+A);
end.
```

ЗАДАЧА 2.2

В листинге 2.45 приведена программа, в начале которой с клавиатуры вводится значение переменной A. Что будет выведено на экран, если введено: A:=5?

Листинг 2.45. Программа к задаче 2.2

```
program listing_2_45;
var
    A,B:integer;
begin
  writeln ('Введите число:');
  readln(A);
  inc(A,2);
  B:= A mod 4;
  inc(B,7);
  A:= B mod 5;
  if (A div 3) > 1 then
     if (B div 5) < 3 then
       write(B);
end.
```

ЗАДАЧА 2.3

Что будет выведено на экран в результате выполнения программы, приведенной в листинге 2.46?

Листинг 2.46. Программа к задаче 2.3

```
program listing_2_46;
var
    A,B:integer;
```

```
begin
  A:=17;
  B:=21;
  if ((A mod 7) > 8) or ((B mod 5) < 3) then
        write(A)
  else
        write(B);
end.
```

ЗАДАЧА 2.4

Что будет выведено на экран в результате выполнения программы, приведенной в листинге 2.47?

Листинг 2.47. Программа к задаче 2.4

```
program listing_2_47;
var
    A:integer;
begin
  A:=8;
  if ((A mod 3) >= 2) and ((A div 3) = 2) then
        write(A*A)
      else
        write(A*A*A);
end.
```

ЗАДАЧА 2.5

Что будет выведено на экран в результате выполнения программы, приведенной в листинге 2.48?

Листинг 2.48. Программа к задаче 2.5

```
program listing_2_48;
var
    A:integer;
begin
  A:=5;
  if ((A mod 3) > 2) and ((A div 3) = 2) then
        write(A)
  else
        write(A*A);
end.
```

ЗАДАЧА 2.6

Определите значение целочисленной S после выполнения фрагмента программы, представленного в листинге 2.49.

Листинг 2.49. Фрагмент программы к задаче 2.6

```
S:=0;
N:=6;
for I:=2 to N do
  bedin
      S:=S+2*I;
  end;
write(S)
```

ЗАДАЧА 2.7

Дан фрагмент программы, который представлен в листинге 2.50. Чему будет равно значение переменных S и T после выполнения фрагмента программы?

Листинг 2.50. Фрагмент к задаче 2.7

```
S:=0; T:=1;
for I:=1 to 5 do
  bedin
     S:= S + T;
     T:=T + 1 + I;
  end
```

ЗАДАЧА 2.8

Определите значение переменной S после выполнения фрагмента программы, представленного в листинге 2.51.

Листинг 2.51. Фрагмент к задаче 2.8

```
S:=0;
for J:=1 to 5 do
  bedin
      S:=S+J;
  end;
```

ЗАДАЧА 2.9

Определите значение целочисленной переменной S после выполнения фрагмента программы, представленного в листинге 2.52.

Листинг 2.52. Фрагмент к задаче 2.9

```
S:=4;
N:=6;
for I:=2 to N do
  bedin
      S:=S+(S mod I);
  end;
```

ЗАДАЧА 2.10

Определите значение переменной C после выполнения фрагмента алгоритма, представленного на рис. 2.25.

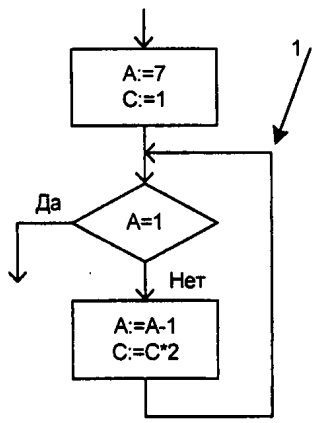


Рис. 2.25. Фрагмент алгоритма к задаче 2.10

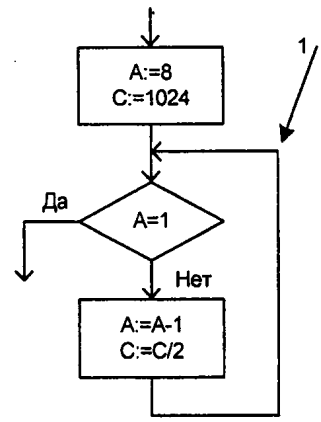


Рис. 2.26. Фрагмент алгоритма к задаче 2.11

ЗАДАЧА 2.11

Определите значение переменной C после выполнения фрагмента алгоритма, представленного на рис. 2.26.

ЗАДАЧА 2.12

Определите значение целочисленных переменных A, B и C после выполнения фрагмента программы, представленного в листинге 2.53.

Листинг 2.53. Фрагмент программы к задаче 2.12

```
A:= 3;
B:= A * (A - 1);
C:= 5 - B;
B:= A - C;
```

ЗАДАЧА 2.13

Что будет выведено на экран в результате выполнения программы, приведенной в листинге 2.54?

Листинг 2.54. Программа к задаче 2.13

```
program listing_2_54;
var
    A,B,C:integer;
begin
  A:=5;
  B:=A;
  A:=A+2*B;
  if A > 10 then
       C:=2*A
  else
            C:=-2*A;
   write(C);
end.
```

ЗАДАЧА 2.14

Что будет выведено на экран в результате выполнения программы, приведенной в листинге 2.55?

Листинг 2.55. Программа к задаче 2.14

```
program listing_2_55;
var
    A,B,C:integer;
begin
  A:=5;
  B:=A-1;
  A:=A+2*B;
  if A > 13 then
       C:=2*A
  else
            C:=-2*A;
   write(C);
end.
```

ЗАДАЧА 2.15

Что будет выведено на экран в результате выполнения программы, приведенной в листинге 2.56?

Листинг 2.56. Программа к задаче 2.15

```
program listing_2_56;
var
    A,B,C:integer;
begin
  A:=5;
  B:=-A;
  A:=A+2*B;
  if A > 3 then
       C:=2*A
  else
            C:=-2*A;
   write(C);
end.
```

ЗАДАЧА 2.16

Что будет выведено на экран в результате выполнения программы, приведенной в листинге 2.57?

Листинг 2.57. Программа к задаче 2.16

```
program listing_2_57;
var
    A,B,C:integer;
begin
  A:=5;
  B:=10;
  A:=A+B;
  if A > 10 then
       C:=2*A
  else
            C:=-3*A;
   write(C);
end.
```

Ответы к задачам и заданиям из ЕГЭ

Задача 2.1

Целочисленное деление 7 на 3 дает 2 в качестве результата. После этого полученное значение складывается с числом 5 (остаток от деления 5 на 10), и в результате в переменную C заносится 7. Переменной A присваивается значение 2 (остаток деления 7 на 5). Оператор условия, которое ложно, приводит к выводу на экран суммы C и A. Это число 9.

Задача 2.2

Вначале значение переменной A увеличивается на 2 (получается 7). После этого в переменной B формируется значение 3. Следующая строка увеличивает B на 7. Осталось вычислить остаток деления 10 на 5. Таким образом, перед условными операторами A = 0, а B = 10. В результате первое условие ложно и на экране после введенного значения мы не увидим ничего.

Задача 2.3

Остаток от деления A на 7 равняется 3, а остаток от деления B на 5 равен 1. В результате одно условие из двух истинно и на экране мы увидим 17.

Задача 2.4

Обе составляющие условия истинны. В результате на экране мы увидим 64.

Задача 2.5

Одна из составляющих условия дает ложный результат. В итоге на экране мы увидим 25.

Задача 2.6

Цикл в представленном здесь программном фрагменте выполняется 5 раз (по I, начиная с 2 до 6 с шагом 1). Приведем результаты вычисления переменной S после каждого шага:

- I:=2; S:=4;
- I:=3; S:=10;
- I:=4; S:=18;
- I:=5; S:=28;
- I:=6; S:=40.

Таким образом, ответ на поставленный вопрос: S = 40.

Задача 2.7

Цикл в данном программном фрагменте выполняется 5 раз (по I, начиная с 1 до 5 с шагом 1). Приведем результаты вычисления переменной S после каждого шага по I:

- I:=1; S:=1; T:=3;
- I:=2; S:=4; T:=6;
- I:=3; S:=10; T:=10;
- I:=4; S:=20; T:=15;
- I:=5; S:=35; T:=21.

Таким образом, значение переменной S = 35, а значение переменной T = 21.

Задача 2.8

Здесь цикл выполняется 5 раз (по J, начиная с 1 до 5). Приведем результаты вычисления переменной S после каждого шага:

- `J:=1; S:=1;`
- `J:=2; S:=3;`
- `J:=3; S:=6;`
- `J:=4; S:=10;`
- `J:=5; S:=15.`

Таким образом, правильный ответ: `S = 15`.

Задача 2.9

Цикл выполняется 5 раз (по I, начиная с 2 до 6 с шагом 1). Приведем результаты вычисления переменной S после каждого шага:

- `I:=2; S:=4` (остаток от деления 4 на 2 равен 0);
- `I:=3; S:=5` (остаток от деления 4 на 3 равен 1);
- `I:=4; S:=6` (остаток от деления 5 на 4 равен 1);
- `I:=5; S:=7` (остаток от деления 6 на 5 равен 1);
- `I:=6; S:=8` (остаток от деления 7 на 6 равен 1).

Таким образом, правильный ответ: `S = 8`.

Задача 2.10

В данном случае следует проанализировать выполнение цикла в соответствии с представленным в виде блок-схемы алгоритмом. Выберем для анализа точку 1 на блок-схеме и будем проверять значения переменных A и C при каждом прохождении цикла:

- 1-й шаг: `A:=6, C:=2;`
- 2-й шаг: `A:=5, C:=4;`
- 3-й шаг: `A:=4, C:=8;`
- 4-й шаг: `A:=3, C:=16;`
- 5-й шаг: `A:=2, C:=32;`
- 6-й шаг: `A:=1, C:=64.`

Таким образом, после последнего шага условие в блок-схеме будет истинным (1 = 1). В результате фрагмент завершается и после него `C = 64`.

Задача 2.11

В данном случае следует проанализировать выполнение цикла в соответствии с представленным в виде блок-схемы алгоритмом. Выберем для анализа точку 1 на блок-схеме и будем проверять значения переменных A и C при каждом прохождении цикла:

- 1-й шаг: A:=7, C:=512;
- 2-й шаг: A:=6, C:=256;
- 3-й шаг: A:=5, C:=128;
- 4-й шаг: A:=4, C:=64;
- 5-й шаг: A:=3, C:=32;
- 6-й шаг: A:=2, C:=16;
- 7-й шаг: A:=1, C:=8.

Таким образом, после последнего шага условие в блок-схеме будет истинным (1 = 1). В результате фрагмент завершается и после него C = 8.

Задача 2.12

Второй оператор в листинге 2.12 приводит к результату: B = 6. Третий оператор приводит к результату: C = -1. Последнее вычисление меняет значение B: B = 4. В результате значение переменной A = 3, B = 4, C = -1.

Задача 2.13

Переменная B принимает значение 5. После этого вычисляется новое значение переменной A, что дает 15. В результате условие выполняется и значение переменной C равняется 30 (2 умножается на 15).

Задача 2.14

Переменная B принимает значение 4. После этого вычисляется новое значение переменной A, что дает 13 (к 5 добавляется 8). В результате условие не выполняется и значение переменной C равняется -26.

Задача 2.15

Переменная B принимает значение -5. После этого вычисляется новое значение переменной A, что дает -5. В результате условие не выполняется и значение переменной C равняется -10.

Задача 2.16

После присвоения значений переменным A и B вычисляется новое значение переменной A, что дает 15. В результате условие выполняется и значение переменной C равняется 30.

ГЛАВА 3

Одномерные массивы

Во многих практических ситуациях приходится работать с большим объемом данных. Для этого удобно использовать массивы, которые являются достаточно простой в плане программирования структурой данных. Использование массива приводит к выделению в памяти набора ячеек под определенным именем. Формально определение *массива* таково: совокупность однотипных данных, хранящихся в последовательных ячейках памяти и имеющих общее имя. Ячейки называются *элементами массива*. Все элементы массива пронумерованы по порядку, а номер называется *индексом* элемента массива.

Важно отметить, что все элементы массива имеют один и тот же тип данных. Для обращения к конкретному элементу массива необходимо указать имя массива и в квадратных скобках индекс элемента.

Массивы могут быть *одномерными* и *многомерными*. В этой главе мы рассмотрим технологию работы с одномерными массивами. В языке Паскаль используется следующее описание одномерного массива:

```
Имя массива:array[тип индекса] of тип элементов;
```

Здесь `тип элементов` — это тип данных, который имеет каждый элемент массива, а `тип индекса` определяет индексы элементов массива. Пример описания массива:

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

Здесь тип индекса является интервальным и изменяется в интервале от 1 до 10, а тип элементов массива — целый. Заметим, что интервальный тип является весьма распространенным при описании индексов массивов, в последующих примерах мы и будем его использовать.

Данная глава построена на разнообразных практических примерах, рассмотрение которых позволит вам сформировать навыки практической работы с одномерными массивами.

Нахождение суммы элементов массива

Исходная ситуация традиционна для работы с массивами: дан массив A[J], где индекс J принимает значения от 1 до N. Будем считать, что элементы массива являются целыми числами (имеют тип **integer**). Сумма элементов массива вычисляется по следующей формуле:

$$S = \sum_{J=1}^{N} A[J]. \qquad (3.1)$$

Алгоритм вычисления S по формуле (3.1) простой и программа, его реализующая, не требует пояснения с использованием блок-схемы. В листинге 3.1 приведена разработка, реализующая решение исходной задачи. Отметим, что для внесения начальной информации мы воспользовались датчиком случайных чисел — стандартной функцией **random**. В программе с помощью конструкции **random**(100) мы обеспечиваем случайное заполнение элементов массива целыми числами от 0 до 99 (включая левую и правую границу интервала). В результате работы мы увидим на экране подсчитанную сумму.

Примечание

Функция **random**(M) выдает случайное целое число в интервале от 0 до M-1.

Листинг 3.1. Вычисление суммы значений элементов массива

```
program listing_3_1;
const
   N=10;
var
   A:array[1..N] of integer;
   J:integer; S:longint;
begin
  for J:=1 to N do
      A[J]:=random(100);
  S:=0;
  for J:=1 to N do
      S:= S + A[J];
   writeln('S=',S);
end.
```

Суммирование элементов массива с учетом условия

Рассмотрим задачу суммирования не всех элементов массива, а только тех, которые удовлетворяют определенному условию. В качестве условия будем считать, что суммироваться должны только те элементы, значения которых кратны 3. В листин-

ге 3.2 приведена необходимая программная разработка. Действия, которые в ней производятся, выглядят так:

- заполнение элементов массива случайными значениями;
- организация цикла по количеству элементов массива;
- проверка условия кратности значения элемента массива 3, и если это условие выполняется, то производится добавление значения рассматриваемого элемента к общей сумме.

Аналогичным образом можно использовать и другие условия при суммировании элементов массива.

Листинг 3.2. Суммирование элементов массива при условии кратности 3

```
program listing_3_2;
const
    N=10;
var
   A:array[1..N] of integer;
   J:integer; Summa:longint;
begin
  for J:=1 to N do
      begin
        A[J]:=random(100);
        writeln(A[J]);
      end;
  Summa:=0;
  for J:=1 to N do
       begin
           if ( A[J] Mod 3 ) = 0 then
                  Summa:= Summa + A[J];
       end;
   writeln('Summa =', Summa);
end.
```

Нахождение среднего арифметического

Среднее арифметическое значение элементов массива вычисляется по следующей формуле:

$$S = \frac{1}{N}\sum_{J=1}^{N} A[J]. \qquad (3.2)$$

Алгоритм вычисления соотношения (3.2) похож на ранее рассмотренное вычисление суммы элементов массива, и в листинге 3.3 приведена программа, реализующая решение данной задачи.

Листинг 3.3. Вычисление среднего арифметического значения элементов массива

```
program listing_3_3;
const
   N=10;
var
   A:array[1..N] of integer;
   J:integer; S:real;
begin
  for J:=1 to N do
      A[J]:=random(100);
  S:= 0;
  for J:=1 to N do
      S:= S + A[J];
  S:= S/N;
  writeln('Srednee=',S);
end.
```

Нахождение среднего арифметического при условии

Внесем небольшие изменения в предыдущую программу. Так, нам необходимо подсчитать среднее значение только для тех элементов, которые расположены в интервале от -15 до 20 (включая границы интервала). При заполнении массива обеспечим внесение в него наряду с положительными значениями еще и отрицательные. Для этого при случайном формировании значений к результату, выдаваемому функцией `random(L)`, следует добавить постоянное отрицательное число:

```
A[J]:=random(100)-50.
```

Программная разработка, решающая поставленную задачу, приведена в листинге 3.4. Здесь для подсчета элементов массива, участвующих в расчете среднего значения, используется переменная `M`.

Листинг 3.4. Вычисление среднего значения элементов массива при условии

```
program listing_3_4;
const
   N=10;
var
   A:array[1..N] of integer;
   J,M:integer; S:real;
begin
  for J:=1 to N do
    begin
              A[J]:=random(100)-50;
```

```
      writeln(A[J]);
    end;
  S:= 0;
  M:=0;
  for J:=1 to N do
      begin
          if ( A[J] >= -15) and ( A[J] <= 20)  then
              begin
                  M:=M+1;
                  S:= S + A[J];
              end;
      end;
  S:= S / M;
  writeln('Srednee=',S);
end.
```

Поиск максимального элемента в массиве

В листинге 3.5 приведена программа поиска максимального элемента в числовом массиве. Сам массив предварительно заполняется случайными целыми числами. Блок-схема алгоритма показана на рис. 3.1.

Листинг 3.5. Поиск максимального элемента в массиве

```
program listing_3_5;
const
    N=10;
var
   A:array[1..N] of integer;
   J,Max:integer;
begin
  for J:=1 to N do
    begin
       A[J]:=random(100);
       writeln(A[J]);
    end;
  Max:= A[1];
  for J:=2 to N do
     begin
       if A[J]>Max then
              Max:=A[J];
     end;
  writeln('Максимальный элемент =',Max);
end.
```

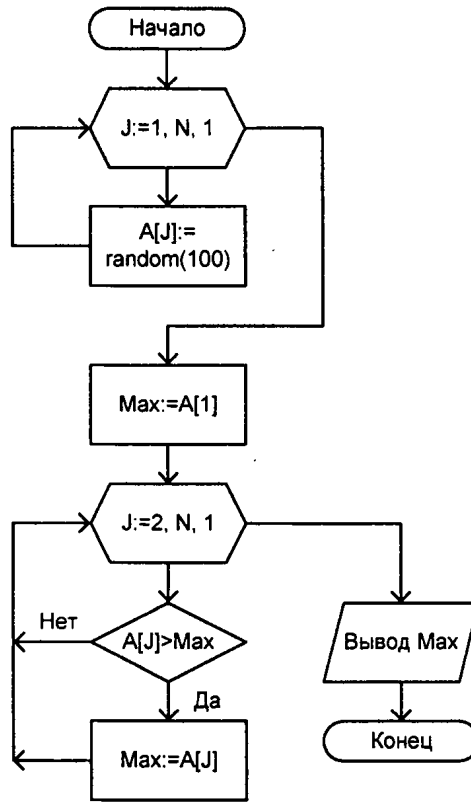


Рис. 3.1. Блок-схема к программе листинга 3.5

Поиск индексов в массиве

Рассмотрим еще один пример. Пусть наша задача заключается в том, чтобы найти индексы максимального и минимального элементов в массиве. В листинге 3.6 приведена программа поиска указанных индексов для массива элементов, предварительно заполненного случайными числами. Блок-схема алгоритма показана на рис. 3.2.

Примечание

Если максимальных или минимальных элементов несколько, то данная программа отображает только индексы первых найденных элементов.

Листинг 3.6. Поиск индексов минимального и максимального элементов массива

```
program listing_3_6;
const
    N=10;
var
   A:array[1..N] of integer;
   J,Jmax,Jmin,Max,Min:integer;
```

```
begin
  for J:=1 to N do
      A[J]:=random(100);
  Max:= A[1];
  Min:= A[1];
  Jmax:=1;
  Jmin:=1;
  for J:=2 to N do
       begin
           if A[J]>Max then
               begin
                  Max:=A[J];
                  Jmax:=J;
               end;
           if A[J]<Min then
               begin
                  Min:=A[J];
                  Jmin:=J;
               end;
       end;
   writeln('Индекс максимального элемента равен', Jmax);
   writeln('Индекс минимального элемента равен', Jmin);
end.
```

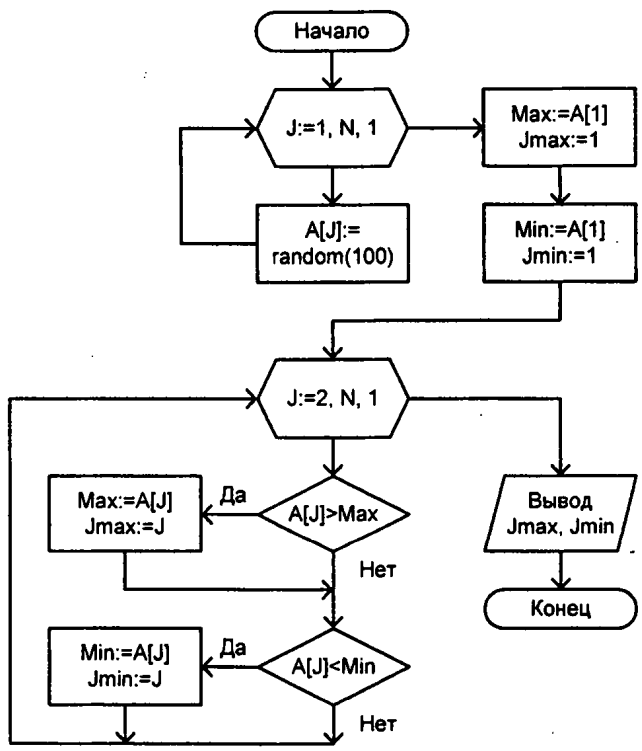


Рис. 3.2. Блок-схема к программе листинга 3.6

Проверка упорядоченности массива

Иногда возникает необходимость проверки упорядоченности массива — все ли элементы расставлены в порядке возрастания. В листинге 3.7 приведена программа, которая попарно анализирует соседние элементы массива. Здесь мы ввели переменную `Flag`, в которую при нахождении нарушения упорядоченности в массиве (если текущий элемент больше последующего) заносится значение 1 (первоначально значение этой переменной инициализируется нулем). Это и является индикатором при формировании сообщения о неупорядоченности массива чисел. В противном случае, если переменная `Flag` остается равной нулю, то можно сказать, что массив упорядочен. Блок-схема алгоритма представлена на рис. 3.3.

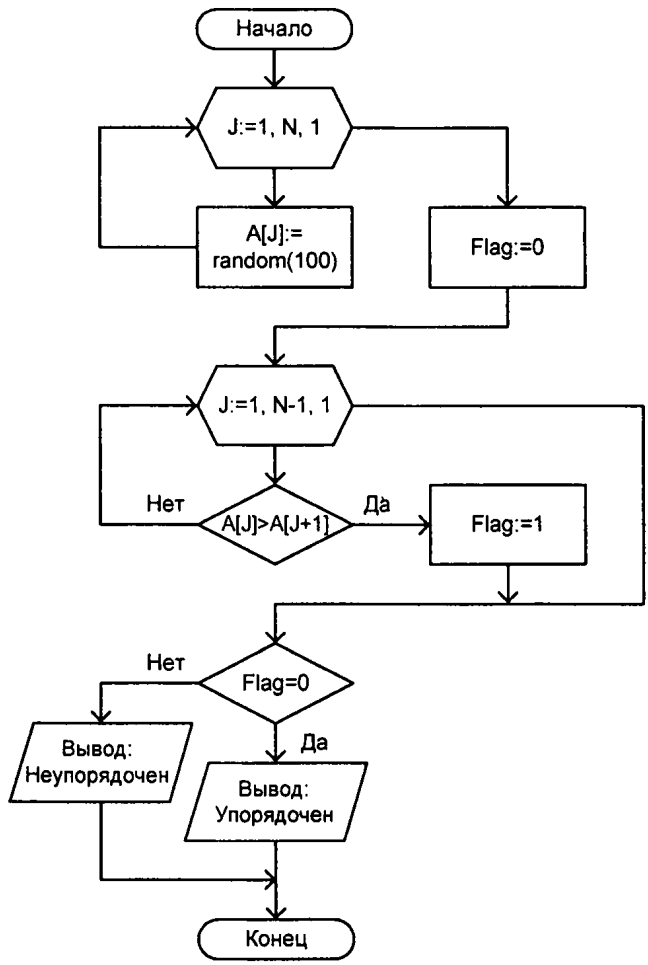


Рис. 3.3. Блок-схема к программе листинга 3.7

Листинг 3.7. Анализ упорядоченности элементов массива

```
program listing_3_7;
const
   N=10;
var
   A:array[1..N] of integer;
   J,Flag:integer;
begin
  for J:=1 to N do
      begin
        A[J]:=random(100);
        writeln(A[J]);
      end;
  Flag:=0;
  for J:=1 to N-1 do
       begin
           if A[J]>A[J+1] then
              begin
                  Flag:=1;
                  break;
              end;
        end;
   if Flag = 0 then
      writeln('Массив упорядочен')
   else
      writeln('Массив неупорядочен');
end.
```

Обмен значений массива

Обмен значений элементов массива достаточно часто встречается при работе с массивами. В программе, приведенной в листинге 3.8, производится следующий вариант обмена значений: первый элемент меняется значением с последним, второй с предпоследним и т. д. На рис. 3.4 показана данная схема обмена.

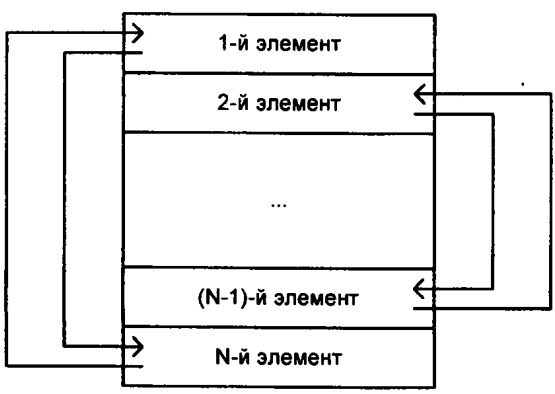


Рис. 3.4. Блок-схема обмена

Сам алгоритм несложен, а его идея заключается в том, что сначала вычисляется количество обменов. Для этого следует число элементов массива поделить на 2 (выполнить целочисленное деление). После этого в цикле по числу обменов выполняется обмен значений элементов массива. Блок-схема алгоритма представлена на рис. 3.5.

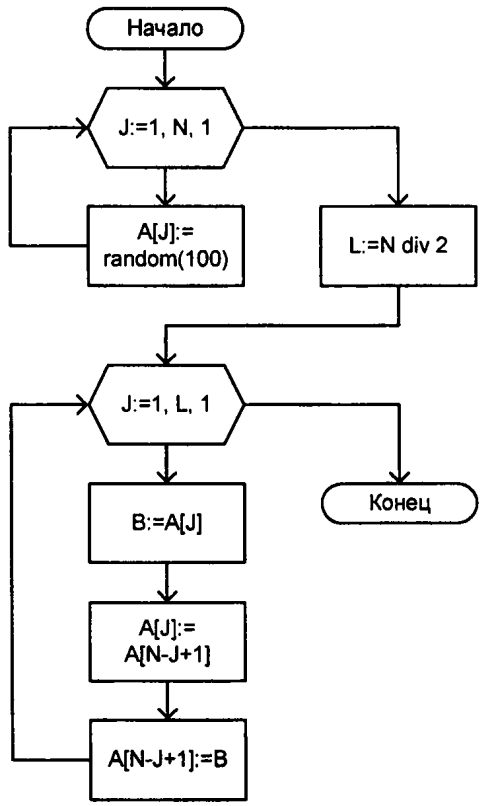


Рис. 3.5. Блок-схема к программе листинга 3.8

Листинг 3.8. Обмен значений элементов массива

```
program listing_3_8;
const
   N=7;
var
   A:array[1..N] of integer;
   J,B,L:integer;
begin
  for J:=1 to N do
    begin
       A[J]:=random(100);
       writeln(A[J]);
    end;
```

```
  L:= N div 2;
  for J:=1 to L do
       begin
           B:= A[J];
           A[J]:= A[N-J+1];
           A[N-J+1]:= B;
       end;
  writeln;
  for J:=1 to N do
       writeln(A[J]);
end.
```

Суммирование соседних элементов массива

Необходимо разработать алгоритм поиска номера первого из двух последовательных элементов в целочисленном массиве, состоящем из 30 элементов, сумма которых максимальна (если таких пар несколько, то можно выбрать любую). Решение этой задачи похоже на поиск номера максимального элемента, а отличием является только то, что здесь требуется анализировать сумму двух соседних элементов массива.

Примечание

Данная задача встретилась в одном из вариантов ЕГЭ в прошлые годы.

В листинге 3.9 приведена необходимая программная разработка. В начале программы в переменную SumMax заносится сумма первых двух элементов, а далее последовательно сравниваются с SumMax суммы следующих пар элементов. В случае когда сумма элементов очередной пары оказывается больше SumMax, то это приводит к обновлению SumMax и фиксированию индекса первого элемента пары в переменной Jmax. В конце программы значение Jmax выводится на печать. Блок-схема алгоритма представлена на рис. 3.6.

Листинг 3.9. Поиск пары соседних элементов с максимальной суммой

```
program listing_3_9;
const
    N=30;
var
   A:array[1..N] of integer;
   J,Jmax,SumMax:integer;
begin
  for J:=1 to N do
      A[J]:=random(100);
  SumMax:= A[1]+A[2];
  Jmax:=1;
```

```
  for J:=2 to N-1 do
      begin
         if ( A[J]+A[J+1] ) > SumMax  then
              begin
                  Jmax:=J;
                  SumMax:= A[J]+A[J+1];
              end;
       end;
   writeln('Jmax =',Jmax);
end.
```

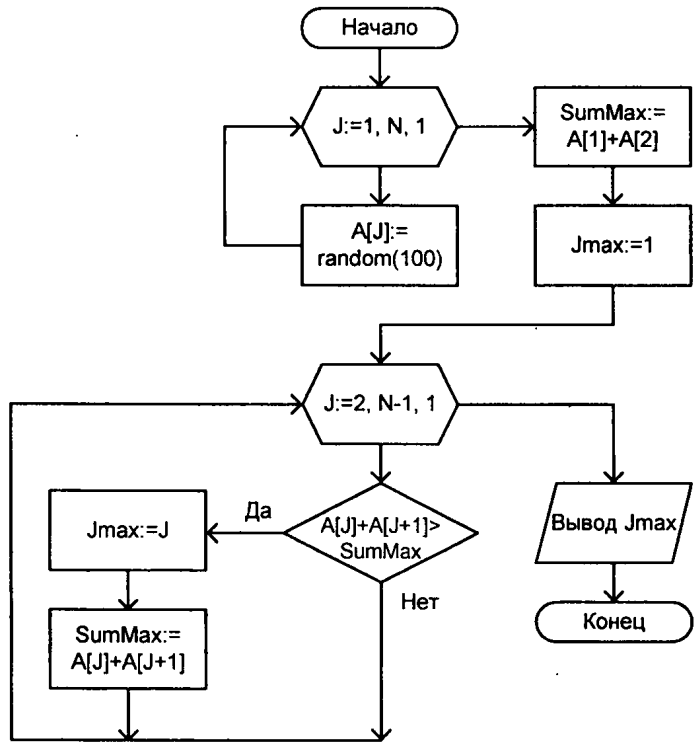


Рис. 3.6. Блок-схема к программе листинга 3.9

Подсчет соседних элементов по условию

Необходимо разработать алгоритм подсчета максимального количества идущих подряд совпадающих элементов в целочисленном массиве, который имеет длину 30. Программа, решающая данную задачу, представлена в листинге 3.10. Учитывая некоторую неочевидность алгоритма, на рис. 3.7 приведена блок-схема, которая предоставляет необходимую информативность. Здесь мы для фиксирования максимального количества подряд идущих элементов определили переменную `MaxNums`, а для хранения текущего числа совпадающих элементов ввели переменную `Nums`.

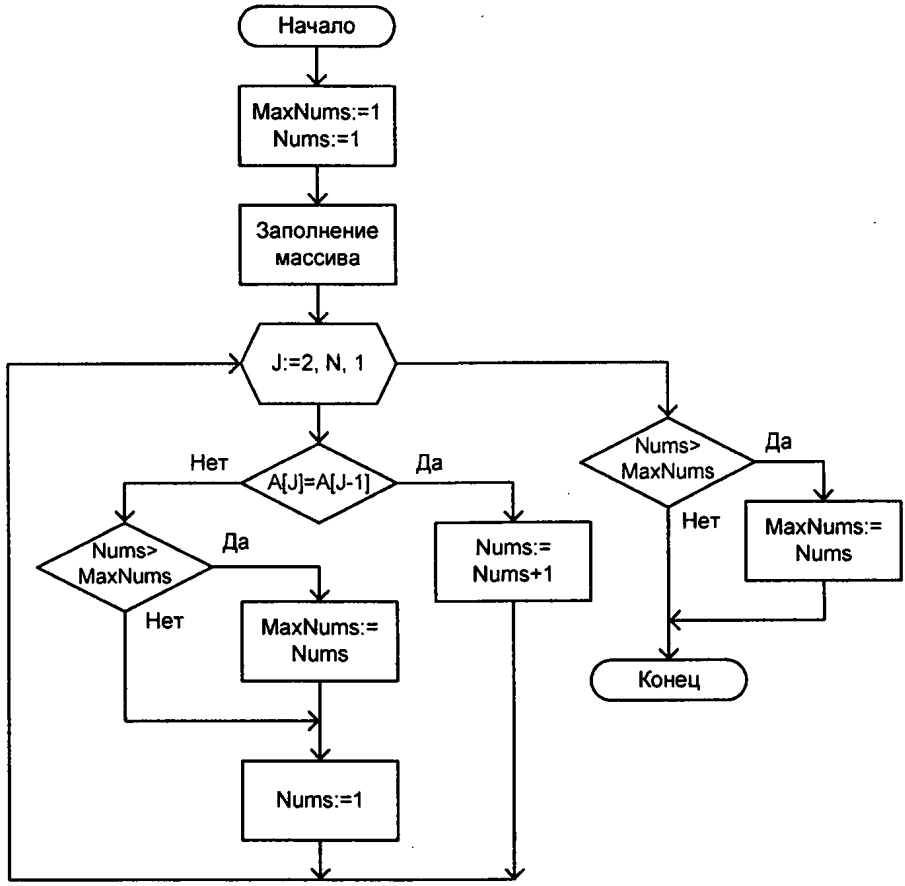


Рис. 3.7. Блок-схема к программе листинга 3.10

В цикле анализируется текущий и предыдущий элементы массива. Если они оказываются равными, то увеличивается значение счетчика `Nums` и анализируется значение следующего элемента массива. Если же предыдущий и текущий элементы не совпадают, то проверяется, больше ли `Nums`, чем `MaxNums`. Если больше, то значение `Nums` фиксируется в качестве нового значения `MaxNums`. И в том, и в другом случае после этого сбрасывается в единицу значение `Nums` и начинается анализ следующего элемента массива.

Примечание

Данная задача встретилась в одном из вариантов ЕГЭ в прошлые годы.

Листинг 3.10. Подсчет максимального количества подряд идущих элементов массива

```
program listing_3_10;
const
   N=30;
```

```
var
   A:array[1..N] of integer;
   J,Nums,MaxNums:integer;
 begin
  Nums:=1;
  MaxNums:=1;
  for J:=1 to N do
      A[J]:=random(10);
  for J:=2 to N do
       begin
          if A[J]=A[J-1] then
            Nums:= Nums+1
          else
            begin
               if Nums > MaxNums  then
                     MaxNums:= Nums;
               Nums:=  1;
             end;
       end;
  if Nums > MaxNums  then
         MaxNums:= Nums;
  writeln(MaxNums);
end.
```

Перенос модулей значений в другой массив

Требуется разработать алгоритм получения из заданного целочисленного массива размером 30 элементов другого массива, который будет содержать модули значений элементов первого массива. При этом нельзя использовать специальную функцию (**abs**), вычисляющую модуль числа.

Примечание

Данная задача встретилась в одном из вариантов ЕГЭ в прошлые годы.

Здесь следует вспомнить, что модуль числа a можно записать так:

$|a|=-a$, если $a<0$;

$|a|=a$, если $a\geq 0$.

Именно это преобразование и необходимо выполнить последовательно над каждым элементом массива. В листинге 3.11 приведена программная разработка для решения данной задачи. Здесь исходным массивом является `A[J]`, а массивом, содержащим модули, — `B[J]`. При заполнении массива с помощью датчика случайных чисел мы обеспечили равновероятное генерирование как положительных, так и отрицательных чисел.

Листинг 3.11. Перенос модулей значений элементов массива

```
program listing_3_11;
const
    N=30;
var
   A,B:array[1..N] of integer;
   J:integer;
begin
  for J:=1 to N do
      A[J]:=random(100)-50;
  for J:=1 to N do
      if A[J]< 0  then
         B[J]:=-A[J]
      else
         B[J]:=A[J];
end.
```

Подсчет количества максимальных элементов

Требуется разработать программу подсчета числа элементов в массиве, значения которых равны максимальному элементу. Число элементов в массиве равно 30.

Примечание

Данная задача встретилась в одном из вариантов ЕГЭ в прошлые годы.

Наиболее простой и очевидный вариант решения (рис. 3.8) заключается в организации двух этапов:

- первоначальный просмотр массива с целью поиска максимального элемента;
- повторный просмотр массива, при котором подсчитывается число элементов, значения которых равны максимальному.

В листинге 3.12 приведено программное решение поставленной задачи данным способом.

Листинг 3.12. Подсчет количества максимальных элементов (вариант 1)

```
program listing_3_12;
const
    N=30;
var
   A:array[1..N] of integer;
   J, Max, Nums:integer;
begin
  for J:=1 to N do
      A[J]:=random(5);
```

```
  Max:=A[1];
  for J:=2 to N do
      if A[J]> Max  then
          Max:=A[J];
  Nums:=0;
  for J:=1 to N do
      if A[J] = Max  then
           Nums:= Nums+1;
end.
```

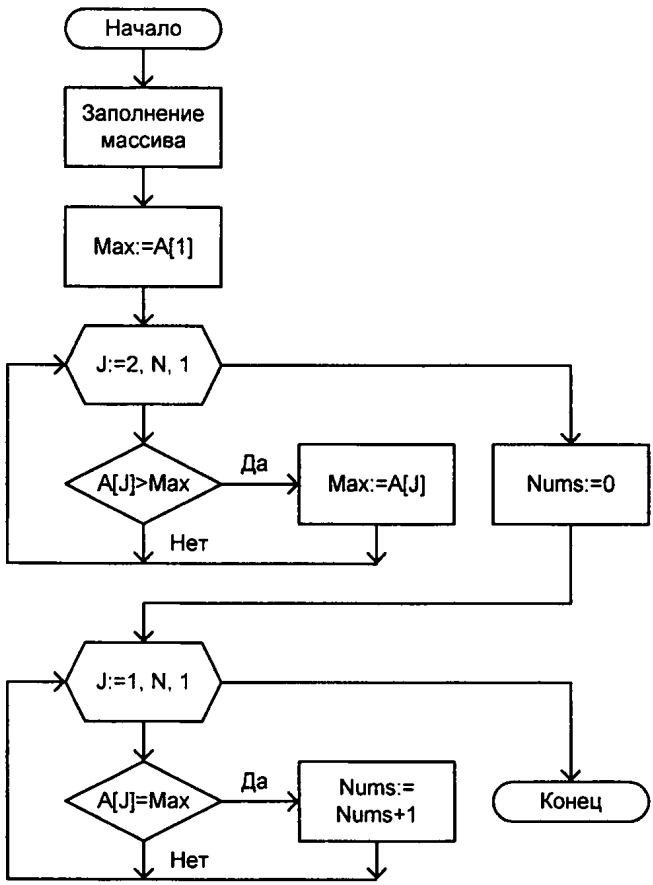


Рис. 3.8. Блок-схема к программе листинга 3.12

Однако такой вариант в инструкции для проверки Единого государственного экзамена рекомендуется оценивать как недостаточно эффективный. Это связано с тем, что в предложенном варианте решения происходит повторный просмотр элементов массива. Реализуем теперь поставленную задачу по-другому. На рис. 3.9 представлена блок-схема алгоритма. Построение решения связано с тем, что мы используем переменную Nums для подсчета количества максимальных значений массива. При

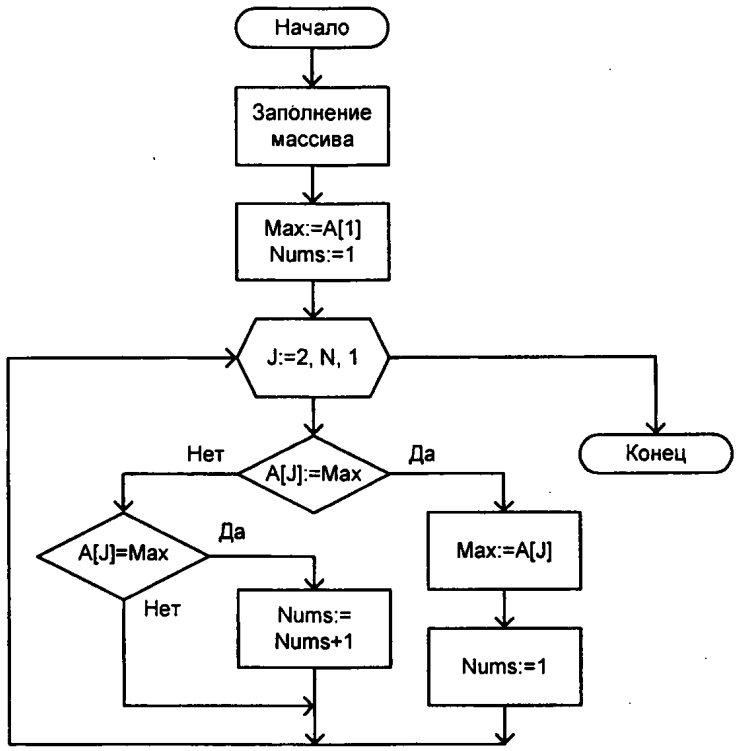


Рис. 3.9. Блок-схема к программе листинга 3.13

нахождении очередного максимума значение этой переменной сбрасывается в единицу, а при нахождении такого же максимального значения увеличивается на единицу. В листинге 3.13 приведена программная реализация алгоритма.

Листинг 3.13. Подсчет количества максимальных элементов (вариант 2)

```
program listing_3_13;
const
    N=30;
var
   A:array[1..N] of integer;
   J, Max, Nums:integer;
begin
  for J:=1 to N do
      A[J]:=random(5);
  Max:=A[1];
  Nums:=1;
  for J:=2 to N do
    begin
      if A[J]> Max  then
       begin
          Max:= A[J];
```

```
          Nums:=1
        end
      else
          if A[J] = Max  then
            Nums:= Nums+1;
   end;
end.
```

Изменение значений элементов массива с заданными свойствами

Необходимо изменить определенные значения в массиве. Так, если значение элемента в массиве отрицательное, то меняется знак элемента.

Скажем, если A[5]=-10, то после коррекции получим: A[5]=10. Кроме того, если значение элемента массива больше 100, то в элемент массива заносится ноль. В листинге 3.14 приведена необходимая программная разработка.

Листинг 3.14. Изменение значений элементов в массиве

```
program listing_3_14;
const
    N=30;
var
   A:array[1..N] of integer;
   J:integer;
begin
  for J:=1 to N do
      A[J]:=random(300)-150;
  for J:=1 to N do
    begin
      if A[J]< 0  then
           A[J]:=-A[J];
      if A[J]> 100  then
           A[J]:=0;
    end;
end.
```

Нахождение индексов элементов с заданными свойствами

Необходимо найти и вывести на экран номера элементов массива, значения которых кратны 4. В листинге 3.15 приведена необходимая программная разработка, а на рис. 3.10 представлена блок-схема алгоритма.

Листинг 3.15. Нахождение индексов элементов массива кратных 4

```
program listing_3_15;
const
    N=30;
var
   A:array[1..N] of integer;
   J:integer;
begin
  for J:=1 to N do
      A[J]:=random(100);
  for J:=1 to N do
    begin
      if  (A[J] Mod 4) = 0  then
          writeln(J);
    end;
end.
```

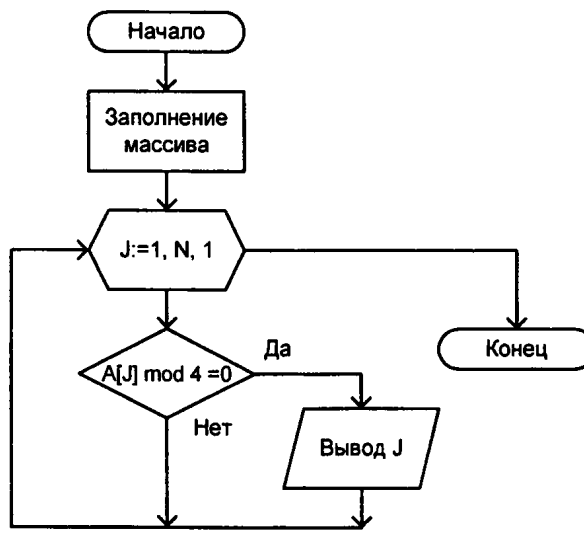


Рис. 3.10. Блок-схема к программе листинга 3.15

Удаление из массива определенного элемента

Допустим, имеется массив из N элементов. Необходимо произвести удаление элемента с определенным индексом (вводимым с клавиатуры). Один из вариантов программы показан в листинге 3.16, а на рис. 3.11 приведена блок-схема данного алгоритма.

Листинг 3.16. Удаление элемента из массива

```
program listing_3_16;
const
    N=30;
var
   A:array[1..N] of integer;
   J,K:integer;
begin
  readln(K);
  for J:=1 to N do
      A[J]:=random(100);
  for J:=K to N-1 do
      A[J]:=A[J+1];
  for J:=1 to N-1 do
      writeln(A[J]);
end.
```

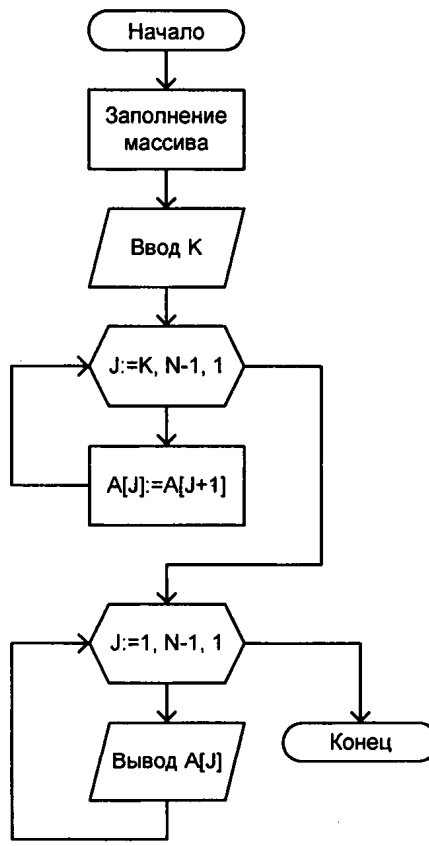


Рис. 3.11. Блок-схема к программе листинга 3.16

Циклическое перемещение элементов массива

Допустим, необходимо произвести циклическое перемещение элементов по такой схеме: второй элемент перемещается на место первого, третий на место второго и т. д. Исходный первый элемент переходит в последний элемент массива.

Один из вариантов программы показан в листинге 3.17, а на рис. 3.12 представлена блок-схема алгоритма.

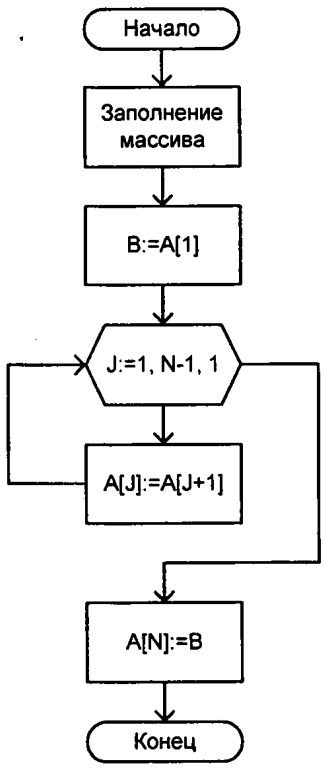


Рис. 3.12. Блок-схема к программе листинга 3.17

Листинг 3.17. Циклическое перемещение

```
program listing_3_17;
const
    N=30;
var
   A:array[1..N] of integer;
   J,B:integer;
begin
  for J:=1 to N do
     A[J]:=random(100);
```

```
  B:=A[1];
  for J:=1 to N-1  do
    begin
         A[J]:=A[J+1];
    end;
  A[N]:=B;
end.
```

Заполнение массива случайными числами

Требуется организовать заполнение массива случайными числами. При этом в массиве не должно оказаться одинаковых элементов.

Один из вариантов программы показан в листинге 3.18, а на рис. 3.13 приведена блок-схема алгоритма заполнения массива.

Листинг 3.18. Заполнение массива случайными числами

```
program listing_3_18;
const
    N=15;
var
   A:array[1..N] of integer;
   J,I,Flag:integer;
begin
 J:=1;
 while J<=N do
    begin
      A[J]:=random(N+5);
      Flag:=0;
      for I:=1 to J-1  do
        begin
          if A[J] = A[I] then
             begin
              Flag:=1;
              break;
          end;
     end;
     if Flag = 0 then
       J:=J+1;
   end;
   for J:=1 to N do
      writeln(A[J]);
end.
```

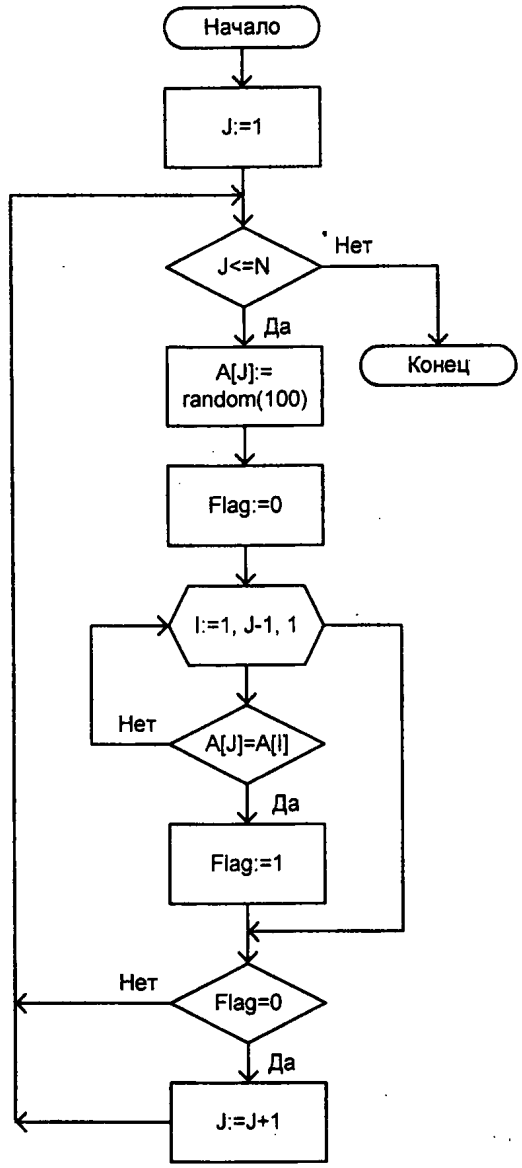


Рис. 3.13. Блок-схема к программе листинга 3.18

Нахождение суммы группы элементов массива

Дан массив целых чисел. Необходимо найти группы из трех подряд идущих элементов, сумма значений которых максимальна. Требуется вывести на экран индекс первого элемента каждой группы и сумму группы элементов. Один из вариантов программы показан в листинге 3.19.

Листинг 3.19. Нахождение группы элементов массива с максимальной суммой

```
program listing_3_19;
const
   N=30;
var
   A:array[1..N] of integer;
   J,M1,M2:integer;
begin
  for J:= 1 to N do
      A[J]:=random(100);
  M1:=A[1]+A[2]+A[3];
  for J:=2 to N-2 do
   begin
     M2:=A[J]+A[J+1]+A[J+2];
     if M2 > M1 then
        M1:=M2;
   end;
 for J:=1 to N-2 do
    begin
     M2:=A[J]+A[J+1]+A[J+2];
     if M1 = M2 then
         writeln(J,' ',M2);
   end;
end.
```

Задания из ЕГЭ за 2008 — 2010 годы

В этом разделе приведены готовые листинги программ, которые вам следует проанализировать. Создания каких-либо программных разработок здесь не требуется. В последующих задачах желательно находить правильные ответы без использования компьютера. В последнем разделе главы приведены правильные ответы.

ЗАДАЧА 3.1

Значения двух одномерных массивов A и B, размером 100 элементов каждый, задаются с помощью представленного в листинге 3.20 фрагмента программы. Вопрос: сколько элементов массива B будут иметь отрицательные значения?

Листинг 3.20. Фрагмент к задаче 3.1

```
for I:=1 to 100 do
 begin
   A(I):=50 - I
 end;
```

```
for I:=1 to 100 do
 begin
   B(I):=A(I) + 49
 end;
```

ЗАДАЧА 3.2

Значения элементов массива A задаются с помощью вложенного цикла во фрагменте, представленном в листинге 3.21. Сколько элементов массива будут кратны 4?

Листинг 3.21. Фрагмент к задаче 3.2

```
for I:=1 to 8 do
  begin
    A[I]:=I;
    A[I]:= A[I] + I*(8-I);
  end;
```

ЗАДАЧА 3.3

Значения элементов массива A размером 5 задаются с помощью цикла во фрагменте, представленном в листинге 3.22. Сколько элементов массива будут кратны 3?

Листинг 3.22. Фрагмент к задаче 3.3

```
for I:=1 to 5 do
  begin
    A[I,J]:= I * (6-I);
  end;
```

ЗАДАЧА 3.4

Значения элементов массива размером 5 задаются с помощью вложенного цикла во фрагменте, представленном в листинге 3.23. Сколько элементов массива будут иметь значения больше 3?

Листинг 3.23. Фрагмент к задаче 3.4

```
for I:=1 to 5 do
  begin
   A[I]:= I*(I-2);
  end;
```

ЗАДАЧА 3.5

Значения элементов массива размером 5 задаются с помощью цикла во фрагменте, представленном в листинге 3.24. Сколько элементов массива будут иметь отрицательные значения?

Листинг 3.24. Фрагмент к задаче 3.5

```
for I:=1 to 5 do
  begin
   if ((I mod 3) = 1) or ((I mod 2) = 0)  then
       A[I]:= I*(5-I)
   else
      A[I]:= I*(4-I);
  end;
```

ЗАДАЧА 3.6

Дан фрагмент программы (листинг 3.25), обрабатывающей одномерный массив A с индексами от 0 до 10. Определить, чему будут равны элементы массива A после выполнения данного фрагмента программы.

Листинг 3.25. Фрагмент к задаче 3.6

```
for I:=0 to 10 do
       A[I]:= 7+I;
for I:=0 to 4 do
  begin
     A[10-I]:= A[I]-1;
     A[I]:= A[10-I]-1;
  end;
```

ЗАДАЧА 3.7

Дан фрагмент программы (листинг 3.26), обрабатывающей одномерный массив A с индексами от 0 до 10. Определить, чему будут равны элементы массива A после выполнения данного фрагмента программы.

Листинг 3.26. Фрагмент к задаче 3.7

```
for I:=0 to 10 do
       A[I]:= 2+I;
for I:=0 to 4 do
  begin
     A[10-I]:= A[I]-1;
     A[I]:= A[I]-2;
  end;
```

Ответы к заданиям из ЕГЭ

Задача 3.1

Проанализируем ситуацию. Во втором цикле отрицательные значения элементов массива B будут в случае, если значения массива A(I) будут иметь значения меньшие, чем -49. Это следует из того, что к значению каждого элемента массива A добавляется 49. Из верхнего цикла видно, что среди элементов массива A таковым является только одно значение (-50), которое получается при I = 100. Следовательно, и в массиве B будет одно отрицательное значение.

Задача 3.2

Проанализируем массив:

- при I:=1 значение A[I] равно 9;
- при I:=2 значение A[I] равно 14;
- при I:=3 значение A[I] равно 18;
- при I:=4 значение A[I] равно 20;
- при I:=5 значение A[I] равно 20;
- при I:=6 значение A[I] равно 18;
- при I:=7 значение A[I] равно 14;
- при I:=8 значение A[I] равно 8.

Всего в массиве 3 элемента, кратных 4.

Задача 3.3

Требуется проанализировать результат вычисления каждого элемента массива:

- при I:=1 значение A[I] равно 5;
- при I:=2 значение A[I] равно 8;
- при I:=3 значение A[I] равно 9;
- при I:=4 значение A[I] равно 8;
- при I:=5 значение A[I] равно 5.

Таким образом, в массиве один элемент, кратный 3.

Задача 3.4

Проанализируем массив:

- при I:=1 значение A[I] равно -1;
- при I:=2 значение A[I] равно 0;
- при I:=3 значение A[I] равно 3;
- при I:=4 значение A[I] равно 8;
- при I:=5 значение A[I] равно 15.

Ответ: два элемента массива будут иметь значения больше 3.

Задача 3.5

Анализ массива показывает:

- при `I:=1` значение `A[I]` равно 4;
- при `I:=2` значение `A[I]` равно 6;
- при `I:=3` значение `A[I]` равно 3;
- при `I:=4` значение `A[I]` равно 4;
- при `I:=5` значение `A[I]` равно –5.

Ответ: один элемент массива будет иметь отрицательное значение.

Задача 3.6

Сначала массив заполняется последовательными числами от 7 до 17 (всего 11 элементов массива). Далее средний элемент массива (имеющий индекс 5) остается неизменным. Первые и последние 5 значений меняются местами и корректируются. Изменение элементов массива представляет собой последовательное присваивание пяти элементам, отсчитываемым с конца, значений на единицу меньших, чем значения в симметричных ячейках (относительно центрального элемента) массива, но отсчитываемых с его начала. Значения элементов начала массива затем сразу же заменяются на значения в симметричных элементах (относительно центрального элемента), уменьшенные еще на одну единицу.

Ответ: после заполнения массива получаем: 5 6 7 8 9 12 10 9 8 7 6.

Задача 3.7

Сначала массив заполняется последовательными числами от 2 до 12 (всего 11 элементов массива). Далее средний элемент массива (имеющий индекс 5) остается неизменным. Первые и последние 5 значений меняются местами.

Изменение элементов массива представляет собой последовательное присваивание пяти элементам, отсчитываемым с конца, значений на единицу меньших, чем значения в симметричных ячейках (относительно центрального элемента) массива, но отсчитываемых с его начала. Значения элементов начала массива затем сразу же заменяются на значения, уменьшенные на 2.

Ответ: после заполнения массива получаем: 0 1 2 3 4 7 5 4 3 2 1.

ГЛАВА 4

Двумерные массивы

Кроме одномерных массивов, в практических задачах часто используются и двумерные массивы. Двумерный массив представляет собой таблицу из однотипных элементов, организованную по строкам и столбцам. Элемент такого массива записывается так: `A[I,J]`, где первый индекс `I` представляет собой номер строки, а второй индекс `J` является номером столбца. Местоположение каждого элемента массива в памяти компьютера определяется этими индексами.

В языке Паскаль используется следующее описание двумерного массива:

```
Имя массива:array[тип индекса строки, тип индекса столбца]
of тип элементов;
```

Описание вполне аналогичное описанию одномерного массива, только в двумерном массиве для индексов используются два параметра: `тип индекса строки, тип индекса столбца`.

Нахождение суммы элементов массива

Исходная ситуация традиционна для работы с двумерными массивами: дан массив `A[I,J]`, где индексы `I`, `J` принимают значения от 1 до `N`. В данном случае будем считать элементы массива целыми числами (имеющими тип **integer**). Сумма элементов двумерного массива вычисляется по следующей формуле:

$$S = \sum_{I=1}^{N} \sum_{J=1}^{N} A[I,J]. \qquad (4.1)$$

Алгоритм вычисления `S` по соотношению (4.1) достаточно простой и программа, его реализующая, не требует пояснения с использованием блок-схемы (как и ряд других программ, рассматриваемых в примерах этой главы). В листинге 4.1 приведена реализация решения данной задачи на Паскале. Для определенности мы установили размерность массива: 10 строк и 10 столбцов. Как и в разработках предыдущих глав, в данной программе обеспечено первоначальное заполнение элементов массива с помощью датчика случайных чисел.

Листинг 4.1. Вычисление суммы значений элементов двумерного массива

```
program listing_4_1;
const
   N=10;
var
  A:array[1..N,1.. N] of integer;
  I,J:integer;
  S:longint;
begin
  for I:=1 to N do
    for J:=1 to N do
      A[I,J]:=random(100);
  S:=0;
  for I:=1 to N do
    for J:=1 to N do
        S:= S + A[I,J];
  writeln('Сумма элементов двумерного массива=',S);
end.
```

Сумма элементов с заданными свойствами

Несколько изменим формулировку предыдущего задания. Будем считать, что требуется просуммировать только те элементы, значения которых являются нечетными числами, а кроме того, эти значения должны располагаться в интервале от 10 до 100. В листинге 4.2 приведена программа, реализующая решение данной задачи.

Листинг 4.2. Вычисление суммы значений элементов массива при условии

```
program listing_4_2;
const
  N=10;
var
  A:array[1..N,1.. N] of integer;
  I,J:integer;
  S:longint;
begin
  for I:=1 to N do
    for J:=1 to N do
      A[I,J]:=random(200);
  S:=0;
  for I:=1 to N do
   for J:=1 to N do
    if ((A[I,J] mod 2) =1) and (A[I,J] >= 10) and (A[I,J] <= 100) then
       S:= S + A[I,J];
   writeln('Сумма выбранных элементов массива равна=',S);
end.
```

Расчет среднего арифметического

В данном случае нам требуется вычислить среднее арифметическое значение только положительных значений элементов двумерного массива. В листинге 4.3 приведена программа, реализующая решение данной задачи. Здесь мы использовали две переменные:

- S — для суммы интересующих нас элементов;
- M — количество элементов, по которым подсчитывается сумма.

Листинг 4.3. Вычисление среднего арифметического

```
program listing_4_3;
const
  N=10;
var
  A:array[1..N,1..N] of integer;
  I,J,M:integer;
  S:real;
begin
  for I:=1 to N do
     for J:=1 to N do
       A[I,J]:=random(100);
  S:=0;
  M:=0;
  for I:=1 to N do
    for J:=1 to N do
      if A[I,J] > 0 then
        begin
          S:= S + A[I,J];
          M:=M+1;
        end;
 S:=S/M;
 writeln('Среднее арифметическое равно=',S);
end.
```

Поиск минимального элемента

В листинге 4.4 приведена программа поиска минимального элемента в двумерном массиве (N на M элементов), заполненном целыми числами. Под поиском подразумевается нахождение номера строки и столбца необходимого элемента в двумерном массиве. При этом, если минимальных элементов несколько, то достаточно найти индексы только одного из них. Блок-схема алгоритма приведена на рис. 4.1.

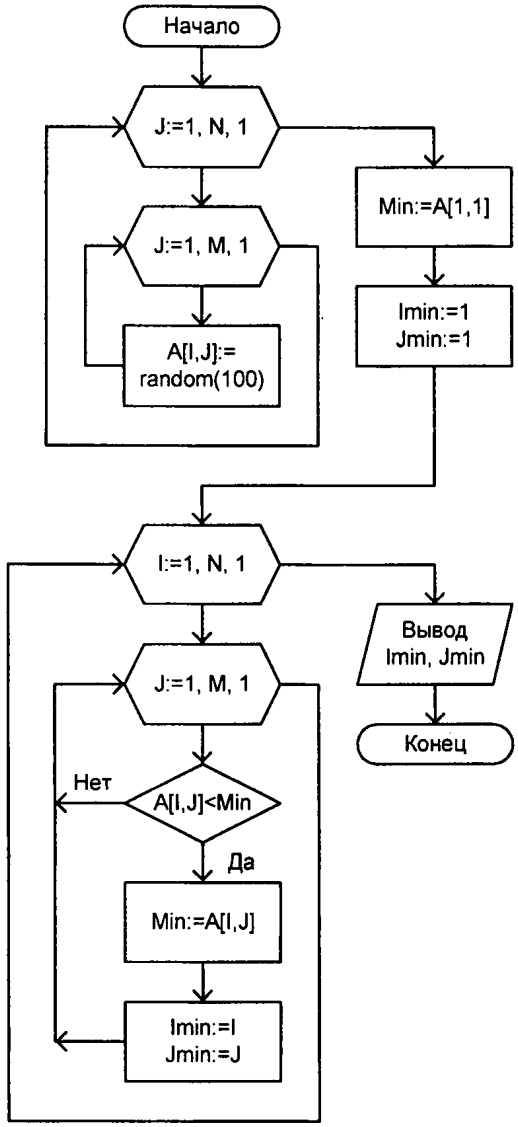


Рис. 4.1. Блок-схема к программе листинга 4.4

Листинг 4.4. Поиск индексов минимального элемента в двумерном массиве

```
program listing_4_4;
const
  N=10; M=20;
var
  A:array[1..N,1..M] of integer;
  I,J,Min,Imin,Jmin:integer;
begin
  for I:=1 to N do
```

```
    for J:=1 to M do
        A[I,J]:=random(100);
  Min:= A[1,1];
  Imin:=1; Jmin:=1;
  for I:=1 to N do
      for J:=1 to M do
       begin
          if( A[I,J] < Min) then
              begin
                 Min:=A[I,J];
                 Imin:=I; Jmin:=J;
              end
       end;
  writeln('I=',Imin,'J=',Jmin);
end.
```

Изменим предыдущую программу так, чтобы в результате на экран выводились индексы всех минимальных элементов. В этом случае (листинг 4.5) нам потребуется дополнительный цикл для просмотра всех элементов.

Листинг 4.5. Поиск индексов всех минимальных элементов в двумерном массиве

```
program listing_4_5;
const
  N=10; M=20;
var
  A:array[1..N,1..M] of integer;
  I,J,Min:integer;
begin
  for I:=1 to N do
     for J:=1 to M do
         A[I,J]:=random(100);
  Min:= A[1,1];
  for I:=1 to N do
     for J:=1 to M do
        if A[I,J] < Min then
                   Min:=A[I,J];
   for I:=1 to N do
     for J:=1 to M do
        if A[I,J] = Min then
            writeln('I=',I,'J=',J);
end.
```

Поиск номера строки с минимальной суммой

Рассмотрим задачу поиска номера строки массива, для которой сумма элементов минимальна. Алгоритм представлен на рис. 4.2, а текст программы в листинге 4.6.

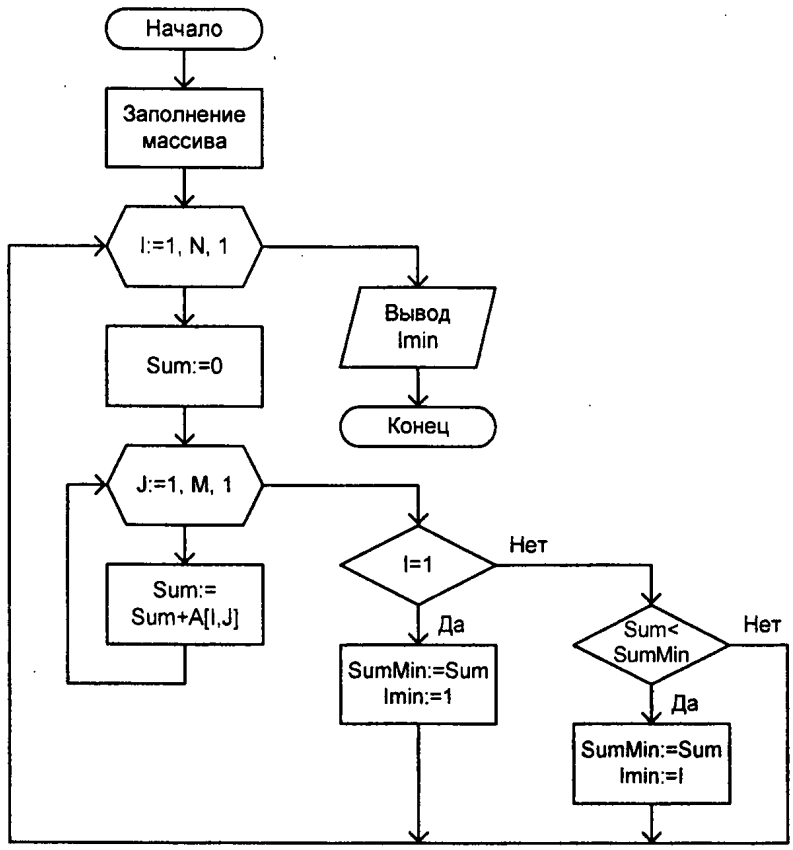


Рис. 4.2. Блок-схема к программе листинга 4.6

Листинг 4.6. Поиск номера строки с минимальной суммой элементов

```
program listing_4_6;
const
  N=10; M=20;
var
  A:array[1..N,1..M] of integer;
  I,J,Imin,Sum,SumMin:integer;
begin
  for I:=1 to N do
     for J:=1 to M do
         A[I,J]:=random(100);
```

```
for I:=1 to N do
 begin
    Sum:=0;
    for J:=1 to M do
       Sum:=Sum+A[I,J];
    if I = 1 then
      begin
         Imin:=1;
         SumMin:=Sum;
      end
    else
      begin
        if( SumMin>Sum ) then
            begin
              SumMin:=Sum;
              Imin:=I;
            end;
      end;
 end;
 writeln('Номер строки с минимальной суммой элементов',Imin);
end.
```

Подсчет числа учащихся

В двумерном массиве хранится информация о количестве учеников в каждом классе всех параллелей школы (параллели нумеруются от 1 до 11). Так, в каждой строке двумерного массива сначала располагается число классов в определенной параллели, а затем последующие элементы содержат число учащихся в каждом классе данной параллели. Пример одной из строк двумерного массива выглядит так:

4 25 23 27 28,

что означает четыре класса в параллели и, соответственно, 25 учеников в первом из классов параллели, 23 ученика во втором классе параллели и т. д.

Необходимо написать программу, которая позволит автоматически определить общее число учащихся в классах с 5-й по 10-ю параллель включительно. Для определенности будем считать, что в каждой параллели не более пяти классов. В листинге 4.7 приведена программа, решающая данную задачу. Для заполнения информации о числе классов в параллелях и числа учащихся мы использовали датчик случайных чисел. Число учащихся в классе варьируется от 20 до 30.

Листинг 4.7. Подсчет числа учащихся в параллелях

```
program listing_4_7;
const
   N=11; M=6;
```

```
var
  A:array[1..N,1..M] of integer;
  I,J,Schet:integer;
begin
  for I:=1 to N do
     A[I,1]:=random(6);
 for I:=1 to N do
   for J:=1 to A[I,1] do
        A[I,J+1]:=20+random(10);
 Schet:=0;
  for I:=5 to 10 do
      for J:=1 to A[I,1] do
         Schet:=Schet+ A[I,J+1] ;
   writeln(' Schet =', Schet);
end.
```

Изменим программу так, чтобы подсчет производился по интервалу параллелей, значения которых вводятся с клавиатуры. В листинге 4.8 приведена необходимая программа.

Листинг 4.8. Подсчет числа учащихся по произвольному интервалу параллелей

```
program listing_4_8;
const
  N=11; M=6;
var
  A:array[1..N,1..M] of integer;
  I,J,Schet,P1,P2:integer;
begin
  writeln('Ввести интервал параллелей');
  readln(P1,P2);
 for I:=1 to N do
     A[I,1]:=random(6);
 for I:=1 to N do
   for J:=1 to A[I,1] do
        A[I,J+1]:=20+random(10);
 Schet:=0;
 for I:=P1 to P2 do
   for J:=1 to A[I,1] do
         Schet:=Schet+ A[I,J+1] ;
writeln(' Schet =', Schet);
end.
```

Определение результата турнира

В двумерном массиве (N на N элементов) хранится информация о результатах хоккейного турнира. В турнире участвовали N команд и элемент массива A[I, J] со-

держит число очков, которые набрала I-я команда в игре на своем поле во встрече с J-й командой. Соответственно, элемент A[J, I] содержит число очков, которые набрала J-я команда при встрече с I-й на своем поле. Будем считать, что за победу команде дается 2 очка, в случае ничьей — одно очко, а при поражении команда очков не получает.

Необходимо определить чемпиона (или несколько команд, набравших максимальное количество очков). В листинге 4.9 приведена программа, позволяющая определить победителя (или победителей) турнира. Здесь мы сначала обеспечиваем формирование результатов турнира с помощью датчика случайных чисел. После этого производится подсчет числа очков, набранных первой командой. И этот результат принимается за максимум, который фиксируется в переменной MaxBalls. Далее последовательно вычисляются набранные очки другими командами. В результате мы получаем максимальное количество очков по итогам турнира, которые набрала команда-чемпион. При этом в дополнительном массиве B фиксируются результаты, набранные каждой командой.

Листинг 4.9. Определение победителя турнира

```
program listing_4_9;
const
   N=10;
var
   A:array[1..N,1..N] of integer;
   B:array[1..N] of integer;
   I,J,MaxBalls,Vr_Max:integer;
begin
  for I:=1 to N do
    for J:=1 to N do
     begin
       if I <> J then
             A[I,J]:=random(3)
       else
             A[I,J]:=0;
     end;
 MaxBalls:=0;
 for I:=2 to N do
    MaxBalls:= MaxBalls + A[1,I];
 for I:=2 to N do
    MaxBalls:= MaxBalls + 2 - A[I,1];
 B[1]:= MaxBalls;
 for I:=2 to N do
   begin
    Vr_Max:=0;
    for J:=1 to N do
       if I <> J then
          Vr_Max:= Vr_Max +A[I,J];
```

```
    for J:=1 to N do
       if I <> J then
          Vr_Max:= Vr_Max +2-A[J,I];
   B[I]:=Vr_Max;
   if Vr_Max > MaxBalls then
       MaxBalls:=Vr_Max;
 end;
for I:=1 to N do
    if B[I] = MaxBalls then
        writeln(I);
end.
```

Расчет доходов по отделу

В двумерном массиве хранится информация о доходах отдела (10 человек) за каждый месяц текущего года. Необходимо подсчитать общую сумму по отделу за каждый квартал. В листинге 4.10 приведена необходимая программа, где после внесения оплат производится группировка данных по кварталам. Отметим назначение двух переменных:

- `L1` — для обозначения первого месяца квартала;
- `L2` — для обозначения последнего месяца квартала.

Листинг 4.10. Расчет доходов по кварталам

```
program listing_4_10;
const
   N=10; M=12;
var
   A:array[1..N,1..M] of integer;
   I,J,L1,L2,L:integer;
   Sum:longint;
begin
  { Формирование зарплат }
  for I:=1 to N do
    for J:=1 to N do
       A[I,J]:=7000+500*random(10);
 { Расчет по кварталам }
 for I:=1 to 4 do
   begin
     Sum:=0;
     L1:=1+(I-1)*3;
     L2:=3+(I-1)*3;
     for J:=1 to N do
       for L:= L1 to L2 do
         Sum:=Sum+A[J,L];
```

```
    writeln(I,Sum)
  end;
end.
```

Анализ средней зарплаты сотрудников

В двумерном массиве хранится информация о доходах отдела (10 человек) за каждый месяц в течение года. Необходимо подсчитать число сотрудников, у которых доходы за год оказались выше средних по отделу. В листинге 4.11 приведена необходимая программа.

Листинг 4.11. Подсчет числа сотрудников по условию

```
program listing_4_11;
const
   N=10; M=12;
var
  A:array[1..N,1..M] of integer;
  I,J,Col:integer;
  Srednee,Sr:real;
begin
  for I:=1 to N do
    for J:=1 to M do
        A[I,J]:=7000+500*random(10);
 Srednee:=0;
 for I:=1 to N do
   for J:=1 to M do
      Srednee:=Srednee+A[I,J];
 Srednee:= Srednee/10;
 Col:=0;
 for I:=1 to N do
  begin
    Sr:=0;
    for J:=1 to M do
       Sr:=Sr+A[I,J];
    if Sr > Srednee then
       Col:=Col+1;
  end;
end.
```

Подсчет элементов по условию

Задан двумерный числовой массив размерности N на N. Необходимо подсчитать число положительных элементов, расположенных выше главной диагонали (располагается от элемента A[1,1] до элемента A[N,N]). В листинге 4.12 приведена необходимая программа.

Просмотр элементов, расположенных выше главной диагонали, реализуется с помощью следующего вложенного цикла:

```
for I:=1 to N do
  for J:=I+1 to N do
```

Листинг 4.12. Подсчет числа элементов по условию

```
program listing_4_12;
const
  N=10;
var
  A:array[1..N,1..N] of integer;
  I,J,Cols:integer;
begin
  for I:=1 to N do
    for J:=1 to N do
        A[I,J]:=-50+random(100);
 Cols:=0;
 for I:=1 to N do
   for J:=I+1 to N do
     if A[I,J] > 0 then
        Cols:=Cols+1;
end.
```

Подсчет суммы элементов по условию

Задан двумерный числовой массив размерности N на N. Необходимо подсчитать сумму отрицательных элементов, расположенных выше главной диагонали, которая располагается от элемента A[1,N] до элемента A[N,1]. В листинге 4.13 приведена необходимая программа.

Листинг 4.13. Подсчет суммы элементов по условию

```
program listing_4_13;
const
  N=10;
var
  A:array[1..N,1..N] of integer;
  I,J,Sum:integer;
begin
  for I:=1 to N do
     for J:=1 to N do
         A[I,J]:=-50+random(100);
```

```
 for I:=1 to N do
   for J:=1 to N-I do
     if A[I,J] < 0 then
        Sum:=Sum+1;
 writeln(Sum);
end.
```

Нахождение индексов элементов

Задан двумерный числовой массив размерностью N на M. Необходимо вывести на экран индексы первых (начиная с начальных столбцов) положительных элементов в каждой строке. В листинге 4.14 приведена необходимая программа.

Листинг 4.14. Нахождение индексов первых положительных элементов

```
program listing_4_14;
const
  N=10; M=20;
var
  A:array[1..N,1..M] of integer;
  I,J:integer;
begin
  for I:=1 to N do
     for J:=1 to M do
          A[I,J]:=-50+random(100);
 for I:=1 to N do
    for J:=1 to M do
        if A[I,J]> 0 then
           begin
              writeln(I,' ',J);
              break;
           end;
end.
```

Рассмотрим еще одну похожую ситуацию. Пусть задан двумерный числовой массив размерностью N на M. Необходимо вывести на экран индексы последних отрицательных элементов в каждой строке. В листинге 4.15 приведена необходимая программа.

Листинг 4.15. Нахождение индексов отрицательных элементов

```
program listing_4_15;
const
  N=10; M=20;
var
  A:array[1..N,1..M] of integer;
  I,J:integer;
```

```
begin
for I:=1 to N do
     for J:=1 to M do
          A[I,J]:=-50+random(100);
  for I:=1 to N do
    for J:=M downto 1 do
        if A[I,J]< 0 then
           begin
              writeln(I,' ',J);
              break;
           end;
end.
```

Нахождение уникальных элементов

Задан двумерный числовой массив размерностью N на M. Необходимо вывести на экран только уникальные элементы в каждой строке. Так, если в определенной строке есть одинаковые элементы, то их выводить не надо. В листинге 4.16 приведена необходимая программа. Здесь перед выводом очередного элемента мы предварительно анализируем предыдущие элементы в данной строке.

Листинг 4.16. Нахождение уникальных элементов в каждой строке

```
program listing_4_16;
const
   N=10; M=10;
var
  A:array[1..N,1..M] of integer;
  I,J,L,Flag:integer;
begin
  for I:=1 to N do
     for J:=1 to M do
          A[I,J]:=-5+random(10);
  for I:=1 to N do
   begin
    for J:=1 to M do
      begin
         Flag:=0;
         for L:=1 to J-1  do
           begin
                if A[I,J]= A[I,L] then
                  begin
                       Flag:=1;
                       break;
                  end;
           end;
```

```
        if Flag=0 then
           write(A[I,J],' ');
      end;
    writeln;
   end;
end.
```

Анализ тестирования

В двумерном массиве размерностью N на M хранится информация о результатах тестирования среди учащихся по пяти темам. Необходимо вывести номера участников (номера строк двумерного массива), которые в сумме набрали более 100 баллов. В листинге 4.17 приведена необходимая программа.

Листинг 4.17. Анализ тестирования учащихся

```
program listing_4_17;
const
  N=10; M=5;
var
  A:array[1..N,1..M] of integer;
  I,J,Sum:integer;
begin
  for I:=1 to N do
      for J:=1 to M do
        A[I,J]:=random(50);
  for I:=1 to N do
    begin
     Sum:=0;
     for J:=1 to M do
        Sum:=Sum+A[I,J];
     if Sum > 100  then
        writeln(I);
    end;
end.
```

Изменение знака элементов

В двумерном массиве размерностью N на M элементов необходимо сделать следующее преобразование: в каждой строке у последнего отрицательного элемента следует поменять знак. В листинге 4.18 приведена необходимая программа.

Листинг 4.18. Изменение знака у правых крайних отрицательных элементов

```
program listing_4_18;
const
   N=10; M=20;
```

```
var
  A:array[1..N,1..M] of integer;
  I,J:integer;
begin
  for I:=1 to N do
     for J:=1 to M do
        A[I,J]:=-50+random(100);
  for I:=1 to N do
    for J:=M downto 1 do
        if A[I,J]< 0 then
           begin
              A[I,J]:=-A[I,J];
              break;
           end;
end.
```

Изменение элементов по условию

В двумерном массиве размерностью N на M элементов необходимо сделать следующее преобразование: в каждой строке максимальный элемент умножить на 10. В листинге 4.19 приведена необходимая программа.

Листинг 4.19. Изменение элементов по условию

```
program listing_4_19;
const
   N=10; M=20;
var
  A:array[1..N,1..M] of integer;
  I,J,Maxsim:integer;
begin
  for I:=1 to N do
    for J:=1 to M do
       A[I,J]:=-50+random(100);
 for I:=1 to N do
  begin
    Maxsim:=A[I,1];
    for J:=2 to M do
        if A[I,J] > Maxsim then
             Maxsim:= A[I,J];
    for J:=1 to M do
        if A[I,J] = Maxsim then
             A[I,J]:= A[I,J]*10;
  end;
end.
```

Тур коня на шахматной доске

Рассмотрим задачу, которая заключается в разработке программы, моделирующей ходы коня на шахматной доске. Сформулируем словесное описание разработки. Пользователь выбирает произвольное положение шахматной фигуры, а затем программа автоматически продвигает коня по шахматной доске с условием, что дважды побывать на одной клетке конь не может. Таким образом маршрут коня завершается, когда нет возможности перехода с текущего поля ни на одну, еще не посещенную клетку.

В качестве аналога шахматной доски выберем двумерный массив 8 на 8 элементов.

Как известно, в шахматах конь может пойти из заданного поля максимум на восемь других полей доски. Если фигура расположена на краю доски, то число возможных ходов уменьшается. Также, если поле для хода уже занято, то конь пойти на него не может.

Варианты ходов шахматного коня с произвольного поля показаны на рис. 4.3, при этом нумерация вариантов в дальнейшем будет отражаться в алгоритме маршрута именно в таком виде. Например, вариант с номером 2 связан с перемещением фигуры на одну клетку вверх и две клетки вправо.

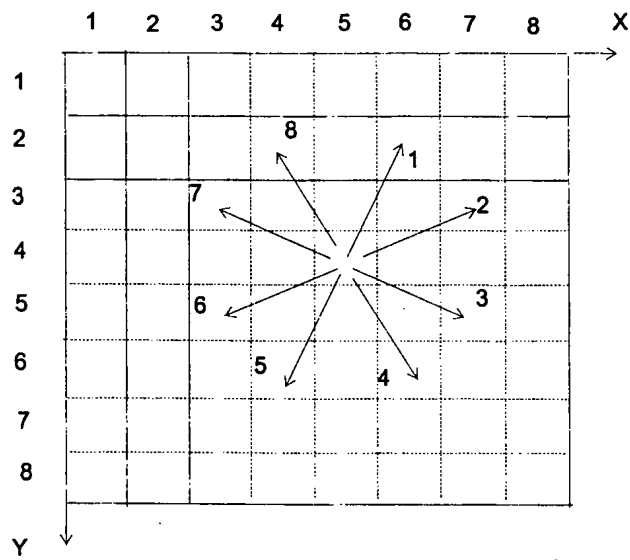


Рис. 4.3. Возможные ходы коня

Сформулируем алгоритм выбора очередного хода. В цикле последовательно просматриваются возможные варианты ходов из текущего поля. Просмотр возможных полей осуществляется последовательно, начиная с первого варианта. Если очередной рассматриваемый вариант хода возможен (данная клетка располагается в пределах доски и конь на этой клетке еще не был), то производится перемещение фигуры в соответствии с этим вариантом. Если ход невозможен, то просматривается

следующий вариант и далее, вплоть до восьмого. Если ни один ход из текущей клетки невозможен, то тур шахматного коня на этом завершается.

Для программного запоминания ходов коня нам потребуется двумерный массив `A[N, N]`. Возможные ходы коня будем называть вариантами, а для обозначения вариантов введем переменную `K`, которая может принимать значения от 1 до 8.

Например, при третьем варианте хода (`K:=3`) к текущему значению координаты `X` добавляется значение элемента массива `Dx[3]`, а к текущему значению координаты `Y` добавляется значение `Dy[3]`.

Таблица 4.1. *Данные для выполнения хода шахматного коня*

K	Новое значение Y	Новое значение X	Значение элемента массива Dy(K)	Значение элемента массива Dx(K)
1	Y-2	X+1	-2	1
2	Y-1	X+2	-1	2
3	Y+1	X+2	1	2
4	Y+2	X+1	2	1
5	Y+2	X-1	2	-1
6	Y+1	X-2	1	-2
7	Y-1	X-2	-1	-2
8	Y-2	X-1	-2	-1

Блок-схема алгоритма приведена на рис. 4.4, а программная реализация — в листинге 4.20.

Листинг 4.20. Тур шахматного коня

```
program listing_4_20;
const
  Dx: array[1..8] of integer =(1,2,2,1,-1,-2,-2,-1);
  Dy: array[1..8] of integer =(-2,-1,1,2,2,1,-1,-2);
var
   Mass: array[1..8,1..8] of integer;
   I,J,K,X,Y,VarX,VarY,Nom:integer;
begin
  for J:=1 to 8 do
    for I:=1 to 8 do
       Mass[I,J]:=0;
   writeln('Ввод номера поля по горизонтали ');
   readln(X);
   writeln('Ввод номера поля по вертикали ');
   readln(Y);
   Nom:=1;
```

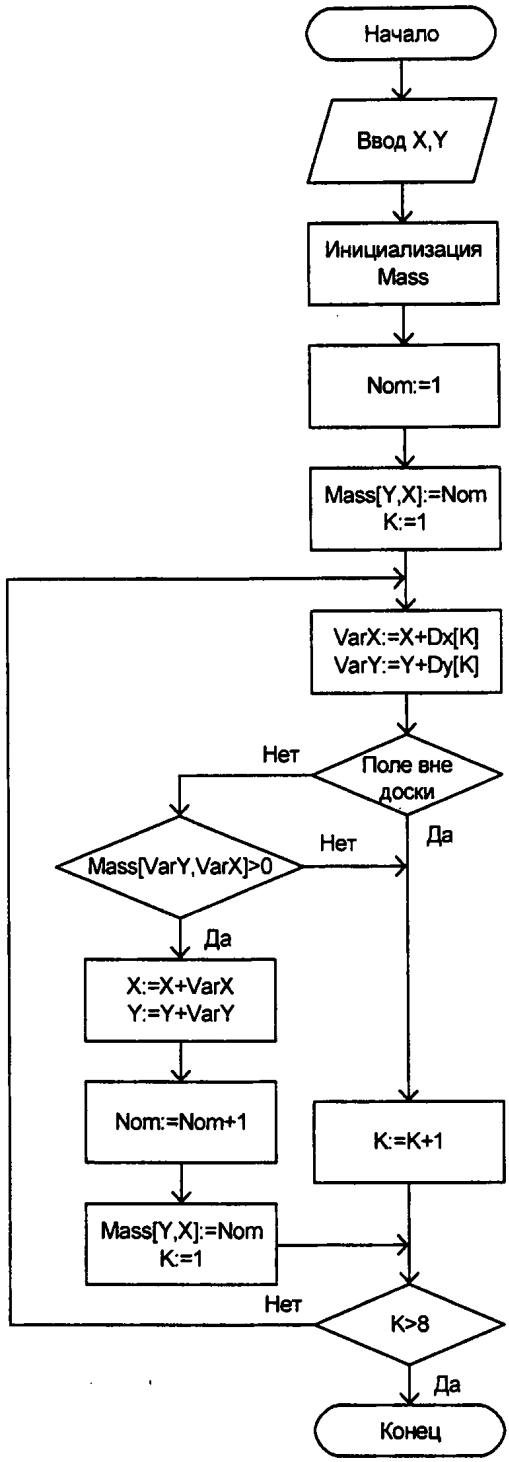


Рис. 4.4. Блок-схема к программе листинга 4.20

```
   K:=1;
   Mass[Y,X]:= Nom ;
   repeat
    VarX:=X+Dx[K];
    VarY:=Y+Dy[K];
    if ( VarX < 1) Or ( VarX >8) Or ( VarY < 1) Or ( VarY > 8) then
        K:=K+1
    else
         if Mass[VarY , VarX ]> 0 then
              K:=K+1
         else
            begin
               X:=VarX;
               Y:=Vary;
               Nom:=Nom+1;
               Mass[Y,X]:= Nom ;
               K:=1
            end
    until K>8;
    for I:=1 to 8 do
      begin
         writeln;
         for J:=1 to 8 do
            write (Mass[I,J]:3);
         writeln;
      end;
end.
```

Задания из ЕГЭ за 2008 — 2010 годы

В заданиях данного раздела требуется проанализировать программный код без использования компьютера и сделать правильный вывод. В последнем разделе главы приведены правильные ответы.

ЗАДАЧА 4.1

Дан фрагмент программы, который представлен в листинге 4.21. Чему равно значение `C[4,3]`, если перед приведенными командами значение `C[4,3]=10`.

Листинг 4.21. Фрагмент программы к задаче 4.1

```
for N:=1 to 6 do
for M:=1 to 5 do
  begin
     C[N,M]:= C[N,M]+(2*N-M);
  end
```

ЗАДАЧА 4.2

Значения элементов двумерного массива A размером 5 на 5 задаются с помощью вложенного цикла во фрагменте, представленном в листинге 4.22. Сколько элементов массива будут больше 10?

Листинг 4.22. Фрагмент к заданию 4.2

```
for I:=1 to 5 do
for J:=1 to 5 do
  begin
     A[I,J]:= I * J;
  end;
```

ЗАДАЧА 4.3

Значения элементов двумерного массива A размером 5 на 5 задаются с помощью вложенного цикла во фрагменте, представленном в листинге 4.23. Сколько элементов массива будут больше 2?

Листинг 4.23. Фрагмент к задаче 4.3

```
for I:=1 to 5 do
for J:=1 to 5 do
  begin
     A[I,J]:= I mod J;
  end;
```

ЗАДАЧА 4.4

Значения элементов двумерного массива размером 7 на 7 задаются с помощью вложенного цикла во фрагменте, представленном в листинге 4.24. Сколько элементов массива будут иметь значения больше 0?

Листинг 4.24. Фрагмент к задаче 4.4

```
for I:=1 to 7 do
  for J:=1 to 7 do
  begin
     B[I,J]:= J - I;
  end;
```

ЗАДАЧА 4.5

Значения элементов двумерного массива A размером 4 на 4 первоначально были равны нулю. Затем значения элементов меняются с помощью вложенного операто-

ра цикла во фрагменте программы, представленном в листинге 4.25. Сколько элементов массива будут равны 1?

Листинг 4.25. Фрагмент к задаче 4.5

```
for N:=1 to 4 do
  for K:=N to 4 do
    begin
      A[N,K]:= A[N,K] + 1;
      A[K,N]:= A[K,N] + 1;
    end;
```

ЗАДАЧА 4.6

Дан фрагмент программы, который представлен в листинге 4.26. Чему равно значение `C[3,5]` после выполнения данного фрагмента.

Листинг 4.26. Фрагмент программы к задаче 4.6

```
C[3,4]:= 5;
C[2,5]:= 7;
C[2,4]:= 2;
for N:=3 to 4 do
for M:=4 to 5 do
    C[N,M]:= C[N-1,M]+ C[N,M];
```

ЗАДАЧА 4.7

Значения элементов двумерного массива `A` размером 4 на 4 задаются с помощью вложенного цикла во фрагменте, представленном в листинге 4.27. Сколько элементов массива будут меньше 5?

Листинг 4.27. Фрагмент к заданию 4.7

```
for I:=1 to 4 do
for J:=1 to 4 do
  begin
     A[I,J]:= I * J+I;
  end;
```

ЗАДАЧА 4.8

Значения элементов двумерного массива `A` размером 3 на 3 задаются с помощью вложенного цикла во фрагменте, представленном в листинге 4.28. Сколько элементов массива будут больше 2?

Листинг 4.28. Фрагмент к задаче 4.8

```
for I:=1 to 3 do
for J:=1 to 3 do
  begin
     A[I,J]:= I div J mod J;
  end;
```

ЗАДАЧА 4.9

Значения элементов двумерного массива размером 5 на 5 задаются с помощью вложенного цикла во фрагменте, представленном в листинге 4.29. Сколько элементов массива будут иметь значения равные 0?

Листинг 4.29. Фрагмент к задаче 4.9

```
for I:=1 to 5 do
  for J:=1 to 5 do
  begin
     B[I,J]:= J mod I;
  end;
```

ЗАДАЧА 4.10

Значения элементов двумерного массива A размером 4 на 4 первоначально были равны единице. Затем значения элементов меняются с помощью вложенного оператора цикла во фрагменте программы, представленном в листинге 4.30. Сколько элементов массива будут равны 0?

Листинг 4.30. Фрагмент к задаче 4.10

```
for N:=1 to 4 do
  for K:=1 to 4 do
    begin
      A[N,K]:= A[K,N] - 1;
    end;
```

ЗАДАЧА 4.11

Дан фрагмент программы (листинг 4.31), обрабатывающей двумерный массив A размером N на N. Представим массив в виде квадратной таблицы, в которой для элемента массива A[I,J] величина I является номером строки, а величина J номером столбца, в котором расположен элемент. Что делает рассматриваемый алгоритм:

- меняет местами элементы двух диагоналей;
- меняет местами элементы диагонали и K-й строки;

- меняет местами элементы диагонали и K-го столбца;
- меняет местами элементы K-й строки и K-го столбца.

Листинг 4.31. Фрагмент к задаче 4.11

```
K:=1;
for I:=1 to N do
begin
      C:= A[I,K];
      A[I,K]:= A[K,I];
      A[K,I]:= C;
    end;
```

Ответы к задачам и заданиям из ЕГЭ

Задача 4.1

Здесь требуется только подставить числа (N:=4 и M:=3) и выполнить арифметическое вычисление: 10 + (2 * 4 – 3) = 15.

Задача 4.2

Всего в массиве 25 элементов и требуется проанализировать результат вычисления каждого элемента массива. Так, результаты больше 10 дают произведения: 5 на 5; 4 на 5; 5 на 4; 4 на 4; 5 на 3; 3 на 5; 4 на 3; 3 на 4. В результате получаем, что число интересующих нас элементов массива равно 8.

Задача 4.3

Всего в массиве 25 элементов и требуется проанализировать результат вычисления остатка от деления I на J. Так, результаты больше 2 дают остатки от деления: 3 на 4; 3 на 5; 4 на 5. В результате получаем ответ 3.

Задача 4.4

Проанализируем двумерный массив по строкам. В первой строке (при I:=1) таких элементов насчитывается 6 штук, во второй (при I:=2) таких элементов 5 штук. Продолжая анализ следующих строк, получим ответы: 4, 3, 2, 1, 0.

В итоге получается, что в массиве число положительных элементов равно 6 + 5 + 4 + 3 + 2 + 1 = 21 значение.

Задача 4.5

Любой способ решения заключается в последовательном анализе выполнения циклов. В табл. 4.2 приведен один из вариантов такого анализа. Здесь первые два столбца соответствуют очередным значениям счетчиков циклов, а последующие два других столбца соответствуют значениям A[N,K] и A[K,N], которые вычисляют-

ся в цикле (заметим, что вложенный цикл вычисляется от N до 4). Таким образом, всего в массиве 6 элементов, которые равны 1.

Таблица 4.2. *Решение задачи 4.5*

N	K	A[N,K]	A[K,N]
1	1	A[1,1]=1	A[1,1]=2
1	2	A[1,2]=1	A[2,1]=1
1	3	A[1,3]=1	A[3,1]=1
1	4	A[1,4]=1	A[4,1]=1
2	2	A[2,2]=1	A[2,2]=2
2	3	A[2,3]=1	A[3,2]=1
2	4	A[2,4]=1	A[4,2]=1
3	3	A[3,3]=1	A[3,3]=2
3	4	A[3,4]=1	A[4,3]=1
4	4	A[4,4]=1	A[4,4]=2

Однако можно предложить и другой вариант анализа (табл. 4.3). В этом случае мы формируем таблицу размером 4 на 4 (визуальное представление массива 4 на 4) и начинаем последовательно выполнять шаги, указанные в циклах. После очередного шага в соответствии с циклами мы корректируем значения рассматриваемых элементов. После заполнения таблицы простой пересчет приводит к тому же значению, равному 6.

Таблица 4.3. *Второй вариант решения задачи 4.2*

N	K=1	K=2	K=3	K=4
N =1	A[1,1]=2	A[1,2]=1	A[1,3]=1	A[1,4]=1
N =2	A[2,1]=0	A[2,2]=2	A[2,3]=1	A[2,4]=1
N =3	A[3,1]=0	A[3,2]=0	A[3,3]=2	A[3,4]=1
N =4	A[4,1]=0	A[4,2]=0	A[4,3]=0	A[4,4]=2

Задача 4.6

Здесь требуется проанализировать работу алгоритма. Так как нас интересует элемент из третьей строки (C[3,5]), то рассмотрим выполнение внутреннего цикла (по M) при N:=3. Так, при M:=4 получим:

```
C[3,4]:= C[2,4]+ C[3,4],
```

что после подстановки приводит к результату, равному 7.

Следующее выполнение внутреннего цикла приводит к результату:

```
C[3,5]:= C[2,5]+ C[3,5],
```

что дает ответ 14.

Задача 4.7

Всего в массиве 16 элементов и требуется проанализировать результат вычисления каждого элемента массива. Так, результат, меньший 7, дают значения:

- `I:=1, J:=1;`
- `I:=1, J:=2;`
- `I:=2, J:=1;`
- `I:=2, J:=2;`
- `I:=1, J:=3;`
- `I:=3, J:=1.`

В итоге получаем, что число интересующих нас элементов массива равно 6.

Задача 4.8

Всего в массиве 25 элементов и требуется проанализировать результат вычисления выражения, где используется целочисленное деление и остаток от деления.

Так, вычисления дают:

- `1 div 1 mod 1 = 0;`
- `1 div 2 mod 2= 2;`
- `1 div 3 mod 3 = 3;`
- `2 div 1 mod 1 = 0;`
- `2 div 2 mod 2 = 2;`
- `2 div 3 mod 3 = 3;`
- `3 div 1 mod 1 = 0;`
- `3 div 2 mod 2 = 2;`
- `3 div 3 mod 3 = 3.`

Из этого можно сделать вывод о том, что в массиве имеются три элемента, значения которых больше 2.

Задача 4.9

Проанализируем двумерный массив по строкам. В первой строке (при `I:=1`) таких элементов (равных 0) насчитывается 5 штук, во второй (при `I:=2`) 2 штуки. Продолжая анализ следующих строк, получим: 1, 1, 1. В итоге: 5 + 2 + 1 + 1 + 1 = 10 элементов массива имеют нулевые значения.

Задача 4.10

Рассмотрим вариант анализа, основанный на данных табл. 4.4. В этом случае мы формируем таблицу размером 4 на 4 (визуальное представление массива 4 на 4) и начинаем последовательно выполнять шаги, указанные в циклах. После очередного шага в соответствии с циклами мы корректируем значения рассматриваемых элементов. После заполнения таблицы простой пересчет приводит к результату: 10 элементов массива имеют нулевые значения.

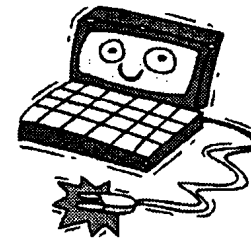
Таблица 4.4. Решение задачи 4.10

N	K=1	K=2	K=3	K=4
N =1	A[1,1]= 0	A[1,2]=0	A[1,3]=0	A[1,4]=0
N =2	A[2,1]=-1	A[2,2]=0	A[2,3]=0	A[2,4]=0
N =3	A[3,1]=-1	A[3,2]=-1	A[3,3]=0	A[3,4]=0
N =4	A[4,1]=-1	A[4,2]=-1	A[4,3]=-1	A[4,4]=0

Задача 4.11

Программа выполняет вычисление в одном цикле по переменной I. Значение индекса K в программе не меняется. Рассмотрим элемент A[2,1]. Этот элемент расположен в 1-м столбце 2-й строки. Как видно из алгоритма, его значение меняется на значение элемента A[1,2]. Соответственно в A[1,2] записывается исходное значение A[2,1]. Аналогично, меняются местами элементы A[3,1] и A[1,3] и т. д. Таким образом, производится замена элементов первого столбца на элементы первой строки. В результате подходит версия ответа — обмен значений элементов K-й строки и K-го столбца.

ГЛАВА 5



Строки и записи

Значительная часть информации, с которой мы работаем, представляет собой текст. Текст фактически является набором символов. Для работы с такими данными в Паскале имеется специальный тип данных — **string**. Если в программе создать переменную данного типа, то в ней можно хранить текстовую строку длиной до 255 символов. Строка может быть и более короткой, но в данном случае в памяти будет отведено место "с запасом". Понятно, что при работе с короткими строками это приводит к неэкономичному использованию оперативной памяти компьютера. В связи с этим в программах на Паскале можно самостоятельно ограничивать число символов, отводимое для строки (от 0 до 255).

Еще один полезный тип данных для работы с большим объемом информации — это *записи*. Запись представляет собой структурированный тип данных, состоящий из фиксированного числа компонентов (элементов). Эти компоненты называются *полями записи*. С помощью записей можно компактно хранить разнородную информацию. Например, информацию о сотрудниках, представляющую совокупность текстовых, числовых и булевых данных.

В этой главе мы рассмотрим различные приемы работы со строками и записями. При этом познакомимся со стандартными функциями Паскаля, предназначенными для работы со строками.

В *главе 2* мы уже упоминали о таком типе данных, как строка. Здесь будут детально рассмотрены приемы работы со строковыми данными, на практических примерах мы познакомимся с типовыми заданиями, касающимися работы со строками.

Описание символьных строк

Тип данных **string**, предназначенный для работы со строками, напоминает массив символов. Однако в отличие от массива со строками, с такими данными можно производить значительно больше действий. Например, строки можно складывать.

Для работы с переменными типа **string** их необходимо описать в разделе **var**. Например, это может выглядеть так:

```
var S:string.
```

В этом случае под строку S выделяется 256 байтов. Можно сказать, что в памяти создается массив из 256 элементов (каждый элемент занимает один байт). Однако непосредственно для символов строки отводится только 255 байтов, а один байт (самый начальный в массиве) предназначен для хранения длины строки. Например, в листинге 5.1 приведено описание переменной строкового типа и занесение в нее одного слова. В этом случае в нулевом элементе массива будет автоматически зафиксирована длина строки: A[0]=6. Далее в последующих элементах размещаются коды символов строки: A[1]= 'П', A[2]= 'Р' и т. д.

Листинг 5.1. Выделение памяти под строку

```
program listing_5_1;
var
  A: string;
begin
  A:= 'Привет';
end.
```

Как уже говорилось в начале главы, если не планируется использовать большие строки, то можно явно указать максимальный размер строки. Например, запись S:string[50] говорит о том, что строка S может содержать от 0 до 50 символов (но не более).

В листинге 5.2 приведен пример более экономичного выделения памяти под строку. В этом случае для строки отводится 11 байтов, из которых 10 предназначены для символов строки, а один байт (байт с нулевым индексом A[0]) предназначен для хранения длины строки.

Листинг 5.2. Выделение памяти под строку ограниченной длины

```
program listing_5_2;
var
  A: string[10];
begin
  A:= 'Привет';
end.
```

Подчеркнем еще раз важный момент — с символами строки можно работать как с элементами одномерного массива. Для этого после имени массива в квадратных скобках указывается индекс интересующего элемента.

Действия над строками

Над строками допустимы следующие операции:

- сцепление (соединение или по-другому сложение нескольких строк);
- присваивание;
- отношения, которые используются для сравнения.

Кроме того, с помощью стандартных операторов (процедур) обеспечивается ввод/вывод при работе со строковыми данными. Для иллюстрации действий над строками в листинге 5.3 приведен пример использования ввода/вывода, присваивания и сложения строковых переменных. Фактическое число символов в строке должно быть не больше числа символов, объявленного в разделе **var** для данной переменной. На примере листинга 5.3 мы создали строковые переменные с ограничением числа символов (ограничили длину строки). Сложение (трех строк) и присваивание строк используется в следующей строке листинга:

```
D:=A+B+C;
```

Листинг 5.3. Пример действий над строками

```
program listing_5_3;
var
  A: string[50];
  B: string[50];
  C: string[10];
  D: string[110];
 begin
  writeln('Введите фамилию ');
  readln(A);
  writeln('Введите имя ');
  readln(B);
  writeln('Введите отдел ');
  readln(C);
  D:=A+B+C;
  writeln(D);
end.
```

При присваивании строк важно подчеркнуть один момент. Поскольку все строковые типы являются совместимыми по присваиванию, то можно присваивать друг другу значения строк разного размера. Однако при этом размеры строк не контролируются. И если значение переменной после выполнения оператора присваивания превышает по длине максимальный размер, указанный при объявлении, то все лишние символы справа отбрасываются. Так, если в рассмотренном примере под переменную D, скажем, отвести 10 символов, то ожидаемый результат на экране мы не увидим.

Частое действие связано со сравнением строк. Здесь никакой новизны относительно ранее рассмотренных примеров с использованием сравнения (встречалось в предыдущих главах) нет. Операции отношения (=, <, >, <>, <=, >=) позволяют выполнить сравнение двух строковых операндов и используются при проверке условий. Сравнение строк производится слева направо до первого несовпадающего символа. В этом случае большей считается та строка, в которой код первого несовпадающего символа больше. Если же строки совпадают по длине и содержат одни и те же символы, то они считаются равными.

Примеры выражений с использованием сравнений строк:

- `'Петя '>'Петя'` — данное выражение истинно (результат — **true**), т. к. в левой части выражения присутствует дополнительный пробел;
- `'Петя'='Вася'` — данное выражение ложно (результат — **false**).

Рассмотрим пример, связанный с вводом с клавиатуры набора (массива) строк. В листинге 5.4 приведена разработка, в которой при повторном вводе уже встречавшейся строки работа программы завершается.

Листинг 5.4. Введение строк с проверкой на идентичность

```
program listing_5_4;
var
  A: array[1..100] of string;
  B:string;
  N,I,Flag: integer;
begin
  Flag:=0;
  N:=1;
  while Flag=0 do
     begin
        writeln('Введите строку');
        readln(B);
        for I:=1 to N-1 do
          if B = A[I] then
               Flag:=1;
        if Flag = 0 then
          begin
            A[N]:=B;
            N:=N+1;
          end;
     end;
   for I:=1 to N do
      writeln(A[I]);
end.
```

Работа со строками как с элементами массивов

Как уже говорилось, строка фактически представляет собой массив символов, для доступа к конкретному символу необходимо указать имя, выбранное для строки, а также в квадратных скобках указать номер символа. Если фактическая длина строки составляет `N` символов, то символы строки имеют номера от `1` до `N` включительно.

Примечание

В предыдущем примере также использовались квадратные скобки. Однако там мы работали с массивом строк, где каждый элемент массива представлял строку в целом.

Рассмотрим пример на данную тему. Так, необходимо по введенной с клавиатуры строке определить, является ли она записью целого числа в десятичной системе счисления. Программа (листинг 5.5) учитывает, что коды чисел расположены плотно и наша задача заключается в проверке: попадает ли каждый введенный символ в интервал от 0 до 9. Здесь мы использовали цикл с предусловием, действие которого заключается в проверке одновременного выполнения двух составляющих условий:

- не превысили ли мы число просматриваемых символов в строке;
- не оказался ли очередной символ не цифрой.

Если очередной просматриваемый символ оказался не цифрой, то это фиксируется в переменной `Flag`. А именно: если очередной просматриваемый символ оказался не цифрой, то в качестве значения в переменную `Flag` заносится единица. В этом случае условие в операторе цикла ложно и выполнение цикла завершается.

В заключительной части программы производится вывод соответствующего сообщения в зависимости от значения переменной `Flag`.

Листинг 5.5. Проверка строки, введенной с клавиатуры

```
program listing_5_5;
var
  A: string[50];
  N, Flag: integer;
 begin
  readln(A);
  N:=1;
  Flag:=0;
  while (N <= length(A)) and (Flag=0) do
    begin
        if ( A[N]<'0') or ( A[N]>'9') then
           Flag:=1;
        N:=N+1;
    end;
  if Flag = 1 then
     writeln('Строка не является целым числом! ')
  else
     writeln('Строка является целым числом! ')
end.
```

В тексте этого листинга использована библиотечная функция работы со строками `length(Str)`, которая вычисляет длину строки `Str`, указанной в качестве параметра. Хотя можно было обойтись и без нее, поскольку мы знаем, что в элементе массива `Str[0]` длина строки фиксируется автоматически.

В Паскале имеется еще ряд стандартных функций и процедур, предназначенных для работы со строками, и в следующем разделе мы о них поговорим.

Строковые процедуры и функции

Для удобства работы в системе Turbo Pascal имеется набор стандартных функций и процедур. В большинстве ситуаций они существенно облегчают работу программиста. После теоретических сведений по процедурам и функциям мы рассмотрим ряд практических примеров с их использованием.

Процедуры для вставки и удаления символов

Для удаления строк используется процедура **delete**, которая имеет следующий формат:

```
delete(Str, Num, N),
```

где `Str` — строка, из которой необходимо удалить `N` символов, начиная с позиции `Num`.

Например, после выполнения программного фрагмента

```
Str:= 'Строка!';
delete(Str,7,1);
```

в переменной `Str` окажется значение: `Строка`.

Если в процедуре **delete** указан номер позиции первого удаляемого символа, превосходящий длину строки, то из строки ничего не удалится.

Для вставки символов используется процедура **insert**, которая имеет следующий формат:

```
insert(Str1, Str2, Num),
```

где `Str1` — строка, которая вставляется в строку `Str2`, начиная с позиции `Num`.

Например, можно предложить следующий вариант использования процедуры вставки символов:

```
Str1:=' Петя! ';
Str2:='Привет,';
insert(Str1,Str2,7);
```

В результате в переменной `Str2` окажется значение: `Привет,Петя!`.

Заметим, что вставка в начало и конец строки эквивалентна сложению (слиянию) строк.

Функции для работы со строками

С одной из функций (**length**) мы уже встречались в предыдущем примере (см. листинг 5.5). В качестве единственного параметра этой функции необходимо передать саму строку (в виде константы или значения переменной строкового типа).

Рассмотрим еще несколько полезных функций. Так, для выделения из строки подстроки используется функция

```
copy(Str1, Num, N),
```

где:

- `Str1` — исходная строка;
- `Num` — позиция выделяемой подстроки;
- `N` — число выделяемых символов.

Результатом выполнения данной функции является выделенная подстрока.

В качестве примера можно предложить следующий вариант использования функции `copy`:

```
Str1:='Петя,Вася,Олег';
Str2:= copy(Str1, 6, 4);
write(Str2);
```

В результате на экране мы увидим имя: `Вася`.

Для сцепления нескольких строк используется функция

```
concat(Str1,Str2,Str3, ...).
```

В качестве итога своей работы функция выдаст строку, являющуюся результатом сложения строк, указанных в списке параметров.

Например, можно предложить следующий вариант использования функции сцепления строк:

```
Str1:='Петя';
Str2:='Вася';
Str3:='Олег';
Str4:=concat(Str1,Str2,Str3, ...)
write(Str4);
```

В результате на экране мы увидим перечисление всех имен: `ПетяВасяОлег`.

Еще одна полезная функция: `pos(Str1,Str2)`. Она позволяет обнаружить первое появление в строке `Str2` подстроки `Str1`. Результат, выдаваемый функцией, имеет целочисленный тип и представляет собой номер позиции, где находится первый символ подстроки `Str1`. Если в `Str2` подстроки `Str1` не найдено, то функция выдает 0.

В качестве примера можно предложить следующий вариант использования функции **pos**:

```
Str1:='Вася';
Str2:='Петя,Вася,Олег';
N:= pos(Str1, Str2);
write(N);
```

В результате на экране мы увидим номер позиции указанной подстроки: 6.

Процедуры преобразования типов

В практических ситуациях часто требуется преобразовывать числовые данные в строки, а также наоборот — последовательность символов необходимо переводить в числовые данные. Для этого в системе Turbo Pascal имеются процедуры **str** и **val**.

Для преобразования числового значения величины Num в строку с именем Stroka предназначена процедура

```
str(Num, Stroka),
```

которая формирует в переменной Stroka результат преобразования, являющийся набором символов.

Например, в следующем фрагменте мы получим из сформированного целочисленного числового значения набор символов:

```
val Stroka:string;
    Num:integer;
begin
  Num:=57;
  str(Num,Stroka);
  Stroka:= 'Число равно'+ Stroka;
  write(Stroka);
end.
```

Здесь после преобразования мы использовали слияние строк, а результат слияния вывели на экран.

При преобразовании вещественных чисел можно указывать необходимый формат. Правила определения формата те же, что уже встречались ранее в книге. Например, в следующем фрагменте мы получили ограниченный (до двух знаков после точки) набор символов, эквивалентный исходному числу:

```
val
    Stroka:string;
    A:real;
begin
  A:=sqrt(2);
  str(A:5:3,Stroka);
  write(Stroka);
end.
```

Другая функция **val** преобразует строковое значение (Stroka) в величину целочисленного или вещественного типа и помещает результат в переменную (Num):

```
val(Stroka, Num, Code),
```

где, кроме уже описанных параметров для вывода уточняющей информации, используется целочисленная переменная Code. Если во время выполнения преобразования ошибки не обнаружено, то Code принимает значение 0. Если же при переводе обнаружена ошибка (не удается перевести указанный набор символов в числовое значение), то Code будет содержать номер позиции первого ошибочного символа.

Например, в следующем фрагменте мы получим в одном случае правильное преобразование, а в другом ошибочный результат:

```
val
    Stroka1,Stroka2:string;
    A1,A2:real;
    Code1,Code2:integer;
begin
   Stroka1:= '589k';
   Stroka2:= '589.7';
   val(Stroka1,A1,Code1);
   writeln(Code1,' ',Stroka1);
   val(Stroka2,A2,Code2);
   wrire(Code2,' ',Stroka2);
end.
```

Примеры программ с обработкой строк

Далее мы рассмотрим несколько программных разработок, которые помогут вам сформировать практические навыки работы со строковыми данными.

Поиск подстроки и удаление подстроки

Необходимо из исходной строки удалить все подстроки со значением `Pete`. Здесь нам потребуется функция **pos** для нахождения подстроки, а также функция **delete** для удаления фрагментов из строки. В листинге 5.6 приведена программная разработка, решающая поставленную задачу. Здесь мы в цикле последовательно корректируем исходную строку до ситуации, когда в ней не останется слова `Pete`.

Листинг 5.6. Удаление из строки фрагментов

```
program listing_5_6;
var
  Str1,Str2:string;
begin
  readln(Str2);
  Str1:= 'Pete';
  while pos(Str1,Str2)> 0  do
      delete(Str2,pos(Str1,Str2),4);
   writeln(Str2)
end.
```

Удаление пар символов при условии

Допустим, необходимо из исходной строки удалить все пары символов, начинающихся с цифры 1. Хотя программа (листинг 5.7) является не очень сложной, будет правильно, если мы приведем ее блок-схему (рис. 5.1).

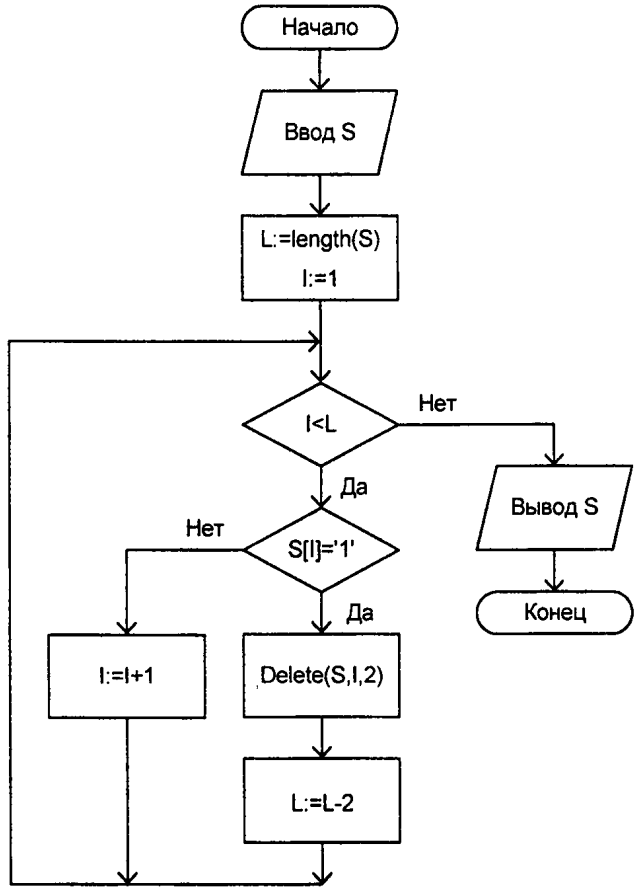


Рис. 5.1. Блок-схема к программе листинга 5.7

Листинг 5.7. Удаление из строки пар символов, начинающихся с цифры 1

```
program listing_5_7;
var
  S: string;
  I,L: integer;
begin
  write('Введите строку');
  readln(S);
  L := length(S);
  I:=1;
  while I < L do
  begin
   if S[I]= '1' then
     begin
        delete(S,I,2);
        L:=L-2;
     end
```

```
    else
      I:=I+1;
  end;
 write(S);
end.
```

Вставка одиночных символов в строку

Необходимо в исходной строке, введенной с клавиатуры, сделать вставки. А именно после сочетания kg необходимо поставить точку, но только если ее там еще нет. Приведем программу (листинг 5.8) и дополним ее блок-схемой (рис. 5.2).

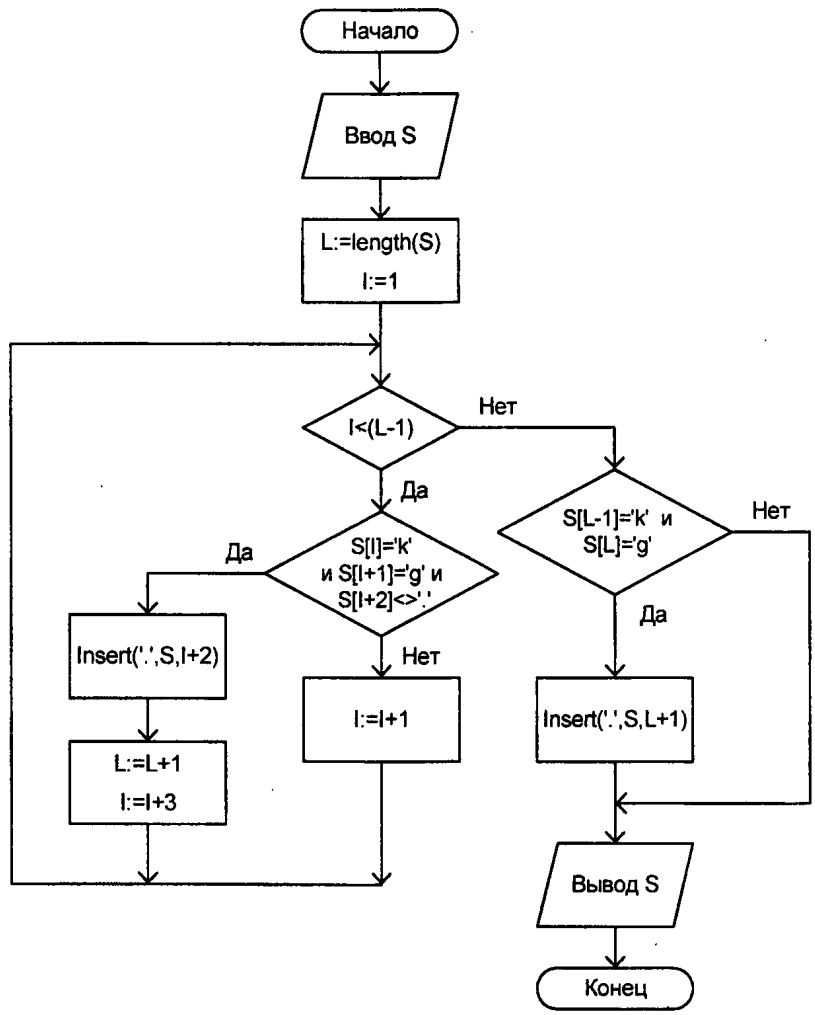


Рис. 5.2. Блок-схема к программе листинга 5.8

Листинг 5.8. Вставка одиночных символов в строку

```
program listing_5_8;
var
  S: string;
  I,L: integer;
  A,B,C:char;
 begin
  writeln('Введите строку');
  readln(S);
  L := length(S);
  I:=1;
  while I < (L-1) do
  begin
    A:= S[I];
    B:= S[I+1];
    C:= S[I+2];
    if (A ='k') and (B='g') and  (C<>'.')  then
      begin
      insert('.',S,I+2);
      L:=L+1;
      I:=I+3;
      end
  else
     I:=I+1;
  end;
    if (S[L-1]='k') and (S[L]='g')  then
      insert('.',S,L+1);
 write(S);
end.
```

Подсчет количества слов в строке

Необходимо подсчитать количество слов в строке, введенной с клавиатуры. Подразумевается, что слова друг от друга могут отделяться одним или несколькими пробелами. При этом несколько пробелов могут располагаться в начале, в середине или в конце строки. Текст необходимой программы приведен в листинге 5.9. Для лучшего пояснения алгоритма на рис. 5.3 дана блок-схема алгоритма.

В этой программе мы вводим переменную `S` для хранения строки, введенной с клавиатуры. Также для технических действий нам потребуются три целочисленные переменные: `I`, `L`, `M`.

Выполнение программного кода начинается с команды **`readln`**`(S)`, позволяющей ввести с клавиатуры набор символов и сохранить их в переменной `S`. После этого с помощью стандартной функции **`length`** вычисляется длина введенной строки. Да-

лее наша задача заключается в последовательном анализе соседних элементов массива `S[I]`, начиная с первого.

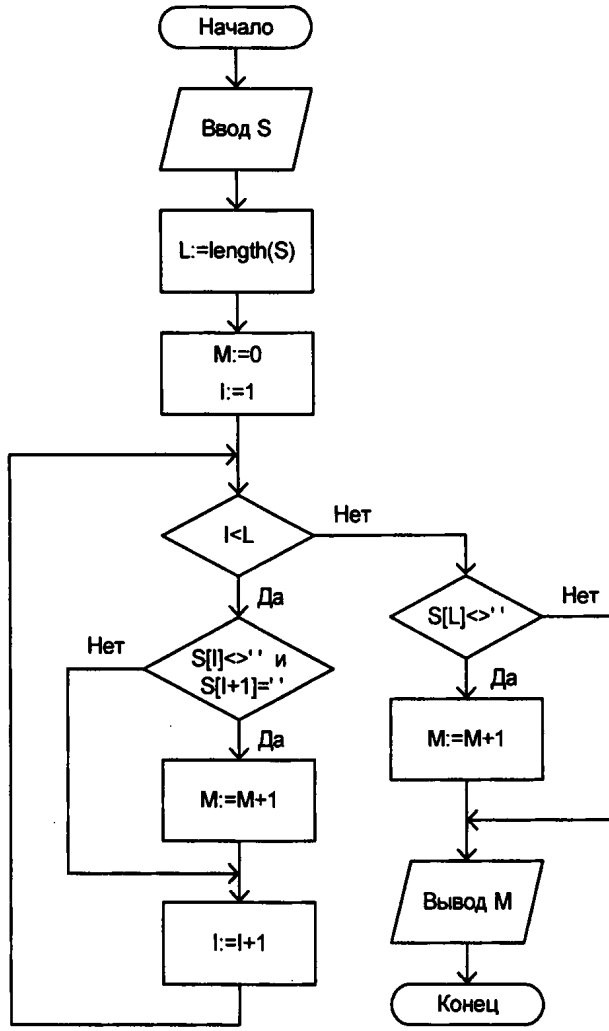


Рис. 5.3. Блок-схема к программе листинга 5.9

Если текущий символ не является пробелом, а в последующем символе пробел присутствует, то в счетчик слов (`M`) добавляется единица. После цикла важно проверить, не является ли последний символ пробелом. Если не является, то к счетчику слов добавляется еще одна единица.

Листинг 5.9. Подсчет числа слов в строке

```
program listing_5_9;
var
  S: string;
  I,L,M: integer;
```

```
 begin
  writeln('Введите строку');
  readln(S);
  L := length(S);
  M:=0;
  I:=1;
  while I < L do
  begin
    if (S[I] <> ' ') and  (S[I+1]=' ') then
        M:=M+1;
    I:=I+1;
  end;
  if   S[L] <> ' ' then
       M:=M+1;
  write(M);
end.
```

Подсчет количества символов фрагмента в строке

Необходимо подсчитать число символов во введенной строке, которые соответствуют одному из символов, присутствующих в предварительно сформированном наборе символов. Например, если сформированный набор представляет собой комбинацию: abc (три символа), то программа должна подсчитать общее число символов из данного списка во введенной строке. Текст необходимой программы приведен в листинге 5.10. Для лучшего пояснения алгоритма на рис. 5.4 дана блок-схема алгоритма.

Листинг 5.10. Подсчет символов фрагмента в строке

```
program listing_5_10;
var
  Nabor: string;
  S: string;
  I,J,L,M,N: integer;
begin
  writeln('Ввести набор интересующих нас символов в строке');
  readln(Nabor);
  N:= length(Nabor);
  writeln('Ввести строку ');
  readln(S);
  L:=length(S);
  M:=0;
  for I:= 1 to  L do
   begin
      for J:= 1 to  N  do
```

```
        begin
           if S[I] = Nabor[J] then
              M:=M+1;
        end;
    end;
    write('Число символов из набора',M);
end.
```

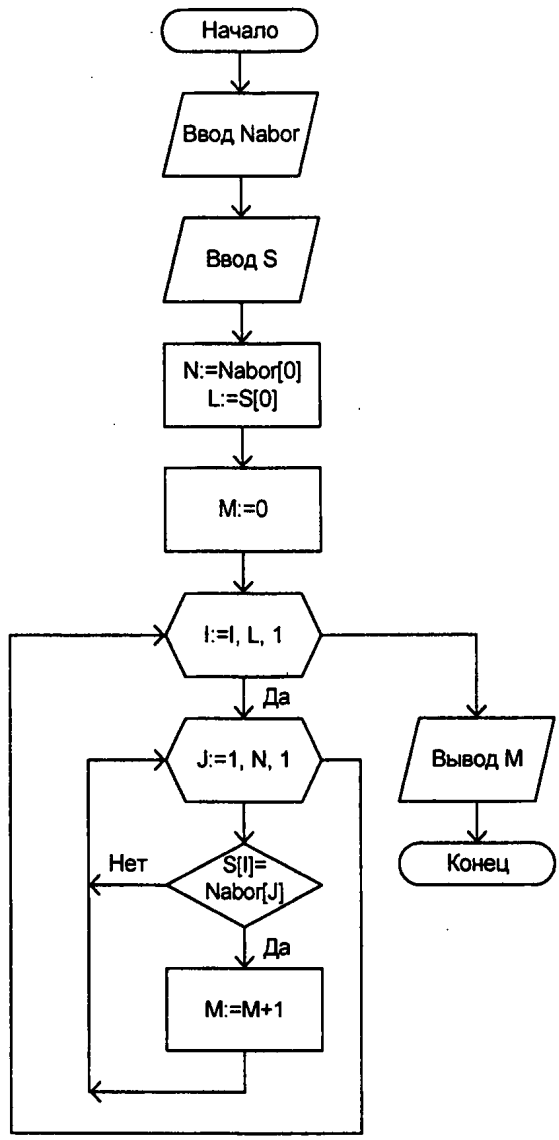


Рис. 5.4. Блок-схема к программе листинга 5.10

Удаление лишних пробелов

При вводе строки с клавиатуры в ней между словами может присутствовать несколько пробелов. Необходимо удалить лишние пробелы и оставить только один. В листинге 5.11 приведена необходимая программная разработка.

Листинг 5.11. Удаление лишних пробелов в строке

```
program listing_5_11;
var
  S: string;
  I,L: integer;
begin
  readln(S);
  L := Length(S);
  I:=1;
  while I < L do
   begin
     if (S[I]= ' ') and (S[I+1]= ' ') then
      begin
         delete(S,I,1);
         L:=L-1;
      end
   else
     I:=I+1;
   end;
 write(S);
end.
```

Вставка слова при условии

Рассмотрим пример на вставку слова в строку. А именно: с клавиатуры последовательно вводится сначала определенное слово, а затем строка из нескольких слов. Необходимо вставить введенное слово после третьего слова этой строки. Здесь мы должны учесть и наличие нескольких пробелов между словами, и возможную вставку слова в конец строки, если в ней всего три слова. В листинге 5.12 приведена программа, а на рис. 5.5 дана блок-схема алгоритма.

Листинг 5.12. Вставка слова в строку

```
program listing_5_12;
var
  S,S1: string;
  I,L,M: integer;
begin
  writeln('Ввести вставляемое слово');
  readln(S1);
```

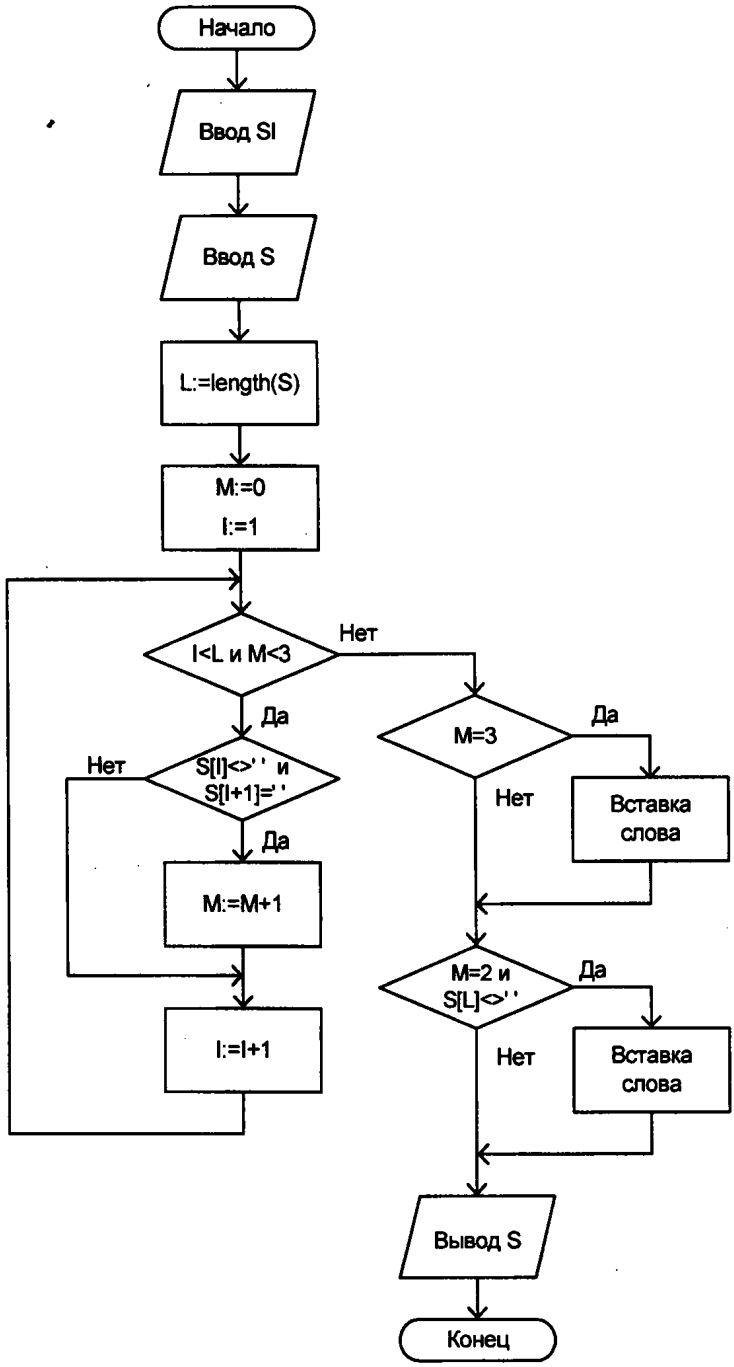


Рис. 5.5. Блок-схема к программе листинга 5.12

```
  writeln('Ввести строку');
  readln(S);
  L := length(S);
  M:=0;
  I:=1;
  while (I < L) and (M<3) do
   begin
     if (S[I] <> ' ') and  (S[I+1]=' ') then
          M:=M+1;
     I:=I+1;
   end;
  if M=3 then
    begin
      Sl:=' '+Sl;
      insert(Sl,S,I);
    end;;
  if ( M=2 ) and ( S[L] <> ' ') then
      begin
         Sl:=' '+Sl;
         insert(Sl,S,L+1);
      end;
  writeln('S =',S);
end.
```

Нахождение суммы цифр

Необходимо подсчитать сумму цифр в строке, а также вывести все имеющиеся цифры в строке на экран (листинг 5.13).

Листинг 5.13. Подсчет суммы цифр в строке

```
program listing_5_13;
var
    A: string[50];
    N, Sum: integer;
begin
  writeln('Ввести строку');
  readln(A);
  N:=1;
  Sum:=0;
  while N <= length(A) do
    begin
        if ( A[N]>='0') and ( A[N]<='9') then
            begin
               Sum:=Sum+Ord(A[N])-Ord('0');
               writeln(A[N]);
            end;
```

```
        N:=N+1;
    end;
   writeln('Сумма цифр равна ', Sum)
end.
```

Удаление из строки цифр

Рассмотрим теперь похожий пример, в котором требуется удалить все цифры из введенной строки (листинг 5.14).

Листинг 5.14. Удаление цифр из строки

```
program listing_5_14;
var
  A: string[50];
  N, L: integer;
 begin
   writeln('Ввести строку');
   readln(A);
   L:=length(A);
   N:=1;
   while N <= L do
     begin
        if ( A[N]>='0') and ( A[N]<='9') then
            begin
               delete(A,N,1);
               L:=L-1;
            end
        else
          N:=N+1;
     end;
   writeln('Строка -', A);
end.
```

Замена в строке

Необходимо программно обеспечить замену в строке A, вводимой с клавиатуры, одной подстроки Str1 на другую подстроку Str2. В листинге 5.15 приведена программная разработка.

Листинг 5.15. Замена одной подстроки на другую

```
program listing_5_15;
var
   A,Str1,Str2: string;
   I,N: integer;
```

```
 begin
   writeln('Введите исходную строку');
   readln(A);
   writeln('Введите подстроку, которую следует заменить');
   readln(Str1);
   writeln('Введите подстроку, которую следует вставить на замену');
   readln(Str2);
   while pos(Str1,A) <> 0 do
  begin
    I:=pos(Str1,A);
    delete(A,I,length(Str1));
    insert(Str2,A,I);
 end;
 writeln(A);
end.
```

Использование массивов строк

Кроме одиночных строк, можно использовать и массивы строк. В данном случае необходимо программно обеспечить ввод в цикле нескольких строк с клавиатуры. Процесс ввода завершается, когда без ввода очередной строки просто нажимается клавиша <Enter>. После ввода массива строк следует вывести количество символов в строке, имеющей максимальную длину. В листинге 5.16 приведена необходимая разработка.

Листинг 5.16. Заполнение массива строк

```
program listing_5_16;
var
   S:array[1..30] of string;
   A: string;
   I,N,Max: integer;
 begin
  N:=0;
  repeat
    writeln('Введите исходную строку');
    readln(A);
    if A <> '' then
     begin
       N:=N+1;
       S[N]:=A;
     end;
  until A = '';
  Max:=0;
  for I:=1 to N do
```

```
    if length(S[I])>Max then
       Max:=length(S[I]);
  writeln(Max);
end.
```

Записи

Как мы уже узнали из предыдущих глав книги, при работе с большим объемом данных используются массивы. Однако в рассмотренных примерах (см. *главы 3*, *4* и *данную главу*) элементы массива имели один из стандартных типов данных. В этом случае мы могли хранить в массиве либо числовые данные (например, год рождения или доход по итогам года), либо строковые (фамилии, названия городов). Однако часто требуется хранить и комбинированную информацию. Так, удобно в элементе данных хранить числовую информацию по каждому сотруднику вместе с текстовой (фамилия, имя и т. д.). Можно сказать, что в этом случае каждый элемент данных имеет структуру, в которой скомбинировано несколько стандартных типов данных. И для такой организации данных в Паскале существуют записи.

Запись — это структурированный тип данных, состоящий из фиксированного числа компонентов. Эти компоненты называются *полями* записи. Каждое поле записи должно иметь один из стандартных типов данных. В целом запись представляет собой гибкий структурированный тип данных.

В программе перед использованием записи сначала необходимо определить ее тип. В определение типа вкладывается назначение имени и выбор полей записи. Формат определения типа записи выглядит так:

```
type
   ИмяТипа=record
       ИмяПоля1: ТипПоля1;
       ИмяПоля2: ТипПоля2;
       ...
       ИмяПоляN: ТипПоляN;
end;
```

Как видно из данной структуры, определение типа записи начинается с идентификатора **record** и выбора имени для типа записи. После этого располагается список полей с указанием их типов. Описание записи завершается ключевым словом **end**.

Например, запись для отражения информации о сотрудниках и их доходах можно организовать так:

```
type
 Stud=record
   Fam: string[50];
   Imya: string[50];
   Dohod: real;
end;
```

Здесь выбрано три поля:

- Fam — для фамилии сотрудника отведена строка, не превышающая 50 символов;
- Imya — для имени сотрудника указана строка, не превышающая 50 символов;
- Dohod — доход (вещественный тип данных).

Значения полей записи можно использовать в выражениях. Для этого при обращении к значению поля необходимо использовать имя переменной (она должна иметь тип рассматриваемой записи) и имя поля, при этом они должны быть разделены точкой. Такая комбинация называется составным именем. Приведем пример доступа к полям записи рассмотренного ранее типа:

```
var
    A:array[1..30] of Stud;
begin
A[1].Fam='Петров';
A[1].Imya ='Иван';
A[1].Dohod=7500.50;
```

Здесь мы создали массив A, каждый элемент которого имеет тип записи Stud. Видно, что полям записи можно присваивать значения, как и обычным переменным.

Важный момент касается организации ввода/вывода. При работе с записями в целом операторы ввода/вывода не работают, т. е. нельзя целиком ввести элемент записи или вывести определенный элемент записи на экран. Однако можно легко работать с полями — использовать их в операторах ввода/вывода. Например, в продолжение приведенного примера можно вывести информацию из полей записи так:

```
write(A[1].Fam, A[1].Imya, A[1].Dohod).
```

Далее мы рассмотрим несколько практических примеров, демонстрирующих приемы работы с записями.

Формирование списка учащихся

Для начала обратимся к задаче, касающейся ввода и хранения информации об учащихся. Так, в наборе записей мы будем фиксировать фамилии, имена и года рождения группы учащихся. Программа должна обеспечить ввод информации и размещение ее в памяти компьютера. После этого необходимо найти наиболее поздний год рождения учащихся и вывести фамилии всех учащихся данного года рождения на экран. В листинге 5.17 приведена эта программа.

Листинг 5.17. Формирование списка учащихся по условию

```
program listing_5_17;
Type
  Stud=record
  Fam: string[50];
```

```
  Imya: string[50];
  God: integer;
end;
var
    A: array[1..50] of Stud;
    N, MaxGod, I:integer;
begin
  writeln('Введите число учащихся');
  readln(N);
  for I:=1 to N do
    begin
       writeln('Введите фамилию');
       readln(A[I].Fam);
       writeln('Введите имя');
       readln(A[I].Imya);
       writeln('Введите год рождения');
       readln(A[I].God);
    end;
    MaxGod:= A[1].God;
    for I:=2 to N do
        if A[I].God > MaxGod then
              MaxGod:= A[I].God;
    for I:=1 to N do
      if A[I].God = MaxGod then
          writeln(A[I].Fam, ' ',A[I].Imya);
end.
```

Анализ оценок учащихся

Допустим, нам необходимо хранить информацию об успеваемости учащихся в группе. Для определенности перечень дисциплин таков:

- математика;
- история;
- информатика.

Программа должна обеспечить ввод фамилий учащихся и оценок. После ввода данных необходимо подсчитать число отличников по математике, а также вывести фамилии тех, кто учится без троек.

Для решения поставленной задачи определим структуру записи так:

```
Type
  Stud=record
  Fam: string[100];
  Matem: integer;
  Istoriya: integer;
  Inform: integer;
end;
```

Здесь для оценок мы отвели поля целого типа: Matem, Istoriya, Inform. Программа, реализующая решение поставленных задач, приведена в листинге 5.18.

Листинг 5.18. Анализ оценок учащихся

```
program listing_5_18;
Type
  Stud=record
  Fam: string[100];
  Matem: integer;
  Istoriya: integer;
  Inform: integer;
end;
var
  A: Stud[50];
  N, I,Cols: integer;
 begin
  writeln('Введите число учащихся');
  readln(N);
  for I:=1 to N
    begin
       writeln('Введите фамилию');
       readln(A[I].Fam);
       writeln('Введите оценки');
       readln(A[I].Matem, A[I].Istoriya, A[I].Inform);
    end;
    Cols:= 0;
    for I:=1 to N
           if A[I].Matem = 5 then
             Cols:= Cols+ 1 ;
  writeln(' На 5 по математике учатся ', Cols );
  for I:=1 to N
    if(A[I].Matem>2) and (A[I].Istoriya>2) and(A[I].Inform > 2)  then
             writeln(' Без троек учатся ', A[I].Fam );
end.
```

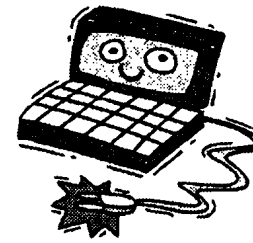
Поиск автомобиля по цене

Известны данные о марках автомобилей и их стоимости. Необходимо обеспечить ввод этих данных в программу, после этого следует найти автомобиль, стоимость которого максимально близка к средней стоимости, вычисленной по всем автомобилям. Программа, реализующая решение поставленных задач, приведена в листинге 5.19.

Листинг 5.19. Нахождение автомобиля по условию

```
program listing_5_19;
Type
  Avto=record
  Nazv:string[100];
  Tsena: longint;
end;
var
  A: array [1..50] of Avto;
  N, I: integer;
  Sr,Delta:real;
 begin
  writeln('Введите число автомобилей');
  readln(N);
  for I:=1 to N do
    begin
       writeln('Введите модель');
       readln(A[I].Nazv);
       writeln('Введите цену');
       readln(A[I].Tsena);
    end;
    Sr:= 0;
    for I:=1 to N do
          Sr:= Sr+ A[I].Tsena ;
     Sr:=Sr/N;
     Delta:= Abs(A[1].Tsena-Sr);
     for I:=2 to N  do
       begin
        if Abs(A[I].Tsena - Sr)< Delta  then
             Delta:= Abs(A[I].Tsena - Sr);
     end;
  for I:=1 to N  do
    if Abs(A[I].Tsena - Sr) = Delta then
       writeln(' Автомобиль ', A[I]. Nazv );
end.
```

ГЛАВА 6



Работа с файлами

В процессе работы на компьютере мы практически постоянно встречаемся с файлами, и в этой главе будут рассмотрены ресурсы языка Паскаль, которые предназначены для работы с файлами. Мы узнаем, каким образом можно программно создать файл и записать в него необходимую информацию, а также извлечь информацию из уже имеющегося файла или дописать в созданный файл нужные сведения.

Файл представляет собой набор данных, хранящихся во внешней памяти компьютера (на жестком диске, компакт-диске и т. д.). У файлов имеется ряд характерных особенностей:

- при создании каждому файлу назначается имя, которое позволяет однозначно его идентифицировать;
- файлы содержат элементы определенного типа.

Система Turbo Pascal разработана для программирования в операционной системе MS-DOS. Это налагает определенные требования к названиям файлов. Имя состоит из двух частей: собственно имени и расширения. Расширение от имени отделяется точкой, за которой следует от одного до трех символов расширения. Расширение имени файла не является обязательным, но его следует использовать. Оно применяется для описания содержимого файла. Это позволяет операционной системе быстро классифицировать и обрабатывать файлы.

В Паскале имеются два основных класса файлов: текстовые и типизированные. *Текстовые* — это файлы, которые состоят из любых символов. Они организуются по строкам, каждая из которых заканчивается символом *конец строки*. Конец самого файла обозначается символом *конец файла*. При записи информации в текстовый файл все данные преобразуются к символьному типу. Просмотреть текстовый файл можно с помощью любого текстового редактора.

Файлы, состоящие из компонентов одного типа, число которых заранее не определено, называются *типизированными*. Например, файл может состоять из целых двухбайтовых чисел (тип `integer`).

Далее мы последовательно рассмотрим теоретические сведения и технологию работы с текстовыми, а затем и с типизированными файлами. В действиях с этими двумя типами файлов имеются общие моменты, но есть и отличия.

Текстовые файлы

Основным элементом текстового файла является символьная строка (совокупность ASCII-кодов). При этом можно работать как со строкой целиком, так и с каждым символом в отдельности. Обращение к символам, хранящимся в текстовом файле, происходит последовательно.

Каждая строка в текстовом файле заканчивается составным символом *конца строки*, который является объединением двух символов: символа *возврата каретки* и символа *перевода строки*. Для такого составного символа вводят обозначение **eoln** (End Of LiNe).

Для начала работы с файлом необходимо связать файловую переменную в программе с файлом на диске. Для этого используется процедура **assign**, которая имеет следующий формат:

```
assign(A,B),
```

где `A` — имя файловой переменной, а `B` — полное имя файла на диске. При этом сама файловая переменная должна быть описана в обычном разделе описаний:

```
var
   A:file of text;
```

Это первый обязательный шаг при работе с текстовыми файлами. В целом начало программы, предназначенной для работы с текстовыми файлами, может выглядеть так:

```
var
   A:file of text;
begin
      assign(A,'info.txt');
```

В данном случае файловая переменная `A` связывается с файлом `info.txt`, который расположен в том же каталоге, что и программа на Паскале. Файл, с которым мы собираемся работать, необязательно должен располагаться в той же папке, что и программа. В такой ситуации следует указать полный путь к файлу. Например, это может выглядеть так:

```
var
   A:file of text;
begin
      assign(A,'C:\primeri\info.txt');
```

Здесь указано, что необходимый файл располагается на диске `C:` в папке `primeri`.

Создание текстового файла

Рассмотрим ситуацию, когда необходимо создать файл и записать в него информацию. Для создания нового файла либо для коррекции уже имеющегося необходимо воспользоваться процедурой **rewrite**.

Если указанного (в процедуре **assign**) файла нет в каталоге, то процедура **rewrite** создаст новый пустой файл. Указатель файлов устанавливается в начало пустого файла. После этого в файл можно записывать информацию. Для этого применяются две процедуры:

- **write** — для записи символов;
- **writeln** — для записи строк.

Если же файл с указанным именем есть в каталоге, то его содержимое очищается и в него можно записывать новую информацию.

Рассмотрим пример (листинг 6.1) создания нового текстового файла и записи в него трех строк, вводимых с клавиатуры. Здесь мы воспользовались только что рассмотренными стандартными процедурами работы с файлами. Кроме того, после действий с файлом его следует закрыть, для чего предназначена процедура **close**(A), где A — имя файловой переменной. В результате выполнения программы на диске будет создан файл prim1.txt, содержащий три строки текста.

Для записи строк в файл мы воспользовались процедурой **writeln** (A, S), где A — файловая переменная, а S — переменная строкового типа, содержащая очередную строку для файла.

Листинг 6.1. Создание текстового файла и внесение в него данных

```
program listing_6_1;
var
  A: text;
  S: string;
  I: integer;
 begin
  assign(A,'prim1.txt');
  rewrite(A);
  for I:=1 to 3 do
   begin
     writeln('Введите строку');
     readln(S);
     writeln(A,S);
  end;
  close(A);
end.
```

Чтение из файла

Возможна ситуация, когда нам требуется открыть имеющийся файл для чтения данных. Для этого после создания файловой переменной следует воспользоваться процедурой **reset**(A), где A — файловая переменная.

В процессе работы с уже имеющимся файлом требуется использовать еще одну функцию — **eof**(A). Данная функция возвращает одно из двух значений:

- **true** — если достигнут конец файла;
- **false** — если еще не достигнут конец файла.

Для чтения строк из файла необходимо воспользоваться процедурой **readln**(A, S), где A — файловая переменная, а S — переменная строкового типа для получения очередной строки из файла.

В листинге 6.2 представлена программа, которая открывает созданный в предыдущем примере текстовый файл, построчно считывает из него данные, а извлеченные данные в цикле последовательно выводятся на экран.

Листинг 6.2. Чтение текстового файла и вывод информации на экран

```
program listing_6_2;
var
  A: text;
  S: string;
 begin
  assign(A,'prim1.txt');
  reset(A);
  while not eof(A) do
    begin
       readln(A,S);
       writeln(S);
    end;
  close(A);
end.
```

Формирование файла с набором символов

Необходимо программно создать текстовый файл из 20 строк, при этом в первой строке в файле должна располагаться буква A, во второй две буквы B, в третьей три буквы C и т. д. в соответствии с последовательностью букв в алфавите. Подразумевается, что все буквы относятся к английскому алфавиту.

В листинге 6.3 представлена программа, которая создает новый текстовый файл и построчно формирует в нем необходимую информацию. Мы воспользовались тем, что коды букв английского алфавита расположены плотно, а также уже знакомыми функциями (см. *главу 1*) работы с символами: **Chr** и **Ord**.

Вывод символов без перевода строки реализован с помощью процедуры **write**(A,Sim), где Sim — очередной символ. Для перевода строки в текстовом файле мы воспользовались процедурой **writeln**(A).

Листинг 6.3. Создание текстового файла с набором символов

```
program listing_6_3;
var
  A: text;
  I,J:integer;
  Sim:char;
 begin
  assign(A,'prim3.txt');
  rewrite(A);
  for I:=1 to 20 do
   begin
     Sim:=Chr(Ord('A')+I-1);
     for J:=1 to I do
         write(A,Sim);
     writeln(A);
  end;
  close(A);
end.
```

Подсчет строк по условию

В имеющемся текстовом файле расположен набор строк. Необходимо подсчитать число строк, которые начинаются с цифр. В листинге 6.4 представлена программа, которая открывает текстовый файл и построчно анализирует имеющиеся в нем данные.

Листинг 6.4. Подсчет строк в файле по условию

```
program listing_6_4;
var
   A: text;
   S: string;
   Cols: integer;
begin
  assign(A,'prim4.txt');
  reset(A);
  Cols:=0;
  while not eof(A) do
    begin
      readln(A,S);
```

```
      if (S[1]>='0') and (S[1]<='9') then
         Cols:=Cols+1;
    end;
   close(A);
   writeln(Cols);
end.
```

Создание файла на основании другого файла

Считаем, что мы располагаем текстовым файлом с набором строк. Необходимо создать другой файл с теми же строками, только в начале каждой строки нужно поставить ее порядковый номер. При этом следует в новом файле отделить пробелом номер строки и содержание самой строки. В листинге 6.5 приведена необходимая программная разработка.

Листинг 6.5. Создание нового файла на основании имеющегося

```
program listing_6_5;
var
   A,B: text;
   S: string;
   I: integer;
begin
  assign(A,'prim5.txt');
  assign(B,'prim51.txt');
  reset(A);
  rewrite(B);
  I:=0;
  while not eof(A) do
    begin
      readln(A,S);
      I:=I+1;
      writeln(B,I,' ',S);
    end;
   close(A);
   close(B);
end.
```

Теперь скорректируем рассмотренную задачу. А именно добавление номеров строк реализуем не в новом файле, а в имеющемся. В листинге 6.6 приведена необходимая программная разработка, где мы воспользовались двумя новыми процедурами:

- **erase**(A) — удаляет неоткрытый файл, задаваемый файловой переменной A;
- **rename**(A,B) — переименовывает файл, задаваемый файловой переменной A в файл с именем B.

Листинг 6.6. Коррекция существующего файла

```
program listing_6_6;
var
  A,B: text;
  S: string;
  I: integer;
begin
  assign(A,'prim6.txt');
  assign(B,'prim61.txt');
  reset(A);
  rewrite(B);
  I:=0;
  while not eof(A) do
    begin
       readln(A,S);
       I:=I+1;
       writeln(B,I,' ',S);
    end;
   close(A);
   close(B);
   erase(A);
   rename(B,'prim6.txt');
end.
```

Обмен символов в файле

Необходимо в текстовом файле произвести замену первого и второго символа в каждой строке (первый символ меняется местами со вторым, при условии наличия в строке не менее двух символов). В листинге 6.7 приведена необходимая программная разработка.

Листинг 6.7. Обмен символов в файле

```
program listing_6_7;
var
  A,B: text;
  S: string;
  C: char;
begin
  assign(A,'prim7.txt');
  assign(B,'prim71.txt');
  reset(A);
  rewrite(B);
  while not eof(A) do
    begin
      readln(A,S);
```

```
      C:= S[1];
      S[1]:= S[2];
      S[2]:= C;
      writeln(B,S);
    end;
    close(A);
    close(B);
    erase(A);
    rename(B,'prim7.txt');
end.
```

Добавление в файл информации

Кроме создания и перезаписи файла, в Паскале существует возможность дополнения информации в имеющемся файле. Для этого предназначена стандартная процедура **append**(A), где A — идентификатор файловой переменной. Данная процедура позволяет открыть имеющийся файл для его дополнения. В этом случае указатель ставится не в начало, а в конец файла, куда будут дописываться новые компоненты. Файловая переменная, как обычно, предварительно должна быть связана с файлом с помощью уже знакомой процедуры **assign**.

Заметим, что после обращения к процедуре **append** текстовый файл доступен только для записи, и функция **eof** всегда принимает значение **true**.

В листинге 6.8 приведена необходимая программа, которая позволяет дописать в конец файла общее число содержащихся в нем строк, а также количество строк, начинающихся с пробела.

Листинг 6.8. Добавление в файл информации о количестве строк

```
program listing_6_8;
var
   A: text;
   S: string;
   Cols1,Cols2: integer;
begin
  assign(A,'prim8.txt');
  reset(A);
  Cols1:=0;
  Cols2:=0;
  while not eof(A) do
    begin
      readln(A,S);
      Cols1:=Cols1+1;
      if S[1]= ' ' then
        Cols2:=Cols2+1;
    end;
```

```
 close(A);
 append(A);
 writeln(A);
 writeln(A,Cols1,' ',Cols2);
 close(A);
end.
```

Программа вывода на экран собственного текста

В листинге 6.9 приведена программа, которая позволяет вывести на экран текст, содержащийся в ней. В соответствии с программным кодом имя файла, включающего программу, должно быть: `list69.pas`.

Листинг 6.9. Вывод собственного текста на экран

```
program listing_6_9;
var
  A: text;
  S: string;
begin
  assign(A,'list69.pas');
  reset(A);
  while not eof(A) do
   begin
     readln(A,S);
     writeln(S);
   end;
 close(A);
end.
```

Нахождение максимальных чисел в строках

Текстовый файл `prim10.txt` состоит исключительно из целых положительных чисел. В листинге 6.10 приведена программа, которая позволяет найти и вывести на экран максимальное число, содержащееся в этом файле.

Листинг 6.10. Нахождение максимального числа в файле

```
program listing_6_10;
var
  A:text;
   I:integer;
   MaxMum:integer;
begin
  assign(A,'prim10.txt');
```

```
  reset(A);
  MaxMum:= 0;
  while not eof(A) do
   begin
     while not eoln(A) do
       begin
         read(A,I);
         if I > MaxMum then
             MaxMum:=I;
       end;
    readln(A);
  end;
  writeln(MaxMum);
end.
```

Нахождение среднего арифметического

Текстовый файл состоит исключительно из целых чисел. В листинге 6.11 приведена программа, которая позволяет вычислить и вывести на экран среднее арифметическое значение по каждой строке.

Листинг 6.11. Нахождение среднего арифметического по каждой строке файла

```
program listing_6_11;
var
  A:text;
  Sr:real;
  I,N:integer;
begin
  assign(A,'prim11.txt');
  reset(A);
  while not eof(A) do
  begin
    N:=0;
    Sr:=0;
    while not eoln(A) do
     begin
       read(A,I);
       N:=N+1;
       Sr:=Sr+I;
     end;
     if ( N > 0) then
        writeln(I, ' ',Sr/N);
    readln(A);
  end;
end.
```

Анализ файла

В текстовом файле необходимо подсчитать количество различных цифр, т. е. вычислить отдельно количество единиц, двоек, троек и т. д. В листинге 6.12 приведена программа, решающая данную задачу. Для подсчета цифр мы создали массив Mass из десяти элементов.

Листинг 6.12. Подсчет количества различных цифр

```
program listing_6_12;
var
  A:text;
  Sum:real;
  Mass:array[0..9] of integer;
  S:string;
  I,N:integer;
begin
  assign(A,'prim12.txt');
  reset(A);
  for I:=0 to 9 do
    Mass[I]:=0;
  while not eof(A) do
   begin
     readln(A,S);
     for I:=1 to length(S) do
        if ( S[I]>= '0') and (S[I]<='9') then
            Mass[Ord(S[I])- Ord('0')]:= Mass[Ord(S[I])- Ord('0')]+1;
   end;
   for I:=0 to 9  do
     writeln(I,' ',Mass[I]);
end.
```

Обмен содержимого файлов

Имеются два текстовых файла. Необходимо поменять местами их содержимое, но при этом не использовать процедуры переименования файлов. В листинге 6.13 приведена необходимая программная разработка, где принято, что исходные файлы имеют следующие имена:

- pr13_1.txt;
- pr13_2.txt.

Листинг 6.13. Обмен содержимого файлов

```
program listing_6_13;
var
  A,B,C: text;
  S,T: string;
```

```
 begin
  assign(A,'pr13_1.txt');
  assign(B,'pr13_2.txt');
  assign(C,'pr13_3.txt');
  reset(A);
  rewrite(C);
  while not eof(A) do
   begin
      readln(A,S);
      writeln(C,S);
   end;
  close(A);
  close(C);
  reset(B);
  rewrite(A);
  while not eof(B) do
   begin
     readln(B,S);
     writeln(A,S);
  end;
  close(A);
  close(B);
reset(C);
rewrite(B);
while not eof(C) do
   begin
     readln(C,S);
     writeln(B,S);
  end;
  close(B);
  close(C);
end.
```

Разделение файла

Имеется текстовый файл. Необходимо создать еще два файла: в один файл войдут короткие строки исходного файла (которые имеют длину меньше десяти символов), а в другой необходимо поместить все остальные строки. В листинге 6.14 приведена программа, где принято, что исходные файлы имеют следующие имена:

- `pr14_1.txt` — исходная информация;
- `pr14_2.txt` — файл с короткими строками;
- `pr14_3.txt` — файл с длинными строками.

Листинг 6.14. Разделение файла на два

```
program listing_6_14;
var
   A,B,C: text;
   S:string;
 begin
  assign(A,'pr14_1.txt');
  assign(B,'pr14_2.txt');
  assign(C,'pr14_3.txt');
  reset(A);
  rewrite(B);
  rewrite(C);
  while not eof(A) do
   begin
    readln(A,S);
    if length(S) <10 then
        writeln(B,S)
    else
       writeln(C,S);
   end;
   close(A);
   close(B);
   close(C);
 end.
```

Добавление в файл информации о количестве символов

На основе текстового файла (`prim15_1.txt`) необходимо создать другой, который будет, в отличие от исходного, включать в конце каждой строки информацию о количестве символов в строке. В листинге 6.15 приведена необходимая программная разработка.

Листинг 6.15. Добавление в файл информации о числе символов в строках

```
program listing_6_15;
var
   A,B: text;
   S:string;
 begin
  assign(A,'pr15_1.txt');
  assign(B,'pr15_2.txt');
  reset(A);
  rewrite(B);
```

```
  while not eof(A) do
    begin
       readln(A,S);
       writeln(B,S,'  ',length(S));
   end;
  close(A);
  close(B);
end.
```

Типизированные файлы

При хранении в файле однородной информации использование текстового файла не очень эффективно. В этом случае системой производится постоянное преобразование данных из символьного вида в другой и обратно. В данной ситуации на практике активно используются типизированные файлы.

Типизированные файлы состоят из последовательности элементов одного типа и длины. При этом число таких элементов не ограничено, а сам файл может быть очень большой. Структура файла определяется тем, что у каждого элемента имеется номер. При этом начальный элемент считается нулевым, следующий — первый и т. д. Номер последнего элемента на единицу меньше длины файла.

В каждый момент времени программе доступен только текущий элемент, на который установлен указатель файла. После выполнения операции чтения или записи для N-го элемента указатель автоматически смещается к следующему по порядку элементу, который будет иметь номер N+1 и который после этого становится доступным для чтения или записи. В Паскале имеется стандартная процедура, позволяющая перемещать указатель и тем самым легко переходить от одного элемента файла к другому.

В отличие от текстового файла типизированный файл не делится на строки, а представляет собой просто набор данных определенного типа. Например, в типизированный файл можно записать массив данных вещественного или целочисленного типа.

Основные моменты работы с типизированными файлами похожи на работу с текстовыми. Это касается открытия, чтения, создания, записи и закрытия файла. Так, для работы с типизированным файлом необходимо в разделе описаний ввести файловую переменную:

```
var Имя переменной :file of тип,
```

где тип — любой тип в системе Turbo Pascal.

Например, описание файловой переменной A для работы с файлом, содержащим числовые данные типа **real**, выглядит так:

```
var
   A:file of real;
```

Следующий шаг после описания файловой переменной заключается в связывании ее с именем файла на диске. Для этого предназначена знакомая процедура **assign**(A,B), где A — имя файловой переменной, а B — имя файла на диске.

Полный пример использования описанных шагов выглядит так:

```
var
    A:file of real;
begin
        assign(A,B);
```

После установки файловой переменной на конкретный файл можно приступить к непосредственной работе с ним.

Открытие типизированного файла можно произвести стандартными способами: **reset** и **rewrite**. Процедура **reset** используется для открытия существующего файла, а процедура **rewrite** — для создания нового файла и его открытия.

Для чтения и записи типизированного файла применяются процедуры **read** и **write**, а использование процедур **readln** и **writeln** запрещено. В связи с тем, что типизированный файл содержит элементы определенного типа, то при использовании операторов **read** и **write** необходимо применять переменные этого определенного типа.

Создание нового типизированного файла

Для создания нового типизированного файла и внесения в него информации необходимо выполнить последовательность шагов:

- создать файловую переменную;
- связать ее с физическим файлом (**assign**);
- открыть файл для записи (**rewrite**);
- внести необходимую информацию в файл (**write**);
- закрыть файл (**close**).

Ключевой в этом перечне является процедура вывода данных в файл:

```
write(A,X1,X2...),
```

где в списке параметров перечисляются переменные для вывода информации. Часто (такая ситуация будет характерна и для последующих примеров) используются операторы цикла, поэтому вместо перечня включается только одна переменная.

Рассмотрим теперь полноценный пример создания файла, внесения в него целочисленной информации и последующего закрытия файла.

В листинге 6.16 приведена необходимая программа, где используются уже знакомые стандартные процедуры работы с файлами. В результате ее выполнения на диске в папке, установленной в качестве рабочей, будет создан файл с набором целочисленных данных. Однако если его открыть в приложении Блокнот, то знакомые цифры будут заменены непонятными значками. Это связано с численным

форматом данных и для извлечения содержимого файла необходимо воспользоваться другой программой на Паскале, которую мы далее рассмотрим.

Листинг 6.16. Создание типизированного файла с целочисленными данными

```
program listing_6_16;
var
   A:file of integer;
   B:integer;
   I,N:integer;
begin
   assign(A,'li_16.txt');
   rewrite(A);
   writeln('Введите число записей в файл');
   readln(N);
   for I:=1 to N do
     begin
           B:=random(100);
           write(A,B);
           writeln(B);
     end;
 close(A);
end.
```

Чтение данных из типизированного файла

Для считывания информации из типизированного файла необходимо выполнить следующую последовательность шагов:

- создать файловую переменную;
- связать ее с физическим файлом;
- открыть файл для чтения;
- считать необходимую информацию из файла (при этом следует проверить, достигнут ли конец файла);
- закрыть файл.

Ключевой в этом перечне является процедура чтения данных из файла:

```
read(A, X1, X2...),
```

где в списке параметров перечисляются переменные, куда будут внесены считанные данные. Однако данная процедура не проверяет достижение конца файла и поэтому для этого следует использовать знакомую функцию `eof(A)`.

Перейдем к практике и рассмотрим ситуацию, которая заключается в открытии созданного в предыдущем примере файла `li_16.txt`. После этого из него выбираются данные и выводятся на экран.

В листинге 6.17 приведена необходимая программа, где используются уже знакомые стандартные процедуры работы с файлами. В результате ее выполнения на экране мы увидим присутствующие в файле данные.

Листинг 6.17. Чтение содержания типизированного файла с целочисленными данными

```
program listing_6_17;
var
   A:file of integer;
   B:integer;
begin
   assign(A,'li_16.txt');
   reset(A);
   while not eof(A) do
      begin
         read(A,B);
         writeln(B);
      end;
  close(A);
end.
```

Последовательный доступ к файлу

Файл представляет собой последовательную структуру данных. После открытия файла становятся доступными последовательно все элементы: первый, второй и т. д. Таким образом, можно последовательно считывать один элемент за другим. Если необходимо считать выборочно только несколько элементов, то придется считывать и предыдущие. В листинге 6.18 приведена программа, в которой считываются первый и третий элементы из ранее созданного файла `li_16.txt`. После считывания программа выводит извлеченные данные на экран.

Листинг 6.18. Выборочное чтение содержания типизированного файла

```
program listing_6_18;
var
   A:file of integer;
   B1,B3:integer;
begin
    assign(A,'li_16.txt');
    reset(A);
    read(A,B1);
    read(A,B3);
    read(A,B3);
    writeln(B1,' ',B3);
    close(A);
end.
```

Прямой доступ к файлу

Последовательная технология работы с содержанием файла неэффективна. Удобнее пользоваться прямым доступом к отдельным элементам файла. В этом случае задается номер интересующего элемента в файле. При прямом доступе к файлу его элементы нумеруются от 0 до N–1, где N — общее число элементов в файле. Заметим, что первый элемент в файле имеет номер 0.

При прямом доступе к файлу, кроме уже знакомых, полезен еще ряд стандартных функций. Так, функция **filesize**(A) позволяет получить число имеющихся элементов в открытом файле A. Эта функция возвращает значение типа **longint**. Если же файл пустой, то данная функция вернет 0.

Рассмотрим пример, в котором открывается ранее созданный файл li_16.txt с набором целых чисел. В листинге 6.19 приведена программа, которая демонстрирует функцию **filesize** и выводит на экран количество записей в файле.

Листинг 6.19. Получение числа данных в файле

```
program listing_6_19;
var
   A:file of integer;
   B:longint;
begin
    assign(A,'li_16.txt');
    reset(A);
    B:=filesize(A);
    writeln(B);
    close(A);
end.
```

Функция **filepos**(A) возвращает текущую позицию в файле A, который должен быть предварительно открыт. Возвращаемое функцией значение имеет тип **longint**. Если файл только что открылся, то **filepos**(A)=0, а после прочтения последнего компонента из файла значение **filepos**(A) совпадает со значением **filesize**(A), что указывает на достижение конца файла.

Также можно организовать и непосредственную проверку текущей позиции в файле:

```
if filepos(A)=filesize(A) then
    writeln('Окончание файла достигнуто');
```

Процедура **seek**(A,N) устанавливает указатель в открытом файле (с файловой переменной A) на компонент с номером N (как следует из предыдущих рассуждений нумерация идет от 0). После такой установки значение компонента файла может быть считано с помощью процедуры **read**.

Рассмотрим пример, касающийся использования данной процедуры. Будем считать, что задача заключается в коррекции ранее созданного файла li_16.txt. В нем необ-

ходимо поменять местами третий и пятый элементы. В листинге 6.20 приведена программа, которая решает поставленную задачу.

Листинг 6.20. Обмен значений в файле

```
program listing_6_20;
var
   A:file of integer;
   B,I,J:integer;
begin
    assign(A,'li_16.txt');
    reset(A);
    { Вывод на экран информации из исходного файла }
    while not eof(A) do
      begin
         read(A,B);
         write(B,' ');
      end;
  seek(A,3);
  read(A,I);
  seek(A,5);
  read(A,J);
  seek(A,3);
  write(A,J);
  seek(A,5);
  write(A,I);
{ Вывод на экран информации из нового файла }
 seek(A,0);
 while not eof(A) do
   begin
      read(A,B);
      write(B,' ');
   end;
  close(A);
end.
```

Рассмотрим еще один пример на эту тему. Необходимо в том же файле `li_16.txt` поменять местами максимальный и минимальный элементы. Считается, что максимальный и минимальный элементы присутствуют в файле в единственном экземпляре. В листинге 6.21 приведена программа, которая решает поставленную задачу.

Листинг 6.21. Обмен максимального и минимального элементов

```
program listing_6_21;
var
     A: file of integer;
```

```
     B,Max,Min: integer;
     Imax,Imin,I: longint;
begin
    assign(A,'li_16.txt');
    reset(A);
    writeln;
   { Вывод на экран информации из исходного файла }
    while not eof(A) do
      begin
         read(A,B);
         write(B,' ');
      end;
    writeln;
    seek(A,0);
    Max:=-MaxInt;
    Min:=MaxInt;
    for I:=0 to filesize(A)-1 do
       begin
          read(A,B);
          if B > Max then
             begin
                Max:=B;
                Imax:=I;
             end;
          if B < Min then
             begin
                Min:=B;
                Imin:=I;
             end;
       end;
       seek(A,Imax);
       write(A,Min);
       seek(A,Imin);
       write(A,Max);
       seek(A,0);
   { Вывод на экран информации из нового файла }
    while not eof(A) do
      begin
         read(A,B);
         write(B,' ');
      end;
   close(A);
end.
```

Процедура **truncate**(A) отсекает часть открытого файла, начиная с текущего компонента, и подтягивает на его место конец файла. Таким образом, с помощью **truncate** можно удалять фрагменты файла.

Рассмотрим пример, в котором необходимо удалить из файла все последующие элементы после нахождения первого максимального элемента. В листинге 6.22 приведена программа, которая решает поставленную задачу.

Листинг 6.22. Удаление фрагмента файла

```
program listing_6_22;
var
  A: file of integer;
  I,Max,B:integer;
begin
   assign(A,'li_16.txt');
   reset(A);
   writeln;
   { Вывод на экран информации из исходного файла }
    while not eof(A) do
      begin
         read(A,B);
         write(B,' ');
      end;
   seek(A,0);
   for I:=0 to filesize(A)-1 do
       begin
         read(A,B);
         If I=0 then
             Max:=B;
         if B > Max then
             Max:=B;
       end;
   seek(A,0);
   for I:=0 to filesize(A)-1 do
    begin
       read(A,B);
       if B = Max then
         break;
    end;
   truncate(A);
   seek(A,0);
  { Вывод на экран информации из нового файла }
    while not eof(A) do
      begin
         read(A,B);
         write(B,' ');
      end;
   close(A);
end.
```

Создание массива данных в файле

Необходимо разработать программу, которая формирует в файле набор вещественных чисел, представляющих собой значения функции синус (для определенности 10 значений). После этого другая программа должна открыть этот файл, вычислить сумму имеющих чисел, а затем эту сумму зафиксировать в файле (заменить имеющиеся в нем данные).

Первая разработка, связанная с формированием значений, приведена в листинге 6.23.

Листинг 6.23. Формирование файла с данными

```
program listing_6_23;
var
  A:file of real;
  B:real;
  I:integer;
begin
  assign(A,'pr_23.dat');
  rewrite(A);
  for I:=1 to 10 do
   begin
     B:=sin(0.1*I);
     write(A,B);
   end;
  close(A);
end.
```

В результате выполнения программы, приведенной в листинге 6.23, на диске появляется файл `pr_23.dat` с набором вещественных чисел.

Теперь перейдем к другой программе (листинг 6.24), которая открывает созданный файл, выполняет вычисление, а затем заменяет в файле информацию.

Листинг 6.24. Обработка файла с данными

```
program listing_6_24;
var
  A: file of real;
  B,C: real;
  I: integer;
begin
  assign(A,'pr_23.dat');
  reset(A);
  C:=0;
```

```
  while not eof(A) do
   begin
     read(A, B);
     C:=C+B;
   end;
  seek(A,0);
  write(A,C);
  truncate(A);
  seek(A,0);
  while not eof(A) do
   begin
     read(A, B);
     writeln(B);
   end;
   close(A);
end.
```

Организация базы данных учащихся

В этом разделе мы рассмотрим еще один практический пример работы с записями. Так, в наборе записей файла будем хранить фамилии, имена и года рождения группы учащихся. Первая программа (листинг 6.25) должна обеспечить ввод информации и размещение ее на диске компьютера.

Листинг 6.25. Формирование списка учащихся в файле

```
program listing_6_25;
Type
  Stud=record
  Fam: string[50];
  Imya: string[50];
  God: integer;
end;
var
  A: file of Stud;
  B: Stud;
  N,I: integer;
 begin
  assign(A,'pr_25.dat');
  rewrite(A);
  writeln('Введите число учащихся');
  readln(N);
  for I:=1 to N do
    begin
       writeln('Введите фамилию');
       readln(B.Fam);
```

```
        writeln('Введите имя');
        readln(B.Imya);
        writeln('Введите год рождения');
        readln(B.God);
        write(A,B);
    end;
 close(A);
end.
```

Вторая программа (листинг 6.26) открывает созданный в предыдущем примере файл и отражает на экране все года рождения учащихся (без повторений).

Листинг 6.26. Формирование списка учащихся по условию

```
program listing_6_26;
Type
  Stud=record
  Fam:string[50];
  Imya: string[50];
  God:integer;
end;
var
   B: Stud;
   A: file of Stud;
   N,I,J,M,Flag:integer;
   Mass: array [1..100] of integer;
 begin
    assign(A,'pr_25.dat');
    reset(A);
    N:=filesize(A);
    M:=0;
    for I:=1 to N do
     begin
       read(A,B);
       Flag:=0;
       for J:=1 to M do
         if B.God = Mass[J] then
             begin
                Flag:=1;
                break;
             end;
       if Flag = 0 then
          begin
             M:=M+1;
             Mass[M]:= B.God;
```

```
            writeln('Year=',B.God);
          end;
      end;
  close(A);
end.
```

Разделение списка учащихся

Необходимо обеспечить формирование файла со списком учащихся. Формат записи выглядит так:

- фамилия и инициалы (строка);
- курс (числовой формат);
- специальность (строка).

После этого необходимо разбить файл на три. В одном файле требуется оставить студентов первого курса, а в двух других, соответственно, второго и всех оставшихся курсов.

Программная разработка, которая заполняет исходный файл информацией, представлена в листинге 6.27. В результате в файле `pr_27.dat` мы получим список всех учащихся.

Листинг 6.27. Формирование списка учащихся в файле

```
program listing_6_27;
Type
  Stud=record
  Fam: string[50];
  Kurs: integer;
  Spets: string[50];
end;
var
  A: file of Stud;
  B: Stud;
  N,I: integer;
 begin
  assign(A,'pr_27.dat');
  rewrite(A);
  writeln('Введите число учащихся');
  readln(N);
  for I:=1 to N do
    begin
       writeln('Введите фамилию');
       readln(B.Fam);
       writeln('Введите курс');
       readln(B.Kurs);
       writeln('Введите специальность');
       readln(B.Spets);
```

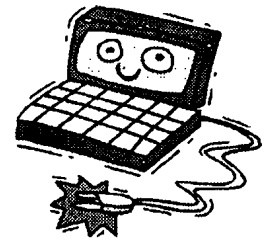
```
        write(A,B);
    end;
 close(A);
end.
```

Следующая программа (листинг 6.28) открывает созданный в предыдущем примере файл и разбивает его на три по различным курсам.

Листинг 6.28. Разбивка списка учащихся по курсам

```
program listing_6_28;
Type
  Stud=record
  Fam: string[50];
  Kurs: integer;
  Spets: string[50];
end;
var
   B: Stud;
   A,F1,F2,F3: file of Stud;
   N,I:integer;
   Mass: array [1..100] of integer;
 begin
    assign(A,'pr_27.dat');
    assign(F1,'pr_281.dat');
    assign(F2,'pr_282.dat');
    assign(F3,'pr_283.dat');
    reset(A);
    rewrite(F1);
    rewrite(F2);
    rewrite(F3);
    N:=filesize(A);
    for I:=1 to N do
      begin
        read(A,B);
        if B.Kurs = 1 then
            write(F1,B)
        else if B.Kurs = 2 then
          write(F2,B)
        else
          write(F3,B);
      end;
   close(A);
   close(F1);
   close(F2);
   close(F3);
end.
```

ГЛАВА 7



Подпрограммы

Большие программы не принято разрабатывать в виде единого целого. Эффективнее большинство разработок разделять на отдельные функциональные компоненты. Для этого в языке Паскаль, как и в других языках программирования, имеется важный ресурс — подпрограммы. Каждая подпрограмма представляет собой самостоятельный фрагмент программного кода. Этот фрагмент, как правило, связан с основной программой с помощью параметров (входных и выходных).

Понятно, что когда программный код программы не очень большой (30—40 строк), то и необходимости выделять в нем отдельные самостоятельные фрагменты нет. Однако решение серьезных алгоритмических задач всегда требует существенно большего количества программных строк. И писать такие программы без разделения их на отдельные фрагменты совершенно не эффективно.

Разбиение программ на подпрограммы разумно по двум причинам. Во-первых, каждая подпрограмма существует в памяти в единственном экземпляре, а вызывать ее можно многократно. Вторая причина касается методики нисходящего программирования. В этом случае алгоритм представляется в виде совокупности подпрограмм, реализующих самостоятельные части алгоритма. Подпрограммы, в свою очередь, могут разбиваться на более мелкие подпрограммы.

Подпрограмма — это именованная, логически завершенная группа операторов, которую можно вызвать для выполнения любое количество раз из различных мест программы. Подпрограммы подразделяются на две категории: *процедуры* и *функции*. Отличие их друг от друга незначительное и заключается в том, что функция выдает значение, являющее результатом ее работы. Это значение можно использовать в выражениях. В качестве примера среди стандартных функций следует упомянуть тригонометрические функции (синус и косинус), с которыми мы уже встречались в этой книге.

Еще одна классификация — все процедуры и функции подразделяются на две группы: встроенные и определенные пользователем. *Встроенные* (стандартные) процедуры и функции могут вызываться по имени без предварительного описания. Их наличие существенно облегчает разработку прикладных программ. Однако в

большинстве случаев некоторые специфические действия требуют написания собственных *пользовательских* процедур и функций.

Вызов пользовательских функций и процедур реализуется простым упоминанием их имени, что похоже на вызов стандартных процедур и функций. Однако пользовательские подпрограммы и функции необходимо предварительно описать (как известно, любое имя в программе должно быть определено). Для этого предназначен раздел описаний. Описать подпрограмму это значит определить ее заголовок и тело (см. далее).

Организация процедур

Процедура включает заголовок и тело процедуры. Заголовок состоит из зарезервированного слова **procedure**, *имени процедуры* и необязательного, заключенного в круглые скобки, списка формальных параметров с указанием типа каждого параметра:

```
procedure Имя процедуры (Формальные параметры);
```

Приведем пример двух заголовков процедур, соответственно, без параметров и с включением параметров:

```
procedure MyProc1;
procedure MyProc2 (A:integer; B:real);
```

Имя процедуры должно быть уникально, т. е. его нельзя использовать повторно в программе для названия процедур.

Примечание

Параметры подпрограммы должны отделяться друг от друга точкой с запятой.

Тело процедуры по своей структуре аналогично обычной программе:

```
begin
{Инструкции исполнительной части}
end;
```

Важно отметить, что в конце процедуры, как и в конце программы, стоит ключевое слово **end** с обязательным завершающим символом — точкой с запятой.

Для обращения к процедуре используется *оператор вызова процедуры*. Он состоит из имени процедуры и списка *фактических* параметров, заключенных в круглые скобки и отделенных друг от друга запятыми. Если процедуре не передается никаких значений, то список параметров отсутствует.

Параметры обеспечивают механизм замены, который позволяет выполнять процедуру с различными исходными данными. Между фактическими параметрами в операторе вызова процедуры и формальными параметрами в заголовке описания процедуры устанавливается однозначное соответствие в результате их перебора слева направо. Количество и тип формальных параметров соответствуют количест-

ву и типу фактических параметров. Заметим, что соответствующие друг другу параметры *не обязательно* должны одинаково обозначаться.

Процедура может возвращать в основную программу необходимые значения. Для этого соответствующие переменные должны быть описаны как параметры-переменные с использованием ключевого слова **var**.

Использование имени процедуры в тексте основной программы приводит к вызову данной процедуры (к выполнению программного кода, содержащегося в ней). После выполнения последнего оператора, имеющегося в процедуре, управление передается в основную программу. В этом случае будут выполняться операторы, следующие за строкой вызова процедуры. Таким образом, если отдельные фрагменты программного кода оформить как процедуры, то в основной программе вместо них останутся только операторы их вызова и основная программа станет существенно компактнее.

Любая процедура может, в свою очередь, иметь вложенные процедуры. При этом каждая процедура имеет совокупность имен. Считается, что все описанные имена видимы внутри процедуры и невидимы снаружи. При входе в процедуру становятся доступны не только имена, определенные внутри нее, но и все имена, указанные выше (имена верхнего уровня). Эти объекты называются *глобальными*, в отличие от *локальных*, определенных внутри процедуры.

Ключевой момент здесь связан с тем, что имена, описанные ниже по тексту программы, недоступны в выше приведенном программном коде. Например, в конструкции:

```
var A;
   procedure B
      var C;
        ...
   end; {Завершение процедуры B}
var D;
```

из процедуры B можно обратиться к переменным A и C, но нельзя обратиться к переменной D.

Имена, определенные локально в процедуре, могут совпадать с глобальными именами. В этом случае локальное имя закрывает глобальное, и оно становится невидимым в процедуре. Например:

```
var A;
procedure S;
 var A;
 ...
end;
```

Здесь глобальная переменная A в процедуре S не видна (локальная переменная закрывает глобальную).

Важное значение имеет соответствие формальных и фактических параметров при вызове процедур. Система Turbo Pascal контролирует соответствие этих параметров.

Любой из формальных параметров процедуры может быть:

- параметром-значением;
- параметром-переменной;
- параметром-константой.

Параметры-значения определяются указанием имени и типом данных:

```
procedure MyProc2 (A:integer; B:real);
```

При использовании параметров-переменных используется ключевое слово `var`:

```
procedure MyProc2 (var A:integer;var B:real);
```

Использование параметров-констант:

```
procedure MyProc2 (const A:integer;const B:real);
```

Определение формального параметра тем или иным способом важно только для вызывающей программы: если формальный параметр объявлен как параметр-переменная, то при вызове процедуры ему должен соответствовать фактический параметр в виде переменной нужного типа.

Если формальный параметр объявлен как параметр-значение или как параметр-константа, то при вызове ему может соответствовать произвольное значение.

Если мы используем параметр-значение, то перед вызовом процедуры это значение вычисляется, а полученный экземпляр копируется во временную память и передается подпрограмме.

Если параметр определен как параметр-переменная, то процедуре передается адрес переменной (место, где переменная располагается). И изменение параметра-переменной в процедуре приводит к изменению самого фактического параметра.

Параметры-переменные используются как средство связи алгоритма, реализованного в подпрограмме, с внешним миром: с помощью этих параметров подпрограмма может передавать результаты своей работы вызывающей программе. При этом в распоряжении программиста есть и другой способ передачи результатов — через глобальные переменные. Однако в соответствии с хорошим стилем программирования рекомендуется там, где возможно, использовать передачу результатов через фактические параметры-переменные.

Параметры-массивы

Из предыдущего изложения использование в качестве параметров числовых типов вполне понятно. Однако, как мы уже знаем, существуют такие типы данных, как массивы и строки. Об их использовании в параметрах процедур мы здесь и поговорим.

Дело в том, что в списке формальных параметров может быть только стандартный или объявленный тип. В связи с этим нельзя объявить следующую процедуру:

```
procedure S (A: array [1..5] of real);
```

Если теперь мы собираемся передать в подпрограмму весь массив, то следует предварительно создать новый тип данных:

```
type
   Ctype = array [1..5] of real;
```

а уже потом использовать его в описании процедуры:

```
procedure S (A: Ctype);
```

Примеры использования процедур

Теперь после рассмотрения необходимых теоретических сведений мы разберем типовые примеры использования процедур в практическом программировании.

Формирование разделяющей линии

Рассмотрим для начала наиболее простую организацию процедуры — без включения параметров. Пример будет касаться добавления оформления при выводе информации на экране. А именно необходимо обеспечить формирование обрамляющих горизонтальных линий в дополнение к информативным данным. В качестве линий будем использовать последовательность символов подчеркивания (_).

Формирование одной такой линии мы обеспечим с помощью процедуры `Line`. Полный текст процедуры и вызывающей (основной) программы представлен в листинге 7.1. Здесь в основной программе производится вычисление и вывод нескольких значений функции синус. При этом начало и завершение вывода значений отмечается горизонтальными линиями.

Листинг 7.1. Использование процедуры для формирования линии

```
program listing_7_1;
procedure Line;
 var
   J:Integer;
 begin
   for J:= 1 to 50  do
      write('_');
   writeln;
 end;
var
   X,Y:real;
   I:Integer;
 begin
 Line;
 for I:= 1 to 5 do
  begin
    X:=0.1*I;
    Y:=sin(X);
```

```
    writeln('X=',X,'  Y=',Y);
  end;
 Line;
end.
```

Поясним теперь более детально содержание листинга 7.1. Начало программы обычное: после заголовка располагается процедура, начинающаяся с ключевого слова **procedure**, за которым следует ее название — `Line`. После названия процедуры до обозначения ее завершения размещается текст. Структура текста процедуры традиционна — область описания переменных и тело процедуры (операторы процедуры должны располагаться между операторами **begin** и **end;**). Содержательной компонентой процедуры является выполнение цикла по выводу 50 символов подчеркивания и перевод на следующую строку.

В основной программе первым выполняемым оператором является вызов процедуры `Line`, что автоматически приводит к выполнению программного кода, содержащегося в процедуре. В результате на экране отображается линия. После этого в основной программе формируется вывод пяти значений функции **sin**. В предпоследней строке основной программы производится еще один вызов процедуры `Line` и линия на экране завершает отражение результатов работы программы.

Рассмотренная ситуация достаточно простая, т. к. из основной программы в процедуру мы не передавали никаких параметров.

Передача символа рисования линии

Теперь усложним ситуацию предыдущего примера — обеспечим передачу в процедуру кода символа для рисования линии. В этом случае линия необязательно будет оформлена с помощью символа подчеркивания. Можно выбрать звездочку или другой понравившийся символ. Текст процедуры вместе с основной программой представлен в листинге 7.2.

Листинг 7.2. Передача кода символа для рисования

```
program listing_7_2;
procedure Line(C:Char);
 var
   J:Integer;
 begin
   for J:= 1 to 50  do
       write(C);
   writeln;
end;
var
   X,Y:real;
   I:Integer;
begin
 Line('*');
```

```
 for I:= 1 to 5 do
  begin
   X:=0.1*I;
   Y:=sin(X);
   writeln('X=',X,'  Y=',Y);
 end;
 Line('/');
end.
```

Итак, мы увидели, каким образом из основной программы в процедуру можно передавать данные. В этом случае после имени процедуры в скобках указываются формальные параметры и их типы. В примере, приведенном в листинге 7.2, мы использовали один параметр с типа **char**.

В процедуре можно использовать формальные параметры так, как будто у нас есть переменные указанных типов. Важно заметить, что такие переменные существуют только до завершения процедуры (когда управление передается в основную программу, они пропадают).

При вызове процедуры (см. листинг 7.2) необходимо в качестве параметра указать символ, которым собираемся отобразить линию. В нашем случае мы использовали два разных символа: `Line('*')` и `Line('/')`.

Передача переменной для символа линии

В предыдущем примере в качестве фактического параметра из основной программы в процедуру передавалось значение константы (тот или иной символ). Однако в качестве фактических параметров могут выступать и переменные. В листинге 7.3 приведен пример, в котором в подпрограмму передается как символ для рисования линии, так и число повторений этого символа при рисовании. Второе значение мы вводим с клавиатуры и записываем в переменную. Далее значение этой переменной передается в процедуру.

Листинг 7.3. Передача значения переменной для рисования линии

```
program listing_7_3;
procedure Line(C:Char;M:Integer);
 var J:Integer;
 begin
   for J:= 1 to M  do
     write(C);
   writeln;
end;
var
   X,Y:real;
   I,N:Integer;
```

```
begin
   writeln('Ввести число повторений символа');
   read(N);
   Line('*',N);
   for I:= 1 to 5 do
    begin
      X:=0.1*I;
      Y:=sin(X);
      writeln('X=',X,'  Y=',Y);
   end;
  Line('/',N);
end.
```

Процедура анализа четности числа

Рассмотрим уже знакомую задачу: необходимо проанализировать, является ли введенное с клавиатуры целое числом четным. Но на этот раз ввод числа реализуем в основной программе, а сам анализ проведем в процедуре. В листинге 7.4 представлена необходимая программная разработка.

Листинг 7.4. Анализ четности целого числа

```
program listing_7_4;
 procedure chislo(Ch:Integer);
 begin
  if Ch mod 2 = 0 then
     writeln('Число четное')
  else
     writeln('Число нечетное')
 end;
 var
   I:Integer;
 begin
  writeln('Введите число');
  readln(I);
  chislo(I);
end.
```

Передача параметров через глобальные переменные

В рассмотренных выше примерах мы использовали передачу информации из основной программы в процедуру с помощью формальных и фактических параметров. Это наиболее стандартный вариант действий. Однако существует еще один ресурс, который приходится иногда использовать. Он связан с применением глобальных параметров.

Так, если описать переменные сразу после заголовка основной программы, то все они будут доступны как в операторах основной программы, так и в процедуре.

В листинге 7.5 приведен пример, в котором используются глобальные переменные для передачи символа и числа повторений этого символа при рисовании.

Листинг 7.5. Пример использования глобальных переменных

```
program listing_7_5;
var
  X,Y:real;
  A:char;
  I,N:Integer;
procedure Line;
var J:Integer;
 begin
   for J:= 1 to N  do
    write(A);
  writeln;
end;
begin
  writeln('Ввести число повторений символа');
  readln(N);
  writeln('Ввести символ');
  readln(A);
  Line;
  for I:= 1 to 5 do
   begin
      X:=0.1*I;
      Y:=sin(X);
      writeln('X=',X,'  Y=',Y);
  end;
  Line;
end.
```

Глобальное описание массива

Рассмотрим пример, в котором будет продемонстрирована технология работы с массивами при использовании их глобального описания. В листинге 7.6 приведен пример, в котором с помощью процедур реализуется ввод значений массива, их вывод на экран и суммирование элементов.

Листинг 7.6. Работа с массивами с использованием глобальных переменных

```
program listing_7_6;
const
  N=5;
type
  Mass=array[1..N] of integer ;
```

```
var
  L:Mass;
  Sum:longint;
procedure InputMass;
var
  I:Integer;
 begin
   for I:= 1 to N  do
     begin
       writeln('Ввести элемент массива');
       readln(L[I]);
     end;
 end;
procedure OutMass;
var
  I:Integer;
 begin
   for I:= 1 to N  do
       writeln(' I=',I, ' L[I]=',L[I]:7);
   end;
procedure SumMass;
var
  I:Integer;
 begin
   Sum:=0;
   for I:= 1 to N  do
      Sum:=Sum+L[I] ;
    writeln(' Sum=',Sum:9);
 end;
begin
  InputMass;
  OutMass;
  SumMass;
end.
```

Рассмотрим еще один пример на тему работы с массивами. Необходимо с помощью процедур обеспечить:

- заполнение массива;
- поиск максимального значения в массиве и вывод его на экран.

В листинге 7.7 приведена полная разработка, включающая основную программу и необходимые процедуры:

Листинг 7.7. Поиск максимального значения в массиве с использованием процедур

```
program listing_7_7;
const
  N=5;
```

```
type
  Mass=array[1..N] of integer;
var
   L:Mass;
   MaxMass:integer;
procedure FormMass;
 var
   I:Integer;
 begin
   for I:= 1 to N  do
    begin
      L[I]:=random(100);
      writeln(L[I]);
    end;
 end;
procedure PoiskMax;
var
  I:Integer;
 begin
   MaxMass:=L[1];
   for I:= 2 to N  do
     if L[I] > MaxMass then
        MaxMass:=L[I];
 end;
begin
  FormMass;
  PoiskMax;
  writeln(' MaxMass=',MaxMass);
end.
```

Здесь мы использовали глобальное описание массива. Если бы мы сделали описание массива перед телом основной программы, то компилятор выдал бы сообщение об ошибке. Этот ошибочный вариант построения программы приведен в листинге 7.8.

Листинг 7.8. Ошибочный вариант описания массива

```
program listing_7_8;
const
  N=5;
var
   MaxMass:integer;
procedure FormMass;
 var
   I:Integer;
 begin
   for I:= 1 to N  do
```

```
    begin
      L[I]:=random(100);
      writeln(L[I]);
    end;
end;
procedure PoiskMax;
var
  I:integer;
 begin
   MaxMass:=L[1];
   for I:= 2 to N  do
     if L[I] > MaxMass then
        MaxMass:=L[I];
 end;
type
  Mass=array[1..N] of integer;
var
   L:Mass;
begin
  FormMass;
  PoiskMax;
  writeln(' MaxMass=',MaxMass);
end.
```

Передача массива через ссылку

Особенность предыдущих примеров данной главы была связана с тем, что информация через параметры передавалась в одном направлении — из основной программы в подпрограмму. Если же требуется возвратить данные из подпрограммы в основную программу, то рассмотренный механизм не работает. Это связано с тем, что в данном случае производится передача параметров по значению.

Существует и другой механизм, называемый *передачей параметров по ссылке* (по адресу). В этом случае в подпрограмму передается не значение переменной, а ее местоположение в памяти (адрес ячейки памяти, которая хранит данную переменную). После этого в подпрограмме можно легко изменить содержимое данного адреса. Для передачи адреса переменной она должна быть описана с помощью ключевого слова **var**.

В листинге 7.9 приведен вариант поиска максимального значения в массиве с использованием передачи параметров по ссылке.

Листинг 7.9. Пример передачи массива в процедуру по ссылке

```
program listing_7_9;
const
  N=7;
```

```
type
  Mass=array[1..N] of integer;
procedure FormMass(var L:Mass);
 var
  I:Integer;
 begin
   for I:= 1 to N  do
     begin
      L[I]:=random(100);
      writeln(L[I]);
     end;
 end;
procedure PoiskMax(var M:Mass; var MaxMass:integer);
var
  I:integer;
 begin
   MaxMass:=M[1];
   for I:= 2 to N  do
    if M[I] > MaxMass then
      MaxMass:=M[I];
 end;
   var
   LL:Mass;
   MaxMass:integer;
begin
  FormMass(LL);
  PoiskMax(LL,MaxMass);
  writeln('MaxMass=',MaxMass);
end.
```

Вычисление факториала

В листинге 7.10 приведен пример вычисления факториала числа с использованием процедуры. Значение факториала передается в основную программу через параметр (M).

Листинг 7.10. Вычисление факториала числа

```
program listing_7_10;
procedure Factorial(N:integer;var M:longint);
var J:integer;
 begin
   if N >= 0 then
     begin
       M:=1;
```

```
      for J:= 1 to N  do
         M:=M*J;
    end
   else
     M:=0;
 end;
var
   M:LongInt;
   N:integer;
begin
writeln('Ввести N');
read(N);
Factorial (N,M);
writeln('N=',N,'  M=',M);
end.
```

Вычисление математических функций

Рассмотрим пример, где требуется с помощью процедуры вычислить значения нескольких математических функций, а результат вычисления передать в основную программу. В листинге 7.11 приведен такой пример для вычисления значений следующих математических функций:

- $y = \sin(x) + 4$;
- $z = \cos(3x) - 1$;
- $w = 2x - 7$.

Листинг 7.11. Вычисление значений математических функций

```
program listing_7_11;
procedure Func(X:Real;var Y:Real; var Z:Real; var W:Real);
begin
   Y:=sin(X)+4;
   Z:=cos(3*X);
   W:=2*X-7;
end;
var
    X,F1,F2,F3:real;
begin
writeln('Ввести X');
readln(X);
Func(X,F1,F2,F3);
writeln('X=',X,'  Y=',F1, '  Z=',F2, '  W=',F3);
end.
```

Обмен значений переменных

Необходимо разработать процедуру, которая будет осуществлять обмен значений двух переменных. Далее с помощью этой процедуры требуется разместить в порядке возрастания три числа, которые вводятся с клавиатуры. Разработка приведена в листинге 7.12.

Листинг 7.12. Перестановка чисел

```
program listing_7_12;
procedure Obmen(var X,Y:integer);
 var
   Z:integer;
 begin
   Z:=X; X:=Y; Y:=Z;
end;
var
   A,B,C:integer;
begin
  write('Ввести три целых числа ',A,B,C);
  readln(A,B,C);
  if A>B then
    Obmen(A,B);
  if B>C then
    Obmen(B,C);
  if A>B then
    Obmen(A,B);
    writeln('A=',A,'  B=',B, '  C=',C);
end.
```

Анализ чисел

Реализуем с помощью процедуры анализ: является ли число простым. Процедура будет иметь два параметра:

- число, которое следует проанализировать;
- переменную, представляющую собой результат, получаемый из процедуры основной программой.

В листинге 7.13 показана разработка, включающая основную программу и процедуру, выполняющую проверку чисел.

Листинг 7.13. Анализ простых чисел

```
program listing_7_13;
procedure Analiz(N:integer;var Res:boolean);
var
  J:integer;
```

```
begin
 Res:=true;
   for J:= 2 to N-1 do
      begin
       if (N mod J) = 0 then
         begin
           Res:= false;
           break;
         end;
      end;
end;
var
    X:integer;
    F:boolean;
begin
   writeln('Введите целое положительное число');
   readln(X);
   Analiz(X,F);
   writeln('Число простое - ',F);
end.
```

Функции пользователя

В большинстве случаев результатом подпрограммы является только одно значение. И в этом случае подпрограмму лучше оформить не как процедуру, а как функцию. Организация функции аналогична процедуре, но имеются два отличия:

- функция передает в программу результат своей работы — единственное значение, носителем которого является имя самой функции;
- имя функции может входить в выражение как операнд (функция возвращает результат в точку своего вызова).

Функция, определенная пользователем, состоит из заголовка и тела функции. Заголовок содержит зарезервированное слово **function** и имя функции. Кроме того, после имени функции в круглых скобках указывается список формальных параметров. А главным отличием от синтаксиса процедуры является указание типа возвращаемого значения. Таким образом, заголовок функции выглядит так:

```
function Имя функции (Формальные параметры): Тип результата;
```

Для полноты картины приведем несколько примеров описаний функций:

```
function Func1 (A:real; B:real): real;
function Func2 (A:integer;  B:real): real;
function Func3(A:boolean;  B:real): integer;
```

Тело функции по своей структуре аналогично обычной процедуре:

```
begin
{Инструкции исполнительной части функции}
end;
```

Заметим, что в разделе операторов функции должен находиться, по крайней мере, один оператор, который присваивает ее имени значение, возвращаемое как результат работы функции.

Далее рассмотрим ряд практических примеров программирования с использованием функций.

Вычисление наибольшего значения

Необходимо разработать функцию, которая вычисляет максимальное из двух вещественных чисел, полученных в качестве аргумента. Требуется также привести вызывающую программу. В листинге 7.14 данная разработка представлена целиком.

Листинг 7.14. Пример функции вычисления наибольшего значения

```
program listing_7_14;
function Comp (A:real; B:real): real;
  begin
     if A > B then
         Comp:=A
     else
         Comp:=B;
  end;
var X1,X2:real;
begin
   writeln('Введите числа');
   read(X1,X2);
   write('Максимум = ',Comp(X1,X2));
end.
```

Вычисление процента

Необходимо разработать функцию, которая вычисляет процент от числа. Эти данные (число и значение в процентах, которое требуется вычислить) передаются в качестве параметров. В листинге 7.15 данная разработка представлена вместе с вызывающей программой.

Листинг 7.15. Функция вычисления значения процента от числа

```
program listing_7_15;
function Protsent (Chislo:real; Prots:real): real;
begin
     Protsent := Chislo * Prots/100;
end;
```

```
var A,B:real;
begin
   writeln('Введите число и процент, который необходимо вычислить');
   read(A,B);
   write('Процент = ', Protsent (A,B));
end.
```

Расчет дохода по вкладу

Нужно разработать функцию, которая вычисляет доход по вкладу. Данные (сумма вклада, годовая ставка по вкладу и срок в днях) передаются в качестве параметров. В листинге 7.16 данная разработка представлена вместе с вызывающей программой.

Листинг 7.16. Функция вычисления дохода по вкладу

```
program listing_7_16;
function Protsent (Summa:real; Prots:real; Srok:real): real;
begin
     Protsent := Summa * (Prots/100)*(Srok/365);
end;
var A,B,C:real;
begin
   writeln('Введите сумму, процент и срок');
   read(A,B,C);
   write('Доход по вкладу = ', Protsent (A,B,C));
end.
```

Анализ текста

Необходимо разработать функцию, которая анализирует строку (вводимую с клавиатуры), и если в ней присутствует английская заглавная буква, то выдаваемое функцией значение должно иметь значение **true**. В противном случае функция выдает значение **false**. В листинге 7.17 данная разработка представлена целиком.

Листинг 7.17. Анализ строки текста

```
program listing_7_17;
function Books (S:string): boolean;
var
   L,I:integer;
 begin
    L:=length(S);
    Books:=false;
```

```
    for I:=1 to L  do
       if (S[I]>= 'A') and (S[I]<= 'Z') then
          begin
              Books:=true;
              break;
          end;
 end;
var
   A:string;
begin
   writeln('Введите строку');
   readln(A);
   write('Заглавные английские буквы - ', Books(A));
end.
```

Функция поиска минимума в одномерном массиве

В *главе 3* мы рассматривали задачу поиска максимального значения в одномерном массиве. В листинге 7.18 приведено решение аналогичной задачи — поиска минимального значения, но при этом основной алгоритм оформлен в виде функции.

Листинг 7.18. Функция поиска минимального значения в массиве

```
program listing_7_18;
const
    N=10;
type
    Mass= array[1..N] of real;
function Books (X:Mass): real;
var
    J:integer;
    Min:real;
begin
    Min:= X[1];
    for J:=2 to N do
       if X[J] < Min then
              Min:=X[J];
    Books:=Min;
end;
var
    J:integer;
    A:Mass;
begin
for J:=1 to N do
  begin
    A[J]:=1000*random;
```

```
    writeln('J=',J, 'A[J]=',A[J]:7:2);
  end;
  writeln('Минимум=',Books(A):7:2);
end.
```

Функция подсчета соседних элементов массива

Необходимо разработать функцию подсчета максимального количества идущих подряд возрастающих элементов в одномерном массиве. В *главе 3* мы решали подобную задачу, где было необходимо найти максимальное количество идущих подряд одинаковых элементов. Здесь требуется разбить алгоритм на две части и разработать вместе с функцией еще и основную программу. Программа, решающая данную задачу, представлена в листинге 7.19.

Листинг 7.19. Функция анализа элементов массива

```
program listing_7_19;
const
    N=15;
type
    Mass= array[1..N] of integer;
function Poisk (A:Mass): integer;
var
    J,Nums,MaxNums:integer;
 begin
  Nums:=1;
  MaxNums:=1;
  for J:=2 to N do
    begin
      if A[J]>A[J-1] then
            Nums:= Nums+1
      else
         begin
          if Nums > MaxNums  then
               MaxNums:= Nums;
               Nums:=  1;
         end;
    end;
  if Nums > MaxNums  then
         MaxNums:= Nums;
  Poisk:=MaxNums;
 end;
var
    J:integer;
    A:Mass;
```

```
begin
for J:=1 to N do
  begin
    A[J]:=random(100);
    writeln('J=',J, 'A[J]=',A[J]);
  end;
    writeln('Число возрастающих элементов=',Poisk(A));
end.
```

Функция изменения значений элементов массива

Необходимо разработать функцию, осуществляющую изменение определенных значений в одномерном массиве. Так, если в массиве есть элементы, равные нулю, то их следует заменить на такое значение элемента массива, которое наиболее близко к нулю. В качестве своего значения функция должна выдать **true**, если такая замена произошла, и **false** — в противном случае. В листинге 7.20 приведена необходимая программная разработка.

Листинг 7.20. Изменение значений элементов в массиве

```
program listing_7_20;
const
    N=11;
type
    Mass= array[1..N] of integer;
function Zamena (var A:Mass): boolean;
var
    J,Delta:integer;
 begin
   Zamena:=false;
   Delta:=MaxInt;
   for J:=1 to N do
    if (Abs(A[J])< Abs(Delta)) and (A[J] <> 0)  then
             Delta:=A[J];
    for J:=1 to N do
      if A[J]=0  then
        begin
           A[J]:= Delta ;
           Zamena:=true;
        end;
 end;
var
   J:integer;
   A:Mass;
begin
for J:=1 to N do
```

```
  begin
    A[J]:=-2+random(5);
    writeln('J=',J, 'A[J]=',A[J]);
  end;
    writeln('Замена элементов -', Zamena(A));
 for J:=1 to N do
   writeln('J=',J, 'A[J]=',A[J]);
end.
```

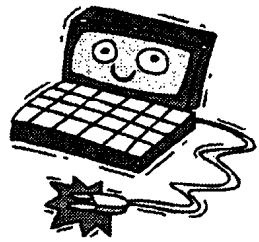
Функция вычисления суммы элементов двумерного массива

Рассмотрим теперь пример работы с двумерным массивом. Необходимо разработать функцию, которая будет суммировать элементы, значения которых являются четными числами. В листинге 7.21 приведена программа, реализующая решение данной задачи.

Листинг 7.21. Функция вычисления суммы значений элементов массива

```
program listing_7_21;
const
    N=11;
type
    Mass= array[1..N,1..N] of integer;
function Raschet (A:Mass): longint;
var
    I,J:integer;
    Sum:longint;
begin
  Sum:=0;
  for I:=1 to N do
   for J:=1 to N do
    if (A[I,J] mod 2) =0 then
         Sum:= Sum + A[I,J];
   Raschet:=Sum;
end;
var
    I,J:integer;
    A:Mass;
begin
 for I:=1 to N do
    for J:=1 to N do
      A[I,J]:=random(10);
    writeln('Сумма элементов=', Raschet(A));
end.
```

ГЛАВА 8



Математические вычисления

В этой главе внимание будет сконцентрировано на задачах, касающихся разнообразных математических вычислений. В билетах Единого государственного экзамена присутствуют такие задания, которые в определенной степени связаны с разделами математики. Конечно, глубоких знаний в математической области они не требуют, однако вопросы, связанные со стандартными функциями и решением несложных уравнений, необходимо знать.

Расчет значений функции

Требуется программно вычислить значения функции в указанном диапазоне изменения аргумента. А именно необходимо построить работающую программу, которая вычисляет значения функции $y = |x + 5|$. Диапазон изменения аргумента составляет от -1 до 5, а шаг приращения аргумента равен 0,5.

Для начала необходимо подсчитать число точек (значений аргумента), в которых следует вычислять значения функции:

$$N = Целая\ часть\left(\frac{X_{max} - X_{min}}{DX} + 1\right), \tag{8.1}$$

где:

- X_{max} — максимальное значение аргумента;
- X_{min} — минимальное значение аргумента;
- DX — шаг приращения аргумента.

В листинге 8.1 приведена необходимая программа для вычисления значений функции. Здесь в цикле от 1 до N последовательно вычисляются значения функции по формуле $y = |x + 5|$.

Листинг 8.1. Программа вычисления значений функции

```
program listing_8_1;
const
   Xmax=5; Xmin=-1; DX=0.5;
var
   X,Y:real;
   J,N:integer;
begin
  N:=trunc((Xmax-Xmin)/DX+1);
  X:=Xmin;
  for J:=1 to N do
     begin
       Y:= Abs(X+5);
       write('  X=', X);
       writeln('  Y=', Y);
       X:=X+DX;
    end;
end.
```

Численное интегрирование

Некоторые школьники знакомы с интегралами и методикой их вычисления. Однако на практике часто встречаются определенные интегралы (напомним, что в отличие от неопределенных у них указываются пределы интегрирования), которые либо нельзя вычислить аналитическими методами, либо вычисление требует очень больших усилий. В этом случае можно использовать численные методы, что обеспечивает вместо точного значения приближенное. Задача, которую мы здесь рассмотрим, заключается в вычислении интеграла:

$$Z = \int_A^B F(x)dx, \qquad (8.2)$$

где Z — искомое значение, которое фактически является площадью фигуры, ограниченной функцией $F(x)$, осью абсцисс и прямыми $x = A$ и $x = B$.

На рис. 8.1 эта задача проиллюстрирована графически.

Существуют методы приближенного вычисления определенного интеграла, которые основываются на замене сложной фигуры (рис. 8.1) на конечный набор элементарных участков. Тогда площадь исходной фигуры легко вычисляется как сумма площадей элементарных участков.

Наиболее простой вариант связан с численным интегрированием по методу прямоугольников. В этом случае интервал от A до B делится точками $x_1, x_2, ..., x_N$ на N равных частей (рис. 8.2).

При этом выполняются следующие условия:

- $x_1 = A$;
- $x_N = B$;
- $h = (B - A) / N$.

В этом случае очередную точку по оси абсцисс можно определить так:

$x_i = x_1 + (i - 1) \cdot h$.

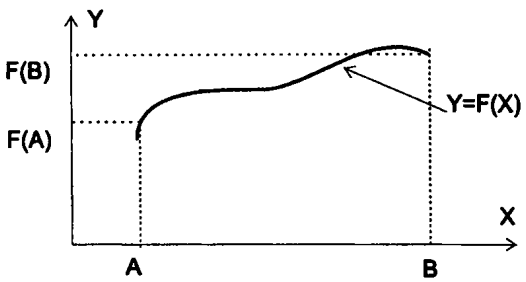


Рис. 8.1. График функции формулы 8.2

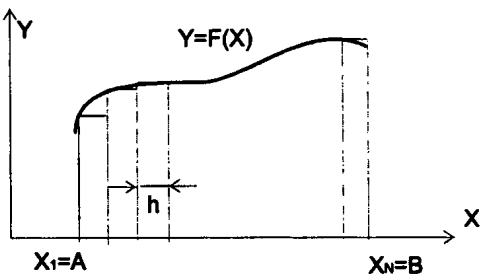


Рис. 8.2. График функции формулы 8.2 с разбивкой на прямоугольники

В соответствии с рис. 8.2 из каждой точки на оси абсцисс проводится перпендикуляр до пересечения с кривой $F(x)$. После этого кривая подынтегральной функции заменяется на ломаную линию, отрезки которой параллельны оси абсцисс. Площадь такой ступенчатой фигуры можно найти как сумму площадей прямоугольников, стороны которых равны h и y_i. Таким образом, площадь отдельного прямоугольника вычисляется:

$$S_i = y_i \cdot h .$$

В этом случае интеграл (8.2) можно приближенно заменить следующим рядом:

$$Z = h \sum_{i=1}^{N-1} F(x_i) . \qquad (8.3)$$

Запрограммировать полученное соотношение достаточно просто, и в качестве примера приведем вычисление интеграла для следующих исходных данных:

- $F(x) = x^3 + \cos(x) + 1$;
- $A = 1$;
- $B = 3$.

В листинге 8.2 приведена необходимая программа для вычисления значений функции. Здесь в цикле от 1 до N последовательно вычисляются значения функции и суммируются площади прямоугольников. В качестве N в приведенной программе мы выбрали значение 1000.

Листинг 8.2. Программа вычисления значений функции

```
program listing_8_2;
function integral(A,B:real;N:integer):real;
var S,H,Xi,Fi:real;
    I:integer;
begin
  H:=(B-A)/N;
  S:=0;
    for I:=1 to N-1 do
       begin
         Xi:=A+H*(I-1);
         Fi:=Xi*Xi*Xi+cos(Xi)+1;
         S:=S+Fi*H;
        end;
 integral:=S;
end;
begin
        write('  Интеграл =', integral (1,3,1000));
end.
```

Однако использование замены функции совокупностью прямоугольников дает невысокую точность. Для ее повышения требуется существенно увеличивать число точек N.

Гораздо большую точность обеспечивает метод трапеций. Если провести ординаты во всех точках деления линии и заменить каждую из полученных "криволинейных" трапеций прямолинейной (рис. 8.3), то приближенное значение интеграла будет равно сумме площадей прямолинейных трапеций. Площадь отдельной трапеции вычисляется:

$$Z_i = \frac{y_i + y_{i+1}}{2} h,$$

где i изменяется от 1 до $N-1$.

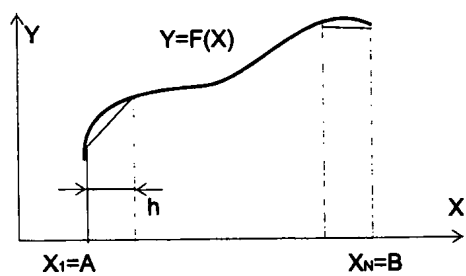


Рис. 8.3. График функции формулы 8.2 с разбивкой на трапеции

Тогда площадь искомой фигуры будет равна сумме всех трапеций:

$$Z=\int_A^B F(x)dx=\frac{h}{2}\sum_{i=1}^{N-1}(y_i+y_{i+1})=h\cdot\left(\frac{y_1+y_N}{2}+\sum_{i=2}^{N-1}y_i\right).$$

В результате можно записать формулу трапеций для численного интегрирования:

$$S=h\cdot\left(\frac{F(A)+F(B)}{2}+\sum_{i=2}^{N-1}F(x_i)\right). \quad (8.4)$$

Теперь применим соотношение (8.3) при вычислении интеграла для следующих исходных данных:

- $F(x)=x^2\cdot\cos(x)$;
- $A=-1$;
- $B=1$.

В листинге 8.3 приведена необходимая программа для вычисления значений данной функции. Здесь в цикле от 1 до N последовательно вычисляются значения функции. Для N в приведенной программе мы выбрали значение 1000.

Листинг 8.3. Пример вычисления интеграла по методу трапеций

```
program listing_8_3;
function integral(A,B:real;N:integer):real;
var S,H,Xi,Fi,FA,FB:real;
    I:integer;
begin
  H:=(B-A)/N;
  S:=0;
    for I:=2 to N-1 do
       begin
         Xi:=A+(I-1)*H;
         Fi:=Xi*Xi+cos(Xi);
         S:=S+Fi;
        end;
       FA:= A*A*cos(A);
       FB:= B*B*cos(B);
       integral:=H*((FA+FB)/2+S);
end;
begin
        write('  Интеграл =', integral (-1,1,1000));
end.
```

Решение уравнений

В билетах Единого государственного экзамена имеется много заданий, касающихся решений уравнений. В этом разделе мы рассмотрим ряд задач в том виде, как они были представлены в экзаменационных билетах.

ПРИМЕР 1

Требуется написать программу, которая решает уравнение AX+B=0 относительно X при любых числах A и B, введенных с клавиатуры. Все числа считаются действительными.

Программист сделал в программе (листинг 8.4) ошибки.

Вам предлагается последовательно выполнить следующие задания:

1. Указать, где имеется ошибка, связанная с вводом информации.
2. Найти другие ошибки в листинге 8.4 и обосновать их.
3. Доработать программу так, чтобы не было случаев ее неправильной работы.

Листинг 8.4. Программа решения уравнения AX+B=0 с наличием ошибок

```
program listing_8_4;
var
  A,B,X:real;
begin
  readln(A, B, X);
  if (B = 0 ) then
     write ('X=0')
  else
    if A = 0 then
       writeln ('решений нет')
    else
       writeln ('решение =',-B/A);
end.
```

Выполнение заданий

1. Учитывая то обстоятельство, что мы решаем уравнение относительно X, вводить значение X с клавиатуры не требуется. Поэтому вместо оператора

   ```
   readln(A, B, X)
   ```

   следует написать оператор ввода только двух параметров:

   ```
   readln(A, B).
   ```

2. Теперь перейдем непосредственно к решению уравнения. Рассмотрим следующие варианты:

   - если A ≠ 0, то в этом случае X = -B/A;
   - если A = 0 и B = 0, то в качестве решения подойдет любое значение X;

- если только A = 0, то решений уравнения нет. Это и нужно отразить в программном решении уравнения.

3. Из рассмотрения листинга 8.4 вытекает, что при равных нулю коэффициентах A и B в качестве решения уравнения получается ноль, однако на самом деле в этом случае решением является любое значение X. С учетом сказанного возможная коррекция программы представлена в листинге 8.5.

Листинг 8.5. Исправленный вариант листинга 8.4

```
program listing_8_5;
var
   A, B, X:real;
begin
 readln(A, B);
 if (B = 0) and (A = 0) then
    write ('Любое значение X')
 else
   if A = 0 then
       writeln ('Решений нет')
   else
       writeln ('Решение ',-B/A);
end.
```

ПРИМЕР 2

Рассмотрим еще один пример, связанный с решением уравнения. Требуется написать программу, которая решает уравнение |X|-2=-A относительно X для любого числа A, вводимого с клавиатуры. Все числа считаются действительными.

Программист сделал в программе (листинг 8.6) ошибки.

Вам предлагается последовательно выполнить следующие задания:

1. Указать, где имеется ошибка, связанная с вводом информации.
2. Найти другие ошибки в листинге 8.6 и обосновать их.
3. Доработать программу так, чтобы не было случаев ее неправильной работы.

Листинг 8.6. Программа решения уравнения |X|-2=-A с наличием ошибок

```
program listing_8_6;
var
   A, X:real;
begin
  readln(A,X);
  if ( 2-A ) < 0  then
     write ('X=',2-A)
```

```
  else
    writeln ('Решений нет')
end.
```

Выполнение заданий

1. Здесь, как и в примере 1, мы решаем уравнение относительно X, значит, вводить значение X с клавиатуры не требуется. Поэтому вместо оператора

   `readln(A, X)`

   нужно написать оператор:

   `readln(A).`

2. А теперь перед коррекцией программы рассмотрим решение уравнения. Модуль числа (в данном случае |X|) не может быть отрицательным, поэтому 2-A должно быть больше или равно 0. Таким образом, после ввода с клавиатуры числа A необходимо проверить выражение 2-A, и если оно отрицательное, то следует вывести сообщение о том, что решений уравнения нет. В противном случае имеются два решения:

   - 2-A;
   - A-2.

3. С учетом сказанного возможная коррекция программы представлена в листинге 8.7.

Листинг 8.7. Исправленный вариант листинга 8.6

```
program listing_8_7;
var
  A,X:real;
begin
  readln(A);
  if (2-A)< 0 then
      writeln('Решений нет')
  else
    begin
        writeln (' Два решения:');
        write ('X1=', 2-A);
        write ('X2=', A-2);
    end;
end.
```

ПРИМЕР 3

Рассмотрим еще один пример на данную тему. Требуется решить уравнение: A|X| = B относительно X для любых чисел A и B, введенных с клавиатуры. Все рассматриваемые числа считаются действительными.

Программист поторопился и написал программу неправильно (листинг 8.8).

Вам предлагается последовательно выполнить следующие задания:

1. Указать, где имеется ошибка, связанная с вводом информации.
2. Найти другие ошибки в листинге 8.8 и обосновать их.
3. Доработать программу так, чтобы не было случаев ее неправильной работы.

Листинг 8.8. Программа решения уравнения A\|X\| = B с наличием ошибок

```
program listing_8_8;
var
  A, B, X:real;
begin
  readln(A,B,X);
  if ( A = 0 ) and ( B = 0 ) then
       write ('Любое число')
  else
       writeln ('X1=',B/A, ' и X2=',-B/A);
end.
```

Выполнение заданий

1. Учитывая, что мы решаем уравнение относительно X, вводить его значение с клавиатуры не требуется. Поэтому вместо оператора

   ```
   readln(A,B,X)
   ```

   следует указать:

   ```
   readln(A,B).
   ```

2. Перед коррекцией программы рассмотрим сначала решение уравнения:

   - если A = 0, то при B = 0 решением является любое X;
   - если A = 0 и B ≠ 0, то уравнение решений не имеет;
   - рассмотрим теперь ситуацию, когда A ≠ 0. Мы должны учесть, что |X| не может быть отрицательным. В связи с этим B/A должно принимать значения не меньше нуля. Таким образом, после ввода с клавиатуры параметров A и B необходимо проанализировать выражение B/A, и если оно отрицательное, то уравнение решений не имеет. Если B = 0, то X = 0. В противном случае решений два:
     - B/A;
     - -B/A.

3. С учетом сказанного возможная коррекция программы представлена в листинге 8.9.

Листинг 8.9. Исправленный вариант листинга 8.8

```
program listing_8_9;
var
  A, B, X:real;
begin
  readln(A,B);
  if ( A = 0 ) and ( B = 0 ) then
       write ('Любое число')
  else
   if A = 0 then
       writeln ('Решений нет')
   else
       if (B/A)< 0 then
           writeln ('Решений нет')
       else
           begin
             if B = 0 then
                writeln ('Решение X=0')
             else
               begin
                   writeln ('2 Решения');
                   writeln ('X1=',B/A,' и X2=',-B/A);
               end;
           end;
end.
```

Квадратное уравнение

Это задание также встречалось в ЕГЭ в прошлые годы. Требовалось написать программу, которая решает уравнение $AX^2 + BX + C = 0$ относительно X для действительных чисел A, B, C, введенных с клавиатуры, о которых заведомо известно, что $A \neq 0$, $B \neq 0$, $C \neq 0$. Была написана программа, представленная в листинге 8.10, в которой программист сделал ошибки.

Вам предлагается последовательно выполнить следующие задания:

1. Указать, где имеется ошибка, связанная с вводом информации.
2. Найти другие ошибки в листинге 8.10 и обосновать их.
3. Доработать программу так, чтобы не было случаев ее неправильной работы.

Листинг 8.10. Программа решения квадратного уравнения с наличием ошибок

```
program listing_8_10;
var
  A, B, C, D, X1, X2:real;
```

```
begin
  readln(A,B,C,X1,X2);
  D:=B*B - 4*A*C;
  if D > 0 then
    begin
     X1:=(-B+sqrt(D))/(2*A);
     X2:=(-B-sqrt(D))/(2*A);
     write ('X1=',X1);
     write ('X2=',X2);
    end
  else
    writeln ('Действительных корней нет');
end.
```

Выполнение заданий

1. Учитывая, что мы решаем уравнение относительно X, вводить с клавиатуры его значение не требуется. Поэтому вместо строки

   ```
   readln(A,B,C,X1,X2)
   ```

   следует указать:

   ```
   readln(A,B,C).
   ```

2. В программе не рассматривается случай, когда дискриминант (D) равен нулю. В этом случае решение уравнения должно выглядеть так:

   ```
   X = -B/(2A).
   ```

   В приведенной программе этот случай не рассматривается, и при `D = 0` программа выдает, что действительных корней нет. Поэтому, задав `A:=1`, `B:=2`, `C:=1`, мы получим, согласно программе листинга 8.10, неправильное решение.

3. С учетом сказанного исправленный вариант программы представлен в листинге 8.11.

Листинг 8.11. Исправленный вариант программы листинга 8.10

```
program listing_8_11;
var
   A, B, C, D, X1, X2:real;
begin
  readln(A,B,C);
  D:=B*B - 4*A*C;
  if D > 0 then
    begin
     X1:=(-B+sqrt(D))/(2*A);
     X2:=(-B-sqrt(D))/(2*A);
     write ('X1=',X1);
     write ('X2=',X2);
   end
```

```
   else
      if D=0 then
         begin
            X1:=(-B)/(2*A);
            write ('X1,2=',X1);
         end
      else
         writeln ('Действительных корней нет');
end.
```

Решение неравенства

Еще одно задание из Единого государственного экзамена прошлых лет. Требуется написать программу, которая решает неравенство AX + B > 0 относительно X для любых чисел A и B, вводимых с клавиатуры. Все числа считаются действительными. Программист поторопился и написал программу некорректно (листинг 8.12).

Вам предлагается последовательно выполнить следующие задания:

1. Указать, где имеется ошибка, связанная с вводом информации.
2. Найти другие ошибки в листинге 8.12 и обосновать их.
3. Доработать программу так, чтобы не было случаев ее неправильной работы.

Листинг 8.12. Программа решения неравенства с наличием ошибок

```
program listing_8_12;
var
  A, B, X:real;
begin
  readln(A,B,X);
  if ( A = 0 ) then
        write ('Любое число')
  else
      if ( A  > 0 ) then
          writeln ('X>',-B/A)
      else
          writeln ('X<',-B/A);
end.
```

Выполнение заданий

1. Как и в предыдущих примерах, ввод значения переменной X является излишним. Поэтому вместо оператора

   ```
   readln(A,B,X)
   ```

   следует написать:

   ```
   readln(A,B).
   ```

2. При A:=0 программа выдает сообщение о том, что решением является любое число. Однако на самом деле решений неравенства нет.

   Сначала необходимо решить неравенство AX + B > 0, и в качестве первого шага выполним очевидное преобразование: AX > -B.

   А теперь рассмотрим следующие ситуации:

   - если A = 0, то решением является любое X, при условии, что B > 0. Это связано с тем, что в этом случае неравенство принимает вид: 0 > -B;
   - если A = 0 и B ≤ 0, то решений нет. Это также связано с тем, что неравенство принимает вид:0 > -B;
   - если A > 0, то в этом случае решением неравенства является: X > -B/A;
   - если A < 0, то решением является: X < -B/A.

3. В листинге 8.13 приведен вариант программной разработки, который соответствует нашим выкладкам.

Листинг 8.13. Исправленный вариант программы листинга 8.12

```
program listing_8_13;
var
  A, B, X:real;
begin
  readln(A,B);
 if  A = 0  then
      if B >0 then
          writeln ('Любое число является решением неравенства')
      else
          writeln ('Решений нет')
 else
    if  A > 0 then
          writeln ('X>',-B/A)
    else
          writeln ('X<',-B/A);
end.
```

Определение принадлежности множеству

На рис. 8.4 приведен набор фигур:

- круг радиуса 10;
- прямая, параллельная оси ординат, проходящая через точку X = 5;
- прямая, проведенная под углом 45 градусов к оси абсцисс.

Необходимо написать программу, позволяющую проверить попадание точки в заштрихованную область.

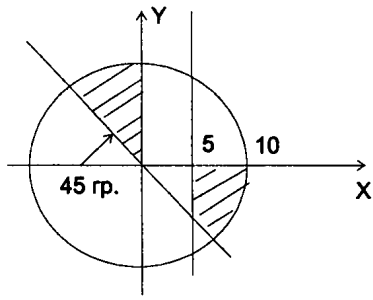


Рис. 8.4. Иллюстрация графической области к листингу 8.14

Здесь нужно просто аккуратно записать условие попадания точки в одну из двух областей.

В листинге 8.14 приведен вариант программной разработки.

Листинг 8.14. Проверка попадания точки в область

```
program listing_8_14;
var
   X, Y, A:real;
begin
  writeln ('Введите координаты точки');
  readln(X,Y);
  A:=sqrt(sqr(X)+sqr(Y));
  if ((A<=10)and(Y>=(-X))and(X<=0)) or ((X>=5)and(Y=<0)and(A<=10) ) then
        writeln ('Да, попадает')
  else
        writeln ('Нет');
end.
```

На рис. 8.5 приведен другой набор фигур:

- круг радиуса 10;
- круг радиуса 5;
- прямая, проведенная под углом 45 градусов к оси абсцисс.

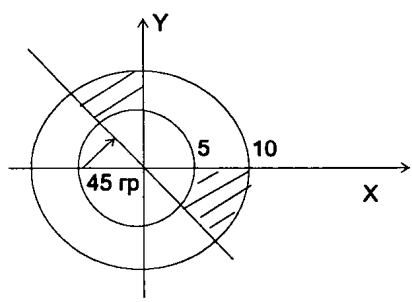


Рис. 8.5. Иллюстрация графической области к листингу 8.15

Необходимо написать программу, позволяющую проверить попадание точки в заштрихованную область, состоящую из двух фрагментов.

Здесь также следует аккуратно записать условие попадания точки в одну из двух областей. В листинге 8.15 приведен вариант программной разработки.

Листинг 8.15. Проверка попадания точки в область

```
program listing_8_15;
var
  X, Y, A:real;
begin
 writeln ('Введите координаты точки');
 readln(X,Y);
 A:=sqrt(sqr(X)+sqr(Y));
 if (A<=10)and(A>=5)and(((Y>=(-X))and(X<=0))or((X>=(-Y))and(Y<=0)))then
        writeln ('Да, попадает')
     else
        writeln ('Нет');
end.
```

И еще одна ситуация (рис. 8.6) с проверкой попадания точки в заштрихованную область. В листинге 8.16 приведен вариант программной разработки.

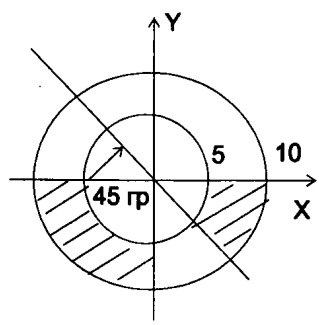


Рис. 8.6. Иллюстрация графической области к листингу 8.16

Листинг 8.16. Проверка попадания точки в область

```
program listing_8_16;
var
  X, Y, A:real;
begin
 writeln ('Введите координаты точки');
 readln(X,Y);
 A:=sqrt(sqr(X)+sqr(Y));
 if (A<=10)and(A>=5)and(((Y<=0)and(X<=0))or((X>=(-Y))and(Y<=0)))then
       writeln ('Да, попадает')
```

```
 else
       writeln ('Нет');
end.
```

Последняя ситуация на рассматриваемую тему приведена на рис. 8.7. Один круг радиуса 10 с центром в начале координат, а второй — радиуса 5 с центром в точке (10,0). В листинге 8.17 отражен вариант программной разработки.

Листинг 8.17. Проверка попадания точки в область

```
program listing_8_17;
var
   X, Y, A,B:real;
begin
 writeln ('Введите координаты точки');
 readln(X,Y);
 A:=sqrt(sqr(X)+sqr(Y));
 B:= sqrt(sqr(X-10)+sqr(Y));
 if (A<=10)and(B<=5)then
       writeln ('Да, попадает')
 else
       writeln ('Нет');
end.
```

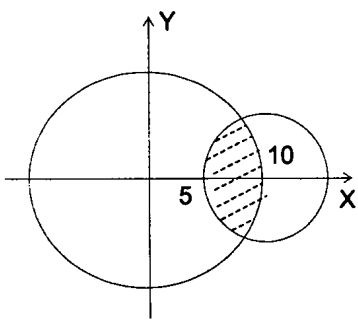


Рис. 8.7. Иллюстрация графической области к листингу 8.17

Метод Монте-Карло

В предыдущих главах мы уже познакомились со стандартной функцией **random**, которая позволяет программно формировать случайные числа. Фактически эта и аналогичные функции в других языках программирования являются *датчиками случайных чисел*.

В математике известен метод Монте-Карло, который позволяет программным способом с помощью датчика случайных чисел решать сложные вычислительные задачи. Сам метод достаточно прост и далее мы его поясним.

Допустим, нам необходимо вычислить площадь сложной фигуры (рис. 8.8). Для этого будем случайным образом формировать точки внутри прямоугольника, окружающего фигуру. На рисунке данный прямоугольник ограничен точками B1, B2, B3, B4. Площадь данного прямоугольника равна произведению длин сторон (на рисунке это C и D). После формирования случайным образом N точек в области рассматриваемого прямоугольника какая-то их часть (обозначим это количество символом M) попадает в фигуру S. В соответствии с методом Монте-Карло формула площади сложной фигуры такова:

$$S = \frac{M}{N} \cdot (C \cdot D). \qquad (8.5)$$

Фактически все содержание метода Монте-Карло заключается в этом соотношении. При этом сам метод не ограничивается только геометрическими задачами вычисления площади. И далее рассмотрим ряд практических примеров, касающихся различных тем, в которых данный метод позволяет быстро найти решение.

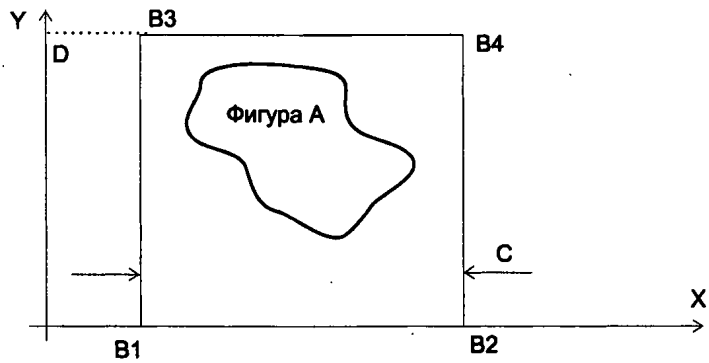


Рис. 8.8. Сложная фигура для вычисления площади

Вычисление площади фигуры

Начнем с уже сформулированной задачи вычисления площади фигуры. Так, необходимо вычислить площадь фрагмента, заключенного внутри линий, которые определяются следующими зависимостями:

$$y = \cos(x)$$

$$y = x^2$$

Для решения организуем формирование точек в области, ограниченной следующими соотношениями:

$$-2 \le x \le 2$$

$$0 \le y \le 1$$

Примечание

В данном случае мы выбрали фигуру, площадь которой легко вычисляется.

Если графически представить исходные линии, то интересующая нас область как раз расположена внутри данного прямоугольника. В листинге 8.18 приведена программная разработка, решающая данную задачу. В программе мы случайным образом формируем 30000 точек, месторасположение которых затем анализируется. Площадь самого прямоугольника, внутри которого формируются точки, равна 4 см^2. В решении задачи мы используем соотношение (8.5).

Листинг 8.18. Использование метода Монте-Карло для вычисления площади фигуры

```
program listing_8_18;
 var
   I,M:Integer;
   S,Y,Y1,Y2,X:real;
 begin
  M:=0;
  for I:=1 to 30000 do
   begin
     X:=-2+4*random;
     Y:=random;
     Y1:=cos(X);
     Y2:=sqr(X);
     if (Y>=Y2) and (Y<=Y1) then
       M:=M+1;
    end;
   S:=(M/30000)*4;
   writeln('Площадь равна = ',S);
end.
```

Рассмотрим еще одну задачу на данную тему. Пусть необходимо вычислить площадь фигуры, ограниченной параболой $y = x^2$, осью абсцисс и прямой $x = 2$.

Организуем формирование точек в области, ограниченной следующими соотношениями:

$$0 \le x \le 2$$

$$0 \le y \le 4$$

Если графически представить исходные линии, то интересующая нас область как раз расположена внутри данного прямоугольника. В листинге 8.19 приведена программная разработка, решающая данную задачу. В программе мы случайным образом формируем 30000 точек, месторасположение которых затем анализируется. Площадь самого прямоугольника, внутри которого формируются точки, равна 8 см^2.

Листинг 8.19. Использование метода Монте-Карло для вычисления площади фигуры

```
program listing_8_19;
 var
   I,M:Integer;
   S,Y,Y1,X:real;
 begin
  M:=0;
  for I:=1 to 30000 do
   begin
     X:=2*random;
     Y:=4*random;
     Y1:=sqr(X);
     if Y<=Y1 then
       M:=M+1;
    end;
   S:=(M/30000)*8;
   writeln('Площадь равна = ',S);
 end.
```

Моделирование бросания игрального кубика

Необходимо смоделировать бросание игрального кубика. По итогам 10000 бросаний требуется отобразить на экране относительную частоту выпадания каждой грани (в какой доле случаев из 10000 выпала та или иная грань).

Для подсчета относительных частот выпадания каждой грани мы используем массив `A` из 6 элементов. Так, элемент `A[1]` будет содержать значение доли случаев выпадания единицы, `A[2]` — двойки и т. д.

Для моделирования подбрасывания кубика мы воспользовались датчиком случайных чисел в следующей редакции:

```
X:=1+random(6).
```

В результате такого использования в переменной `X` будут случайно и равновероятно формироваться целые числа от 1 до 6, что соответствует граням кубика.

В листинге 8.20 приведена программная разработка, решающая данную задачу.

Листинг 8.20. Статистика подбрасываний кубика

```
program listing_8_20;
 var
   I,X:Integer;
   A:array[1..6] of real;
 begin
  for I:=1 to 6 do
     A[I]:=0;
```

```
  for I:=1 to 10000 do
   begin
     X:=1+random(6);
     A[X]:=A[X]+1;
   end;
 for I:=1 to 6 do
  begin
     A[I]:= A[I]/10000;
     writeln(I,' ',A[I]);
  end;
end.
```

Статистика подбрасывания монет

Необходимо смоделировать эксперимент по подбрасыванию двух монет: подсчитать статистику комбинаций:

- орел, орел;
- орел, решка;
- решка, решка.

Разработка учитывает уже рассмотренные в предыдущем примере технические приемы, поэтому какого-либо комментария не требуется. В листинге 8.21 приведена программная разработка, решающая данную задачу.

Листинг 8.21. Статистика подбрасываний двух монет

```
program listing_8_21;
 var
   I,X,Y:Integer;
   A:array[1..3] of real;
 begin
  for I:=1 to 3 do
     A[I]:=0;
  for I:=1 to 10000 do
   begin
     X:=random(2);
     Y:=random(2);
     if (X = 0) and (Y = 0) then
       A[1]:=A[1]+1
    else
       if (X = 1) and (Y = 1) then
          A[2]:=A[2]+1
       else
         A[3]:=A[3]+1
   end;
```

```
   writeln(' Доля комбинаций (орел, орел)  равна ',A[1]/10000);
   writeln(' Доля комбинаций (решка, решка)  равна ',A[2]/10000);
   writeln(' Доля комбинаций (орел, решка)  равна ',A[3]/10000);
end.
```

Задания из ЕГЭ за 2008 — 2010 годы

В этом разделе приведены типовые задачи из билетов Единого государственного экзамена. В задачах присутствуют готовые листинги программ, которые вам необходимо проанализировать. Также требуется предложить свой правильный вариант программной разработки.

ЗАДАЧА 8.1

Напишите программу, которая проверяет, удовлетворяет ли пара чисел $(x; y)$ двойному неравенству $2 < x^2 + y^2 < 10$.

Программист поторопился и написал программу неправильно (листинг 8.22).

Вам предлагается последовательно выполнить следующие задания:

1. Найти ошибки в листинге 8.22 и обосновать их.
2. Доработать программу так, чтобы не было случаев ее неправильной работы.

Листинг 8.22. Программа с ошибками к задаче 8.1

```
program listing_8_22;
var
  X, Y, S:real;
begin
  readln(X,Y);
  S:=Sqr(X) + Sqr(Y);
  if (S > 2) or (S < 10) then
    writeln ('Удовлетворяет');
  else
    writeln ('Удовлетворяет');
end.
```

Выполнение заданий

1. В данном случае должны выполняться одновременно два условия, поэтому в операторе условия необходимо использовать ключевое слово **and**.

   Вместо строки

   ```
   if (S > 2) or (S < 10) then
   ```

   нужно написать:

   ```
   if (S>2) and (S<10) then
   ```

Программа согласно листингу 8.22 ошибочно (при `X:=5` и `Y:=5`) выдаст сообщение, что пара точек удовлетворяет неравенству.

В программе используются два одинаковых сообщения о выводе информации.

2. С учетом сказанного исправленный вариант программы представлен в листинге 8.23.

Листинг 8.23. Исправленный вариант программы листинга 8.22

```
program listing_8_23;
var
  X, Y, S:real;
begin
  readln(X,Y);
  S:=Sqr(X) + Sqr(Y);
  if (S > 2) and (S< 10) then
    writeln ('Удовлетворяет')
  else
    writeln ('Не удовлетворяет');
end.
```

ЗАДАЧА 8.2

Напишите программу, которая проверяет, удовлетворяет ли пара чисел $(x; y)$ двойному неравенству $-4 < x^2 - y^2 < 7$.

Программист поторопился и написал программу неправильно (листинг 8.24).

Вам предлагается последовательно выполнить следующие задания:

1. Найти ошибки в листинге 8.24 и обосновать их.
2. Доработать программу так, чтобы не было случаев ее неправильной работы.

Листинг 8.24. Программа с ошибками к задаче 8.2

```
program listing_8_24;
var
  X, Y, S:real;
begin
  readln(X,Y);
  S:=Sqr(Y) - Sqr(X);
  if (S > -4) and ( S < 7) then
    writeln ('Удовлетворяет')
  else
    writeln ('Удовлетворяет');
 end.
```

Выполнение заданий

1. В листинге 8.24 неправильно написано выражение, составляющее компоненту неравенства:

   ```
   S:= Sqr(Y) – Sqr(X);
   ```

   Нужно написать:

   ```
   S:= Sqr(X) – Sqr(Y);
   ```

   Программа согласно листингу 8.24 ошибочно (при `X:=0` и `Y:=10`) выдаст сообщение, что пара точек удовлетворяет неравенству.

   В программе используются два одинаковых сообщения о выводе информации.

2. С учетом сказанного исправленный вариант программы представлен в листинге 8.25.

Листинг 8.25. Исправленный вариант программы листинга 8.24

```
program listing_8_25;
var
  X, Y, S:real;
begin
  readln(X,Y);
  S:= Sqr(X) -Sqr(Y) ;
  if (S > -4) and (S< 7) then
    writeln ('Удовлетворяет')
  else
    writeln ('Не удовлетворяет');
end.
```

ЗАДАЧА 8.3

Напишите программу, которая проверяет, удовлетворяет ли пара чисел $(x; y)$ двойному неравенству $5 \le x^2 + y^2 \le 9$.

Программист поторопился и написал программу неправильно (листинг 8.26).

Вам предлагается последовательно выполнить следующие задания:

1. Найти ошибки в листинге 8.26 и обосновать их.
2. Доработать программу так, чтобы не было случаев ее неправильной работы.

Листинг 8.26. Программа с ошибками к задаче 8.3

```
program listing_8_26;
var
  X, Y, S:real;
begin
  readln(X,Y);
  S:=Sqr(Y) + Sqr(X);
```

```
  if (S >= 5) or ( S <= 9) then
    writeln ('Удовлетворяет')
  else
    writeln ('Удовлетворяет');
end.
```

Выполнение заданий

1. В листинге 8.26 неправильно написано условие:

   ```
   if (S >= 5) or ( S <= 9) then
   ```

   Нужно написать:

   ```
   if (S >= 5) and ( S <= 9) then
   ```

   Программа согласно листингу 8.26 ошибочно при `X:=1` и `Y:=10` выдаст сообщение, что пара точек удовлетворяет неравенству.

   В программе используются два одинаковых сообщения о выводе информации.

2. С учетом сказанного исправленный вариант программы представлен в листинге 8.27.

Листинг 8.27. Исправленный вариант программы листинга 8.26

```
program listing_8_27;
var
  X, Y, S:real;
begin
  readln(X,Y);
    S:=Sqr(Y) + Sqr(X);
  if (S >= 5) and ( S <= 9) then
    writeln ('Удовлетворяет')
  else
    writeln ('Не удовлетворяет');
end.
```

ГЛАВА 9

Обработка данных

Эта глава непосредственно связана с наиболее трудоемкой компонентой Единого государственного экзамена. В целом программирование и алгоритмизация задач являются основным содержанием части *C* билетов Единого государственного экзамена, где содержатся четыре задания, которые ориентированы на проверку умения составлять и анализировать алгоритмы. При этом данная часть традиционно вызывает наибольшие трудности у учащихся. Это вполне объяснимо, т. к. здесь требуется как самостоятельное написание достаточно сложных программ, так и анализ и коррекция уже готовых, часто непростых разработок. И данная глава преследует цель подготовить учащихся именно к части *C* билетов Единого государственного экзамена. Глава состоит из подробного рассмотрения как типовых заданий по информатике в целом, так и непосредственно заданий Единого государственного экзамена прошлых лет.

Анализ тестирования учащихся

Рассмотрим следующую задачу. Пусть в некотором вузе студенты проходят предварительное тестирование, по результатам которого они могут быть допущены к ранней сдаче экзаменов. Тестирование проводится по трем темам. По каждой теме учащийся может набрать от 0 до 100 баллов. При этом к ранней сдаче экзамена допускаются учащиеся, набравшие по результатам тестирования не менее 30 баллов по каждой из трех тем. Известно, что общее количество участников тестирования не превосходит 300.

Необходимо обеспечить ввод начальных данных в программу. При этом в первой строке вводится количество студентов, принимавших участие в тестировании (N).

Далее следуют N вводимых строк, имеющих следующий формат:

- фамилия — это информация о фамилии студента, являющаяся строкой (в плане типа данных), не превышающей 20 символов;
- имя — строка (также речь идет о типе данных) не более 15 символов;
- баллы — три целых числа, разделенных пробелами (эти числа соответствуют баллам, набранным по результатам тестирования по каждой из трех тем).

При этом фамилия и имя, а также имя и баллы разделены одним пробелом.

Пример возможной введенной строки с информацией по студенту:

```
Петренко Наталья 58 75 35.
```

Разработанная программа должна выводить на экран фамилии и имена студентов, допущенных к сдаче экзамена (фамилии и имена можно выводить в произвольном порядке).

Перед написанием программы составим блок-схему (рис. 9.1). Здесь после ввода N (числа студентов) организуется цикл, в котором сначала считываются символы,

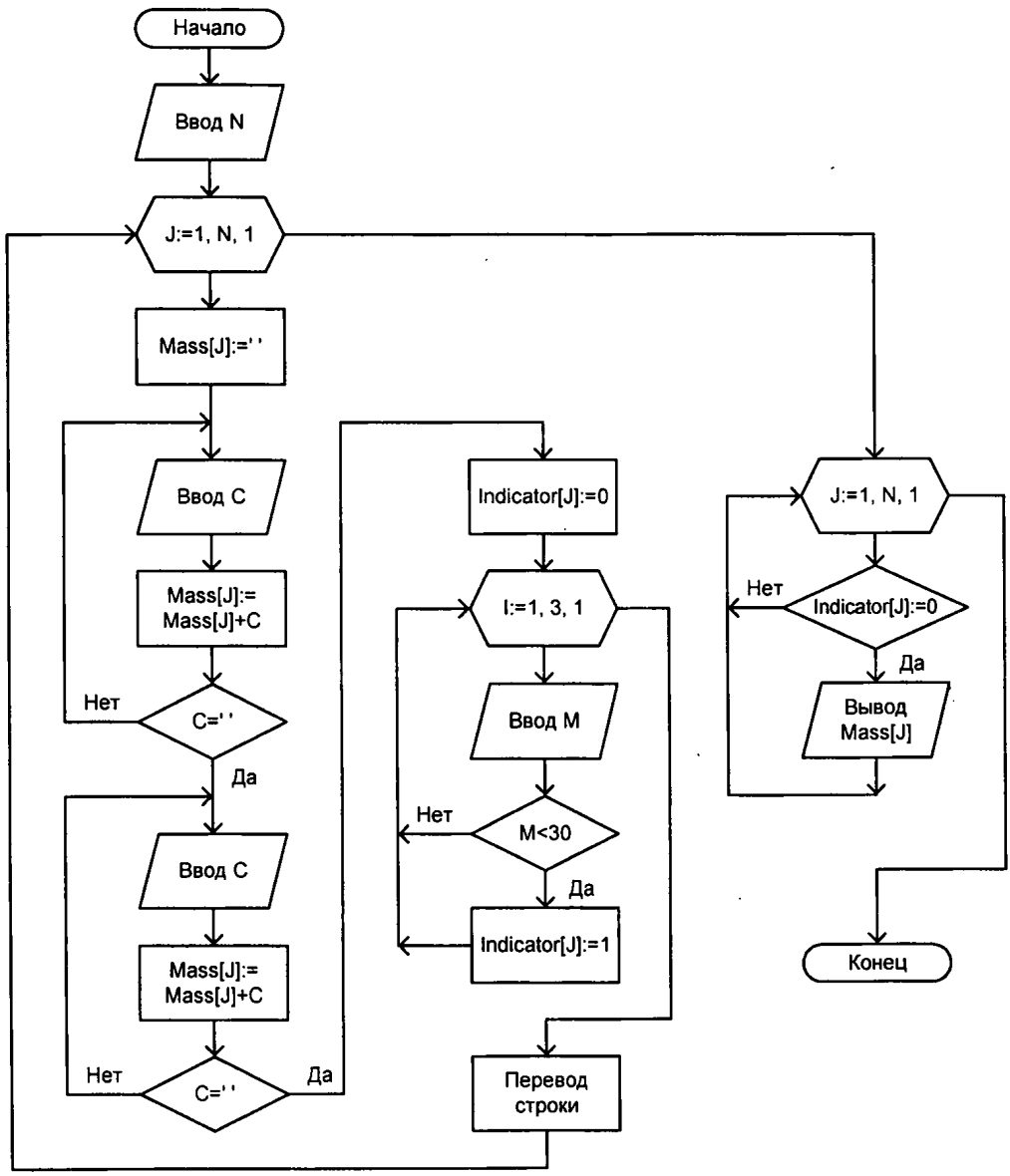


Рис. 9.1. Блок-схема к программе листинга 9.1

составляющие фамилию, а затем символы, составляющие имя. Эти сведения заносятся в элемент массива Massiv[J], который предназначен для хранения информации о фамилии и имени J-го студента.

Еще один массив (имя которого — Indicator) предназначен для фиксирования прохождения каждым учащимся трех тем. Условно будем считать, что если все три темы успешно пройдены J-го студентом, то в элементе Indicator[J] это отмечается значением 0, в противном случае записывается 1. И во вложенном цикле (от 1 до 3) последовательно проверяются три целых числа, отражающие количество набранных баллов по темам. Если хотя бы по одной теме число баллов окажется меньше 30, то в индикаторе (Indicator[J]) это отмечается единицей. После завершения цикла по вводу всей информации организует новый цикл по перебору всех участников. Здесь, если значение Indicator[J] равно нулю, на экран выводится элемент массива Mass[J]. Таким образом, мы достигаем поставленной цели. В листинге 9.1 приведена программа, функционирующая в соответствии с построенной блок-схемой.

Листинг 9.1. Анализ тестирования учащихся

```
program listing_9_1;
var
  Mass: array[1..300] of string[36];
  Indicator: array[1..300] of integer;
  N,J,I,M:integer;
  C:char;
begin
  readln(N);
  for J:=1 to N do
   begin
     Mass[J]:='';
     repeat
        read(C);
        Mass[J]:= Mass[J]+C;
     until C=' '; { Считана фамилия студента }
     repeat
        read(C);
        Mass[J]:= Mass[J]+C;
     until C=' '; { Считано имя студента }
     Indicator[J]:=0;
     for I:=1 to 3 do
      begin
         read(M);
         if ( M < 30) then
              Indicator[J]:=1;
      end;
    readln;
  end;
```

```
  for J:=1 to N do
   if Indicator[J]=0 then
      writeln (Mass[J]);
end.
```

Можно предложить небольшое улучшение программы. Так, если по какой-либо теме балл оказался меньше 30, то количество баллов, набранных по следующей теме, можно уже не проверять. В этом случае фрагмент листинга 9.1, выполняющий анализ тем, следует заменить на такой набор операторов:

```
for I:=1 to 3 do
 begin
    read(M);
    if ( M < 30) then
         begin
           Indicator[J]:=1;
           break;
         end;
 end;
```

В *главе 5* мы рассматривали возможности записей. В приведенной разработке вместо двух массивов можно использовать один, но из элементов комбинированного формата. В листинге 9.2 приведен вариант программы отбора необходимых учащихся с использованием создания нового типа данных.

Листинг 9.2. Анализ тестирования учащихся (вариант 2)

```
program listing_9_2;
Type
  Stud=record
  FamImya:string[36];
  Indicator:boolean;
end;
var
  Mass: array[1..300] of Stud;
  N,J,I,M:integer;
  C:char;
begin
  readln(N);
  for J:=1 to N do
   begin
     Mass[J].FamImya:='';
      repeat
         read(C);
         Mass[J].FamImya:= Mass[J].FamImya+C;
      until C=' '; { Считана фамилия студента }
      repeat
         read(C);
         Mass[J].FamImya:=Mass[J].FamImya+C;
```

```
      until C=' '; { Считано имя студента }
      Mass[J].Indicator:=true;
      for I:=1 to 3 do
       begin
         read(M);
          if ( M < 30) then
               begin
                  Mass[J].Indicator:=false;
                  break;
               end;
       end;
    readln;
  end;
  for J:=1 to N do
   if Mass[J].Indicator=true then
      writeln (Mass[J].FamImya);
end.
```

Отчет по олимпиаде

В этой задаче на вход программы подаются сведения о номерах школ учащихся, которые участвовали в олимпиаде. При этом в первой строке вводится количество учащихся (N). Далее следуют N строк, имеющих следующий формат:

- фамилия — фамилия учащегося (тип данных — строка не более 20 символов);
- инициалы — строка, состоящая из 4-х символов (буква, точка, буква, точка);
- номер школы — двузначное число.

При этом фамилия и инициалы, а также инициалы и номер школы разделены одним пробелом.

Один из вариантов ввода строки с клавиатуры выглядит так:

```
Петренко Н.С. 55.
```

В программе необходимо обеспечить вывод на экран информации о том, в какой школе было меньше всего участников (таких школ может быть несколько). При этом необходимо отразить информацию только по школам, пославшим хотя бы одного участника. Следует учитывать, что N<1000.

Блок-схема алгоритма приведена на рис. 9.2. Для работы нам понадобится массив из 99 элементов: элемент с индексом 1 будет хранить сведения о количестве участников из школы с номером 1, элемент с индексом 2 будет хранить сведения о количестве участников из школы с номером 2 и т. д.

Во все элементы массива мы предварительно заносим нули (перед началом подсчета от каждой школы 0 участников). Как и в предыдущем примере, после ввода числа учащихся N производится последовательный ввод N строк указанного ранее формата. Так как нас интересует только из какой школы участник (фамилии нас не

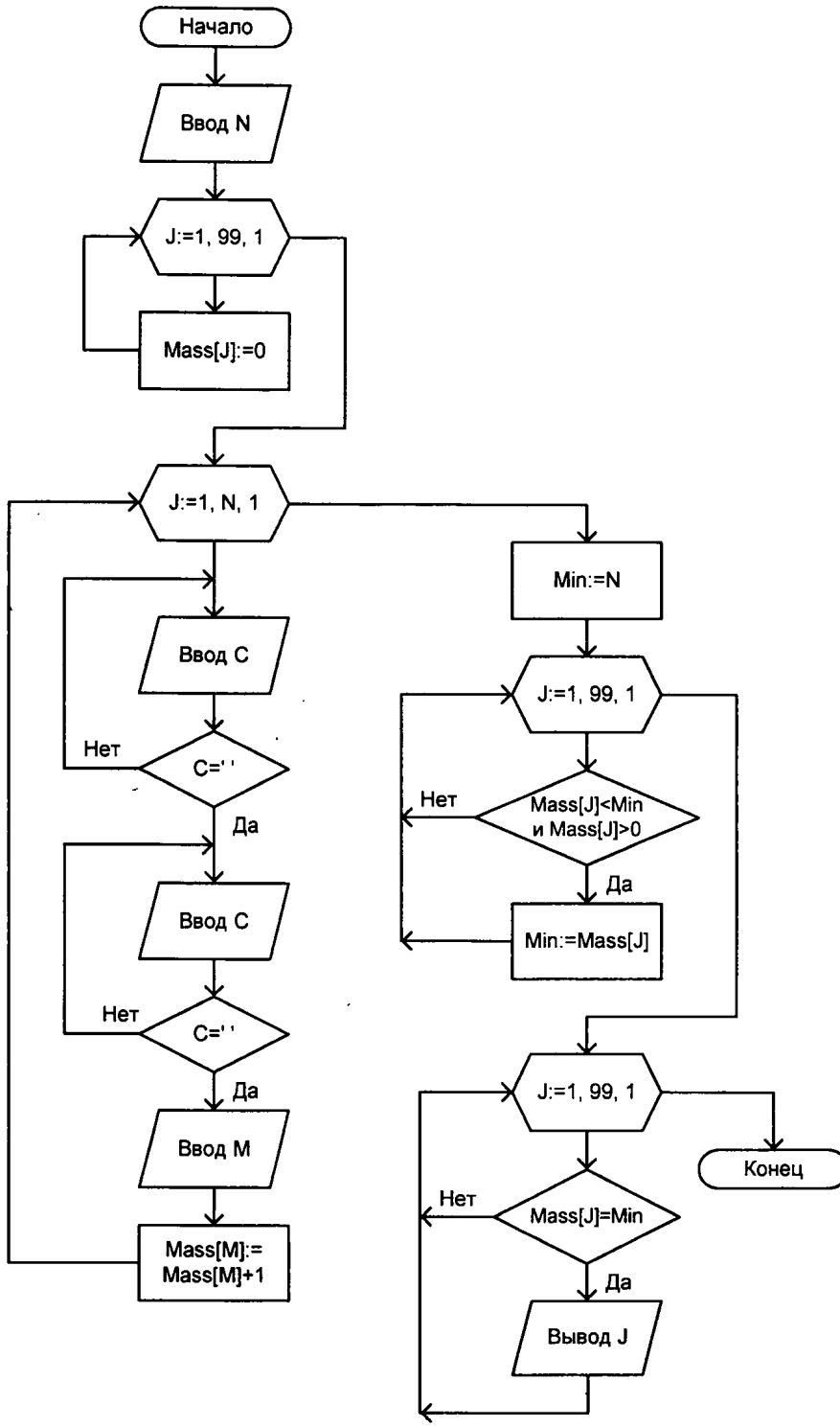


Рис. 9.2. Блок-схема к программе листинга 9.3

интересуют), то следует просто прибавить единицу в определенный элемент массива. Например, если в очередной вводимой строке считан учащийся из школы с номером 25, то в элемент массива с индексом 25 добавляется единица. После такой обработки строк необходимо найти минимальный элемент в массиве (только при этом необходимо учесть, что значение элемента не должно равняться нулю). В завершающей части алгоритма с помощью еще одного цикла на экран выводятся соответствующие школы. В листинге 9.3 представлена программа, которая реализует рассмотренный алгоритм.

Листинг 9.3. Формирование отчета по олимпиаде

```
program listing_9_3;
var
  Mass: array[1..99] of integer;
  N,J,M,Min:integer;
  C:char;
begin
  readln(N);
  for J:=1 to 99 do
    Mass[J]:=0;
  for J:=1 to N do
   begin
      repeat
         read(C);
     until C=' '; { Считана фамилия }
      repeat
         read(C);
      until C=' '; { Считаны инициалы }
      readln(M); { Считан номер школы }
      Mass[M]:= Mass[M]+1;
   end;
   Min:=N;
   for J:=1 to 99 do
     if (Mass[J] < Min) and (Mass[J]>0) then
        Min:= Mass[J];
   for J:=1 to 99 do
     if Mass[J]=Min  then
          writeln(J);
end.
```

Сертификаты

А теперь рассмотрим такую задачу. Пусть некоторый вуз принимает сертификаты по трем предметам (математика, русский язык и информатика). Программа получает входную информацию об этих сертификатах. Так, в первой строке вводится

количество абитуриентов. После этого вводится еще N строк, каждая из которых имеет следующий формат:

- фамилия — строка, содержащая не более 20 символов;
- оценка1 — целое число от 0 до 100;
- оценка2 — целое число от 0 до 100;
- оценка3 — целое число от 0 до 100.

Пример входной строки:

```
Петров 45 76 23.
```

Необходимо составить список трех учеников, набравших наибольшее количество баллов. Если и среди других окажутся люди, набравшие такой же балл, то их также необходимо включить в список. Следует учитывать, что N<300.

На рис. 9.3 и 9.4 приведена блок-схема алгоритма, а в листинге 9.4 представлена необходимая программа. В начале программы мы отвели массивы Mass и Poits для хранения информации о фамилии и набранных баллах каждого участника. После ввода числа учеников производится считывание N строк с информацией о фамилиях и набранных баллах.

Для фиксирования трех лучших результатов мы используем переменные Max1, Max2 и Max3 (Max1 предназначается за самого большого значения, Max2 — для следующего в порядке возрастания и т. д.). При просмотре элементов массива Poits руководствуемся правилом: если очередное значение оказывается больше Max1, то обновляются значения всех переменных Max1, Max2 и Max3. Если же данное условие не выполняется, то производится сравнение очередного значения с Max2 (в этом случае возможно обновление значений только двух переменных). Последнее условие проверяет Max3, и если очередное значение Poits[J] оказывается больше Max3, то Max3 обновляется. Таким образом, после описанного просмотра всех элементов Points[J] мы получим три лучших результата. Далее в цикле по числу участников производится вывод фамилий.

Листинг 9.4. Список наилучших сертификатов

```
program listing_9_4;
var
Mass: array[1..300] of string[50];
Points: array[1..300] of integer;
N,J,I,Max1,Max2,Max3,M:integer;
C:char;
begin
  readln(N);
  for J:=1 to N do
   begin
      Mass[J]:= '';
      repeat
         read(C);
         Mass[J]:= Mass[J]+ C;
```

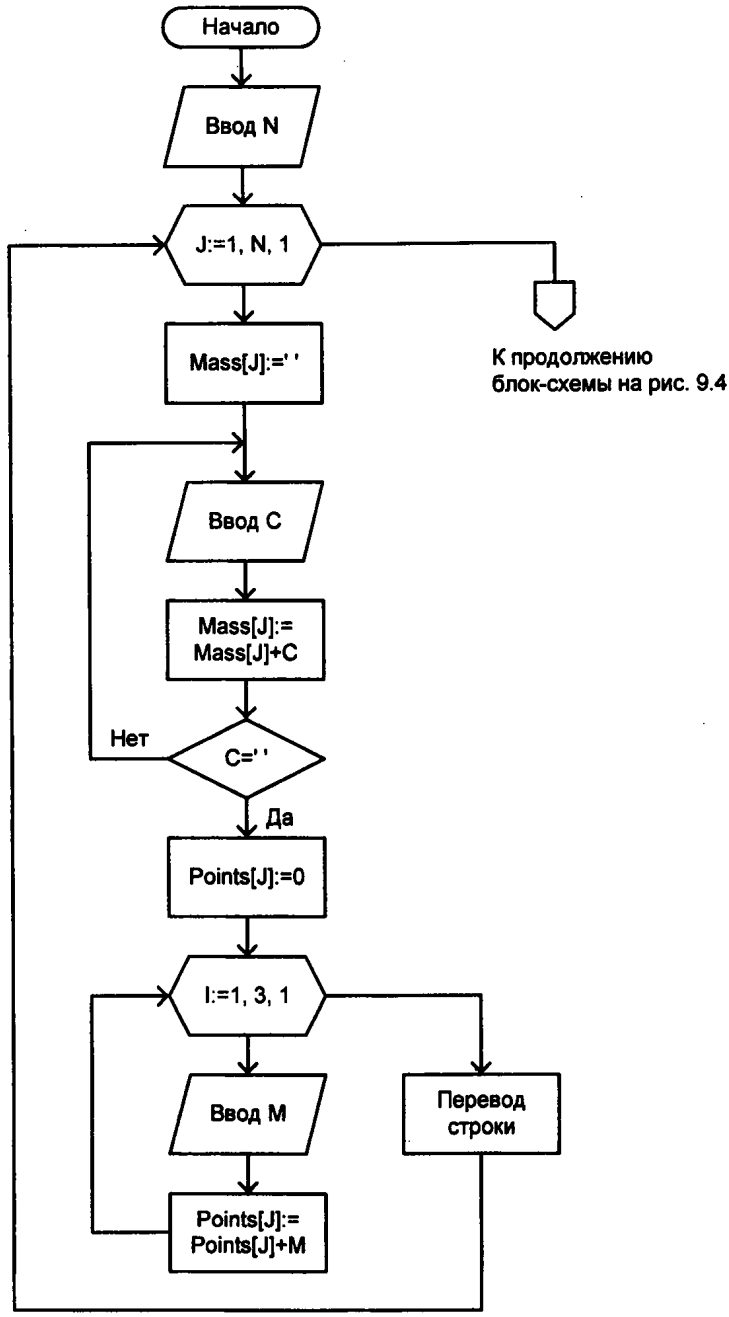


Рис. 9.3. Блок-схема (1 часть) к программе листинга 9.4

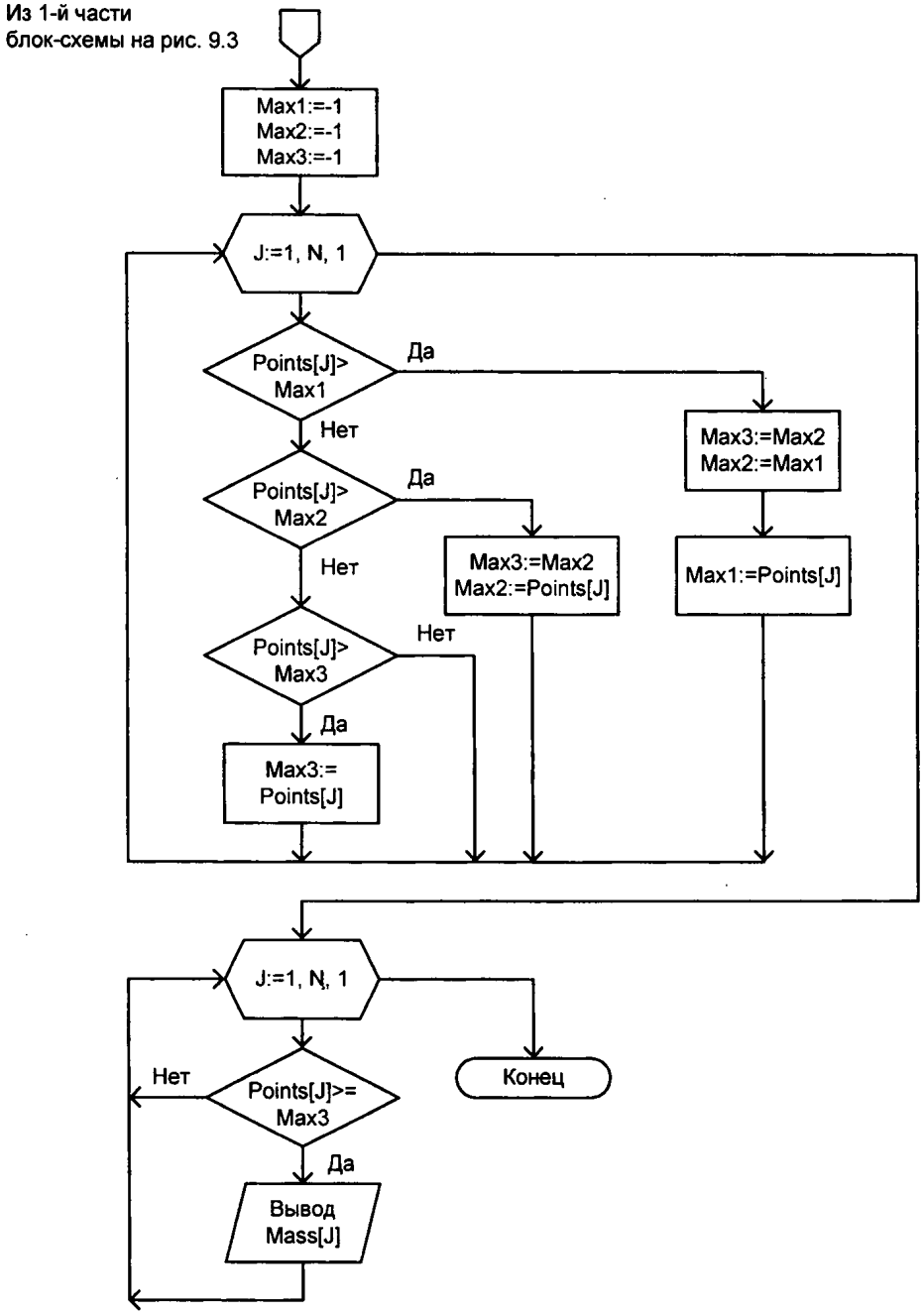


Рис. 9.4. Блок-схема (2 часть) к программе листинга 9.4

```
      until C=' '; { Считана фамилия }
      Points[J]:=0;
      for I:=1 to 3 do
        begin
          read(M);
          Points[J]:=Points[J]+M;
       end;
       readln;
   end;
   Max1:=-1; Max2:=-1; Max3:=-1;
   for J:= 1 to N do
    begin
      if Points[J] > Max1 then
          begin
           Max3:= Max2; Max2:= Max1; Max1:= Points[J];
          end
      else
          if Points[J] > Max2 then
            begin
               Max3:= Max2; Max2:= Points[J];
            end
      else
          if Points[J] > Max3 then
                Max3:= Points[J];
  end;
  for J:=1 to N do
   if (Points[J]>= Max3 ) then
      writeln (Mass[J]);
end.
```

В предложенном варианте разработки мы выполнили условие задания, однако фамилии учащихся при таком алгоритме отражаются не в порядке уменьшения набранных баллов. Поэтому исправим эту недоработку. Кроме того, реализуем хранение информации об учащихся и набранных ими баллами в одном массиве записей. Второй вариант программы поиска лучших сертификатов приведен в листинге 9.5.

Листинг 9.5. Список наилучших сертификатов (вариант 2)

```
program listing_9_5;
Type
  People=record
  Fam:string[50];
  Points: integer;
end;
var
  Mass: array[1..300] of People;
```

```
  N,J,I,Max1,Max2,Max3,M:integer;
  C:char;
begin
  readln(N);
  for J:=1 to N do
   begin
      Mass[J].Fam:= '';
      repeat
         read(C);
         Mass[J].Fam:= Mass[J].Fam+ C;
      until C=' '; { Считана фамилия }
      Mass[J].Points:=0;
      for I:=1 to 3 do
        begin
           read(M);
           Mass[J].Points:= Mass[J].Points+M;
       end;
       readln;
   end;
   Max1:=-1; Max2:=-1; Max3:=-1;
   for J:= 1 to N do
    begin
      if Mass[J].Points > Max1 then
          begin
           Max3:= Max2; Max2:= Max1; Max1:= Mass[J].Points;
          end
      else
          if Mass[J].Points > Max2 then
            begin
               Max3:= Max2; Max2:= Mass[J].Points;
            end
      else
          if Mass[J].Points > Max3 then
                Max3:= Mass[J].Points;
  end;
  for J:=1 to N do
   if (Mass[J].Points= Max1 ) then
      writeln (' 1 место ', Mass[J].Points,' ',Mass[J].Fam);
  for J:=1 to N do
   if (Mass[J].Points= Max2 ) then
      writeln (' 2 место ', Mass[J].Points,' ',Mass[J].Fam);
 for J:=1 to N do
   if (Mass[J].Points= Max3 ) then
      writeln (' 3 место ', Mass[J].Points,' ',Mass[J].Fam);
end.
```

Результаты экзамена

Теперь рассмотрим задачу, где на вход программы подаются сведения о результатах экзаменов. Общее количество учеников не превосходит 100. Так, в первой строке вводится количество учеников N. После этого вводится еще N строк, каждая из которых имеет следующий формат:

- фамилия — строка, содержащая не более 20 символов;
- имя — строка, содержащая не более 15 символов;
- баллы — четыре числа, разделенные пробелами.

При этом фамилия и имя, а также имя и баллы разделены одним пробелом. Необходимо составить программу, которая будет выводить список учеников, набравших максимальную сумму баллов, а также количество таких учеников.

На рис. 9.5 и 9.6 приведена блок-схема алгоритма, а в листинге 9.6 представлена необходимая программа, которая реализует рассмотренный алгоритм. Действия, выполняемые в соответствии с алгоритмом, достаточно точно поясняются блок-схемой и каких-либо дополнительных комментариев не требуют.

Листинг 9.6. Формирование списка лучших учеников

```
program listing_9_6;
var
  Mass: array[1..100] of string;
  Points: array[1..100] of integer;
  N,J,I,K,M,Max:integer;
  C:char;
begin
  readln(N);
  for J:=1 to N do
   begin
     Mass[J]:= '';
     repeat
       read(C);
       Mass[J]:= Mass[J]+ C;
     until C=' '; { Считана фамилия }
     repeat
       read(C);
       Mass[J]:= Mass[J]+ C;
     until C=' '; { Считано имя }
     Points[J]:=0;
     for I:=1 to 4 do
     begin
        read(M);
        Points[J]:=Points[J]+M;
     end;
     readln;
   end;
```

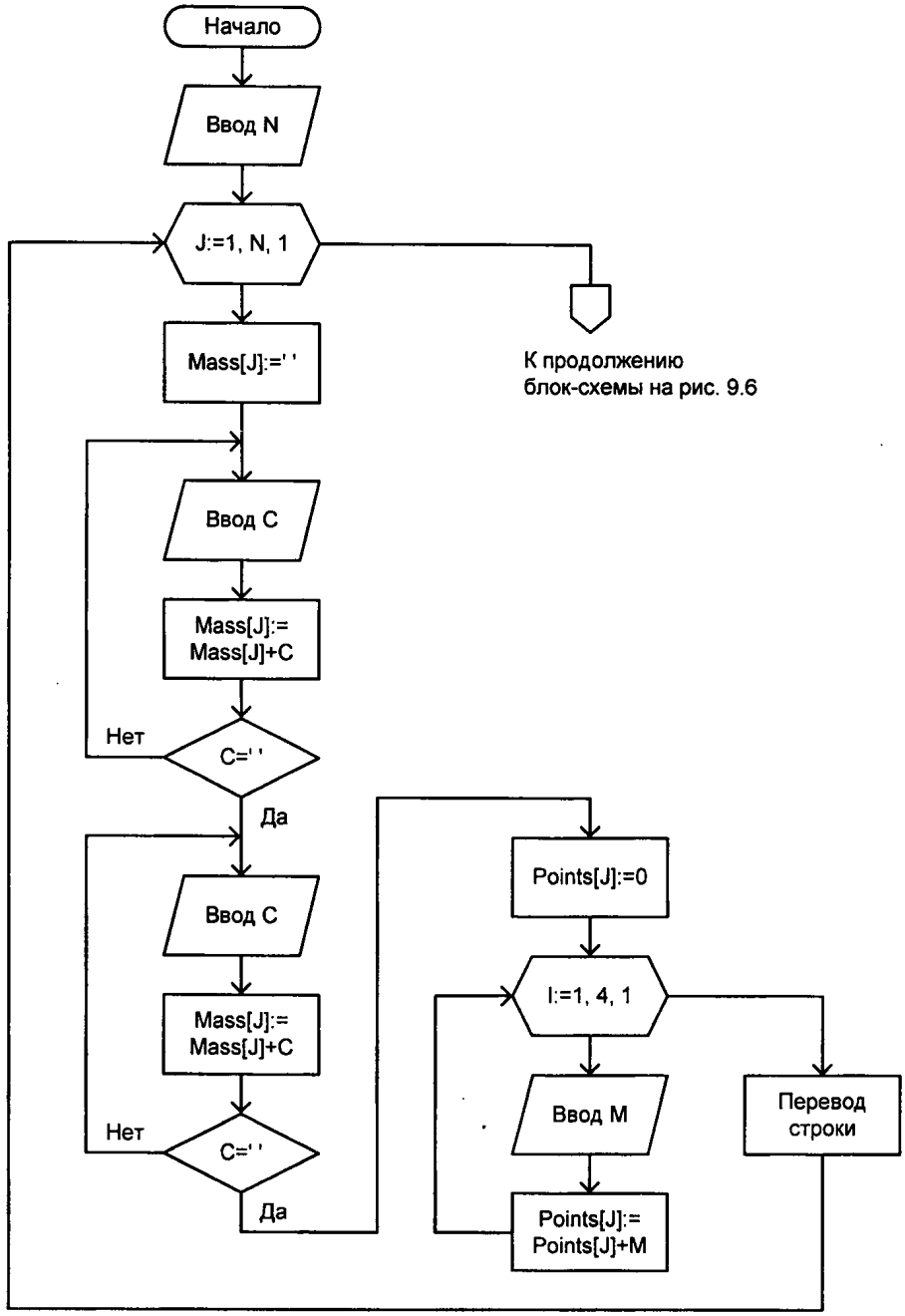


Рис. 9.5. Блок-схема (1 часть) к программе листинга 9.6

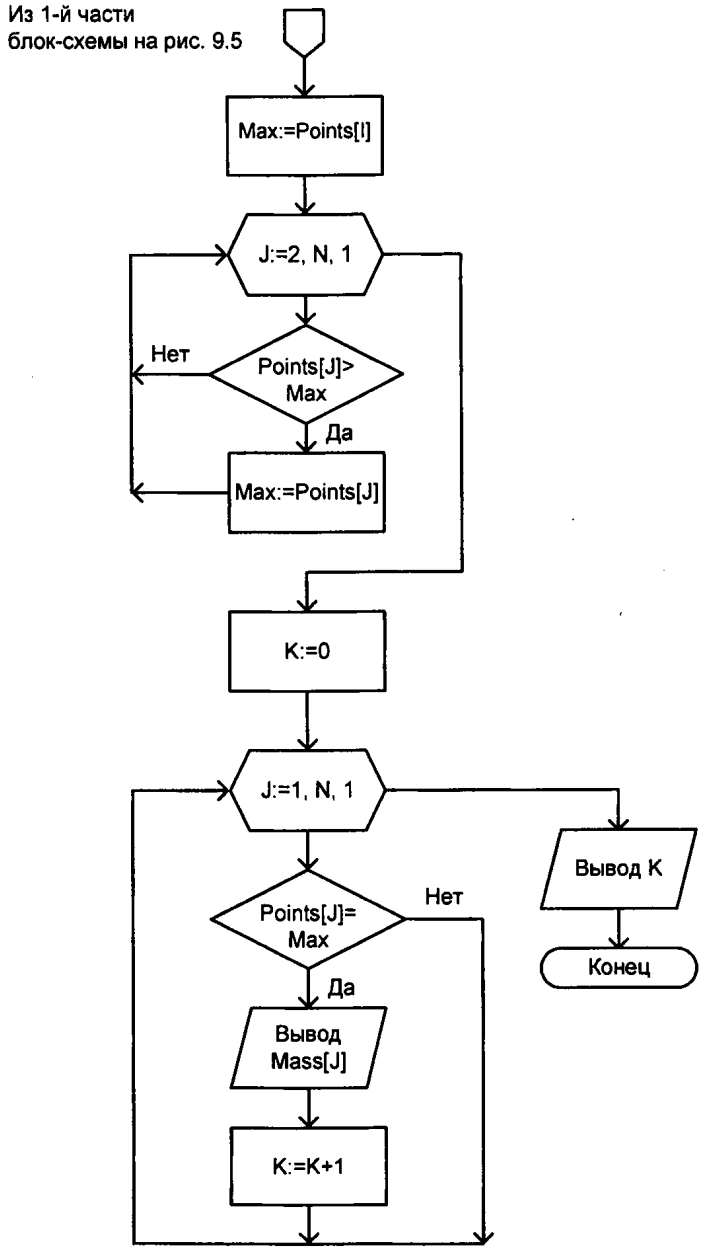


Рис. 9.6. Блок-схема (2 часть) к программе листинга 9.6

```
   Max:= Points[1];
   for J:= 2 to N do
      if Points[J] > Max then
          Max:= Points[J];
   K:=0;
   for J:= 1 to N do
      if Points[J] = Max then
       begin
         writeln(Mass[J]);
         K:=K+1;
       end;
  writeln('Число учеников, получивших максимальный балл = ',K);
end.
```

Аналогично предыдущему примеру приведем вариант программы (листинг 9.7) с использованием записей.

Листинг 9.7. Формирование списка лучших учеников (вариант 2)

```
program listing_9_7;
Type
  People=record
  Fam:string;
  Points: integer;
end;
var
  Mass: array[1..100] of People;
  N,J,I,K,M,Max:integer;
  C:char;
begin
  readln(N);
  for J:=1 to N do
   begin
     Mass[J].Fam:= '';
     repeat
       read(C);
       Mass[J].Fam:= Mass[J].Fam + C;
     until C=' '; { Считана фамилия }
     repeat
       read(C);
       Mass[J].Fam:= Mass[J].Fam + C;
     until C=' '; { Считано имя }
     Mass[J].Points:=0;
     for I:=1 to 4 do
     begin
        read(M);
        Mass[J].Points:= Mass[J].Points+M;
     end;
```

```
    readln;
  end;
  Max:= Mass[1].Points;
  for J:= 2 to N do
     if Mass[J].Points> Max then
        Max:= Mass[J].Points;
  K:=0;
  for J:= 1 to N do
     if Mass[J].Points = Max then
      begin
        writeln(Mass[J].Fam);
        K:=K+1;
      end;
 writeln('Число получивших максимальный балл = ',K);
end.
```

Полупроходной балл

Допустим, что на вход программы подаются сведения о сертификатах абитуриентов. Общее количество абитуриентов не превосходит 500. В первой строке программы вводится количество абитуриентов N, а во второй вводится количество мест K, на которые они претендуют. После этого вводится еще N строк, каждая из которых имеет следующий формат:

- фамилия и инициалы — строка, содержащая не более 50 символов;
- баллы по первому предмету;
- баллы по второму предмету;
- баллы по третьему предмету.

Баллы представляют собой три числа, разделенные пробелами (каждое число является целым в интервале от 0 до 100).

При этом фамилия с инициалами и баллы разделены одним пробелом (считается, что между фамилией и инициалами пробела нет — просто инициалы пишутся большими буквами). Необходимо составить программу, которая будет выводить список абитуриентов, набравших полупроходной балл, а также количество таких учеников. Или же необходимо указать, что полупроходного балла нет.

Примечание

Полупроходным называется такой балл, когда лишь часть абитуриентов, набравших его, попадает в число поступивших.

На рис. 9.7 и 9.8 приведена блок-схема алгоритма, а в листинге 9.8 представлена необходимая программа, которая реализует рассмотренный алгоритм. Массив `Points` используется для подсчета числа абитуриентов, набравших конкретное количество баллов. Например, элемент `Points[300]` содержит число учащихся,

набравших 300 баллов, соответственно `Points[299]` — число учащихся, набравших 299 баллов.

В массивах `Mass` и `Balls` фиксируется фамилия и число баллов абитуриента. Например, `Mass[1]` содержит фамилию первого учащегося, а в `Balls[1]` фиксируется число баллов, которое данный учащийся набрал.

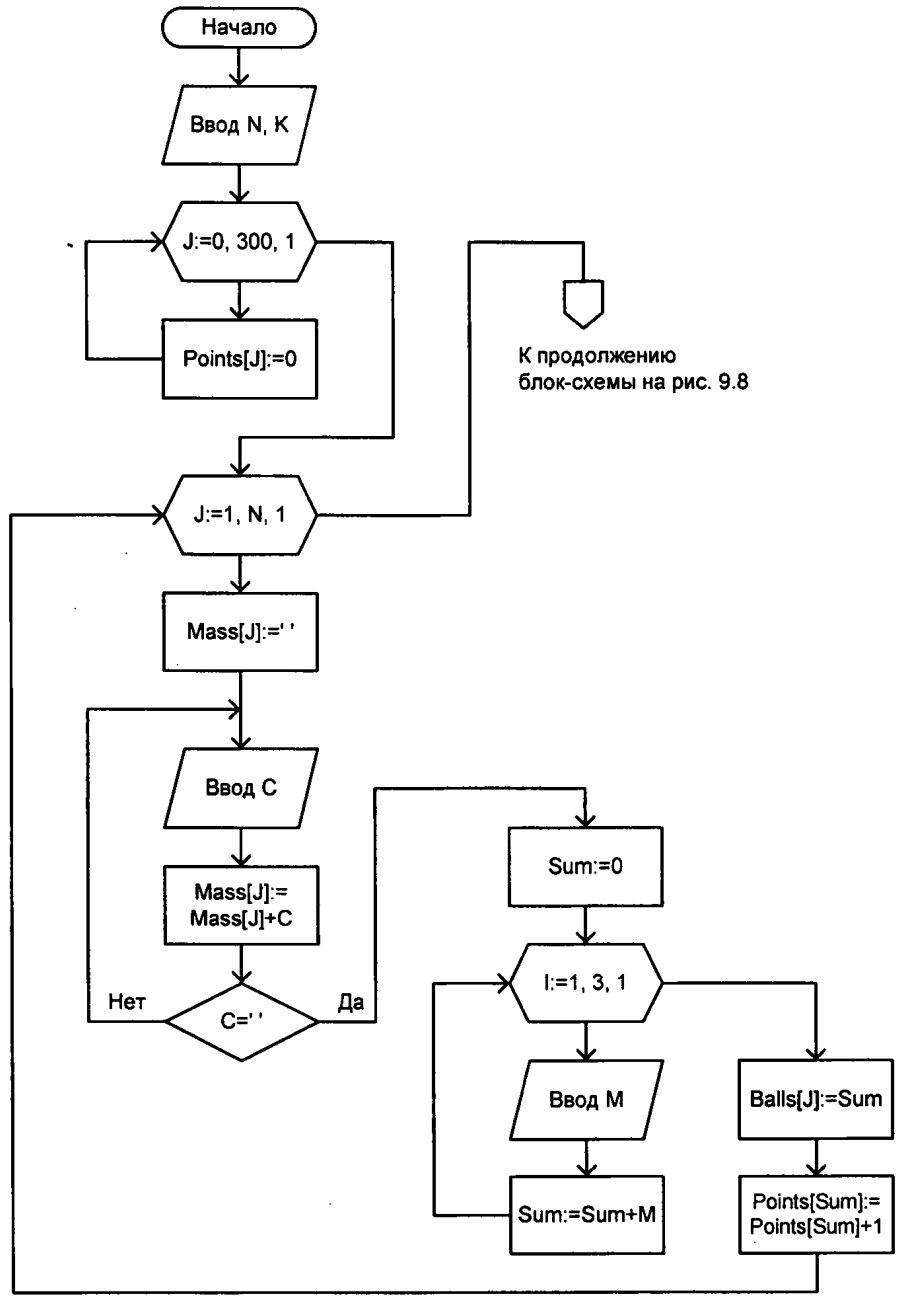


Рис. 9.7. Блок-схема (1 часть) к программе листинга 9.8

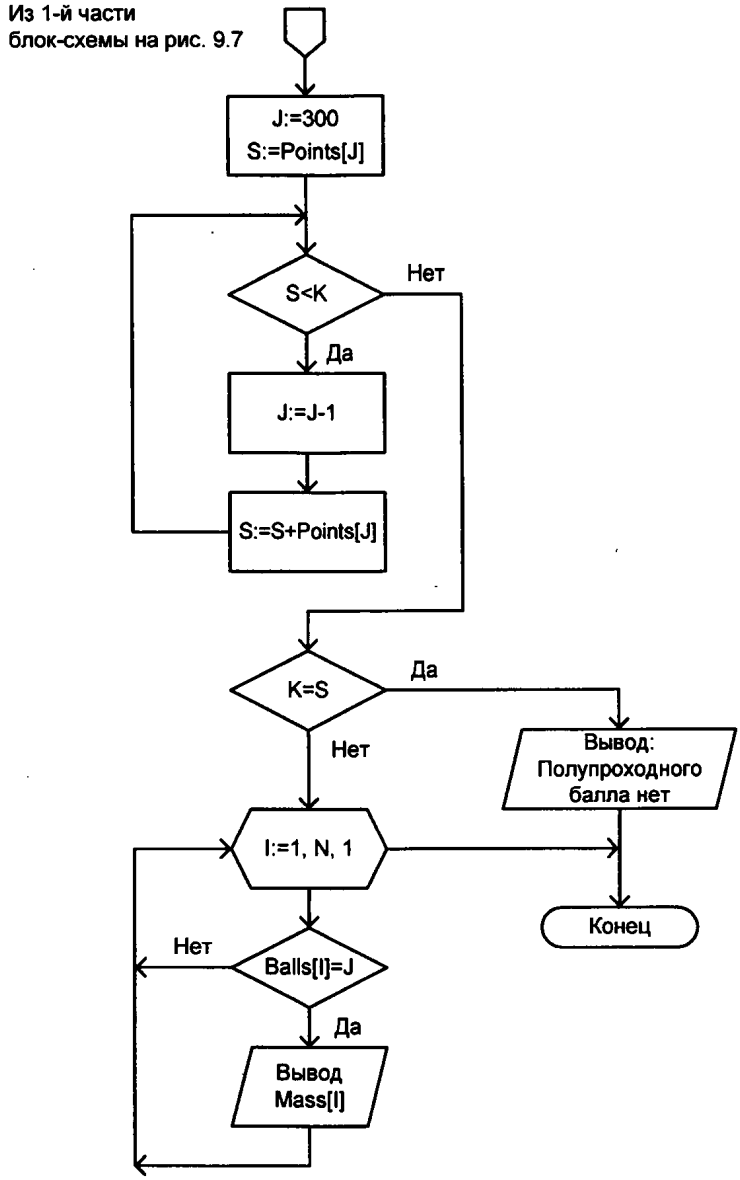


Рис. 9.8. Блок-схема (2 часть) к программе листинга 9.8

После ввода всех данных производится поиск полупроходного балла. Для этого используется цикл с предусловием:

```
while S < K do
    begin
        J:=J-1;
        S:= S + Points[J];
   end;
```

В результате в переменной J фиксируется значение полупроходного балла, а в переменной S — число учащихся, которые претендуют на K мест. В число S входят те, кто набрал или проходной балл, или полупроходной. Далее в случае наличия полупроходного балла производится вывод фамилий учащихся, которые его набрали.

Листинг 9.8. Формирование списка учеников с полупроходным баллом

```
program listing_9_8;
var
  Mass: array[1..500] of string[50];
  Balls: array[1..500] of integer;
  Points: array[0..300] of integer;
  N, K , J , I, Sum, S, M :integer;
  C:char;
begin
  readln(N);
  readln(K);
  for J:=0 to 300 do
       Points[J]:=0;
  for J:=1 to N do
   begin
     Mass[J]:= '';
     repeat
       read(C);
       Mass[J]:= Mass[J]+ C;
     until C=' '; { Считана фамилия с инициалами }
     Sum:=0;
     for I:=1 to 3 do
     begin
        read(M);
        Sum:= Sum +M;
     end;
     Balls[J]:=Sum;
     Points[Sum]:= Points[Sum]+1;
     readln;
   end;
   J:=300;
   S:= Points[J];
   while S < K do
     begin
         J:=J-1;
         S:= S + Points[J];
    end;
if K=S then
   writeln('Полупроходного балла нет')
```

```
else
   begin
     for I:= 1 to N do
      if Balls[I] = J then
         writeln(Mass[I]);
   end;
end.
```

Аналогично предыдущему примеру приведем вариант программы (листинг 9.9) с использованием записей.

Листинг 9.9. Формирование списка учащихся с полупроходным баллом (вариант 2)

```
program listing_9_9;
Type
  People=record
  Fam: string[50];
  Balls: integer;
end;
var
  Mass: array[1..500] of People;
  Points: array[0..300] of integer;
  N, K , J , I, Sum, S, M :integer;
  C:char;
begin
  readln(N);
  readln(K);
  for J:=0 to 300 do
       Points[J]:=0;
  for J:=1 to N do
   begin
     Mass[J].Fam:= '';
     repeat
       read(C);
       Mass[J].Fam:= Mass[J].Fam + C;
     until C=' '; { Считана фамилия с инициалами }
     Sum:=0;
     for I:=1 to 3 do
     begin
        read(M);
        Sum:= Sum +M;
     end;
     Mass[J].Balls:=Sum;
     Points[Sum]:= Points[Sum]+1;
     readln;
   end;
```

```
   J:=300;
   S:= Points[J];
   while S < K do
     begin
         J:=J-1;
         S:= S + Points[J];
     end;
if K=S then
   writeln('Полупроходного балла нет')
else
   begin
     for I:= 1 to N do
      if Mass[I].Balls = J then
         writeln(Mass[I].Fam);
   end;
end.
```

Сортировка

Сортировкой называется процесс размещения заданного множества объектов в определенном порядке (в частности, если речь идет о числах, то рассматривается сортировка в порядке убывания либо в порядке возрастания).

Существует много различных методов сортировки [5]. При этом простые методы предполагают элементарную программную реализацию, а сложные отличаются более изощренными программными действиями. Далее мы рассмотрим ряд алгоритмов сортировки применительно к массивам чисел.

Сортировка выбором

Сортировка выбором заключается в том, что сначала в неупорядоченном массиве выбирается минимальный элемент. Этот элемент исключается из дальнейшей обработки, а оставшаяся последовательность элементов принимается за исходную. Этот процесс повторяется до тех пор, пока все элементы не будут выбраны. В результате выбранные элементы образуют упорядоченную последовательность.

Для реализации данной идеи удобнее всего использовать дополнительный массив. После первого просмотра найденный минимальный элемент размещается на первом месте в этом дополнительном массиве (рис. 9.9). При втором просмотре мы должны найти следующий элемент в порядке возрастания. Для этого необходимо исключить уже найденный минимальный элемент. Лучший вариант состоит в том, чтобы вместо минимального элемента записать число, заведомо превосходящее все имеющиеся элементы массива. Тогда найденный при втором просмотре элемент размещается на втором месте в дополнительном массиве. Этот процесс аналогичным образом продолжается для третьего, четвертого и последующих элементов.

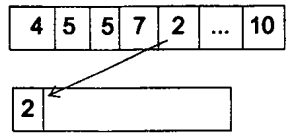


Рис. 9.9. Размещение минимального элемента в дополнительном массиве

В листинге 9.10 приведена программа, реализующая рассматриваемый алгоритм сортировки. В начале программы производится заполнение элементов массива с помощью датчика случайных чисел. Далее мы отыскиваем максимальный элемент для замены очередных найденных минимальных значений массива. После этого происходит поиск минимального элемента, который переносится в дополнительный массив. Затем производятся уже описанные действия по последовательному переносу элементов. Алгоритм проиллюстрирован блок-схемой (рис. 9.10).

Листинг 9.10. Сортировка массива выбором

```
program listing_9_10;
const
   N=10;
var
   A,B:array[1..N] of integer;
   J,I,Jmin,Min,Max:integer;
begin
  for J:=1 to N do
      A[J]:=random(100);
  Max:=A[1];
  for J:=2 to N do
    if A[J]> Max then
          Max:=A[J];
 for I:=1 to N do
  begin
    Min:=A[1];
    Jmin:=1;
    for J:=2 to N do
       begin
           if A[J]< Min then
              begin
                  Min:= A[J];
                  Jmin:=J;
              end;
       end;
     B[I]:=Min;
     A[Jmin]:=Max;
  end;
end.
```

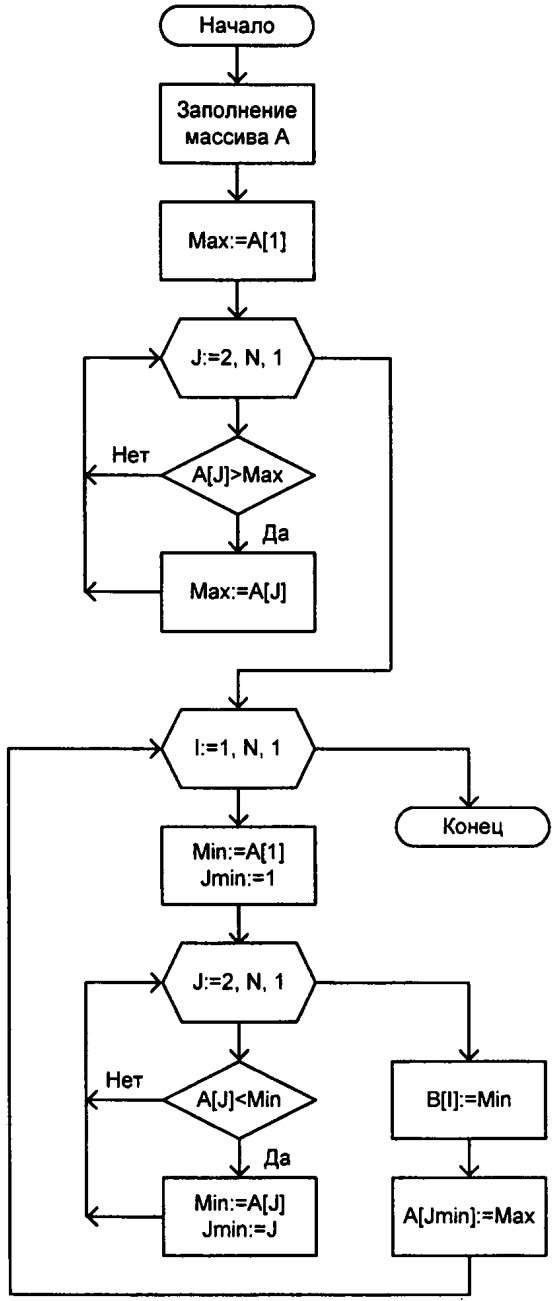


Рис. 9.10. Блок-схема к программе листинга 9.10

Рассмотренный вариант сортировки массива методом выбора обладает двумя недостатками:

- для его реализации требуется дополнительный массив;
- для нахождения минимального элемента и его индекса на каждом проходе приходится просматривать все элементы массива.

Указанные недостатки устраняются, если все изменения проводить в исходном массиве. Отсортировать его можно следующим образом:

1. Найти минимальный элемент среди всех элементов массива и поменять его местами с первым элементом.
2. Найти минимальный элемент среди группы, начинающейся со второго элемента, и поменять его местами со вторым элементом.
3. Найти минимальный элемент среди группы, начинающейся с третьего элемента, и поменять его местами с третьим элементом и т. д.

На последнем этапе определяется минимальный элемент среди двух последних (по номеру индекса), после чего найденный элемент меняется местами с предпоследним элементом исходного массива. В листинге 9.11 приведена реализация описанного алгоритма, а на рис. 9.11 представлена непосредственно блок-схема алгоритма.

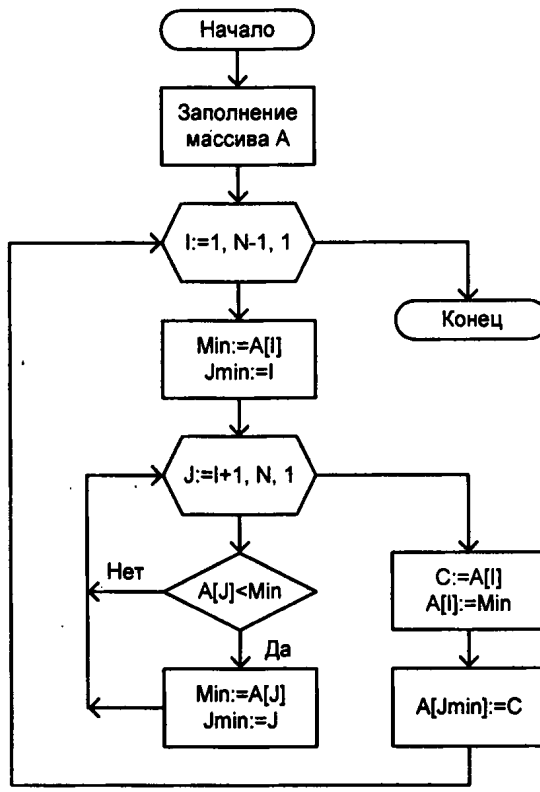


Рис. 9.11. Блок-схема к программе листинга 9.11

Листинг 9.11. Сортировка выбором без использования дополнительного массива

```
program listing_9_11;
const
   N=10;
```

```
var
   A:array[1..N] of integer;
   J,I,Max,Min,Jmin,C:integer;
begin
  for J:=1 to N do
      A[J]:=random(100);
  for I:=1 to N-1 do
  begin
    Min:=A[I];
    Jmin:=I;
    for J:=I+1 to N do
       begin
           if A[J]< Min then
              begin
                  Min:=A[J];
                  Jmin:=J;
              end;
       end;
     C:= A[I];
     A[I]:=Min;
     A[Jmin]:=C;
  end;
  for J:=1 to N do
    writeln('A=',A[J]);
end.
```

Сортировка обменом значений

Сортировка обменом значений заключается в том, все соседние элементы попарно сравниваются друг с другом и меняются местами, если предшествующий элемент больше последующего. В результате максимальный элемент постепенно смещается вправо. После первого такого просмотра массива максимальный элемент займет крайнее правое положение. Затем процесс просмотра повторяется, и свое место занимает второй по величине элемент, который также исключается из дальнейшего рассмотрения.

Однако описанная процедура оказывается неэффективной, если массив изначально практически отсортирован. В этом случае будут лишние просмотры, что приводит к увеличению времени выполнения программы. Возможен более эффективный вариант, связанный с фиксированием факта обмена. Так, если факт обмена при очередном просмотре массива не фиксируется, то сортировка завершается. В листинге 9.12 приведена программная реализация указанного способа, который называется *пузырьковой сортировкой* (или *сортировкой методом пузырька*). Здесь для фиксирования факта обмена мы создали переменную `Flag`, которая перед началом очередного просмотра сбрасывается в 0. Если в процессе просмотра происходит обмен значений, то переменная `Flag` принимает значение, равное 1. Это является поводом просмотреть массив еще раз.

Листинг 9.12. Сортировка с помощью метода пузырька

```
program listing_9_12;
const
   N=10;
var
   A:array[1..N] of integer;
   J,M,Vr,Flag:integer;
begin
  for J:=1 to N do
      A[J]:=random(100);
 M:=N;
 Flag:= 0;
  while Flag = 0 do
   begin
    Flag:=0;
    for J:=1 to M-1 do
        if A[J]> A[J+1] then
           begin
               Vr:=A[J];
               A[J]:=A[J+1];
               A[J+1]:=Vr;
               Flag:=1;
           end;
    M:=M-1;
   end;
end.
```

Анализ числа учащихся в классах

Рассмотрим задачу, где на вход программе подаются сведения об учениках некоторой средней школы. В первой строке сообщается количество учеников N, а каждая из следующих вводимых строк имеет такой формат:

- фамилия — строка, содержащая не более 20 символов;
- имя — строка, содержащая не более 15 символов;
- класс — год обучения (от 1 до 11) и заглавная буква (от А до Я) без пробела.

Фамилия и имя, а также имя и класс разделены одним пробелом.

Пример входной строки выглядит так:

```
Иванов Петр 10Б
```

Требуется написать как можно более эффективную программу, которая будет выводить на экран информацию о параллелях (годе обучения) с наименьшим числом

учеников. Программа должна выводить на экран в первой строке количество учеников в искомых параллелях, а во второй строке — в порядке возрастания номера этих параллелей через пробел. Например,

```
30
1 7 11
```

В рассматриваемой задаче информация о фамилиях и именах учащихся нам не требуется и, следовательно, выделять память для хранения этой информации мы не будем. Основная трудность связана с тем, что из названия класса необходимо выделить только числовую часть (отбросить букву). Блок-схема алгоритма представлена на рис. 9.12 и 9.13. Начальные шаги аналогичны тем, которые встречались в наших предыдущих примерах. Мы считываем символы, составляющие фамилию и имя. После этого наша задача заключается в том, чтобы считать оставшуюся часть строки, которая содержит номер параллели. В результате в соответствующий элемент массива добавляется единица. В листинге 9.13 приведена программная реализация задачи.

Листинг 9.13. Извлечение информации о параллелях

```
program listing_9_13;
var
Mass: array[1..11] of integer;
N,J,I,Dl,Min,Paral,Ves:integer;
S:string[3];
C:char;
begin
  readln(N);
  for J:=1 to 11 do
    Mass[J]:=0;
  for J:=1 to N do
   begin
      repeat
         read(C);
     until C=' '; { Считана фамилия }
      repeat
         read(C);
      until C=' '; { Считаны инициалы }
      readln(S);
      Dl:=Length(S);
      Paral:=0;
      Ves:=1;
      for I:=Dl-1 downto 1 do
        begin
          Paral:=Paral+ (Ord(S[I])-Ord('0'))*Ves;
          Ves:=Ves*10;
        end;
```

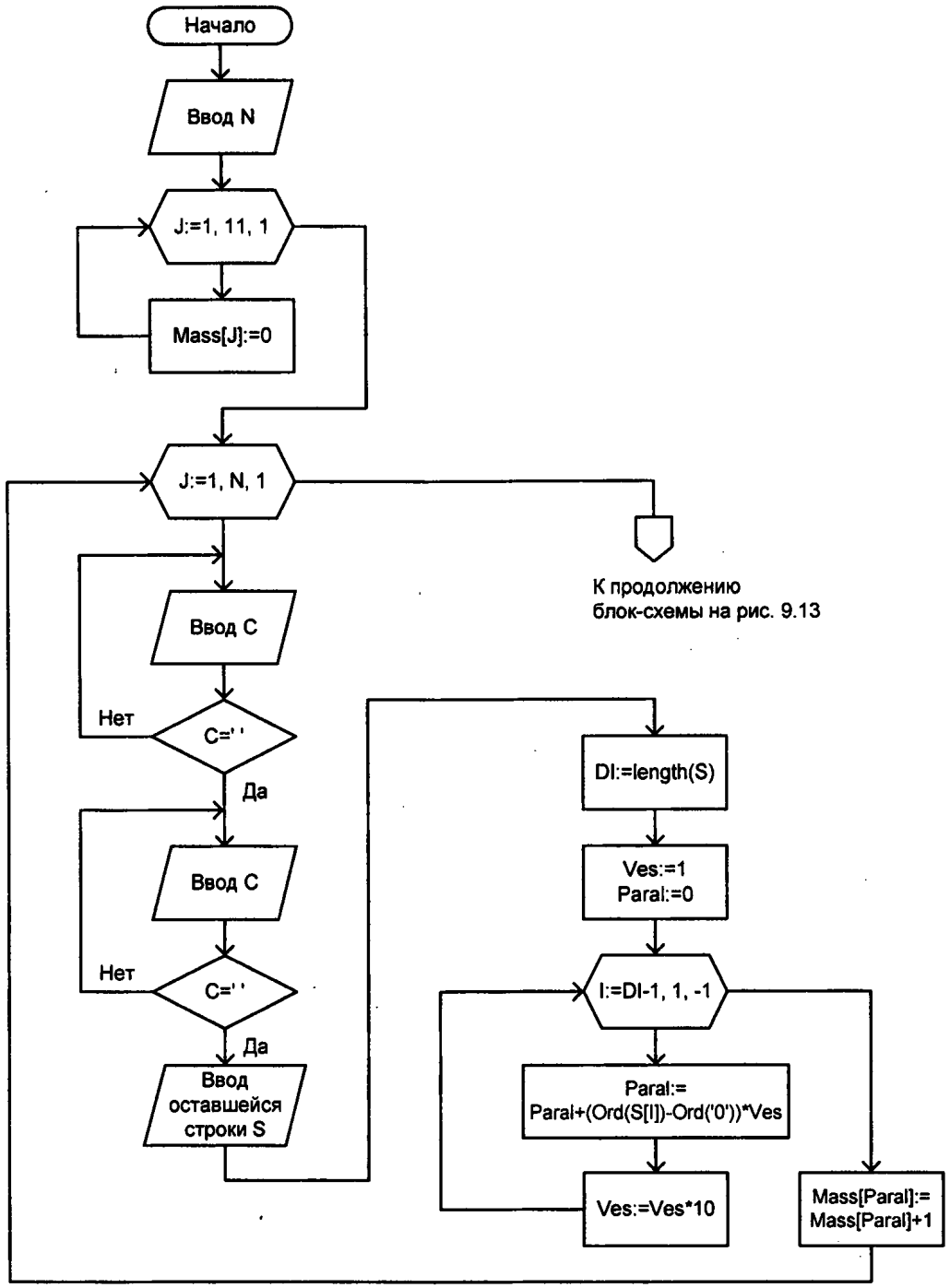


Рис. 9.12. Блок-схема (1 часть) к программе листинга 9.13

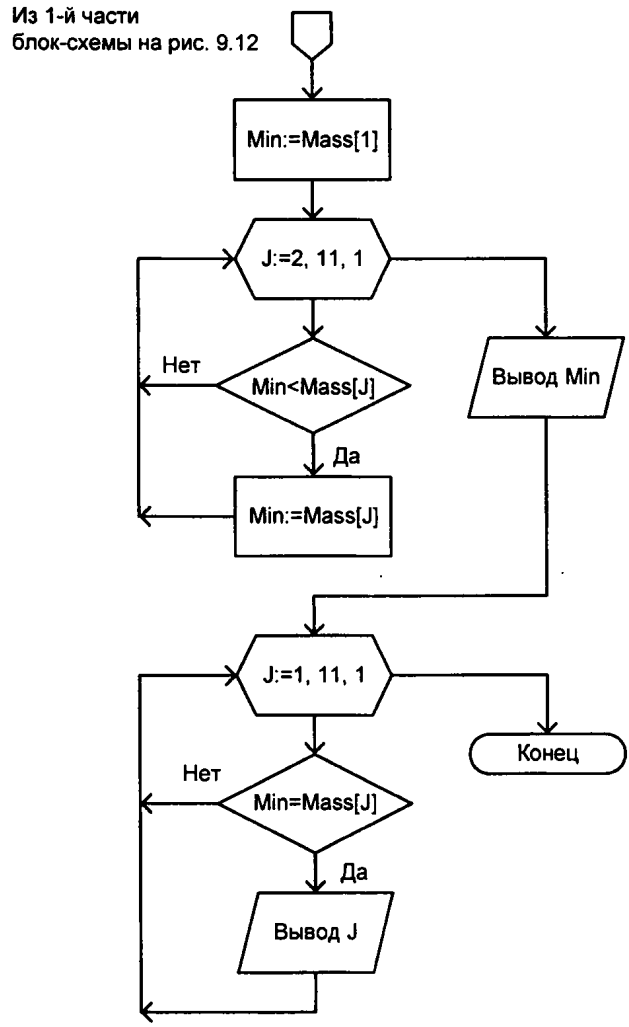


Рис. 9.13. Блок-схема (2 часть) к программе листинга 9.13

```
      Mass[Paral]:= Mass[Paral]+1;
     { write( Mass[Paral] );}
   end;
   Min:=Mass[1];
   for J:=2 to 11 do
     if Mass[J] < Min then
          Min:= Mass[J];
   writeln(Min);
   for J:=1 to 11 do
      if Mass[J]=Min then
          write(J,' ');
end.
```

Статистика температуры

В этой задаче на вход программе подаются 365 строк, которые содержат информацию о среднесуточной температуре всех дней года. Формат каждой из строк следующий:

- дата в виде dd.mm (на запись номера дня и номера месяца в числовом формате отводится строго два символа, день от месяца отделен точкой);
- значение температуры — число со знаком плюс или минус с точностью до одной цифры после десятичной точки (записано через пробел после даты).

Данная информация отсортирована по значению температуры, т. е. хронологический порядок нарушен. Требуется написать как можно более эффективную программу, которая будет выводить на экран информацию о месяцах с минимальной среднемесячной температурой. Найденные минимальные значения следует выводить в отдельной строке для каждого месяца в виде:

- номера месяца;
- значения среднемесячной температуры, округленного до одной цифры после десятичной точки.

Блок-схема алгоритма представлена на рис. 9.14 и 9.15, а в листинге 9.14 приведена программная реализация задачи.

В программе мы задействовали два массива:

- `Mass` — одномерный массив из 12 элементов для накопления суммы температур в течение каждого месяца;
- `Mass2` — одномерный массив из 12 элементов для подсчета количества измерений температуры в течение каждого месяца.

В цикле по строкам данных (365 строк) сначала считываются три символа (день и точка в качестве разделителя), которые отбрасываются, а затем два символа, которые определяют номер месяца. Символьное представление месяца далее мы преобразуем в числовое. После этого из строки считывается значение температуры, что фиксируется в массивах `Mass` и `Mass2`.

Далее необходимо подсчитать среднюю температуру в течение каждого месяца. Для этого организуется цикл от 1 до 12, где последовательно выполняется:

```
Mass[J]:=Mass[J]/Mass2[J]
```

Далее все уже знакомо: вычисляется минимальная температура и на экран выводятся номера месяцев.

Листинг 9.14. Извлечение информации о средней температуре

```
program listing_9_14;
const
   N=365;
```

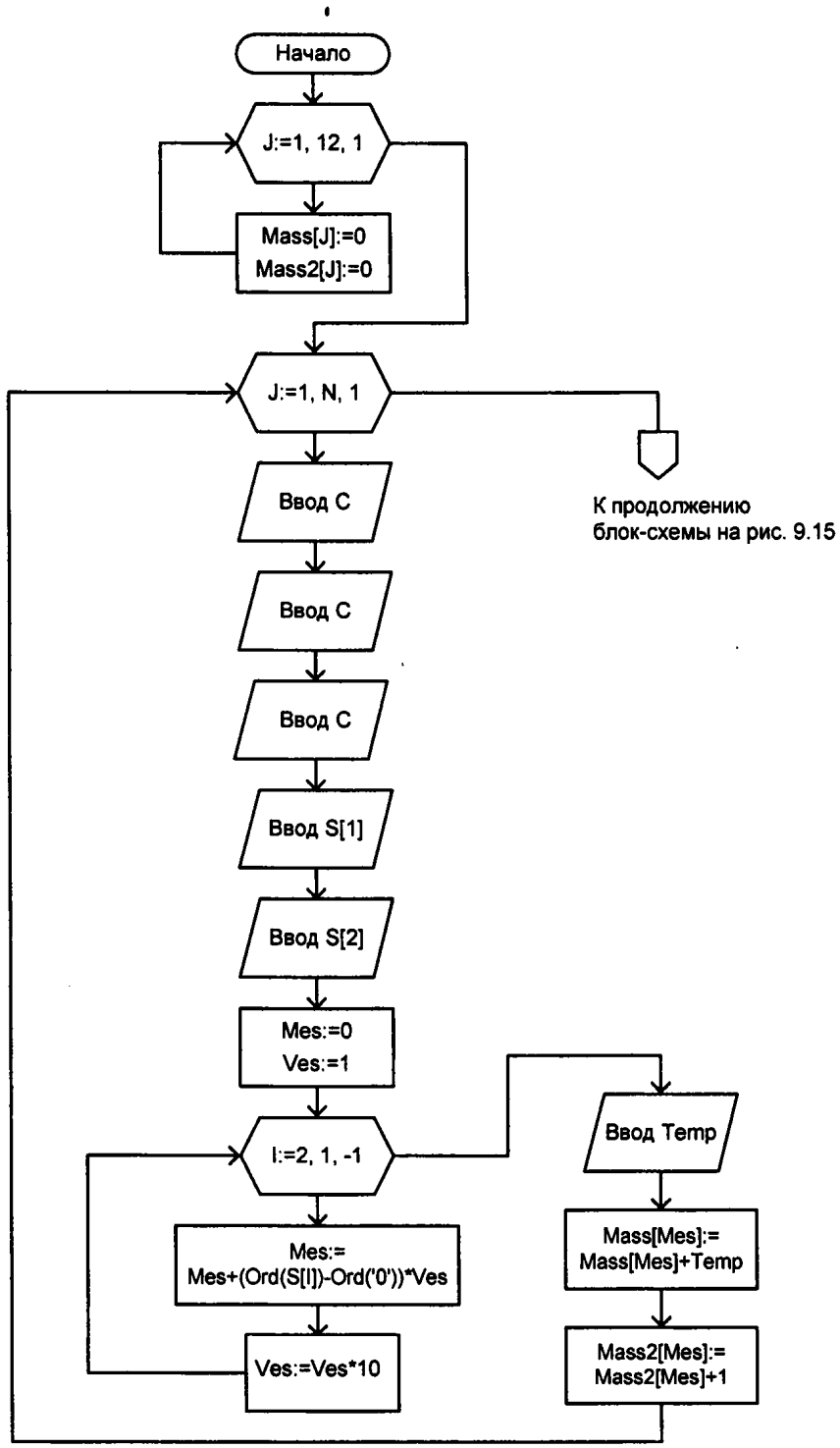


Рис. 9.14. Блок-схема (1 часть) к программе листинга 9.14

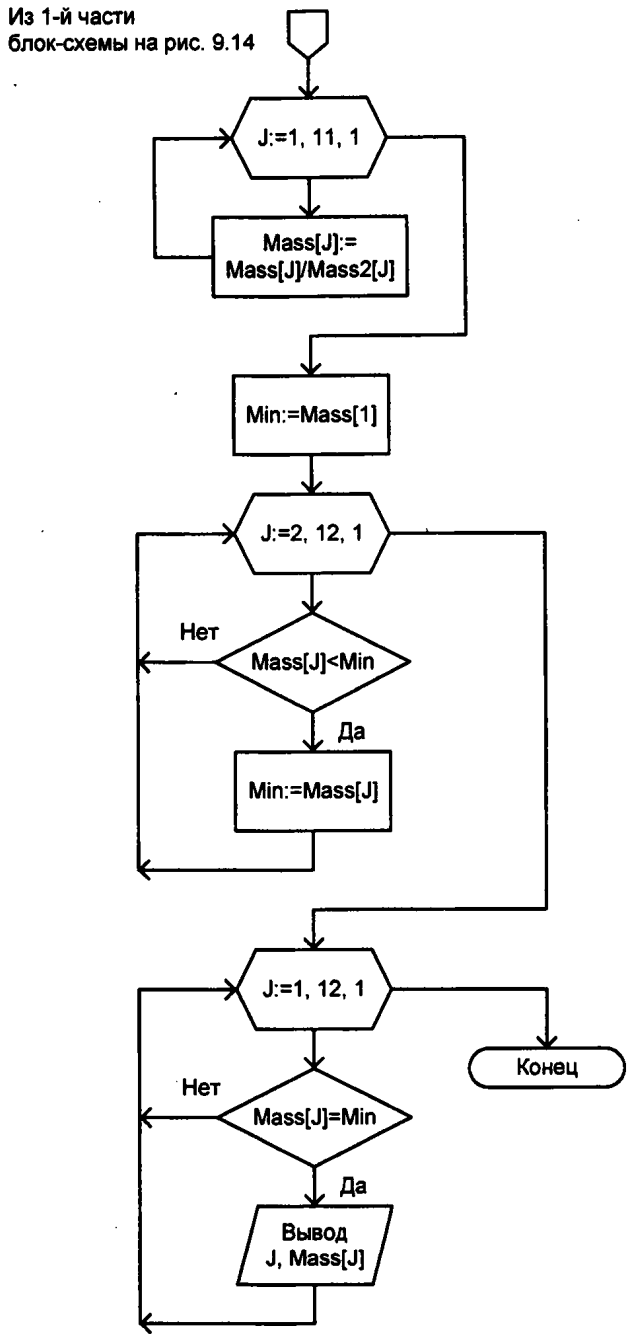


Рис. 9.15. Блок-схема (2 часть) к программе листинга 9.14

```
var
Mass: array[1..12] of real;
Mass2: array[1..12] of integer;
J,I,Mes,Ves:integer;
Temp,Min:real;
S:string[2];
C:char;
begin
  for J:=1 to 12 do
    begin
       Mass[J]:=0;
       Mass2[J]:=0;
    end;
  for J:=1 to N do
    begin
       read(C);read(C);read(C);
       read(S[1]);read(S[2]);
       Mes:=0;
       Ves:=1;
       for I:=2 downto 1 do
         begin
            Mes:=Mes+ (Ord(S[I])-Ord('0'))*Ves;
            Ves:=Ves*10;
         end;
       readln(Temp);
       Mass[Mes]:= Mass[Mes]+Temp;
       Mass2[Mes]:= Mass2[Mes]+1;
    end;
    for J:=1 to 12 do
       Mass[J]:=Mass[J]/Mass2[J];
    Min:=Mass[1];
    for J:=2 to 12 do
      if Mass[J] < Min then
           Min:= Mass[J];
    for J:=1 to 12 do
      if Mass[J]=Min then
          writeln('Номер месяца =',J,'Средняя температура = ',Mass[J]);
end.
```

Формирование результатов тестирования в файле

Рассмотрим следующую задачу. Будем считать, что в файле располагается информация о результатах прохождения тестов учащимися. Тестирование проводилось по пяти темам и формат представления информации в текстовом файле такой:

- строка с фамилией учащегося;
- строка с результатами этого учащегося из пяти целых чисел (считаем, что каждое число определяется из интервала от 0 до 10);
- строка с фамилией другого учащегося;
- строка с результатами другого учащегося и т. д.

Для определенности: общее количество участников тестирования не превосходит 1000.

Программная разработка должна произвести чтение данных из файла, затем отобрать учащихся, которые набрали максимальное количество баллов. После этого необходимо произвести запись фамилий отобранных учащихся вместе с их результатами тестирования в типизированный файл. Формат записи типизированного файла такой:

- строка с фамилией учащегося;
- 5 целых чисел с результатами тестов.

Далее необходимо привести еще одну программу, которая будет считывать данные из типизированного файла, а также выводить их на экран.

Перед написанием программы составим блок-схему (рис. 9.16). Здесь после создания файловой переменной `A1` организуется цикл, в котором из текстового файла считываются строки. Эти сведения заносятся в элементы массива `B[J]`. После этого выполняются действия, которые нам знакомы по предыдущим главам: поиск максимального значения и запись данных в типизированный файл.

В листинге 9.15 приведена программа, функционирующая в соответствии с построенной блок-схемой.

Листинг 9.15. Обработка текстового файла с результатами тестирования

```
program listing_9_15;
Type
  Stud=record
  Fam: string[50];
  Balls: array[1..5] of integer;
end;
var
  A1: text;
  A2: file of Stud;
  B: array[1..1000] of Stud;
  I,J,N,MaxMass,Sum:integer;
begin
  assign(A1,'prim15.txt');
  reset(A1);
  N:=0;
```

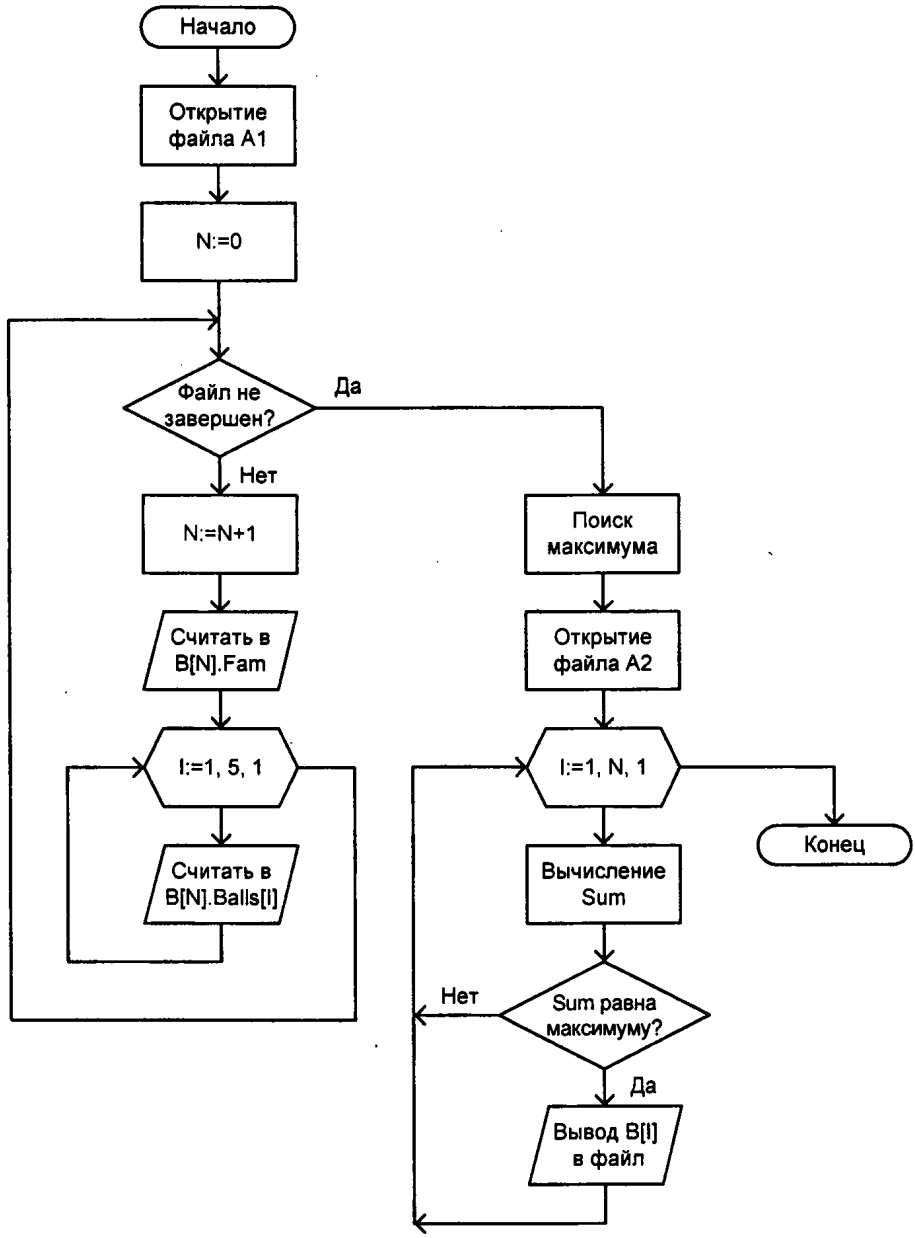


Рис. 9.16. Блок-схема к программе листинга 9.15

```
while not eof(A1) do
 begin
    N:=N+1;
    readln(A1,B[N].Fam);
    for I:=1 to 5 do
       read(A1,B[N].Balls[I]);
    readln(A1);
 end;
```

```
    MaxMass:= 0;
    for I:=1 to N do
      begin
         Sum:=0;
         for J:=1 to 5  do
             Sum:=Sum + B[I].Balls[J];
         if Sum > MaxMass then
             MaxMass:= Sum;
      end;
  assign(A2,'prim15.dat');
  rewrite(A2);
  for I:=1 to N do
    begin
       Sum:=0;
       for J:=1 to 5 do
             Sum:=Sum + B[I].Balls[J];
       if Sum = MaxMass then
           write(A2,B[I]);
    end;
  close(A1);
  close(A2);
end.
```

Таким образом, с помощью программы, представленной в листинге 9.15, мы обеспечили формирование типизированного файла prim15.dat. Теперь очередь следующего этапа, который связан с разработкой другой программы, позволяющей извлечь информацию из файла prim15.dat и представить ее на экране (листинг 9.16).

Листинг 9.16. Вывод информации из типизированного файла на экран

```
program listing_9_16;
Type
  Stud=record
  Fam: string[50];
  Balls: array[1..5] of integer;
end;
var
  A1:file of Stud;
  Z: Stud;
  I:integer;
begin
  assign(A1,'prim15.dat');
  reset(A1);
  while not eof(A1) do
  begin
      read(A1,Z);
```

```
      write(Z.Fam);
      for I:=1 to 5 do
        write(' ',Z.Balls[I]);
      writeln;
    end;
  close(A1);
end.
```

Формирование в файле отчета об олимпиаде

Допустим, что в файле располагается информация об учащихся из различных школ, которые участвовали в олимпиаде.

Формат представления данных сведений в *текстовом* файле такой:

- строка с фамилией учащегося;
- строка из двух целых чисел, одно из которых является номером школы, а второе — номером класса (без буквы).

Далее следуют две строки с аналогичной информацией о другом учащемся и т. д. Для определенности: общее количество участников олимпиады не превосходит 500, а номера школ расположены в интервале от 1 до 99.

Программа должна обеспечить вывод информации в другой *(типизированный)* файл о школе/школах, имеющих наибольшее число участников (таких школ может быть несколько).

Формат записи *типизированного* файла выглядит так:

- целое число с номером школы;
- целое число, соответствующее числу учащихся этой школы на олимпиаде.

После формирования файла необходимо разработать еще одну программу, которая будет считывать данные из созданного типизированного файла и выводить их на экран.

В листинге 9.17 приведена первая программа, которая считывает данные из текстового файла и формирует типизированный файл.

Организация подсчета количества учащихся из различных школ связана с использованием целочисленного массива `Mass` из 99 элементов (по условию номера школ расположены в интервале от 1 до 99). Считывание очередного учащегося приводит к изменению в этом массиве одного из элементов. Например, если очередной учащийся из школы с номером 15, то значение элемента `Mass[15]` увеличивается на 1. В результате после просмотра файла с учащимися мы получим в массиве `Mass` количество представителей от различных школ.

После этого в программе (листинг 9.17) осуществляется нахождение максимума в массиве `Mass`. И в заключительной части выполняется запись в типизированный файл prim17.dat сведений о школе/школах и числе ее представителей.

Листинг 9.17. Формирование отчета в типизированном файле

```
program listing_9_17;
Type
  Dannye =record
  Shkola: integer;
  Cols: integer
end;
var
  A1: text;
  A2: file of Dannye;
  A: Dannye;
  B: string;
  C,D: integer;
  Mass: array[1..99] of integer;
  J,M:integer;
begin
  for J:=1 to 99 do
     Mass[J]:=0;
  assign(A1,'prim17.txt');
  reset(A1);
  while not eof(A1) do
   begin
      readln(A1,B); { Считывание фамилии учащегося }
      readln(A1,C,D); { Считывание номера школы и класса }
      Mass[C]:= Mass[C]+1;
   end;
   M:=0;
   for J:=1 to 99 do
     if Mass[J] > M then
         M:= Mass[J];
   assign(A2,'prim17.dat');
   rewrite(A2);
   for J:=1 to 99 do
     if Mass[J]= M  then
        begin
          A.Shkola:=J;
          A.Cols:=M;
          write(A2,A);
        end;
  close(A1);
  close(A2);
end.
```

Следующий этап связан с разработкой другой программы, которая позволит извлечь информацию из файла и представить ее на экране (листинг 9.18).

Листинг 9.18. Извлечение информации из типизированного файла

```
program listing_9_18;
Type
  Dannye =record
  Shkola: integer;
  Cols: integer
end;
var
  A: file of Dannye;
  Z: Dannye;
 begin
  assign(A,'prim17.dat');
  reset(A);
  while not eof(A) do
    begin
      read(A,Z);
      write(Z.Shkola);
      write(' ',Z.Cols);
      writeln;
   end;
  close(A);
end.
```

Отчет о результатах экзамена

Необходимо разработать программу, которая будет после ввода с клавиатуры результатов экзамена записывать их в типизированный файл. Общее количество учащихся в группе предварительно (перед внесением оценок) вводится с клавиатуры. Информация о сдаче экзамена вносится с клавиатуры построчно: одна строка содержит фамилию с инициалами, а следующая — оценку. Далее в следующих двух строках с клавиатуры вводятся сведения по следующему учащемуся. Формат типизированного файла для записи результатов экзамена такой:

- фамилия учащегося (строка, не более 50 символов);
- оценка (целое число).

Будем считать, что в качестве вариантов оценок рассматриваются такие: 5, 4, 3, 2, 1. Последняя оценка (1) условно означает отсутствие учащегося на экзамене.

Необходимо также разработать еще одну программу, которая после формирования типизированного файла с оценками будет формировать статистику (сколько каких оценок получено по итогам экзамена в группе). Эту статистику необходимо вывести как на экран, так и в типизированный файл.

Формат этого второго типизированного файла такой:

- вариант оценки (целое число);
- процент таких оценок в группе (целое число, при этом дробная часть отбрасывается).

В листинге 9.19 представлена первая программа, которая реализует ввод оценок и запись их в типизированный файл первого указанного типа. Здесь мы создали новый тип данных `Otsenka`, позволяющий хранить информацию об оценке конкретного учащегося. Для записи в файл данных определена переменная `Z` типа `Otsenka`. При внесении информации о конкретном учащемся используются поля записи:

- `Z.Fam` (фамилия учащегося);
- `Z.Ots` (оценка).

Листинг 9.19. Формирование типизированного файла с оценками учащихся

```
program listing_9_19;
Type
   Otsenka=record
   Fam:string[50];
   Ots:integer;
end;
var
  A:file of Otsenka;
  Z: Otsenka;
  I,N: integer;
begin
writeln('Введите число учащихся');
readln(N);
assign(A,'prim19.dat');
rewrite(A);
for I:=1 to N do
     begin
       writeln('Введите фамилию учащегося');
       readln(Z.Fam);
       writeln('Введите его оценку');
       readln(Z.Ots);
       write(A,Z);
     end;
close(A);
end.
```

Следующий этап, как уже говорилось, связан с разработкой другой программы (листинг 9.20), которая позволяет извлечь информацию из файла и сформировать статистику. Эта статистика отображается на экране, а также параллельно сохраняется в типизированном файле prim20.dat.

В приведенной разработке (листинг 9.20) нам понадобится тип данных (Otsenka) для работы с оценками учащихся и тип данных (Stat) для формирования статистики.

Организация подсчета оценок связана с использованием целочисленного массива X из пяти элементов. Считывание очередной оценки приводит к изменению в этом массиве одного из элементов. Например, если оценка очередного учащегося 5, то значение элемента X[5] увеличивается на 1. В результате после просмотра файла с успеваемостью мы получим в массиве X количество различных оценок по результатам экзамена. Далее требуется перевести результат в проценты и отбросить в соответствии с условием дробную часть. Заключительная часть связана с выводом данных на экран и в типизированный файл.

Листинг 9.20. Формирование статистики по проведенному экзамену

```
program listing_9_20;
Type
   Otsenka=record
   Fam:string[50];
   Ots:integer;
end;
Type
   Stat=record
   Ots:integer;
   Protsent:integer;
end;
var
  A1:file of Otsenka;
  A2:file of Stat;
  Z1: Otsenka;
  Z2: Stat;
  X: array[1..5] of integer;
  I,N:integer;
begin
  assign(A1,'prim19.dat');
  reset(A1);
  for I:=1 to 5 do
      X[I]:= 0;
  N:=0;
  while not eof(A1) do
    begin
      read(A1,Z1);
      X[Z1.Ots]:= X[Z1.Ots]+1;
      N:=N+1;
    end;
  close(A1);
  assign(A2,'prim20.dat');
```

```
  rewrite(A2);
  for I:=1 to 5 do
     begin
       Z2.Ots:=I;
       Z2.Protsent:=trunc(100*X[I]/N);
       write(A2,Z2);
       writeln('Оценка ', Z2.Ots,' Процент =', Z2.Protsent  );
     end;
  close(A2);
end.
```

Итак, разработки этой главы продемонстрировали уровень наиболее сложных заданий части *С* билетов Единого государственного экзамена. Примерно половина приведенных примеров взята из билетов ЕГЭ прошлых лет. И если вы разобрались с рассмотренными примерами программ, то можно с уверенностью сказать, что к ЕГЭ вы готовы!

В заключение хотим порекомендовать читателям познакомиться с рядом очень хороших книг по разработке алгоритмов и программированию, список которых приводится далее в этой книге.

ПРИЛОЖЕНИЕ

Описание компакт-диска

На сопроводительном компакт-диске вы найдете все программы на языке Паскаль, которые представлены в книге. Также на компакт-диске присутствуют файлы, упоминаемые в нескольких главах книги.

В табл. П1 приводятся пояснения относительно содержания папок, имеющихся на компакт-диске.

Таблица П1. *Папки сопроводительного компакт-диска*

Название папки	Описание
Glava 1	Тексты примеров программ, описанных в первой главе
Glava 2	Тексты примеров программ, описанных во второй главе
Glava 3	Тексты примеров программ, описанных в третьей главе
Glava 4	Тексты примеров программ, описанных в четвертой главе
Glava 5	Тексты примеров программ, описанных в пятой главе
Glava 6	Тексты примеров программ, описанных в шестой главе
Glava 7	Тексты примеров программ, описанных в седьмой главе
Glava 8	Тексты примеров программ, описанных в восьмой главе
Glava 9	Тексты примеров программ, описанных в девятой главе

Список используемой литературы

1. Кашаев С. М., Шерстнева Л. В. Самостоятельная подготовка к ЕГЭ по информатике. Необходимая теория и достаточная практика. — СПб.: БХВ-Петербург, 2009. — 464 с.: ил.
2. Рапаков Г. Г., Ржеуцкая С. Ю. Turbo Pascal для школьников и студентов. — СПб.: БХВ-Петербург, 2009. — 352 с.: ил.
3. Культин Н. Б. Turbo Pascal в задачах и примерах. — СПб.: БХВ-Петербург, 2008. — 256 с.: ил.
4. Златопольский Д. М. Сборник задач по программированию. — 2-е изд. — СПб.: БХВ-Петербург, 2007. — 240 с.: ил.
5. Златопольский Д. М. Программирование: типовые задачи, алгоритмы, методы. — М.: Бином, 2007. — 233 с.: ил.
6. Златопольский Д. М. Задачи по программированию. 7–11 классы: Книга для учителя (учебно-методическое издание). — М.: Первое сентября, 2001. — 208 с.
7. Сафронов И. К. Готовимся к ЕГЭ. Информатика. — СПб.: БХВ-Петербург, 2009. — 368 с.: ил.

Предметный указатель

М

О

П

Р

С

Т

У

Ф

Ц

Ч

Э