Д. М. Ушаков, Т. А. Юркова

ПАСКАЛЬ
ДЛЯ ШКОЛЬНИКОВ

ПИТЕР®

Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Ростов-на-Дону · Екатеринбург · Самара · Новосибирск
Киев · Харьков · Минск
2010

Денис Михайлович Ушаков,
Татьяна Анатольевна Юркова

Паскаль для школьников

Заведующий редакцией	*А. Кривцов*
Ведущий редактор	*В. Шачин*
Редактор	*А. Вахитов*
Художник	*Л. Адуевская*
Иллюстрации	*М. Шендерова*
Корректоры	*С. Беляева, Н. Лукина*
Верстка	*А. Келле-Пелле*

ББК 32.973-018.1я7
УДК 681.3.06(075)

Ушаков Д. М., Юркова Т. А.

У93 Паскаль для школьников. — СПб.: Питер, 2010. — 256 с.: ил.

ISBN 978-5-469-00492-9

Эта книга — не учебник, а скорее помощник в освоении языка программирования Паскаль, с которым на уроках информатики знакомятся все школьники. Она состоит из бесед, посвященных практическим вопросам программирования и решения задач. Многочисленные примеры позволяют лучше понять, как разработать алгоритм, написать собственную программу, правильно оформить ее текст. Советы и примечания обращают внимание читателей на важные детали, позволяют избежать подводных камней, более эффективно писать программы.

Книга написана школьными преподавателями информатики, имеющими большой опыт многолетней практической работы.

ISBN 978-5-469-00492-9

Подписано к печати 04.12.09. Формат 84×108/32. Усл. п. л. 13.44. Доп. тираж 3500.
Заказ № 2654-4.
Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2;
95 3005 — литература учебная.
ООО «Лидер», 194044, Санкт-Петербург, Б. Сампсониевский пр., дом 29а.
Отпечатано по технологии CtP в ИПК ООО «Ленинградское издательство».
195009, Санкт-Петербург, ул. Арсенальная, д. 21/1.
Телефон/факс: (812) 495-56-10.

Содержание

Вступление

Что такое язык программирования? Любая задача, которую решает компьютер, записывается в виде последовательности команд. Такая последовательность называется программой. Команды, конечно, должны быть представлены на языке, понятном компьютеру. Один из таких языков — язык программирования Паскаль. Он разработан швейцарским профессором Николаусом Виртом специально для обучения студентов программированию. К особенностям языка относится также и его структурность. То есть программа легко разбивается на более простые, непересекающиеся блоки, те, в свою очередь, на еще более простые блоки. Это также облегчает программирование. В 1979 году язык был утвержден в качестве стандартного. Вирт назвал его в честь французского ученого Блеза Паскаля, изобретателя счетной машины. Язык Паскаль прост, логичен и эффективен. Он получил распространение во всем мире. Наши беседы построены на конкретных примерах программ. Длительных теоретических пояснений нет, поэтому крайне необходимо внимательно читать комментарии в текстах программ!

Итак, начинаем первую беседу сразу с первой программы на Паскале.

Желаем удачи!

Благодарности

Авторы выражают благодарность за помощь, оказанную в работе над книгой: ученикам школ 183, 489, 550, 366 за то, что они решали предложенные задачи; сотрудни-

кам кафедры АСУ СПб. ГУАП за то, что рядом с ними было приятно работать; профессору Е. А. Круку и к. т. н. С. В. Федоренко, без которых не возникла бы идея этой книги; руководителю проекта В. Ю. Шачину за массу ценных советов и помощь в доведении проекта до конца.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу электронной почты comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

Все файлы, описываемые в книге, вы сможете найти по адресу http://www.piter.com/download.

Подробную информацию о наших книгах вы найдете на веб-сайте издательства http://www.piter.com.

ТЕМА 1

Как написать простую программу на Паскале

Изучение Паскаля мы начнем сразу с конкретных примеров, которые помогут вам почувствовать процесс программирования вживую. Шаг за шагом мы проведем вас через все стадии программирования. Мы хотим, чтобы вы сразу ощутили уверенность, чтобы вы поняли: «Если я смог написать одну программу, значит, смогу и вторую»!

Урок 1.1. Выводим сообщение на экран дисплея

Нашей первой программой будет программа-приветствие. Она просто выведет текст на экран компьютера и завершит свою работу. В этом уроке мы рассмотрим основные правила оформления программы.

СОВЕТ

Внимательно читайте комментарии, они находятся в фигурных скобках {}.

Пример 1.1. Первая программа

```
program First; { Первая строка – заголовок программы.
                 program – служебное слово;
                 First – имя нашей программы,
                 его вы придумываете сами.
                 В конце строки стоит
                 точка с запятой.
                 При перечислении инструкций Паскаля
                 между ними нужно
                 ставить ";".
                 Дальше идет тело программы.
                 Оно всегда начинается со слова begin }
begin          { Здесь в конце строки нет
                 точки с запятой.
```

```
                    Следующая команда, или оператор,
                    выводит слово ПРИВЕТ на экран;
                    после вывода курсор остается на той же
                    строке в конце текста;
                    текст для вывода всегда заключается
                    в апострофы.}
    write ('Привет, '); { В конце строки обязательна
                    точка с запятой}
                  { Следующий оператор выведет на экран
                    слово ДРУЗЬЯ! и переведет курсор
                    на следующую строку, так как символы
                    "ln" в операторе writeln означают
                    "line" - по-английски "строка" }
    writeln ('друзья!');
    writeln('Это вторая строка') { Здесь в конце строки
                    не обязательна точка с запятой, так как
                    это последний оператор.
                    Проще говоря, перед end точку с запятой
                    можно не ставить }
end.                { Словом end заканчивается
                    тело программы;
                    в конце обязательно стоит точка }
```

ЗАМЕЧАНИЕ

Очень важно понимать, когда нужно ставить точку с запятой, а когда нет. Основное правило — точка с запятой должна ставиться при перечислении инструкций. Так, в рассмотренном только что примере точка с запятой стоит между разделом program *и разделом тела программы, а также в теле программы при перечислении операторов. На последнем операторе* writeln *перечисление заканчивается, поэтому мы не поставили точку с запятой.*

Урок 1.2. Как заложить эту программу в компьютер?

В этом уроке мы рассмотрим и выполним все действия, необходимые для работы в среде Паскаль. Мы условно

разделили их на 7 шагов. Может быть, такой длинный процесс покажется вам утомительным. Ну, что ж поделаешь: тяжело в учении — легко в бою.

Обнадежьте себя тем, что эти 7 шагов приведут вас к первому конкретному результату. Это начало большого пути!

Сначала рассмотрим, какие этапы должен пройти пользователь (программист) для того, чтобы увидеть на экране правильные результаты работы своей программы.

Этапы создания компьютерной программы

- **1 этап.** Редактирование текста программы (команда Edit). На этом этапе пользователь набирает свою программу в символах выбранного им языка программирования.
- **2 этап.** Компиляция программы (команда Compile). В результате программа пользователя переводится из символов языка программирования в двоичный код компьютера. При обнаружении ошибок происходит возврат к 1 этапу.
- **3 этап.** Построение программы (команда Build). Подгружаются библиотечные модули[1], и готовая программа в двоичном коде сохраняется на диске. При обнаружении ошибок происходит возврат к 1-му этапу.
- **4 этап.** Запуск программы на выполнение (команда Run). При обнаружении ошибок происходит возврат к 1 этапу.

Прохождение всех этих этапов заложено в интегрированной среде Паскаль.

---

[1] О том, что такое библиотечные модули, см. замечание далее по тексту.

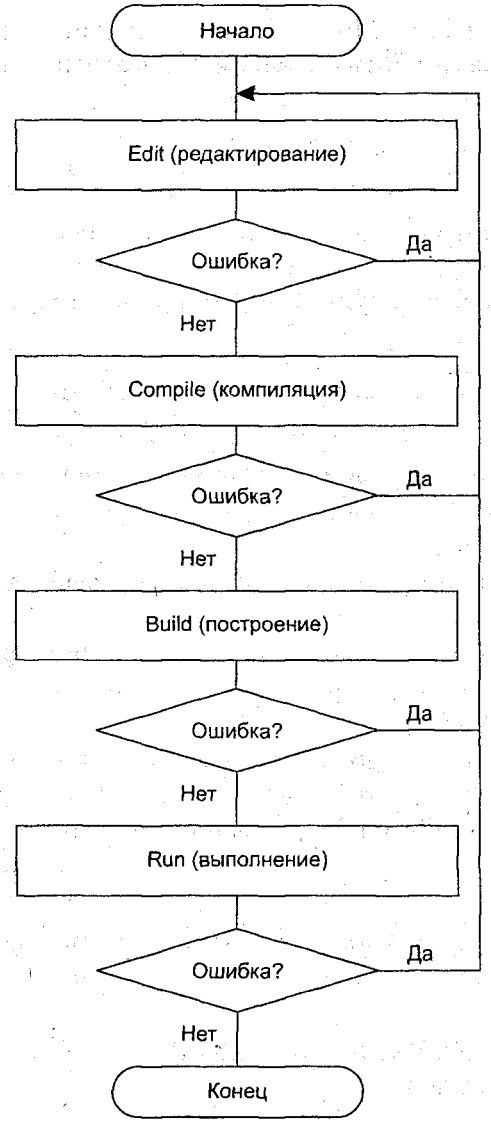


Рис. 1.1. Схема прохождения этапов создания программы на компьютере

ЗАМЕЧАНИЕ

Почти любая компьютерная программа состоит из блоков (модулей), каждый из которых выполняет какое-то одно действие. Программу составляют из этих блоков, как из кирпичиков. В 9-й теме мы научимся создавать такие блоки самостоятельно. Однако даже в самых простых случаях приходится выполнять действия, без которых не обходится почти ни одна программа — например, выводить информацию на экран или вводить что-то с клавиатуры. Так как эти действия нужны всем, для них ввели специальные названия (например, write *и* writeln*) и заложили непосредственно в программу Паскаль. Они-то и называются библиотечными модулями. Их машинные коды хранятся в специальных файлах (библиотеках) и подсоединяются к вашим программам на этапе построения.*

Вам предлагается выполнить последовательно следующие действия.

1. Запуск среды Паскаль

- Щелкните на значке Паскаль, если он существует на вашем Рабочем столе, или найдите на диске файл turbo.exe или bp.exe (исполняемые файлы с программой среды Паскаль) с помощью функции поиска Windows (Пуск ▸ Найти).

ЗАПОМНИТЕ!

Клавиша F10 *вызывает главное меню (можно также щелкнуть мышью).*

Клавиша Esc *осуществляет возврат из любого вызванного режима к предыдущему.*

При успешном запуске интегрированной среды Паскаль вы увидите на экране следующие элементы:

- главное меню (строка сверху);
- окно редактирования (синее поле посередине);
- описания функциональных клавиш (строка внизу).

СОВЕТ

Эти советы касаются пользователей Паскаля для Windows.

Скорее всего, вы запускаете Паскаль из Windows, создав для этого ярлык (например, на Рабочем столе). Однако Паскаль — программа для MS-DOS, поэтому в ходе работы с ней вы столкнетесь с несколькими особенностями.

Во-первых, программы для MS-DOS обычно запускаются в полноэкранном текстовом режиме. Это кажется нам неудобным хотя бы потому, что частота мерцания монитора в этом режиме составляет всего 60 Гц, в то время как для безопасности глаз считается необходимым хотя бы 85 Гц. Обычно графический режим Windows настроен на максимальную частоту обновления. Мы рекомендуем вам перейти после запуска Паскаля в оконный режим работы (комбинация клавиш Alt+Enter*) и развернуть окно во весь экран.*

Во-вторых, переключатель клавиатуры для MS-DOS-программ не имеет никакого отношения к обычному переключателю клавиатуры Windows. Запомните: в Паскале переключение на русскую раскладку клавиатуры производится нажатием правого Ctrl+Shift*, а обратно на латинскую — нажатием левого* Ctrl+Shift*. При этом индикатор клавиатуры на панели задач не меняется — не обращайте на него внимания. Обычно при переключении раскладки для MS-DOS компьютер также издает звуковой сигнал.*

В-третьих, окно программы для MS-DOS крайне не рекомендуется закрывать щелчком на кнопке закрытия окна Windows. Это действие приведет к аварийному принудительному закрытию Паскаля и, скорее всего, к потере несохраненной программы. Закрывать Паскаль нужно средствами Паскаля — с помощью команды Exit *в меню* File *или комбинации клавиш* Alt+X *на клавиатуре.*

2. Работа в окне редактирования Edit

- Перейдите в главное меню (клавиша F10).
- Выберите пункт File с помощью клавиш перемещения курсора (←, →).
- Нажмите клавишу Enter.

Появится раскрывающееся меню File:

New	
Open	F3
Save	F2
Save as	
Save all	
----------	------
Change dir	
Print	
Printer setup	
DOS shell	
Exit	Alt+X

Комбинации клавиш, указанные справа от названия команды, дают возможность сразу войти в этот режим.

- Выберите команду New с помощью клавиш перемещения курсора (↑, ↓).
- Нажмите клавишу Enter.

На экране откроется пустое окно, озаглавленное NONAME00.PAS. Это имя, данное средой по умолчанию вашей будущей программе (точнее, файлу, в котором будет храниться ваша будущая программа). Если вы повторите описанную операцию, то раскроется еще одно окно, но уже с именем NONAME01.PAS. Так можно раскрыть достаточное число окон редактирования. Для переключения окон достаточно, удерживая нажатой клавишу Alt, нажать клавишу с цифрой, представляющей порядковый номер окна. Эту операцию также выполняет клавиша F6.

- Итак, курсор находится в левом верхнем углу окна редактирования. Наберите вашу первую программу, но без комментариев. В конце каждой строки нажимайте Enter. Программа будет выглядеть следующим образом:

Пример 1.2. Первая программа без комментариев

```
program First;
begin
   write('Привет, ');
   writeln('друзья!');
   writeln('Это вторая строка')
end.
```

СОВЕТ

Компилятору языка Паскаль безразлично, какие буквы вы используете при наборе программы: строчные или заглавные. В наших примерах вам предлагается стандартный вариант использования регистра символов в программе. При написании текста программы вы можете свободно вставлять пробелы и перевод строки между независимыми операторами. Но при этом позаботьтесь о том, чтобы вашу программу было легко читать!

При работе в редакторе полезно знать основные комбинации клавиш (табл. 1.1).

Таблица 1.1. Комбинации клавиш редактора Паскаля

Действие	Клавиша
Образование новой строки	Enter
Переход в начало строки	Home
Переход в конец строки	End
Переход в начало файла	Ctrl+Home
Переход в конец файла	Ctrl+End
Включение режима замены символов	Insert (Ins)
Включение режима вставки символов	Insert (Ins)
Удаление текущего символа	Delete (Del)
Удаление предыдущего символа	Backspace (Bs) — правая клавиша в ряду цифровых клавиш
Перемещение курсора по тексту	←, →, ↑, ↓
Страница вверх	Page Up (PgUp)
Страница вниз	Page Dn (PgDn)
Склеивание двух строк	1. Переход в конец 1-й строки 2. Клавиша Delete
Удаление текущей строки	Ctrl+Y

ЗАМЕЧАНИЯ

*Клавишами управления (*PgUp, PgDn, Ins, Del *и т. д.) на вспомогательной цифровой клавиатуре можно пользоваться при выключенном режиме* Num Lock.

В главном меню есть пункт Edit. *Раскрывающееся меню, которое появляется при выборе этого пункта, позволяет выполнять различные операции с текстом.*

Выделить фрагмент текста можно с помощью клавиш управления курсором, держа нажатой клавишу Shift, *а также путем перетаскивания курсора мыши по выделяемому фрагменту.*

С выделенным фрагментом можно работать почти так же, как в Windows, — удалять, вырезать, копировать,

вставлять. В Паскале, как и в Windows, есть буфер обмена, куда можно скопировать выделенный фрагмент, а потом вставить из него в другое место. Причем вставлять из буфера можно несколько раз. Сочетания клавиш для работы с буфером обмена используются не совсем привычные для Windows (табл. 1.2).

Таблица 1.2. Комбинации клавиш для работы с буфером обмена среды Паскаль

Действие	Комбинация клавиш
Удалить выделенный фрагмент	Ctrl+Delete
Вырезать в буфер (Cut)	Shift+Delete
Копировать в буфер (Copy)	Ctrl+Insert
Вставить из буфера (Paste)	Shift+Insert

Задание 1.1. Напишите (в редакторе Паскаля) программу, которая выводит на экран фразу «Всем привет!» 80 раз — в таблице из 20 строк по 4 столбца.

Подсказка. Для задания расстояния между колонками используйте несколько пробелов. Напишите сначала только один оператор write, который выведет одну фразу (не забудьте про пробелы). Затем скопируйте его еще 3 раза, чтобы получить целую строку. В конце не забудьте поставить переход на следующую строку (writeln). После этого скопируйте получившийся текст еще 19 раз. Копировать begin и end не нужно!

3. Сохранение программы в файле на диске

- В главном меню выберите File (клавиши Alt+F).
- В раскрывающемся меню File выберите команду Save as.... (Паскаль откроет окно Save File As.)
- Переключайтесь между элементами окна клавишей Tab (или щелчком мыши).

- В поле ввода Save file as введите имя, под которым собираетесь сохранить файл, например MY.PAS, и нажмите Enter (расширение .pas набирать не обязательно).

ЗАМЕЧАНИЯ

В поле Files *отображаются имена файлов текущего каталога в соответствии с маской, установленной в поле* Save file as. *Когда вы выберете нужный файл в поле* Files *и нажмете* Enter, *то его имя автоматически появится в поле ввода* Save file as. *Кнопка* OK *служит для подтверждения выбранных действий. Кнопка* Cancel *отменяет все действия и закрывает диалоговое окно. Кнопка* Help *выводит окно с подсказкой. В дальнейшем при сохранении файла под текущим именем достаточно нажатия клавиши* F2 *(*Save*).*

Так как Паскаль — программа MS-DOS, то и сохранять файлы рекомендуется под именами MS-DOS. Рекомендуем имя файла составлять не более чем из восьми латинских букв и цифр, без пробелов и знаков препинания.

4. Запуск компилятора

- Выберите меню Compile (Alt+C).
- Выберите в окне режима Compile операцию Compile.
- Нажмите клавишу Enter.

Если компилятор не обнаружил ошибок, то на экране появляется окно с сообщением: Compilation successful: press any key (Компиляция прошла успешно: нажмите любую клавишу).

Окно остается на экране до тех пор, пока вы не нажмете какую-либо клавишу. Если обнаружена ошибка, курсор устанавливается на ошибку в окне редактирования и выдается сообщение об ошибке. Получить подробную информацию о найденной ошибке можно, нажав сразу клавишу F1. Esc — возврат в окно редактирования.

ЗАМЕЧАНИЕ

Справку по языку Паскаль можно получить, если установить курсор с помощью клавиатуры на служебное слово Паскаля и нажать комбинацию клавиш Ctrl+F1 .

СОВЕТ

Быстро запустить процесс компиляции можно нажатием клавиш Alt+F9.

5. Выполнение программы

Объединяем работу построителя (команда Build) и запуск нашей программы на выполнение.

- В главном меню выберите Run (Alt+R).
- В раскрывающемся меню Run выберите команду Run.
- Нажмите клавишу Enter.

ЗАМЕЧАНИЯ

Быстро запустить программу можно нажатием комбинации клавиш Ctrl+F9.

Процессы компиляции и запуска программы на выполнение можно объединить, вызвав команду Run *сразу после набора текста программы.*

6. Просмотр результатов работы программы

- Одновременно нажмите клавиши Alt+F5 для перехода к экрану пользователя.
- Просмотрите результаты работы программы.
- Для возвращения в среду Паскаль нажмите любую клавишу.
- Если результат работы вас не удовлетворяет, вернитесь к редактированию текста программы.

7. Выход из среды Паскаль

❖ Нажмите Alt+X.

Самый простой и быстрый способ выйти из среды Паскаль — нажать комбинацию клавиш Alt+X. Если ваша программа при этом не была сохранена, появится сообщение о том, что файл был изменен, и предложение его сохранить. Другой способ выйти из Паскаля — выбрать в меню File команду Exit. Напоминаем, что щелчок мышью на «крестике» в правом верхнем углу окна Паскаля в системе Windows является не выходом, а аварийным завершением работы. При этом набранная вами программа будет, скорее всего, потеряна.

Задание 1.2. Напишите программу, которая выводит на экран текст:

```
Важно
не путать Write
и Writeln!
```

Урок 1.3. Оформление текста на экране

До сих пор мы выводили текст шрифтом белого цвета на черном экране, начиная с той позиции, где в настоящее время находится курсор. А нельзя ли выводить текст более красиво — например, цветными буквами в центре экрана?

Для реализации такой возможности в комплект Паскаля входит особый дополнительный модуль. Он называется CRT (это английская аббревиатура, обозначающая электронно-лучевую трубку — название модуля подчеркивает, что он умеет управлять способами вывода на экран).

Модуль не входит в стандарт языка, он является расширением возможностей Паскаля для IBM-совмести-

мых компьютеров (на которых мы с вами и работаем). Этот модуль содержит набор программ (процедур), которые позволяют задавать цвет символов, очищать экран, устанавливать курсор в любую позицию экрана и выполнять множество других полезных действий.

Рассмотрим принцип работы с модулем CRT и его основные процедуры.

Как мы уже говорили, Паскаль работает в текстовом режиме. Это означает, что информация на экран выводится в виде символов, каждый из которых отображается на экране в определенной позиции, как бы в клеточке. Экран при этом можно себе представить как таблицу из 25 строк и 80 столбцов (рис. 1.2). Каждая ячейка этой таблицы имеет 2 координаты — x и y, где x — номер столбца, y — номер строки. Строки нумеруются сверху вниз, начиная с единицы до 25, столбцы — слева направо, с 1-го до 80-го. То есть левый верхний угол экрана имеет координаты (1,1), правый верхний — (80,1), а левый нижний — (1,25). Символы можно выводить на экран 16 различными цветами, которые кодируются числами от 0 до 15. Каждому коду соответствует свой цвет. Полную таблицу кодов можно получить, набрав, например, слово red в окне редактора Паскаля и нажав Ctrl-F1 на клавиатуре.

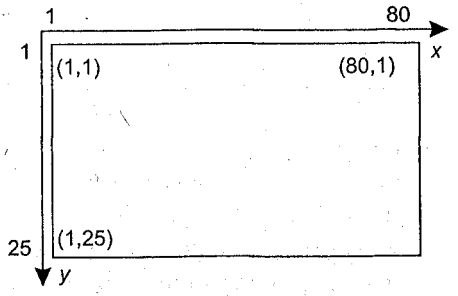


Рис. 1.2. Схема нумерации позиций экрана для модуля CRT

Внимательно разберите следующую программу:

Пример 1.3. Использование модуля CRT

```
program Second;
{ Использование возможностей модуля Crt при выводе
  на экран. Для того чтобы в программе можно было
  использовать дополнительные библиотечные функции
  (нам сейчас нужен блок функций CRT), необходимо
  в начале программы указать это в специальной секции
  объявления библиотечных модулей.
  Она начинается словом uses.
  Затем через запятую перечисляются подключаемые модули.
  Заканчивается список символом ";"}
uses Crt;                { Crt - имя подключаемого модуля.
                           Блок заканчивается символом ";" }
{ Начало основной программы }
begin
   TextBackGround (3); { Вызов процедуры для выбора
                         фонового цвета.
                         В скобках указан номер
                         выбранного цвета.
                         В данном случае "3"
                         означает светло-голубой
                         цвет.
                         Вместо номера можно написать
                         название цвета: black, red,
                         green, blue, magenta, cyan, ... }
   ClrScr;           { Процедура очистки экрана.
                       Указав цвет фона до команды
                       ClrScr, мы тем самым залили
                       экран светло-голубым цветом }

   TextColor (14);   { Процедура выбора цвета
                       выдаваемых символов.
                       В скобках указан номер
                       выбранного цвета.
                       В данном случае желтый цвет.
                       Обратите внимание:
                       команда TextColor не меняет
                       цвет символов, уже имеющихся
                       на экране! Она лишь
                       устанавливает цвет, которым
                       будут выведены следующие
                       символы }
```

```
  GoToXY (40,10);      { Процедура установки
                         курсора в точку экрана
                         с координатами X=40, Y=10 }

  Writeln(' Все отлично!!!');
                       { Вывод текста в 10 строку,
                         начиная с позиции 40 }

  Delay (1000)         { Процедура временной задержки
                         на 1000 мкс.
                         На современных компьютерах
                         Delay(1) обычно работает
                         быстрее, чем 1/1000 секунды.
                         Поэтому задержка в данном
                         случае будет меньше секунды }
end.                   { Конец программы }
```

Задание 1.3. Написать программу, выводящую два любых сообщения в левом верхнем и правом нижнем углах экрана. Каждое сообщение выводить своим цветом.

ЗАМЕЧАНИЕ

Прежде чем писать программу на языке программирования, стоит описать задачу словами пошагово — то есть придумать алгоритм задачи. Алгоритм можно представить в виде блок-схемы (рис. 1.3). Основные блоки, чаще всего используемые в таких схемах, см. в приложении 1.

Алгоритм, представленный на рисунке, называется *линейным*, так как все его шаги проходятся обязательно и последовательно один за другим.

Алгоритм не зависит от языка, на котором вы программируете. Хотя в дальнейших задачах при *детализации* отдельных шагов мы будем учитывать возможности языка.

Сейчас ваша задача — представить каждый шаг (блок) алгоритма на языке Паскаль и оформить программу по образцу примера 1.3.

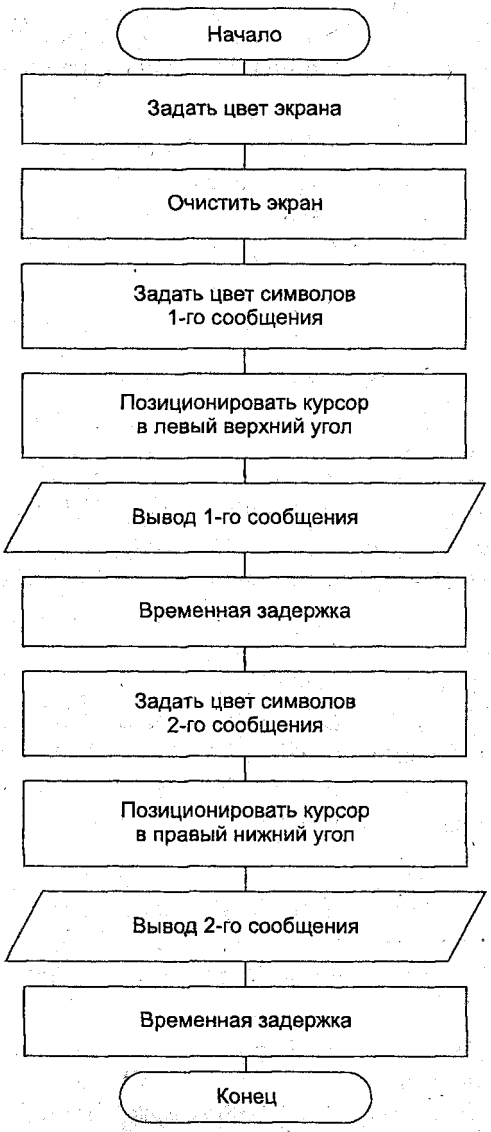
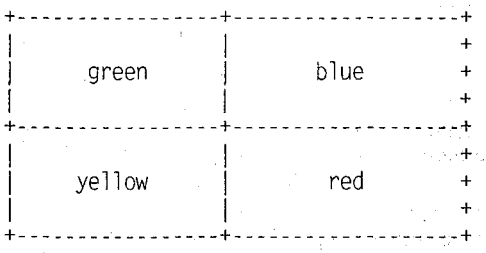


Рис. 1.3. Алгоритм вывода цветных сообщений в левом верхнем и правом нижнем углах экрана

Задание 1.4. Написать программу, которая очищает экран и выводит слова red, green, blue, yellow каждое своим цветом в центр четвертей экрана (если экран условно разбить на 4 части, как показано на рисунке):



Задание 1.5. Левый столбец таблицы содержит действия, которые выполняет некоторый оператор. Правый столбец содержит операторы языка Паскаль. Поставьте в соответствие элементам из левого столбца таблицы элементы из правого столбца.

1. Очистка экрана	A. Crt;
2. Позиционирование курсора в левый нижний угол экрана	B. TextBackGround(red); ClrScr;
3. Заказ красного цвета фона экрана	C. Write('Happy New Year');
4. Заливка экрана красным цветом	D. GoToXY(78,1);
5. Вывод в текущую позицию экрана 'Happy New Year' с переходом курсора на новую строку	E. TextColor(red);
6. Позиционирование курсора в правый верхний угол экрана	F. TextColor(12); Write('Hello');
7. Установка красного цвета текста	G. GoToXY(1,23);
8. Вывод в текущую позицию экрана 'Happy New Year' без перехода курсора на новую строку	H. Writeln('Happy New Year');
9. Библиотека среды Паскаль для работы в текстовом режиме	I. Begin end
10. Начало и конец тела программы	J. TextBackGround(red);
11. Вывод в центр экрана 'Hello'	K. ClrScr;
12. Вывод текста 'Hello' цветом № 12	L. GotoXY(35,12); Write('Hello');

Ответ: 1–K, 2–G, 3–J, 4–B, 5–H, 6–D, 7–E, 8–C, 9–A, 10–I, 11–L, 12–F.

Выводы

1. Любую задачу можно представить в виде последовательности шагов — алгоритма. Одна из форм записи алгоритма — блок-схема, которая в дальнейшем переводится на конкретный язык программирования.
2. В структуре программы на языке программирования Паскаль обязательно присутствует тело программы. Его формируют операторы Begin и End. Между Begin и End с помощью других операторов задаются определенные действия.
3. Вывод информации на экран осуществляют операторы write и writeln.
4. При выполнении некоторых действий в Turbo-среде используются библиотечные модули языка Паскаль. Имена этих модулей объявляются в разделе uses.
5. Для красивого вывода на экран используется модуль Crt. Он позволяет очищать экран (ClrScr), менять позицию курсора (GotoXY), а также цвет символов (TextColor) и фона (TextBackGround).

Контрольные вопросы

1. Какими словами начинается и заканчивается тело любой программы на Паскале?
2. Из каких этапов состоит процесс создания компьютерной программы?
3. Как запустить Паскаль? Как выйти из среды Паскаль? Как запустить программу в среде Паскаль?

4. Какие действия совершают операторы write и writeln? В чем состоит разница между ними?
5. Какие действия нужно совершить, чтобы скопировать часть программы в другое место? Какие комбинации клавиш при этом нужно использовать?
6. Как просмотреть результат работы программы?
7. Что такое текстовый режим работы? Опишите правило задания координат определенной точки экрана.
8. Какой модуль позволяет выводить информацию на экран красиво и в цвете? Как установить курсор в нужную позицию экрана?
9. Что такое линейный алгоритм?

ТЕМА 2

Как включить в работу числовые данные

Мы рассмотрели простейшие действия — вывод на экран информации, причем всегда одной и той же. Однако компьютеры были придуманы для автоматизации сложных вычислений, для быстрого выполнения математических операций (иначе говоря — для работы с числами). И до сих пор эта их функция остается главной: ведь вся компьютерная информация хранится в форме чисел. Цель нашей второй темы — научить вас работать с различными видами чисел.

Урок 2.1. Начнем с простого: целые числа

Любые данные, с которыми вы работаете, необходимо где-то хранить. Все данные, с которыми работает программа, должны находиться в основной памяти. Основная память состоит из ячеек (байтов), каждая из которых имеет *адрес*, то есть порядковый номер (рис. 2.1). В этих ячейках мы и будем хранить данные.

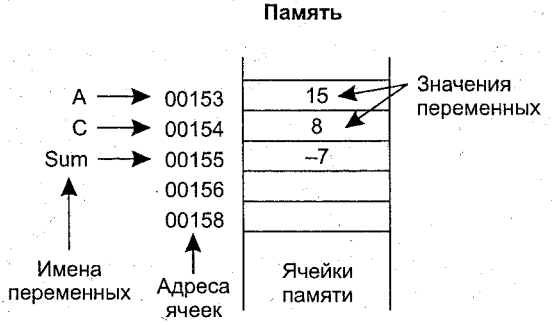


Рис. 2.1. Хранение переменных в памяти компьютера

Понятие переменной

Данные нашей программы принято называть *величинами*. Величины, которые меняются, называют *переменными*, а те, которые не меняются — *постоянными*.

Величину (число), хранящуюся в ячейке, называют *значением ячейки*. Программа работает с адресами и значениями ячеек памяти. Но нам, людям, неудобно работать с адресами — это большие числа и они для нас ничего не значат. Поэтому ячейкам, с которыми будет работать программа, принято давать *имя*, или, что то же самое, *идентификатор*. В специальной таблице программа-компилятор будет запоминать, какому имени какой адрес ячейки памяти соответствует. Итак, мы будем иметь дело только с именами ячеек и с их значениями.

ЗАПОМНИТЕ!

Идентификатор (имя) всегда должен начинаться с латинской буквы, после которой может следовать некоторое число латинских букв, цифр или символов подчеркивания (_). В имени не должно быть пробелов, запятых или других непредусмотренных знаков. В Паскале 7.1 учитываются лишь первые 63 символа.

Тип Integer. Оператор присваивания. Вывод на экран

Теперь рассмотрим работу с самыми простыми переменными. В них можно хранить только целые числа. Для хранения целых чисел в Паскале используется специальный тип данных — Integer.

Внимательно читайте комментарии к программе в следующем примере!

Пример 2.1. Работа с целочисленными переменными

```
Program Product;
{ Далее идет раздел описания переменных. Он всегда
```

```
  начинается со слова var (от variable – переменная) }
var

   A,B,C: integer;    { Имена в списке – через запятую;
                        в конце списка через двоеточие
                        указывается тип данных:
                        integer – целый }

Begin                 { Началось тело программы }

   A:=5;              { Это оператор присваивания.
                        В данном случае запись означает,
                        что в переменную (ячейку) A
                        записали число 5.
                        Не путайте с записью A=5 !!! }

   writeln(A);        { Выводим на экран содержимое
                        переменной A.
                        Имя A не заключено в апострофы! }

   writeln('A');      { Вывод на экран символа A }

   A:=A+1;            { Запишем в переменную A число,
                        которое до этого в ней было,
                        но увеличенное на 1 }
   B:=7;

   C:=A*B;            { * – это операция умножения }

   writeln('Product=',C)   { Вывод содержимого ячейки C
                             с пояснительным текстом }

end.                  { Здесь кончается тело программы }
```

При запуске программа выведет на экран следующее:

```
5
A
Product=42
```

ЗАПОМНИТЕ!

В результате выполнения оператора присваивания в ячейку помещается новое число. Старое содержимое ячейки при этом пропадает.

Справа от оператора присваивания может стоять число или любое выражение. Слева может стоять только имя переменной. Выражения слева быть не может — иначе Паскаль не будет знать, в какую ячейку памяти поместить результат.

Тип результата выражения справа от оператора присваивания должен быть таким, чтобы помещаться в переменную слева от «:=».

Операции с типом Integer

Рассмотрим операции, которые можно выполнять с целыми числами и целочисленными переменными.

Пример 2.2. Операции с переменными целого типа

```
Program Action;
var
   A,B,C: integer;
begin
   A:=17;
   B:=3;

   { Операция умножения: }
   C:=A*B;      writeln('17 * 3=',C);

   { Деление нацело: }
   C:=A div B; writeln('17 div 3=',C);

   { Вычисление остатка от деления: }
   C:=A mod B; writeln('17 mod 3=',C);

   { Сложение: }
   C:=A+B;      writeln('17 + 3=',C);

   { Вычитание: }
   C:=A-B;      writeln('17 - 3=',C)
end.
```

При запуске программа выведет на экран следующее:

```
17 * 3=51
17 div 3=5
17 mod 3=2
```

```
17 + 3=20
17 – 3=14
```

Рассмотрим еще несколько примеров операций div и mod. Для успешного понимания результатов этих операций нужно вспомнить 2-й класс и деление столбиком (рис. 2.2).

```
– 23 | 5      23 div 5 = 4
  20 | 4      23 mod 5 = 3
  3
```

Рис. 2.2. Пример целочисленного деления столбиком

Частая ошибка: не забудьте, что все действия мы производим только с целыми числами! Не нужно продолжать деление, когда делимое (это то, что мы делим) оказывается меньше делителя (это то, на что мы делим). То, что осталось от делимого, называется *остатком*. Это и есть результат операции mod. Целое число, которое получилось в результате деления, называется *целочисленным частным*. Это результат операции div.

Проверим себя, вспомнив 2-й класс:

```
5 div 2=2; 5 mod 2=1;
6 div 2=3; 6 mod 2=0;
40 div 6=6; 40 mod 6=4;
3 div 5=0; 3 mod 5=3.
```

Результат вычисления операций div и mod для отрицательных чисел оказывается не совсем таким, как положено в математике (когда остаток всегда неотрицателен). Зато он более понятен. Другими словами, результат нужно посчитать отдельно от знаков, а потом добавить знак в соответствии с правилами математики:

```
(-10) div 3=-3; (-10) mod 3=-1;
(-3) div 5=0; (-3) mod 5=-3.
```

Задание 2.1. Даны 3 целых числа — *A*, *B*, *C*. Вычислить их сумму и произведение.

Продумаем алгоритм решения данной задачи (рис. 2.3).

Представьте каждый шаг алгоритма на языке Паскаль.

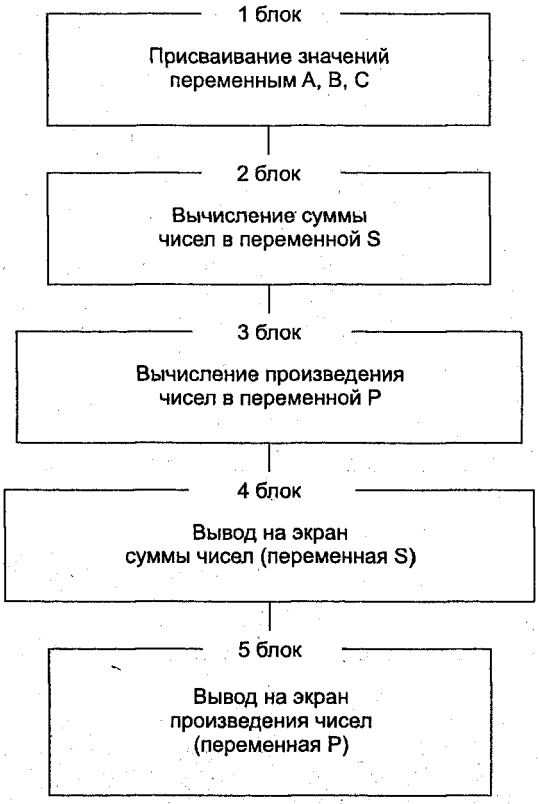


Рис. 2.3. Блок-схема алгоритма решения задания 2.1

Задание 2.2. Дана длина ребра куба (целое число). Найти объем куба и площадь его боковой поверхности.

Стандартные функции типа Integer

Многие стандартные действия с числовыми данными выполняются путем вызова функций из библиотеки

Паскаля. Такие функции называются *стандартными функциями.*

Пример 2.3. Демонстрация стандартных функций

```
Program Infunct;
var
   A,B,C: integer;
begin
   A:=-2;

   { Функция Abs (X) вычисляет абсолютное значение
     аргумента X, то есть модуль X }
   B:=Abs(A); writeln('Abs(-2)=',B);

   { Функция Sqr (X) возводит в квадрат аргумент X }
   C:=Sqr(B); writeln('Sqr(2)=',C);
   C:=Sqr(B+B); writeln('Sqr(2+2)=',C)

end.
```

При запуске программы вывод на экран:

```
Abs(-2)=2
Sqr(2)=4
Sqr(2+2)=16
```

Задание 2.3. Вычислите значение следующего выражения: $|39 \cdot 54 - 84^2|$.

Задание 2.4. В переменные A и B записаны целые числа (оператором присваивания, например, A:=20; B:=13). Поменяйте числа в этих переменных местами.

Будьте внимательны! Если записать A:=B, вы потеряете число 20 и получите в двух переменных число 13! Воспользуйтесь третьей переменной — C (рис. 2.4).

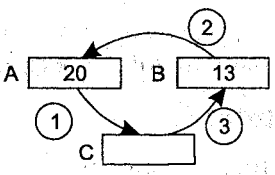


Рис. 2.4. Схема обмена значений двух переменных через третью ячейку. В кружках указан порядок операторов присваивания

Задание 2.5. Выполните задание 4 без использования третьей переменной. Используйте действия сложения и вычитания.

Как представляются переменные целого типа в памяти компьютера

Вся информация в компьютере хранится в виде последовательностей нулей и единиц. Информация, для записи которой используется всего два знака: 0 и 1, называется *двоичной*. Информация в компьютере хранится в виде двоичных кодов (комбинации из нулей и единиц). Память мы представляем, как последовательность ячеек, каждая из которых имеет свой адрес (см. рис. 2.1).

Стандартная длина ячейки — 8 бит, что равно 1 байт. В такую ячейку можно записать двоичный код длиной 8 бит.

Для переменной типа integer выделяется ячейка длиной в 2 байт = 16 бит. Такая ячейка получает символьное имя — имя переменной, и вы обращаетесь к ней не по адресу, а по имени.

Крайняя левая позиция выделяется для знака числа:

- 0 — число положительно;
- 1 — число отрицательно.

Остальные 15 позиций выделяются для записи самого числа в двоичном виде (рис. 2.5).

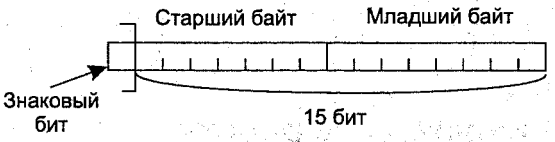


Рис. 2.5. Распределение двоичных разрядов (бит) при хранении числа типа integer

На 15 позициях можно получить 2^{15} двоичных кодов. Самое маленькое число состоит из 15 нулей, самое

большое — из 15 единиц. Поскольку счет начинается с нуля, получаем всего $(2^{15} - 1)$ положительных двоичных чисел. С учетом знака числа (+ или –) получаем, что числа типа integer имеют диапазон представления $-2^{15}...+2^{15} - 1$, или –32 768 ... +32 767.

Тип integer является основой для нескольких производных типов — со знаком и без знака (табл. 2.1).

Таблица 2.1. Целочисленные типы данных языка Паскаль

Наличие знака	Тип переменной	Формат (длина) в байтах	Диапазон	
			Запись с порядком	Обычная запись
Без знака	byte	1	$0 ... 2^8 - 1$	0 ... 255
	word	2	$0 ... 2^{16} - 1$	0 ... 65 535
Со знаком	shortint	1	$-2^7 ... 2^7 - 1$	–128 ... 127
	integer	2	$-2^{15} ... 2^{15} - 1$	–32 768 ... 32 767
	longint	4	$-2^{31} ... 2^{31} - 1$	–2 147 483 648 ... 2 147 483 647

Задание 2.6. Считая, что операция умножения и операция возведения в квадрат имеют одинаковую сложность, запишите оптимальным образом выражения:

а) x^5;

б) x^6;

в) x^8;

г) x^9;

д) x^{10}.

Урок 2.2. Включаем в работу вещественные числа

Что делать, если число не целое, то есть имеет десятичную точку?

Описание вещественного типа данных (Real)

Опишем в программе переменные для хранения не целых чисел. Такие переменные имеют тип real — вещественный тип.

Пример 2.4. Работа с типом real

```
Program Optel;
var
   A,B,C: real;
begin
   A:=3.5;
   B:=7.6;
   C:=A+B;
   writeln('Сумма=',C)
end.
```

При запуске программы вывод на экран:

```
Сумма=1.1100000000E+01
```

Форматы записи вещественных переменных

В примере 2.4 переменная вещественного типа будет выдана на экран в особой, *экспоненциальной* форме. Числа с десятичной точкой могут записываться в двух формах:

1. Обычная форма.

   Примеры:

   - 0,7 может быть записано как 0.7 или .7;
   - −2,1 может быть записано как −2.1.

2. Запись с экспонентой: число представляется в виде мантиссы, то есть дробной части числа, умноженной на 10 в некоторой степени.

   Примеры:

   - $2700 = 2{,}7 \cdot 10^3$. Число 10 записывается в виде буквы E, а за ней идет величина степени: 2.7E3;
   - $0{,}002 = 2 \cdot 10^{-3}$ соответствует запись 2E-3.

Вещественные операции

Пример 2.5. Операции с переменными вещественного типа

```
program Operation;
var
   A,B,C: real;
begin
   A:=17.3;
   B:=3.4;
   C:=A*B; writeln('A*B=',C);

   { / – это операция деления }
   C:=A/B; writeln('A/B=',C);

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

При запуске программа выведет на экран следующее:

```
A*B= 5.8820000000E+01
A/B= 5.0882352941E+00
A+B= 2.0700000000E+01
A–B= 1.3900000000E+01
```

ЗАМЕЧАНИЕ

Для вещественных чисел нет таких проблем с операцией деления, как для целых чисел. Операция «/» – это обычное деление.

Задание 2.7. Вычислите выражение:

$$\frac{7.478937 - 89.2456}{883.5995 + 618.332} \cdot 76.2833.$$

Стандартные функции типа Real

Пример 2.6. Стандартные функции с вещественными переменными

```
Program AllFunc;
var
   A,B: real;
```

```
begin
   A:=2.0;
   B:=Sqr(A); writeln('Sqr(2.0)=',B);
   B:=Abs(-A); writeln('Abs(-2.0)=',B);

   B:=Sqrt (A);       { Вычисление квадратного корня }
   writeln('Sqrt(2)=',B);

   B:=Sin (A);        { Вычисление синуса }

   { Зададим вывод вещественного числа
     не в экспоненциальной, а в обычной форме. На экране
     под значение переменной "B" закажем 6 позиций. Из них
     3 позиции выделим для цифр справа от десятичной точки }
   writeln('Sin(2)=',B:6:3);

   B:=Cos (A);        { Вычисление косинуса }
   writeln('Cos(2)=',B:6:3);

   B:=Arctan (A);     { Вычисление арктангенса }
   writeln('Arctan(2)=',B);

   B:=Ln (A);         { Вычисление логарифма }
   writeln('Ln(2)=',B);

   B:=Exp (A);        { Возведение числа e в степень A }
   writeln('Exp(2)=',B);

   B:=Pi;             { Вычисление числа Пи }
   writeln('Pi=',B)
end.
```

При запуске программа выведет на экран следующее:

```
Sqr(2.0)= 4.0000000000E+00
Abs(-2.0)= 2.0000000000E+00
Sqrt(2)= 1.4142135624E+00
Sin(2)= 0.909
Cos(2)=-0.416
Arctan(2)= 1.1071487178E+00
Ln(2)= 6.9314718056E-01
Exp(2)= 7.3890560989E+00
Pi= 3.1415926536E+00
```

ЗАМЕЧАНИЕ

Вы, вероятно, заметили, что Паскаль содержит мало функций. Он не умеет вычислять даже те функции, которые вычисляет обычный инженерный калькулятор! Что же это за язык программирования, скажете вы! Ответ на это прост: Паскаль разрабатывался не для вычислений (как, например, Фортран), а для обучения.

Запись математических выражений

Имеющихся в Паскале функций достаточно для вычисления других, более сложных. Вот несколько примеров:

$$\operatorname{tg} x = \frac{\sin x}{\cos x},\ \operatorname{ctg} x = \frac{\cos x}{\sin x},\ \log_y x = \frac{\ln x}{\ln y},\ x^y = e^{y \ln x}.$$

Например, чтобы вычислить $(2x+3)^{1+\cos x}$, мы напишем на Паскале:

```
exp( (1+cos(x)) * ln(2*x+3) )
```

Обратите внимание на то, что при записи выражений на языке Паскаль нужно тщательно задумываться о приоритетах операций. Например, выражение $\frac{x+1}{2x}$, записанное в виде x+1/2x, содержит сразу три ошибки. Во-первых, приоритет операции деления выше, чем у сложения, поэтому для правильного вычисления числителя его надо взять в скобки: (x+1). Во-вторых, Паскаль не понимает, что означает 2x. Это мы привыкли, что в математике операцию умножения в таких случаях опускают. Паскалю требуется, чтобы она была указана явно: (x+1)/2*x. Но даже это выражение все еще содержит ошибку. Дело в том, что умножение и деление имеют одинаковый приоритет и выполняются слева направо. Значит, при такой записи сначала выполнится деление, а потом результат будет умножен на x. Нужно либо поставить знамена-

тель в скобки и написать (x+1)/(2*x), либо, для ленивых, поставить вместо умножения деление: (x+1)/2/x. Порядок вычисления в этом случае будет не такой, как требует условие, однако результат будет таким же: ведь поделить на $2x$ — это все равно, что поделить на 2, а потом результат — на x!

ЗАПОМНИТЕ!

Аргументы функции всегда пишутся в скобках. То есть если у функции нет аргументов (как у Pi*, например), то скобки после ее имени не нужны. Если же аргументы есть, то после имени функции вы должны обязательно открыть скобку, перечислить аргументы и не забыть закрыть скобку. Например,* sin 2x *в Паскале нужно записывать как* sin(2*x).

Задание 2.8. Напишите программу для вычисления дискриминанта квадратного уравнения. Коэффициенты задайте в программе через оператор присваивания.

Продумаем алгоритм решения данной задачи (рис. 2.6).

Запишите каждый шаг алгоритма на языке Паскаль.

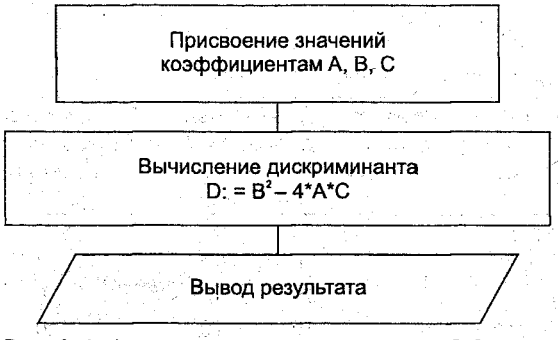


Рис. 2.6. Алгоритм выполнения задания 2.8

Задание 2.9. Вычислите выражение:

$$\sqrt{\operatorname{arctg}^2\left(\sin\frac{3.15}{6.1}\right)+53.7}.$$

Как представляются переменные вещественного типа в памяти компьютера

Вы познакомились с экспоненциальной формой представления числа. В ней можно выделить две части: мантиссу, то есть значащие цифры числа, и порядок — степень десятки (в общем случае это степень основания системы счисления, в которой записано данное число). Ячейка памяти, выделенная для переменной вещественного типа, должна содержать следующие элементы: знак числа, знак порядка, значение порядка и значение мантиссы — естественно, все в двоичном представлении (рис. 2.7).

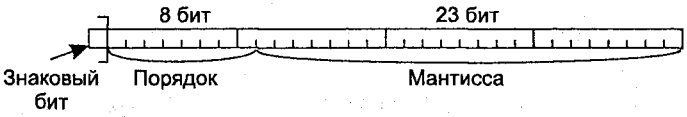


Рис. 2.7. Распределение двоичных разрядов (бит) при хранении числа типа real

Порядок и знак порядка занимают вместе 8 бит и хранятся, вообще говоря, немного хитрее. Но это нам сейчас не важно. Главное — понять принцип хранения чисел с плавающей точкой.

Таблица 2.2. Вещественные типы данных языка Паскаль

Тип переменной	Формат (длина) в байтах	Примерный диапазон абсолютных значений	Количество значащих десятичных цифр
Одинарный (single)	4	$10^{-45} \ldots 10^{38}$	7 или 8
Вещественный (real)	6	$10^{-39} \ldots 10^{38}$	11 или 12
Двойной (double)	8	$10^{-324} \ldots 10^{308}$	15 или 16
Расширенный (extended)	10	$10^{-4932} \ldots 10^{4932}$	19 или 20

Урок 2.3. Как совместить переменные целого и вещественного типа

В программе могут одновременно встречаться переменные разных типов. Как их совместить?

Преобразование типов

Пример 2.7. Одновременное использование вещественных и целых типов

```
Program Mix;
var
   N,K: integer;
   A,B: real;
begin
   N:=4;
   A:=3.6;

   B:=N;              { В переменную типа real можно
                        записать целое число }

   writeln('B=',B);

   { В переменную типа integer нельзя записать
     вещественное число! Чтобы все-таки поместить число
     типа real в переменную типа integer, нужно явно
     указать, что делать с дробной частью числа. Есть
     два варианта: }

   N:=Trunc(A);       { Функция Trunc(X) возвращает
                        целую часть числа X,
                        то есть отбрасывает дробную часть }
   writeln('Trunc(3.6)=',N);

   K:=Round(A);       { Функция Round(X) округляет
                        до ближайшего целого }

   writeln('Round(3.6)=',K)
end.
```

При запуске программа выведет на экран следующее:

```
B:= 4.0000000000E+00
Trunc(3.6)=3
Round(3.6)=4
```

Еще раз уточним правила преобразования типов: для хранения данных типа integer используется 2 байт, а для real необходимо 6 байт. Это значит, что число типа integer можно поместить в ячейку типа real (целая часть будет равна этому числу, а дробная будет равна нулю). А вот число типа real в ячейку типа integer никак не поместится. Чтобы все-таки поместить его туда, нужно явно указать, что делать с дробной частью числа. Для этого предусмотрены функции trunc и round. Обе они возвращают результат типа integer.

Что делать, если в программе нужно записать сложное математическое выражение? В каком порядке будут выполняться действия?

Правила приоритета в выполняемых действиях

1. Действия над переменными, стоящими в скобках, выполняются в первую очередь.
2. После вычисления значений всех скобок вычисляются все функции.
3. После функций выполняются умножение и деление. Они имеют одинаковый приоритет.
4. Следующие по приоритету — сложение и вычитание.
5. Операции одинакового приоритета выполняются слева направо.

Действия над данными разных типов

Сведем воедино операции и функции по работе с вещественными и целыми величинами (табл. 2.3).

Поясним написанное в таблице. Мы разделили все функции/операции на 6 категорий:

1. Результат операций +, – и * зависит от типа аргументов. Если хоть один из них имеет тип Real, то и результат будет иметь тип Real. Это объясняется тем, что у данных типа Real есть дробная часть, а у Integer —

нет. Даже если в вещественной переменной хранится целое число, оно все равно имеет дробную часть, только она равна нулю. То есть если хотя бы у одного из аргументов есть дробная часть, то в результате выполнения операции она никуда не исчезает. Поэтому результат тоже имеет дробную часть (Real).

Таблица 2.3. Операции и функции для типов integer и real

Операция/ функция	Тип данных 1-го аргумента	Тип данных 2-го аргумента	Тип данных результата
+,-,*	Integer	Integer	Integer
	Integer	Real	Real
	Real	Integer	
	Real	Real	
/	Не важен		Real
Div, mod	Только Integer		Integer
Abs, Sqr	Integer	—	Integer
	Real	—	Real
Sqrt, Sin, Cos, Arctan, Ln, Exp, Pi	Не важен	—	Real
Trunc, Round	Не важен	—	Integer

2. Результат операции вещественного деления по определению всегда имеет дробную часть.
3. Операции целочисленного деления определены только для целых чисел. Поэтому результат тоже всегда целое число.
4. Функции Abs и Sqr определены для обоих типов данных. Поэтому тип их результата зависит от типа аргумента. Для целого аргумента результат имеет целый тип, для вещественного — вещественный.
5. Функции Sqrt, Sin, Cos, Arctan, Ln, Exp, Pi по определению являются вещественными. (На самом деле это связано с особенностями вычисления Паскалем этих функций. Он вычисляет их приближенно путем разло-

жения в ряд. Такой метод не предполагает целого результата в виде целого числа. Более того, значения этих функций всегда вычисляются приближенно.)

6. Функции Trunc и Round предназначены для преобразования типов. Они явно указывают на то, что сделать с дробной частью числа. Поэтому это единственный способ получить на Паскале из дробного числа целое.

Изложенные выше сведения позволяют нам понимать, что за выражение написано в чужой программе, какое оно будет иметь значение и какого типа будет результат.

Задание 2.10. Вычислите выражение и укажите тип результата:

```
Abs(12 mod 7*4/2-350 div 15)+2
```

Сначала расставим операции в соответствии с их приоритетами (табл. 2.4).

Таблица 2.4. Расстановка операций в примере 2.8

№	Операция	Пояснение
1	12 mod 7	Сначала выполняются действия в скобках. Скобки в данном случае ограничивают аргумент функции Abs. В скобках сначала выполняются операции типа «умножение-деление», а потом — «сложение-вычитание». Операции mod и div осуществляют целочисленное деление. Их приоритет такой же, как у операций «*» и «/». В первой группе таких операций три. Выполняем их слева направо. Поэтому сначала выполняем mod, затем — «*», а потом — «/»
2	(12 mod 7) * 4	
3	(12 mod 7*4) / 2	
4	350 div 15	Теперь выполняем вторую группу операций типа «умножение-деление». В данном случае это одинокая операция div

продолжение

Таблица 2.4 (*продолжение*)

№ п.п	Операция	Пояснение
5	(12 mod 7*4/2) – – (350 div 15)	Теперь пришла пора объединить результаты первых двух групп операцией «–». Она выполняется последней в скобках, так как имеет наименьший приоритет
6	Abs(...)	Теперь пора посчитать результат функции
7	Abs(...) + 2	Последний оператор — сложение

Нам представляется удобным расставлять приоритеты, обводя операции (рис. 2.8). Так оказывается легче понять, что уже вычислено и что еще предстоит вычислить.

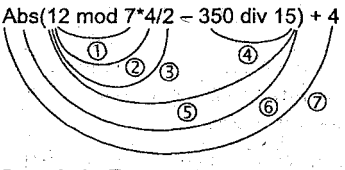


Рис. 2.8. Порядок выполнения операций в задании 2.10

Теперь определим тип и результаты каждого действия (табл. 2.5).

Таблица 2.5. Вычисление результатов каждого действия из задания 2.10

№	Операция	Результат и его тип	Пояснение
1	12 mod 7	5 Integer	Остаток от деления 12 на 7 равен 5. Результат операции mod всегда целый
2	5 * 4	20 Integer	Результат умножения двух целых — целое число
3	20 / 2	10.0 Real	Результат вещественного деления всегда вещественный. Хотя 20 делится нацело на 2, мы специально приписали «.0» к результату, чтобы подчеркнуть и не забыть, что результат вещественный и имеет дробную часть

№	Операция	Результат и его тип	Пояснение
4	350 div 15	23 Integer	При целочисленном делении нас интересует только целая часть частного. Результат операции div всегда целый
5	10.0 - 15	-5.0 Real	При вычитании одно из чисел имеет дробную часть. Команды от нее избавиться не было, поэтому результат тоже имеет дробную часть. Значит, результат имеет тип Real
6	Abs(-5.0)	5.0 Real	Abs меняет отрицательный знак аргумента на положительный. Так как аргумент имеет дробную часть, а команды от нее избавиться не было, результат тоже содержит дробную часть
7	5.0 + 2	7.0 Real	Те же соображения, что и в 5-м пункте. Результат имеет дробную часть

Ответ: 7.0. Real

Задание 2.11. Дано действительное число X. Напишите программу для вычисления:

- целой части числа X;
- числа X, округленного до ближайшего целого;
- числа X без дробных цифр.

Сборник заданий по вариантам см. в приложении 3.

Урок 2.4. Ввод и вывод данных

Заносить данные в ячейки памяти можно не только оператором присваивания, но и путем непосредствен-

ного ввода с клавиатуры. Это удобно тем, что в программу при каждом запуске можно вводить разные начальные значения, что добавляет ей универсальности.

Вводим переменные с клавиатуры

Пример 2.8. Ввод с клавиатуры значения переменной N

```
program Inp;
uses Crt;
var
   N: integer;
begin
   ClrScr;
   write('Введите число с клавиатуры:');

   readln(N);      { Здесь программа приостановится
                     и будет ожидать ввода с клавиатуры.
                     Наберите на клавиатуре число,
                     например 153, и нажмите
                     клавишу Enter}

   writeln('Вы ввели число ', N);

   readln          { Это оператор пустого ввода. Здесь
                     программа опять приостановится и
                     будет ожидать нажатия клавиши Enter.
                     За это время вы успеете просмотреть
                     вывод на экране. Этот прием мы
                     рекомендуем использовать, чтобы не
                     нажимать Alt+F5 после окончания
                     работы программы }
end.
```

Красивый вывод на экран

Рассмотрим еще одну задачу: задать с клавиатуры цвет фона (экрана), символов и координат для вывода текста, а затем вывести текст в окно с заданными координатами.

Продумаем алгоритм решения данной задачи (рис. 2.9).

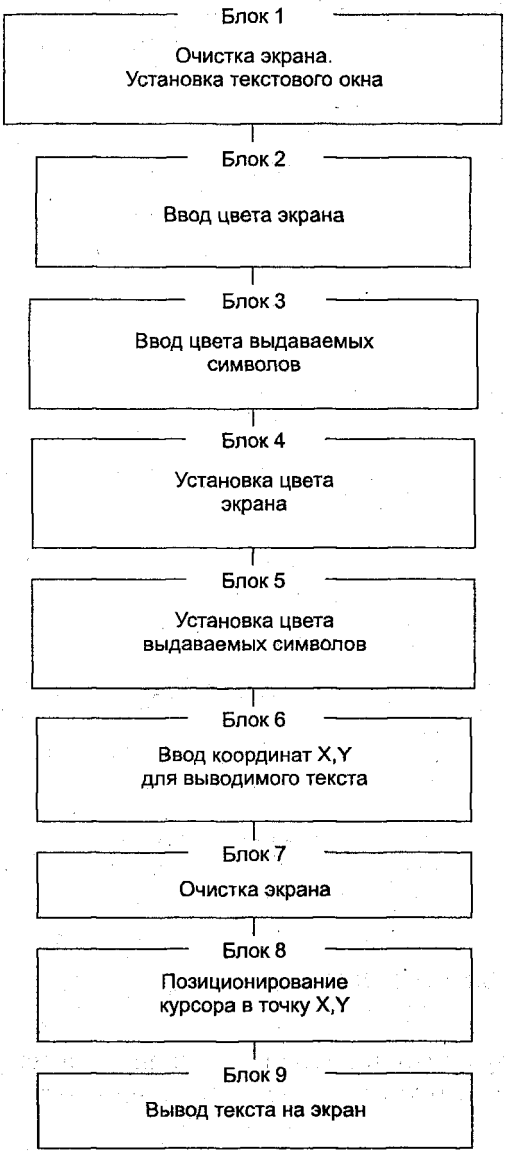


Рис. 2.9. Алгоритм решения задачи из примера 2.9

Пример 2.9. Красивый вывод текста

```
program Inp_Color;
uses
   Crt;          { Подключение библиотечного модуля Crt }
var

  { Опишем переменные, где будут храниться
    цвет экрана – C11, и цвет выдаваемых символов – C12 }

   C11,C12: 0..15;   { 0..15 – этот тип для переменных
                       называется интервальным. В данном
                       случае значения переменных могут
                       меняться в интервале от 0 до 15.
                       Здесь мы имеем отрезок (интервал)
                       базового типа integer. Палитра
                       цветов лежит именно в этом
                       интервале }

   X,Y: integer;     { Для хранения координат }

begin

   { Блок 1: }
   ClrScr;

   { Ниже следует вызов процедуры Window(X1,Y1,X2,Y2) из
     модуля Crt, которая определяет на экране текстовое
     окно. (X1,Y1) – координаты верхнего левого угла
     окна, (X2,Y2) – координаты нижнего правого угла }
   Window(20,5,60,20);

   { Блок 2: }
   write('Введите цвет для экрана:');

   readln(C11);       { Здесь работа программы
                        приостанавливается и ожидается
                        ввод номера цвета экрана
                        в переменную C11. Во время работы
                        программы следует ввести число и
                        нажать Enter }

   { Блок 3: }
   write('Введите цвет для символов:');

   readln(C12);      { Ожидается ввод номера цвета
                       символов в переменную C12. Во время
```

```
                работы программы следует ввести
                число и нажать Enter }

            { Блок 4: }
  TextBackGround(C11);      { Выбор фонового цвета }

  { Блок 5: }
  TextColor(C12); { Выбор цвета выводимых символов }

  { Блок 6: }
  writeln('Введите координаты X и Y');

  readln(X,Y);    { Ожидается ввод координат для
                    выводимого текста. Необходимо
                    ввести два числа (координаты) через
                    пробел и нажать Enter.
                    Помните, что координаты внутри
                    текстового окна отсчитываются
                    от его левого верхнего угла.
                    В нашем окне 16 строк и 41 столбец}

  { Блок 7: }
  ClrScr;       { Функция очистки экрана в данном
                  случае очистит не весь экран, а
                  только заданное текстовое окно }

  { Блок 8: }
  GoToXY(X,Y);{ Позиционирование курсора в точку
                с координатами X,Y }

  { Блок 9: }
  writeln(' Мы отлично вводим с клавиатуры!');

  readln      { Этот "пустой" оператор readln
                задерживает нас в экране пользователя.
                Возврат в среду Паскаль происходит
                после нажатия Enter }
end.
```

Задание значений переменных датчиком случайных чисел

Есть еще один способ занесения данных в переменные — вызов датчика случайных чисел, когда компьютер сам выдает случайное число из указанного диапазона.

Зададим этим способом цвет экрана. Вместо второго блока в программе поставим вызов датчика случайных чисел Random(X). Но перед этим обязательно нужно выполнить процедуру инициализации датчика, иначе он будет выдавать нам неслучайные числа:

```
Randomize;          { Инициализация датчика случайных чисел
                      проводится один раз в программе }

C11:=Random(16);{ В результате переменной C11
                  присваивается целое случайное число
                  из диапазона 0..15.
                  Результат функции Random(N) -
                  случайное число из диапазона 0..N-1 }
```

ЗАМЕЧАНИЕ

На самом деле настоящий программный датчик случайных чисел создать невозможно. Функция Random *выдает псевдослучайные числа. Это значит, что числа, выдаваемые функцией, порождаются по определенной закономерности. Эта закономерность придумана так, чтобы казалось, что числа получаются случайными. Однако если не использовать процедуру* randomize, *каждый раз при запуске программы последовательность чисел будет одинаковой. Процедура* randomize *«встряхивает» начальное значение последовательности. После этого порождаемая последовательность чисел становится почти уникальной.*

Рассмотрим еще несколько примеров использования функции Random.

Пусть нам необходимо получить случайное двузначное число, то есть число от 10 до 99. Логика проста: нам нужно одно случайное число из 90 (именно столько двузначных чисел). Значит, будем использовать функцию Random, которая выдаст одно число из 90, то есть Random(90). Но результат тогда будет лежать в диапазоне 0..89, а нам требуется 10..99! Значит, нужно сдвинуть

полученный диапазон вправо на 10. Это делает операция +10. В итоге получаем следующее: Random(90)+10.

Задание 2.12. Напишите программу, которая выдает сообщение в текстовое окно. Координаты окна и координаты для сообщения должны вводиться с клавиатуры. Цвет экрана и цвет символов задайте с помощью датчика случайных чисел.

Задание 2.13. Напишите программу, которая выдает сообщение на полный экран (без текстового окна). Координаты сообщения, цвет экрана и цвет символов задайте с помощью датчика случайных чисел.

ЗАПОМНИТЕ!

Заливка экрана или текстового окна выполняется вызовом двух процедур: установка цвета экрана — процедурой TextBackGround(...)*, а очистка экрана — процедурой* ClrScr.

Урок 2.5. Зачем нужны константы в программе?

Бывают случаи, когда некоторые величины не меняются по ходу выполнения программы. Для удобства работы с такими величинами в Паскале предусмотрена отдельная категория — константы.

Для начала простой пример:

Пример 2.10. Расчет скорости тела при падении с башни

```
Program Piza;
var
   G,V,H: real;
begin
   G:=9.8;  { Эта переменная всегда имеет одно значение
              и не изменяет его
              по ходу выполнения программы }

   write('Введите высоту башни:');
   readln(H); V:=Sqrt(2*G*H);
```

продолжение ⇩

Пример 2.10 (*продолжение*)

```
    writeln('Скорость падения':20,V:7:3);
            { На выводимый текст выделяется 20 позиций }

    readln
end.
```

Для наглядности в формуле вычисления скорости падения мы использовали переменную G, которая, однако, в действительности *не менялась*. И это неудивительно: она представляет ускорение свободного падения, которое, как известно, есть величина *постоянная*.

Для того чтобы явно указать в программе, что величина G не может измениться, перенесем ее описание в особый раздел — раздел описания постоянных величин, или *констант* (см. следующий пример). Тем самым мы покажем Паскалю и человеку, который будет читать нашу программу, что эту величину нельзя менять по ходу выполнения программы. Она получает свое значение один раз и не может быть изменена.

Пример 2.11. Программа при использовании констант — более логична и читабельна

```
Program Piza;

const       { Это раздел описания констант.
              Он находится перед разделом var }

   G=9.8;   { Тип константы определяется автоматически,
              по форме записи числа. В данном случае
              из-за наличия десятичной точки
              это тип real }
var V,H: real;
begin
   write('Введите высоту башни:');
   readln(H); V:=Sqrt(2*G*H);

   writeln('Скорость падения ',V:6:3);
            { Чтобы текст и число не "слиплись",
              после текста внутри апострофов добавлен
              пробел }
   readln
end.
```

Использование констант выполняет еще две функции. Во-первых, описывая величину в разделе констант мы подстраховываем сами себя, чтобы случайно не изменить ее в программе. (Вам, наверное, это замечание кажется глупым: как можно так ошибиться! Но при написании больших, многостраничных программ это становится актуальным.)

Во-вторых, константы оказываются нужны при объявлении новых типов данных — массивов. Об этом мы поговорим в теме 8.

Задание 2.14. Вычислите длину окружности и площадь круга. Радиус должен вводиться с клавиатуры.

Выводы

1. Данные, с которыми работает программа, хранятся в ячейках. Каждая ячейка имеет имя и тип данных. Изменяемые ячейки называются *переменными*, неизменяемые — *постоянными*.
2. Ячейки, используемые в программе, описываются в разделах const и var.
3. Для хранения целых чисел используется тип данных integer, а для хранения вещественных — real.
4. Каждый тип данных имеет свои операции и функции. Особенно это важно для типа integer, который имеет две операции деления — div и mod.
5. Для преобразования значений типа real к типу integer используются специальные функции — trunc и round.
6. При записи арифметических выражений нужно помнить о приоритете операций и о типе данных, который получается в результате.
7. Начальные значения переменных можно задавать путем ввода с клавиатуры или с помощью датчика случайных чисел.

Контрольные вопросы

1. Где хранятся все данные, с которыми работает программа?
2. Чем различаются понятия *имя ячейки*, *адрес ячейки* и *значение ячейки*?
3. Какой тип данных используется для хранения целых чисел? А для дробных?
4. Что следует сделать, если в программе используется величина, не изменяющаяся за все время работы программы?
5. В чем отличие между операциями mod, div и /?
6. Зачем нужны функции trunc и round? В чем между ними разница?
7. Какое максимальное значение может принимать переменная типа integer? Что делать, если необходимо сохранить целое число, в 10 раз большее этого значения?
8. Как записать на Паскале «2,5 в степени 16,7»?
9. Что означает запись «1E5», «3E-4», «.2E7»?
10. Что нужно использовать, чтобы изменить приоритет выполнения математических операций?
11. Чему равно и какой тип данных имеет выражение trunc(sqrt(2+52 div 8))-sqr(15 mod 4/3)?

ТЕМА 3

Учимся работать с символами

В предыдущей теме мы рассмотрели типы данных, позволяющие хранить и обрабатывать числа — целые и дробные. Но, перефразируя известную поговорку, не числами едиными живет программист. Кроме чисел, Паскаль умеет также работать с символьной информацией. Для каждого символа в программе выделяется отдельная ячейка со всеми присущими ячейке параметрами — именем и значением.

Урок 3.1. Как компьютер понимает символы

Под символами мы понимаем буквы и все те значки, которые вы видите на клавиатуре. В Паскале переменные для хранения символов имеют тип Char.

За каждым символом закреплен свой числовой код. Все коды сведены в таблицу.

Кодовая таблица ASCII

Обычно для хранения символов используют код, называемый ASCII (американский стандартный код обмена информацией).

Таблица 3.1. Фрагмент таблицы ASCII (таблица кодов символов)

Код	Двоичный код	Символ	Код	Двоичный код	Символ
48	00110000	0	65	01000001	A
49	00110001	1	66	01000010	B
50	00110010	2	67	01000011	C
57	00111001	9	89	01011001	Y
			90	01011010	Z

Как видите, цифры здесь — не числовые данные, а тоже символы, каждый из которых имеет свой код. В компьютере коды записаны в двоичном виде. На каждый код выделено 8 бит, то есть 1 байт. Получаем $2^8 = 256$ двоичных кодов. Всего в таблице ASCII 256 кодов: наименьшее значение кода 00000000, наибольшее — 11111111 (это 255 в двоичном виде).

Описание типа Char и стандартные функции

Пример 3.1. Демонстрация стандартных функций для работы с типом Char

```
Program Letter1;
var
   N: Integer;
   X: Char;
begin
   X:='L';     { В символьную переменную X
                 записали символ 'L' }
   writeln(X);

   N:=Ord (X); { Функция Ord возвращает код символа,
                 занесенного в переменную X, то есть код
                 буквы 'L' }

   writeln(N);
   X:='A';
   writeln(X);
   X:=Chr (N); { Функция Chr возвращает символ
                 по заданному коду. Сейчас в переменной X
                 оказался символ 'L' - именно его код мы
                 только что записали в переменную N }
   writeln(X);
   readln
end.
```

При выполнении программа выведет на экран следующее:

```
L
76
A
L
```

Пример 3.2. Ввод символов с клавиатуры

```
Program Letter2;
var
   X,Y:Char;
begin
   writeln('Введите несколько символов:');
   readln(X);
   writeln(X);
   writeln('Введите еще несколько символов:');
   readln(X,Y);
   writeln(X,Y);
   readln
end.
```

Запустив программу на выполнение, введите с клавиатуры последовательность символов (например, ABC) и нажмите Enter. Программа выведет единственный символ:

```
A
```

В ответ на второе предложение введите с клавиатуры CAT. На экране получим следующее:

```
CA
```

ЗАМЕЧАНИЕ

Переменная типа Char *принимает только один символ из введенной строки. При вводе символы не заключаются в апострофы. Таким образом, в первом случае из введенных символов запомнился только один, во втором — два.*

Можно определять и символьные константы:

```
const Leto='X';
```

Урок 3.2. Тип Char — порядковый тип!

В таблице кодов вы могли заметить такую закономерность:

'0'<'1'<'2'<'3'<...<'9'<...<'A'<'B'<'C'<...<'X'<'Y'<'Z'... (коды символов упорядочены).

Таким образом, для каждого элемента типа Char всегда есть предшествующий и последующий элементы. Такой тип данных называется *порядковым*. Тип Char — порядковый тип. Тип Integer также является порядковым.

Пример 3.3. Стандартные функции, применяемые к порядковому типу

```
Program Letter3;
var
   X1,X2,X3,X4: Char;
begin
   X1:='L';
   writeln(X1);

   X2:=Pred (X1); { Функция Pred возвращает
                    предшествующий элемент
                    относительно значения
                    переменной X1 }

   writeln('Pred=',X2);

   X3:=Succ (X1); { Функция Succ возвращает
                    последующий элемент
                    относительно значения
                    переменной X1 }

   writeln('Succ=',X3);
   readln
end.
```

При выполнении программа выведет на экран следующее:

```
L
Pred=K
Succ=M
```

Задание 3.1. Напишите программу расшифровки 4-буквенного однословного сообщения. Для получения 4 букв нужно ввести 3 строки:

- из первой строки прочесть только первую букву;

- из второй строки прочесть только первую букву;
- из третьей строки прочесть первую и вторую буквы.

Далее расшифровать полученные четыре буквы по такому алгоритму: вместо первой и третьей букв подставить соответственно буквы, отстоящие от них по алфавиту на две буквы назад, а вторую и четвертую буквы оставить без изменения.

Для проверки возьмите пример, приведенный ниже.

Ввод:

```
FINISHED
OR
PENDING?
```

На выводе должно быть слово DONE.

Задание 3.2. Известно, что коды прописных (заглавных) букв латинского алфавита следуют в таблице непрерывно друг за другом. Коды строчных букв латиницы также следуют непрерывно друг за другом на расстоянии 32 символов от прописных (ниже по таблице). Если ord('A') = 65, то ord('A')+32 = 97, и это код строчной буквы «a», то есть chr(ord('A')+32) = 'a'. Напишите программу, в которой вы вводите прописную букву (только латиницу!), а получаете ее строчной эквивалент, и наоборот, по строчной букве получаете соответствующую прописную.

ЗАМЕЧАНИЕ

С русскими символами такого порядка нет из-за особенностей организации кодовой таблицы. В частности, строчные буквы в таблице следуют не подряд, а с разрывом в середине алфавита.

Выводы

1. Все символы хранятся в компьютере в виде кодов.
2. Обычно для кодирования символов применяется таблица ASCII.

3. Каждому символу соответствует свой код.
4. Для преобразования символов в коды и обратно применяют функции ord и chr.
5. Тип Char является порядковым типом.
6. Коды буквы латинского алфавита идут последовательно.
7. Русские буквы хранятся в таблице символов ASCII с разрывом в последовательности кодов.
8. Для получения следующего и предыдущего символов используют соответственно функции succ и pred.

Контрольные вопросы

1. Сколько всего различных символов кодируется таблицей ASCII?
2. Какой объем памяти требуется для кодирования одного символа? А 15 символов?
3. Какой тип данных в Паскале предназначен для хранения символьной информации? Сколько символов можно поместить в одну переменную этого типа?
4. Какой код у буквы «F»? Какой символ кодируется кодом 87?
5. В программе определены 3 переменные (a,b,c:char;). В ответ на инструкцию readln(b,a,c); пользователь ввел текст Леша. В каком месте памяти оказалась каждая из введенных букв?
6. Каков будет результат выполнения инструкции c:=succ(pred(succ('D')))?
7. Какое значение получит переменная i в операторе i := pred(ord('F')-2)?

ТЕМА 4

Джордж Буль и его логика

В этой теме мы обсудим вопросы, связанные с Решениями. Именно так, с большой буквы. Мы разберем, как же компьютер принимает решения. Конечно, компьютеру не приходится принимать такие сложные решения, как человеку. Однако логики и определенности в поведении компьютера куда больше. Собственно, никаких колебаний у него и не бывает. Каждый раз, когда компьютер принимает решение, оно четко и окончательно — или да, или нет! Согласитесь, людям подобной решимости зачастую не хватает.

Урок 4.1. Необходим еще один тип — логический!

Поговорим о философии, а именно — о логике.

Логика оперирует утверждениями. Любое логическое утверждение может быть либо истинным, либо ложным. При решении многих задач возникает ситуация, когда требуется проверить некоторое условие (сформулированное в виде утверждения) и в зависимости от результата проверки (истинности утверждения) произвести те или иные действия:

- если условие выполняется, то результатом будет «истина»,
- если условие не выполняется, то результатом будет «ложь».

Например, утверждение «4 > 3» является истинным, а утверждение «2 > 3» — ложным.

Такие выражения называются *булевскими* (по имени английского математика Джорджа Буля). Область

математики, которая изучает действия с булевскими выражениями, называется *булевской алгеброй* или *алгеброй логики.*

Логический тип данных (Boolean)

Для хранения результата проверки условия введен логический тип — Boolean. Переменные такого типа называются *булевскими переменными.*

Пример 4.1. Булевские переменные в программе

```
Program Bool1;
var X: integer;
   Bol:Boolean;
begin
   X:=4;

   Bol:=X > 3;   { Это утверждение истинно }
   writeln(Bol);

   Bol:=X < 3;   { Это утверждение ложно }
   writeln(Bol);

   readln
end.
```

При выполнении программы на экране мы получим следующее:

```
TRUE
FALSE
```

Что такое TRUE и FALSE? Это значения логического выражения: TRUE означает «истина», а FALSE — «ложь».

Операции отношения

В логических условиях используются операции отношения. Вот как они записываются на языке Паскаль (табл. 4.1).

Таблица 4.1. Запись операций отношения на языке Паскаль

Операция отношения	Запись на языке Паскаль
Меньше	<
Меньше или равно	<=
Больше	>
Больше или равно	>=
Равно	=
Не равно	<>

Ввод-вывод булевских переменных

Булевские переменные можно выводить на экран, но нельзя вводить с клавиатуры. Для этого приходится вводить переменную другого типа, сравнивать ее с образцом и по результатам сравнения устанавливать значение логической переменной.

Пример 4.2. Как ввести с клавиатуры переменную булевского типа

```
program BooleanInput;
var eat:boolean;
   ch:char;
begin
   write('Ты хочешь есть [y/n]?:');
   readln(ch);
   eat:= ch = 'y';
   writeln('Твой ответ:',eat);
   readln
end.
```

Урок 4.2. Логические (булевские) операции

Часто принимаемое решение зависит от результата не одного, а нескольких утверждений. Например, «Вася получит сегодня пятерку, если придет на урок **И** правильно выполнит задание». Значит, нужно научиться

объединять результаты нескольких утверждений и принимать общее решение. В приведенном примере этим объединением служит союз «И».

Логическое умножение (конъюнкция)

В алгебре логики операции сравнения в логических выражениях можно комбинировать с помощью логических операций. Обсудим их по порядку.

Рассмотрим утверждение:

$x < 7$ и $x > 3$

Два отношения связаны союзом «и» (and).

Согласно правилам булевской алгебры, комбинация двух логических выражений, связанных между собой союзом «и», всегда является истинной, если истинны оба выражения.

Эта операция называется *логическим умножением*, или *конъюнкцией*. В Паскале она обозначается как and.

Таблица 4.2. Таблица истинности для операции логического умножения

Операнд 1	Операция	Операнд 2	Результат
True	And	True	True
True	And	False	False
False	And	True	False
False	And	False	False

Логическое сложение (дизъюнкция)

Рассмотрим утверждение:

$x > 100$ или $x < 10$

Выражения можно связывать союзом «или» (or).

Логическое выражение, связанное союзом «или», всегда ложно (false), если ложны обе его части. Во всех других случаях результатом будет «истина» (true).

Эта операция называется *логическим сложением*, или *дизъюнкцией*. В Паскале она обозначается как or.

Таблица 4.3. Таблица истинности для операции логического сложения

Операнд 1	Операция	Операнд 2	Результат
True	Or	True	True
True	Or	False	True
False	Or	True	True
False	Or	False	False

Исключающее ИЛИ (сложение по модулю 2)

Рассмотрим утверждение:

либо $x > 5$, либо $x < 0$

Выражения связаны парой «либо-либо».

Такое утверждение истинно, когда истинно только одно из двух составляющих его утверждений.

Можно сформулировать это иначе: логическое выражение истинно (true), если его операнды различны.

Данная операция называется *исключающим ИЛИ*. В Паскале она обозначается как xor.

Таблица 4.4. Таблица истинности для операции исключающего ИЛИ

Операнд 1	Операция	Операнд 2	Результат
True	Or	True	False
True	Or	False	True
False	Or	True	True
False	Or	False	False

Как еще можно получить эти результаты? Представим значение true как 1 (логическую единицу), а false — как 0 (логический нуль).

Теперь сложим эти значения и возьмем остаток от деления полученного результата нацело на 2 (mod 2). Эта операция называется также *сложением по модулю 2*. Ясно, что результат будет всегда меньше двух.

Логическое отрицание (инверсия)

Рассмотрим утверждение:

не $(x > 100)$

Выражение отрицается частицей «не».

Результат операции противоположен отрицаемому утверждению. Если утверждение было истинным (true), результатом будет false («ложь»). И наоборот, если утверждение было ложным (false), то получится true («истина»).

Эта операция называется *логическим отрицанием*, *логическим НЕ*, или *инверсией*. В Паскале она обозначается как not.

Таблица 4.5. Таблица истинности для операции логического НЕ

Операнд	Результат операции Not
True	False
False	True

Применение логических операций в программе

Пример 4.3. Логические операции в программе

```
Program Bool_1;
var X: Integer;
   Bol, OnBol, Rez: Boolean;
Begin
   X:=4;
   Bol:=X>3;
   OnBol:=X<3;
   writeln('Bol=',Bol);
   writeln('OnBol=',OnBol);
```

```
    Rez:=Bol and OnBol;
    writeln('Bol and OnBol=',Rez);
    Rez:=Bol or OnBol;
    writeln('Bol or OnBol=',Rez);
    Rez:=not Bol;
    writeln('not Bol=',Rez);
    readln
end.
```

При выполнении программы имеем на экране следующее:

```
Bol=TRUE
OnBol=FALSE
Bol and OnBol=FALSE
Bol or OnBol=TRUE
not Bol=FALSE
```

Пример 4.4. Составление логических выражений

```
Program Bol_2;
{ Введем логические переменные, которые будут определять
  характеристики студента.
  Составим выражения, определяющие, является ли студент
  первокурсником, получающим стипендию }

var
   Price:Boolean;      { Определяет наличие
                         стипендии у студента }

   Kurs1:Boolean;      { Определяет, является ли
                         студент первокурсником }

   Rezult:Boolean;     { Определяет результат }

begin
   Price:=True;        { Пусть наш студент получает
                         стипендию }

   Kurs1:=True;        { Пусть студент - первокурсник }

   Rezult:=Price and Kurs1;
   writeln('Студент - первокурсник со стипендией? -',
           Rezult);
```

продолжение ⇩

Пример 4.4 (*продолжение*)

```
    Price:=False;          { Пусть наш студент не получает
                             стипендию }

    Rezult:=Price and Kurs1;
    writeln('Студент - первокурсник со стипендией? - ',
            Rezult);
    readln
end.
```

При выполнении программы имеем на экране следующее:

```
Студент - первокурсник со стипендией? - TRUE
Студент - первокурсник со стипендией? - FALSE
```

Задание 4.1. Определите в программе 4 логических переменных, которые содержат следующую информацию о людях:

Married — «истина», если человек женат (замужем),
Blond — «истина», если у человека светлые волосы,
Male — «истина», если человек — мужчина,
Employed — «истина», если человек работает.

Составьте логические выражения, с помощью которых можно определить, является ли человек:

1) замужней женщиной;
2) неженатым мужчиной;
3) незамужней блондинкой;
4) безработной незамужней женщиной;
5) либо неженатым, либо безработным, либо и тем и другим.

Приоритет логических операций

При объединении нескольких логических операций нужно помнить, что они, подобно математическим операциям, подчинены правилам приоритета (табл. 4.6). Для изменения приоритета выполняемых действий необходимо использовать скобки.

Таблица 4.6. Приоритет логических операций

Приоритет	Логическая операция
1 (самый высокий)	Not
2	And
3 (самый низкий)	Or, Xor

Операции xor и or имеют одинаковый приоритет, и значит, при отсутствии скобок, выполняются слева направо.

ЗАПОМНИТЕ!

Это очень важно! Приоритет любой операции сравнения ниже, чем любой логической операции. Это значит, что при объединении сравнений при помощи логических операций каждое сравнение необходимо взять в скобки. Например, утверждение $2 \le x \le 4$ *должно быть записано как* (2<=x) and (x<=4).

Сборник заданий по вариантам см. в приложении 4.

Выводы

1. Существуют утверждения, относительно которых можно однозначно сказать, истинны они или ложны.
2. Для записи результатов таких утверждений, а также для анализа условий необходим еще один тип данных — boolean.
3. Данные этого типа могут принимать одно из двух значений: true («истина») или false («ложь»).
4. При составлении выражения для анализа условий используются операции отношений: >, <, >=, <=, =, <>.
5. Для объединения нескольких логических утверждений в одно используются логические операции (and,

or, xor, not), результаты которых формируются в соответствии с таблицами истинности.

6. Порядок выполнения действий определяется скобками и приоритетами операций.

Контрольные вопросы

1. Что такое логическое утверждение?
2. Сколько различных значений могут принимать логические утверждения? Как они обозначаются?
3. Как записать на Паскале утверждение «икс не равен пяти»?
4. Как записать на Паскале утверждение «игрек не принадлежит отрезку [3, 5]»?
5. Чему равен результат операции true or false and false?
6. Правильно ли записано выражение (x<0) or (x+2)>3?
7. Чему будет равен результат операции true xor false xor true?

ТЕМА 5

Анализ ситуации и последовательность выполнения команд

Все алгоритмы и программы, которые мы до настоящего момента рассматривали, были *линейными*, то есть выполнялись последовательно, шаг за шагом, инструкция за инструкцией, независимо от введенных данных. Однако часто бывает необходимо выполнять разные действия в зависимости от того, какое решение было принято. В данной теме мы рассмотрим, как менять порядок выполнения команд по результатам проверки некоторого условия.

Урок 5.1. Проверка условия и ветвление в алгоритме

В предыдущей беседе мы познакомились с логическими выражениями. Именно они используются в Паскале для организации ветвления.

Пусть стоит такая задача:

Если X > 3, то выводим на экран X.

Блок-схема алгоритма решения этой задачи выглядит так (рис. 5.1):

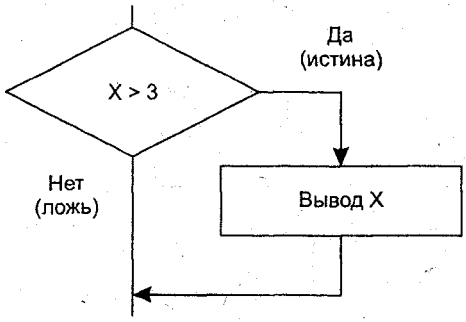


Рис. 5.1. Блок-схема алгоритма, выводящего число, если оно больше трех

На языке Паскаль такую схему обрабатывает условный оператор if.

Полная и неполная форма оператора if

Формат записи оператора if следующий: if <условие> then <оператор>.

Пример:

```
if X > 3 then writeln(X);
```

Под условием здесь понимается любое выражение, результат которого имеет тип boolean.

Это *неполная форма алгоритма с ветвлением.*

Изменим немного нашу задачу:

Если X >= 3, то вывести на экран X, иначе вывести текст 'X < 3'.

В том и другом случае X необходимо увеличить на 1.

Здесь используется расширенный условный оператор — *полная форма ветвления*: if ... then ... else ....

Формат записи оператора: if <условие> then <оператор-да> else <оператор-нет> (см. блок-схему на рис. 5.2).

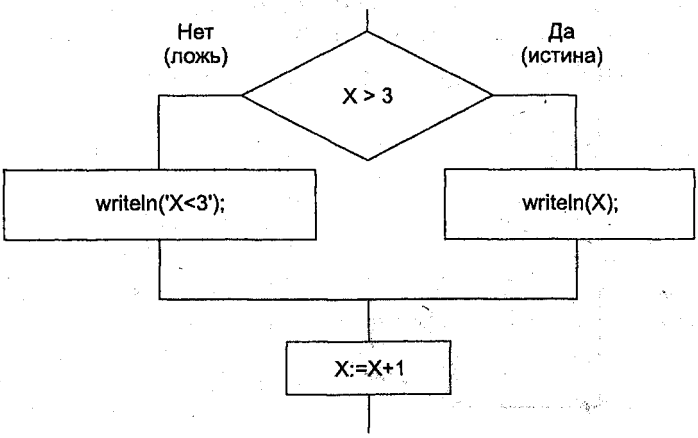


Рис. 5.2. Блок-схема алгоритма, выводящего число, если оно больше трех, или сообщение «X<3» в противном случае

В общем случае структурная схема условного оператора выглядит так (рис. 5.3).

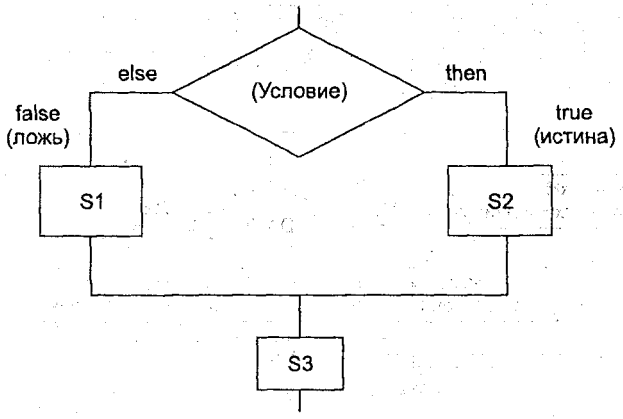


Рис. 5.3. Структурная схема условного оператора If

(S1, S2, S3 — условные обозначения операторов.)

ЗАПОМНИТЕ!

Перед else *нельзя ставить точку с запятой!*

Независимо от формы записи условия, после окончания оператора if программа снова «соединяется» и продолжает выполнять операторы, стоящие после структуры if. Это наглядно демонстрирует рис. 5.3. Оператор S3 выполняется независимо от того, каким будет результат проверки условия.

Рассмотрим задачу определения количества корней квадратного уравнения по дискриминанту (рис. 5.4).

ЗАПОМНИТЕ!

При использовании вложенных операторов if *слово* else *относится к последнему* if, *у которого нет еще* else .

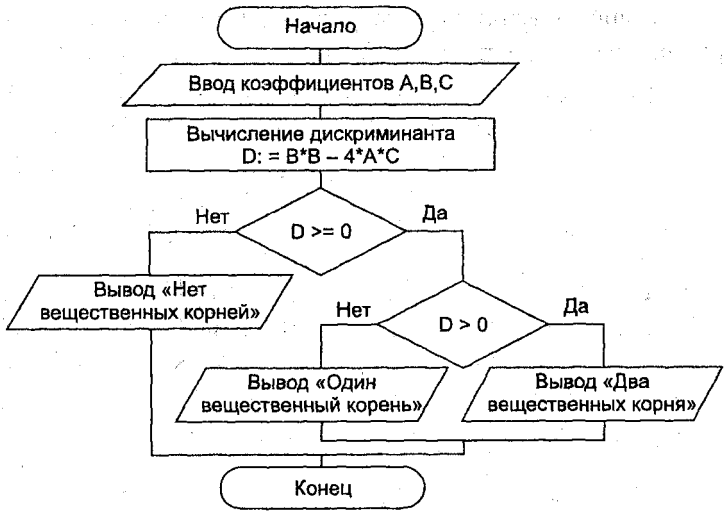


Рис. 5.4. Блок-схема алгоритма, определяющего количество корней квадратного уравнения по дискриминанту

Пример 5.1. Анализ дискриминанта квадратного уравнения

```
program Diskr;
var
   A,B,C,D: real;
begin
   write('Введите коэффициенты A,B,C:');
   readln(A,B,C);
   D:=SQR(B)-4*A*C;
   if D >= 0 then
      if D > 0 then
         writeln('Два вещественных корня')
      else
         writeln('Один вещественный корень')
   else
      writeln('Нет вещественных корней');
   readln
end.
```

В этой задаче используется вложенный оператор if.

Оформление программ

Возможно, вы уже обратили внимание, что во всех приводимых нами примерах мы слегка (на 2-3 позиции) сдвигаем операторы вправо. Таким способом мы выделяем, например, блок описания переменных var, а также операторы основной программы относительно begin и end. Это мы делаем сознательно и хотели бы рекомендовать вам поступать так же. Паскаль прекрасно поймет вашу программу, даже если вы ее всю наберете в одну строку. Однако через несколько дней в ней будет тяжело разобраться даже автору, не говоря уже о других людях.

Мы рекомендуем всегда сдвигать вправо вложенные фрагменты программы относительно точки вложения. Так/например, список переменных, определяемых в разделе var, следует сдвигать относительно слова var, список операторов основной программы — относительно begin и end, а операторы, вложенные в структуру if, — относительно if и else.

При этом соответствующие пары операторов begin и end рекомендуется располагать друг под другом, на одинаковом расстоянии от левого края.сЭто позволяет в сложной программе отследить, какому оператору begin какой оператор end соответствует, и, например, найти пропущенный оператор.

« | ЗАМЕЧАНИЕ

ЛвЯ ***Текоменуц&пся nftu Haбofie nftozfuuuitc/газцже после onefiatHufia*** `begin` ***писаЖь onefiatnofi*** `end` ***и notfuut t/же межуц ними вс(Яавля1нь вложенные onefiatnofurt. Э/ilo позволяет избежсипь cutnJ/аций с появлением непсфмых*** `begin/end` .

StfiuM Metiioq&M. [гешхменус/е&ся пользовшпься /пакже nftu на-Jofte anoan/гофов и скобок: HaJfictf? левую скобки, с/юзц по-апавыпе п/гавцю и rfioiqa t/же вписывайте >пекап межуц ними.

Задание 5.1. Нарисуйте блок-схему алгоритма и напишите программу, которая анализирует введенное с клавиатуры число и выдает на экран:

- удвоенное значение числа, если число положительное;
- абсолютное значение числа, если число отрицательное.

Задание 5.2. Нарисуйте блок-схему алгоритма и напишите программу, которая анализирует введенное с клавиатуры число на четность и сообщает о результате. Используйте операцию нахождения остатка от деления числа на 2.

Урок 5.2. Блоки операторов

Управляющая структура if ... then может показаться негибкой, так как после служебного слова then должен стоять только один оператор. Если вы напишете два оператора подряд (например, if x<>0 then y:=1/x; x:=0;), то второй оператор выполнится в любом случае, независимо от проверяемого условия.

Если требуется выполнить последовательность действий (несколько операторов подряд), то ее заключают в блок, образуемый операторами begin и end.

Пример:

```
if X > 3 then begin S1; S2; S3; S4 end;
```

Здесь S1–S4 символически обозначают операторы.

Эта группа (begin S1; S2; S3; S4 end) называется *составным оператором*, или *операторной скобкой*. Она как бы говорит компилятору, что данный блок операторов нужно рассматривать как единое целое.

Пример 5.2. Вычисление корней квадратного уравнения

```
Program Quad;
var
  A,B,C: real;{ Переменные для хранения коэффициентов }
```

продолжение

Пример 5.2 (*продолжение*)

```
    D: real;    { Переменная для дискриминанта }
    X1,X2: real;{ Переменные для получения корней }
begin
   writeln('Введите коэффициенты A, B, C:');
   readln(A,B,C);
   D:=Sqr(B) - 4*A*C;
   if D < 0 then
      writeln('Уравнение не имеет вещественных корней')
   else
      if D=0 then
         writeln('У уравнения один корень',
                 -B/(2*A):6:2)
      else

      { Ниже идет составной оператор }
      begin
         X1:=(-B + Sqrt(D))/(2*A);
         X2:=(-B - Sqrt(D))/(2*A);
         writeln('У уравнения два корня:',
                 X1:6:2, X2:6:2)
      end;

   readln
end.
```

Задание 5.3. Нарисовать блок-схему алгоритма и написать программу, в которой с клавиатуры вводится код режима работы Kd и цвет Cl. На экран выводится сообщение. При этом, если Kd = 0, то сообщение выводится символами цвета Cl, а если Kd = 1, то сообщение выводится символами цвета Cl+16.

На экране во втором случае вы получите мигающие символы, так как код их цвета больше 15. В этом случае в качестве номера мигающего цвета Паскаль использует остаток от деления кода цвета нацело на 16.

ЗАМЕЧАНИЕ

В качестве проверяемых условий можно использовать логические операции.

Рассмотрим задачу: ввести три числа A, B, C и определить, равны ли введенные числа. Блок-схема алгоритма показана на рис. 5.5.

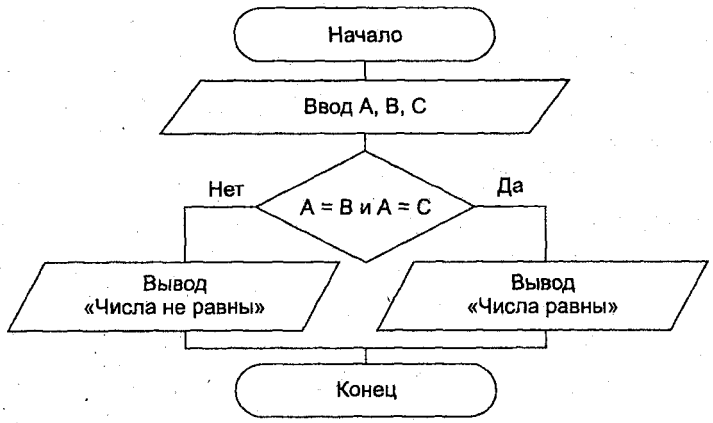


Рис. 5.5. Блок-схема алгоритма, определяющего, равны ли друг другу три введенные с клавиатуры числа

Пример 5.3. Проверка равенства трех чисел, введенных с клавиатуры

```
Program Mx;
var
   A,B,C: integer;
begin
   writeln(' Введите 3 числа:');
   readln(A,B,C);
   if (A = B) and (A = C) then
      writeln('Числа равные')
   else
      writeln('Числа не равны');
   readln
end.
```

Задание 5.4. Ввести три числа A, B, C и определить максимальное из них. Нарисуйте блок-схему алгоритма и напишите программу.

Задание 5.5. Ввести четырехзначное целое число и определить, является ли оно палиндромом, или «перевертышем» (такими, например, являются числа 6666 и 3223). Нарисуйте блок-схему алгоритма.

Подсказка: для выделения отдельных разрядов числа используются операции div и mod.

Задание 5.6. Ввести три числа A, B, C. Если ни одно из чисел не равно нулю, то в переменную K записать среднее арифметическое трех чисел.

Задание 5.7. Введите значение X и, используя график функции (рис. 5.6), определите значение Y. Заполните блок-схему алгоритма (рис. 5.7).

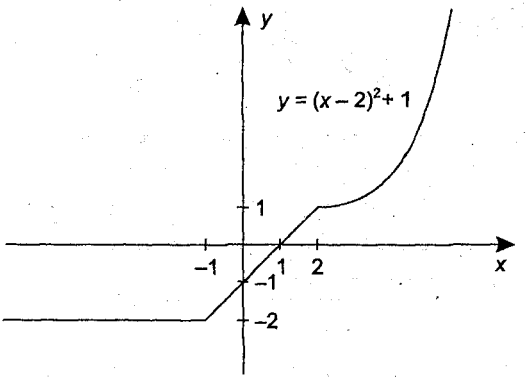


Рис. 5.6. График кусочно-заданной функции для задания 5.7

Задание 5.8. Положение фигуры на шахматной доске (8 × 8) описывается двумя числами — номером горизонтали и номером вертикали. Ввести с клавиатуры координаты ферзя (X, Y) и координаты любой фигуры (M, N). Проверить, находится ли фигура под ударом. Ферзь бьет по вертикали, горизонтали и диагонали.

Задание 5.9. Введите число с клавиатуры. Если это число четное и кратно 7, то выведите свое имя на экран

красным (red) цветом, иначе — выведите его зеленым (green) цветом. Предварительно заполните блок-схему алгоритма (рис. 5.8).

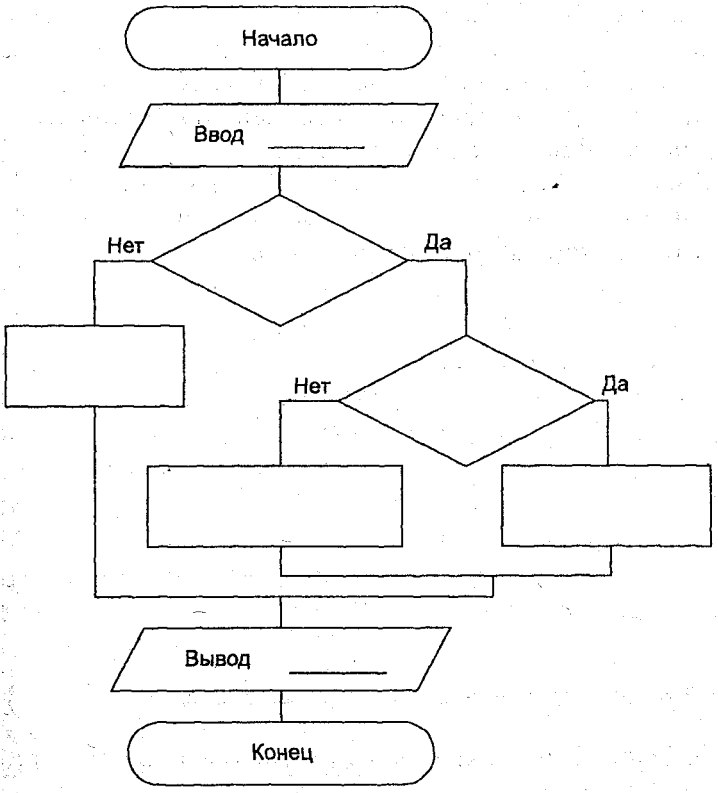


Рис. 5.7. «Слепая» блок-схема алгоритма вычисления кусочно-заданной функции, изображенной на рис. 5.6

Задание 5.10. Ввести три числа A, B, C и определить среднее из них (то, которое больше одного, но меньше другого). Нарисуйте блок-схему алгоритма и напишите программу.

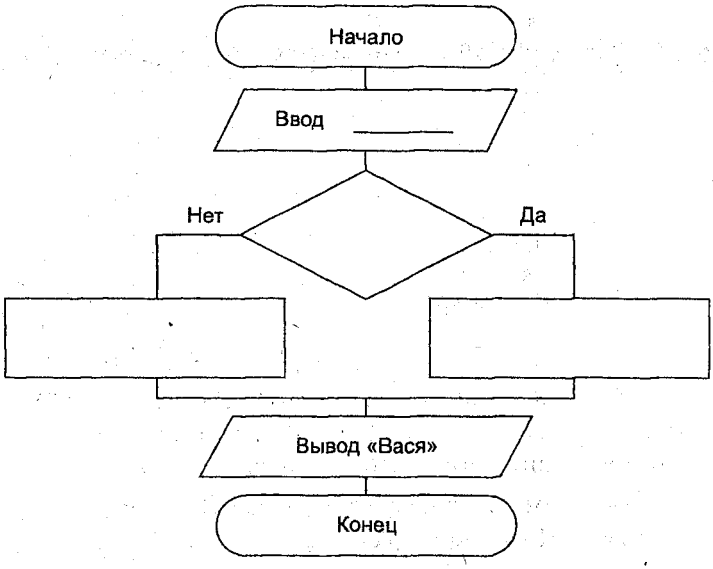


Рис. 5.8. «Слепая» блок-схема алгоритма, выводящего слово «Вася» красным цветом на экран, если введенное число кратно 7, и зеленым цветом в противном случае

Урок 5.3. Ветвление по ряду условий (оператор case)

Оператор if позволяет выполнять переходы на ту или иную ветвь по значению булевского (логического) условия. Используя несколько операторов if, можно производить ветвление по последовательности условий.

Пример 5.4. Преобразование введенного целого числа из диапазона (0..4) в его словесное представление

```
Program Digit1;
var
   Num: integer;
begin
   write('Введите число:');
```

```
    readln(Num);
    if Num = 0 then
       writeln('Нуль');
    if Num = 1 then
       writeln('Один');
    if Num = 2 then
       writeln('Два ');
    if Num = 3 then
       writeln('Три ');
    if Num = 4 then
       writeln('Четыре');
    readln
end.
```

Выполним ту же задачу, используя другую управляющую структуру — оператор выбора case ... of.

Формат записи оператора таков:

```
Case <выражение порядкового типа> of
<значение1>:<оператор1>;
...
<значениеN>:<операторN>
else <оператор>
end
```

Пример 5.5. Использование структуры case ... of для перевода целого числа в его словесное представление

```
Program Digit2;
var Num: integer;
begin
   write('Введите число:');
   readln(Num);
   case Num of
     0: writeln('Нуль');
     1: writeln('Один');
     2: writeln('Два ');
     3: writeln('Три ');
     4: writeln('Четыре')
     else writeln('Введено другое число')
 end;
 readln
end.
```

Ниже приведена блок-схема алгоритма решения этой задачи (рис. 5.9).

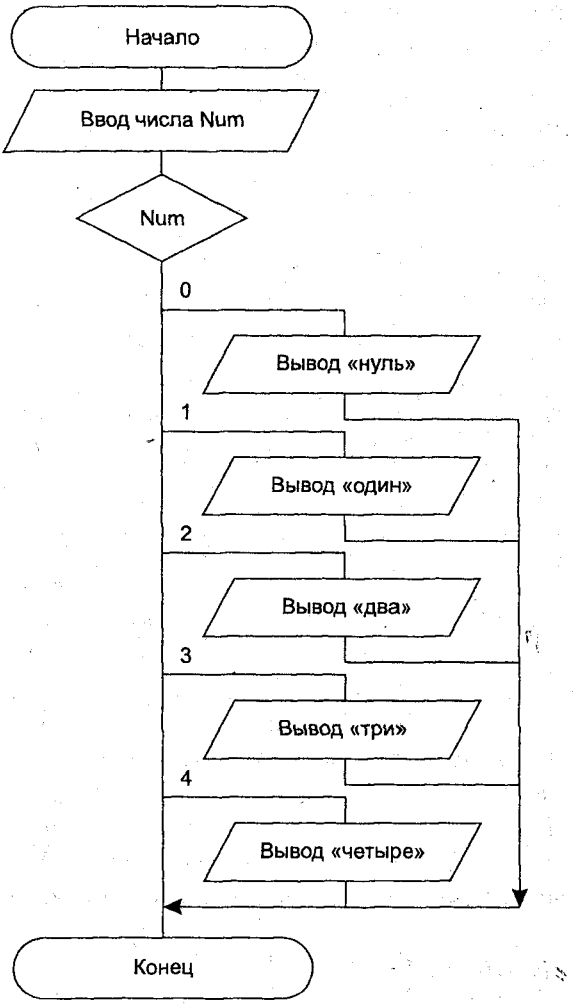


Рис. 5.9. Блок-схема алгоритма, выводящая словесное представление введенного с клавиатуры числа (не большего 4)

Переменная Num является *селектором* в операторе case. По значению селектора происходит переход на соответствующую метку.

Селектор должен принадлежать к порядковому типу (то есть он не может иметь тип real)!

Пример 5.6. Определение номера квартала по введенному номеру месяца

```
Program Digit3;
var
   Num:integer;
begin
   write('Введите номер месяца:');
   readln(Num);
   case Num of
      1,2,3 : writeln('Первый квартал');
      4,5,6 : writeln('Второй квартал');
      7..9  : writeln('Третий квартал'); { метка 7..9 –
                                           интервал чисел
                                           от 7 до 9;
                                           то же самое,
                                           что и 7,8,9 }
      10..12: writeln('Четвертый квартал')
      else writeln('Некорректный ввод')
   end;
   readln
end.
```

При переходе на метку может выполняться целый блок операторов, который оформляется с помощью структуры begin ... end.

Задание 5.11. Написать программу, в которой в переменную типа Char вводится символ с клавиатуры. Программа выдает сообщение о том, какой символ был введен:

- цифра от 0 до 9;
- латинская строчная буква;
- латинская заглавная буква.

При записи меток в операторе case можно использовать интервальный тип. Например, интервал для латинских заглавных букв записывается: 'A'..'Z'.

Если нужно учесть строчную латиницу, то интервал для меток будет выглядеть так: 'A'..'Z','a'..'z'.

Выводы

1. Существуют задачи, решение которых включает анализ логического условия. Такие задачи описываются разветвляющимся алгоритмом (сравните с линейным алгоритмом).
2. При ветвлении анализируется логическое выражение и, в зависимости от его результата, выполняется та или иная ветвь алгоритма.
3. В Паскале оператор ветвления называется if. Он имеет две формы записи — полную и неполную.
4. При полной форме записи if в случае истинности логического условия выполняется один блок программы (после слова then), а в случае ложности — другой (после слова else).
5. При неполной записи оператора if блок else опускается.
6. При переходе на ту или иную ветвь алгоритма после анализа логического условия возможно выполнение блока операторов, который оформляется с помощью структуры begin ... end.
7. Точка с запятой слева и справа от then и от else не ставится.
8. В случае, когда анализируемое выражение может иметь более двух значений, и при разных значениях нужно выполнять разные инструкции, используют оператор case.

9. Оператор case должен заканчиваться ключевым словом end. Это один из тех редких случаев, когда количество операторов begin в программе не будет совпадать с количеством операторов end.
10. Чтобы текст программы был более понятен, вложенные (подчиненные) блоки операторов принято оформлять со сдвигом вправо, лесенкой. При каждом следующем вложении операторы сдвигают еще на несколько позиций вправо.

Контрольные вопросы

1. Чем отличается линейный алгоритм от ветвления?
2. Какие ключевые слова используются в Паскале для организации ветвления? Что находится между ними?
3. Чем полное ветвление отличается от неполного?
4. Как оформлять текст программы, чтобы он был понятнее?
5. Что необходимо использовать, если в случае истинности некоторого условия нужно выполнить несколько операторов?
6. Как быть, если в случае истинности некоторого условия никаких действий выполнять не требуется, а в случае ложности нужно выполнить несколько действий?
7. Какую управляющую структуру Паскаля нужно использовать, если проверяемое выражение может принимать несколько возможных значений, и в каждом случае необходимо выполнить разные действия?
8. В каком случае количество операторов begin в программе не должно соответствовать количеству операторов end?

ТЕМА 6

Многократно повторяющиеся действия

Иногда необходимо повторить определенные действия в программе. Повторение некоторой последовательности действий называется *циклом*. Саму последовательность повторяющихся действий называют *телом цикла*.

Если число повторений известно заранее, то используется структура, которая называется *циклом с заданным (известным) числом повторений*, или *циклом со счетчиком*. Этот вид цикла является частным случаем цикла с условием. Мы начинаем с этого вида цикла в силу его простоты и наглядности.

Урок 6.1. Оператор цикла for

На языке Паскаль повторение некоторой последовательности действий известное число раз выполняет оператор for. Подсчет количества выполняемых действий осуществляется при помощи специальной переменной — *счетчика*. Поэтому цикл for называют иногда циклом со счетчиком. Цикл for на Паскале может быть представлен в двух формах. Первая форма последовательно наращивает счетчик:

for <переменная порядкового типа>:=<начальное значение> to <конечное значение> do <оператор>

Вторая форма последовательно уменьшает счетчик:

for <переменная порядкового типа>:=<начальное значение> downto <конечное значение> do <оператор>

Оператор for с последовательным увеличением счетчика

Пример 6.1. Вывод на экран квадратов чисел от 1 до 10

```
Program Test1;
var N: integer;                                    продолжение
```

Пример 6.1 (*продолжение*)

```
begin
   for N:=1 to 10 do    { Переменная N будет меняться
                          от 1 до 10 с шагом 1 }

      writeln(sqr(N));  { Эта строка – тело цикла.
                          Оно выполняется 10 раз }

 readln
end.
```

Поясним пример 6.1. Переменная N является *счетчиком цикла*. Счетчик цикла всегда должен иметь порядковый тип (то есть он не может иметь тип real). В операторе for указаны его начальное и конечное значения. Начальное значение не обязательно равно 1! При первом выполнении тела цикла N = 1, при втором — N = 2, и т. д. При последнем выполнении тела цикла N = 10. Каждый раз перед выполнением тела цикла текущее значение N сравнивается с конечным. После каждого выполнения тела цикла переменная N увеличивается на 1. Блок-схема алгоритма представлена на рис. 6.1.

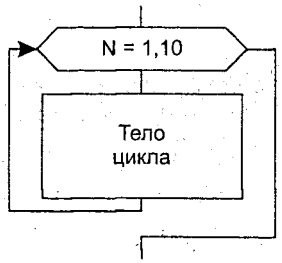


Рис. 6.1. Блок-схема организации цикла в примере 6.1

Как только N превысит конечное значение, выполнение цикла прекращается. Считается, что после окончания цикла переменная цикла не определена (то есть в разных реализациях языка Паскаль она может принимать разные значения). Иными словами, неправиль-

но считать, что после окончания цикла переменная-счетчик цикла имеет какое-то определенное значение.

Крайне не рекомендуется внутри цикла самостоятельно менять счетчик цикла, особенно в сторону уменьшения. Это может привести к «зацикливанию» программы (бесконечному повторению тела цикла).

Оператор for с последовательным уменьшением счетчика

Счетчик может изменяться с шагом –1. Это вторая форма оператора for (for ... downto ... do).

Пример 6.2. Вывод на экран кубов чисел от 11 до 5

```
Program Test2;
var
   N: integer;
begin
   for N:=11 downto 5 do   { Счетчик N изменяется
                             с шагом -1 }

      write(N*N*N:5);      { Эта строка – тело цикла;
                             оно выполняется 8 раз,
                             так как N изменяется
                             от 11 до 5 с шагом -1 }

   writeln;                { Этот оператор нужен,
                             чтобы закончить вывод чисел
                             в одну строку }

   readln
end.
```

Урок 6.2. Применение циклов со счетчиком

Можно организовать выполнение одного цикла внутри другого. В этом случае различают внешний и внутренний циклы — например, когда при каждом значении счетчика внешнего цикла нужно несколько раз выпол-

нить какое-то действие (внутренний цикл). Счетчик внешнего цикла изменяется медленнее, чем счетчик внутреннего.

Цикл в цикле

Рассмотрим задачу вывода последовательности пар чисел:

```
1 1
1 2
1 3
1 4
2 1
2 2
2 3
2 4
3 1
3 2
3 3
3 4
```

Блок-схема алгоритма решения задачи показана на рис. 6.2.

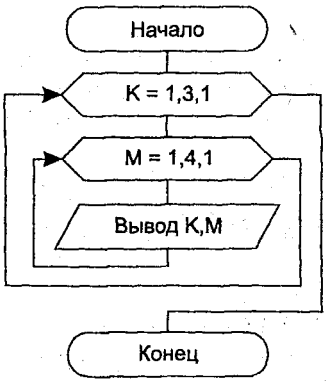


Рис. 6.2. Блок-схема алгоритма с вложенными циклами, выводящего последовательность из примера 6.2

Пример 6.3. Использование цикла в цикле

```
Program Test3
var
   K,M: integer;
begin
   for K:=1 to 3 do
      for M:=1 to 4 do

         writeln(K,' ',M); { Пробел в апострофах между
                             K и M нужен для того,
                             чтобы эти числа не
                             сливались друг с другом }

   readln
end.
```

Для каждого значения переменной K переменная M меняется от 1 до 4. Нетрудно подсчитать, что в этом случае оператор writeln выполнится 12 раз.

Задание 6.1. Вывести на экран 6 раз свое имя.

Задание 6.2. Вывести на экран таблицу умножения для 5 чисел от 9 до 4.

Задание 6.3. Вывести на экран коды таблицы ASCII от 0 до 255 и их символы. Выводить парами код и символ.

Трассировка

Для понимания чужой программы и для проверки правильности написания своей используют метод пошагового выполнения программы с отслеживанием значений всех переменных.

Пример 6.4. Вычисление суммы чисел от 6 до 10

```
Program Test4;
var
   N: integer;   { Это будет счетчик цикла for }

   S: integer;   { В этой переменной будем
                   накапливать сумму }
```

продолжение ⇙

Пример 6.4 (*продолжение*)

```
begin
   S:=0;          { Вначале обнулим сумматор }

   for N:=6 to 10 do

      S:=S + N;  { Эта строка – тело цикла.
                   При его выполнении каждый раз
                   к S прибавляется очередное N.
                   Переменную S можно сравнить
                   с аккумулятором, в котором
                   накапливается сумма }

   writeln('Сумма чисел=', S:6);
   readln
end.
```

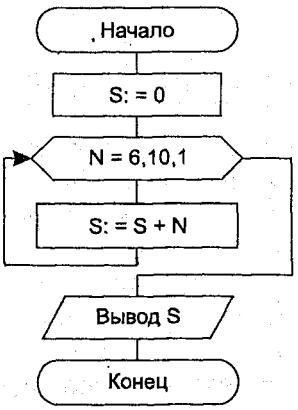


Рис. 6.3. Блок-схема алгоритма вычисления суммы чисел от 6 до 10

Для проверки правильности работы программы рекомендуется пошагово отслеживать изменение всех переменных после выполнения каждого оператора программы. Такой процесс называется *трассировкой*. Продемонстрируем этот прием (табл. 6.1).

В результате работы программы на экране получим число 40.

Таблица 6.1. Трассировка программы из примера 6.4

Оператор	Условие	N	S	Примечание
S:=0			0	
for N:= 6 to 10 do	Да	6		
S:= S + N			6	0 + 6 = 6
For N:= 6 to 10 do	Да	7		
S:= S + N			13	6 + 7 = 13
For N:= 6 to 10 do	Да	8		
S:= S + N			21	13 + 8 = 21
For N:= 6 to 10 do	Да	9		
S:= S + N			30	21 + 9 = 30
For N:= 6 to 10 do	Да	10		
S:= S + N			40	30 + 10 = 40
For N:= 6 to 10 do	Нет	11		
writeln('Сумма чисел ', S:3)		???		На экране: Сумма чисел=40

Для операторов, выполняющих проверку условий (if, for и т. п.) в столбце «Условие» принято указывать результат проверки. В данном случае в цикле for проверяется условие продолжения цикла.

Символы «???» подчеркивают, что значение счетчика цикла по выходе из цикла считается неопределенным.

Метод трассировки очень помогает при отладке программы, когда программа выдает не тот результат, который должна выдать. Осуществляя пошаговую трассировку, мы вникаем в логику работы программы и на каждом шаге проверяем, правильны ли были наши рассуждения при ее написании.

Вычисление суммы ряда

Рассмотрим задачу вычисления суммы ряда:

$$\frac{1}{1\cdot 1}+\frac{1}{2\cdot 2}+\frac{1}{3\cdot 3}+\frac{1}{4\cdot 4}+\frac{1}{5\cdot 5}.$$

Здесь мы имеем ряд дробей, у которых в знаменателях записаны квадраты чисел от 1 до 5.

Рассмотрим каждую дробь как произведение двух дробей, например:

$$\frac{1}{3 \cdot 3} = \frac{1}{3} \cdot \frac{1}{3}.$$

В общем виде это можно записать так:

$$\frac{1}{N \cdot N} = \frac{1}{N} \cdot \frac{1}{N}.$$

Блок-схема алгоритма решения задачи представлена на рис. 6.4.

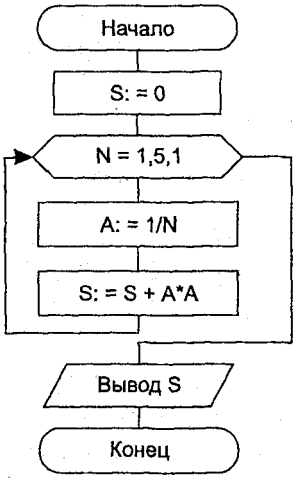


Рис. 6.4. Блок-схема алгоритма вычисления суммы ряда

Пример 6.5. Вычисление суммы ряда

```
Program Test5;
var
   N: integer;       { Это будет счетчик цикла for }
```

```
    S: real;          { В этой переменной будем
                        накапливать сумму.
                        Выбрали для нее тип real,
                        так как искомая сумма будет
                        нецелым числом }

    A: real;          { Для записи очередной дроби 1/N
                        (тоже число не целое) }
begin
    S:=0;             { Первоначально сумматор обнулим }

    for N:=1 to 5 do
    begin
      A:=1 / N;       { Эти строки - тело цикла.
                        В цикле необходимо выполнить
                        два оператора, поэтому объединяем
                        их в блок  begin ... end }

      S:=S + A*A
    end;
    writeln('Сумма чисел=', S:6:4);
    readln
end.
```

Задание 6.5. Написать программу вычисления $n!$ (факториал числа n), где n положительно.

Определение *факториала*:

$0! = 1$

$1! = 1$

$2! = 1 \cdot 2$

$3! = 1 \cdot 2 \cdot 3$

.....

$n! = 1 \cdot 2 \cdot 3 \cdot .... \cdot (n-1) \cdot n$

Другими словами, $n!$ — это произведение первых n натуральных чисел.

Каждый следующий результат (обозначим его P) получается путем умножения предыдущего результата (предыдущего P) на счетчик, который пробегает значения от 1 до n. Обозначим значение счетчика буквой k.

Получаем общий вид выражения: $P = P \cdot k$ (то есть воспользуемся рекуррентной формулой вычисления факториала: $n! = (n-1)! \cdot n$).

Программа должна быть организована так: с клавиатуры вводится число n (n — положительно), а затем на экран выдается таблица факториалов чисел до n включительно.

Задание 6.6. Написать программу вычисления суммы ряда $S = 1 + 2 + 3 + 4 + 5 + 6$. Нарисовать блок-схему и заполнить таблицу трассировки. Убедиться при трассировке, что сумма равна 21.

Таблица 6.2. Заготовка для таблицы трассировки алгоритма из задания 6.6

Оператор	Условие	S	k (счетчик)	Примечание

Задание 6.6. Написать программу вычисления суммы ряда для n слагаемых (n вводится с клавиатуры):

$$\frac{1}{1\cdot 2\cdot 3}+\frac{1}{2\cdot 3\cdot 4}+\frac{1}{3\cdot 4\cdot 5}+\frac{1}{4\cdot 5\cdot 6}+\cdots$$

Задание 6.7. Используя возможности модуля Crt для работы с экраном в текстовом режиме, написать программу, которая 16 раз меняет цвет экрана и выводит любой текст на новом фоне в центр экрана.

Пояснение: разумно, если цвет фона и параметр цикла будут одной переменной (палитра цветов изменяется в диапазоне 0–15).

Задание 6.8. Используя возможности модуля Crt, напишите программу, в которой символ «звездочка» (*)

пробегает по всему периметру экрана из верхнего левого угла.

Пояснение: в программе организуйте 4 цикла. В качестве счетчика используйте координаты X и Y. Нарисуйте блок-схему алгоритма.

Попробуйте изменить программу, используя всего два цикла: в одном цикле звездочки бегут сразу по верхней и нижней строкам экрана, в другом — сразу по левому и правому краю. Пусть каждая следующая звездочка выводится случайным цветом.

Задание 6.9. По экрану разбросайте 1000 звездочек в случайном месте случайным цветом с небольшой задержкой. Не забудьте инициализировать датчик случайных чисел в начале программы — один раз! Нарисуйте блок-схему алгоритма.

Выводы

1. Для организации многократно повторяющихся действий с заранее известным числом повторений используется оператор цикла for.
2. Счетчик цикла всегда имеет порядковый тип.
3. Счетчик цикла изменяется с шагом +1, если оператор имеет форму
   for ...:=... to .... do
4. Счетчик цикла изменяется с шагом −1, если оператор имеет форму
   for ...:=... downto .... do
5. Чтобы узнать, сколько раз выполнится тело цикла for, нужно найти разность между крайними значениями счетчика (по модулю) и прибавить к результату 1.
6. Не рекомендуется изменять счетчик цикла в теле цикла.

7. Если внутри цикла for поставить еще один цикл for, то количество раз, которое выполнится тело внутреннего цикла, равно произведению числа повторений внешнего цикла на число повторений внутреннего.
8. Для проверки правильности работы алгоритма его выполняют вручную, шаг за шагом, отслеживая изменения всех переменных. Это называется трассировкой.

Контрольные вопросы

1. Какой оператор нужно использовать, чтобы вывести в каждой строке экрана слово «Привет»?
2. Чем отличаются формы «to» и «downto» оператора for?
3. Переменные какого типа должны использоваться в качестве счетчика цикла for?
4. Сколько раз выполнится тело внутреннего цикла:

```
for i:=2 to 6 do
  for j:=5 downto 3 do
    writeln('*');
```

5. Написанная программа выдает странный результат. Вероятно, программа написана с ошибкой. Как понять, где содержится ошибка?
6. Требуется последовательно присвоить переменной N значения всех трехзначных чисел. Напишите оператор, присваивающий переменной N нужные значения.

ТЕМА 7
Циклы с условием

Цикл со счетчиком for, рассмотренный в предыдущей теме, отлично выполняет свои функции, когда число повторений тела цикла известно к моменту его начала (или известны начальное и конечное значения счетчика, что, впрочем, то же самое). Однако такая «радужная» картина встречается в программировании далеко не всегда. Часто приходится решать задачи, когда число повторений цикла неизвестно и определяется лишь постепенно, после некоторого количества повторений тела цикла. В этом случае применяют другую разновидность цикла — *цикл с условием*. В языке Паскаль циклов с условием предусмотрено два: условие цикла может проверяться перед телом цикла или после него.

Урок 7.1. Цикл с предусловием

В первой разновидности цикла условие проверяется перед выполнением тела цикла. Поэтому данное условие правильно будет назвать *условием продолжения цикла*. Цикл такого вида называется *циклом с предусловием*.

Цикл будет повторяться до тех пор, пока проверка этого условия будет давать результат «истина» (true), то есть пока условие выполняется. Если условие сразу оказывается ложным, цикл не будет выполнен ни разу.

Описание цикла с предусловием

Запишем цикл с предусловием на языке блок-схем (рис. 7.1).

Вот как этот тип цикла реализуется на языке Паскаль:

```
While <логическое условие> do <оператор-тело_цикла>
```

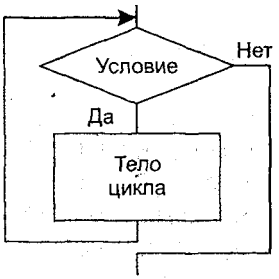


Рис. 7.1. Блок-схема цикла с предусловием

Так же как при использовании цикла for и оператора if, после служебного слова do предполагается только один оператор.

Если в теле цикла нужно выполнить несколько операторов, оно оформляется как блок begin ... end.

ЗАПОМНИТЕ!

После служебных слов then, else, do *(в операторах* if, for, while*) должен стоять только один оператор! Если необходимо выполнить несколько операторов, они должны быть взяты в операторные скобки (перед операторами нужно поставить* begin*, после —* end*).*

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

Приближенное вычисление суммы бесконечного ряда

Рассмотрим задачу, в которой требуется написать программу приближенного вычисления суммы:

$$1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\cdots$$

с точностью ε.

Считается, что нужное приближение получено с заданной точностью ε (эпсилон), если вычислена сумма

нескольких первых слагаемых, и очередное слагаемое оказалось по модулю меньше, чем данное малое положительное число ε. Тогда считается, что это и все последующие слагаемые можно уже не учитывать.

На каждом следующем шаге цикла будем максимально использовать сделанное нами на предыдущих шагах.

Если уже получено $x^{i-1}/(i-1)!$, то для вычисления $x^i/i!$ достаточно умножить предыдущий результат на x/i.

Блок-схема алгоритма решения задачи приведена на рис. 7.2.

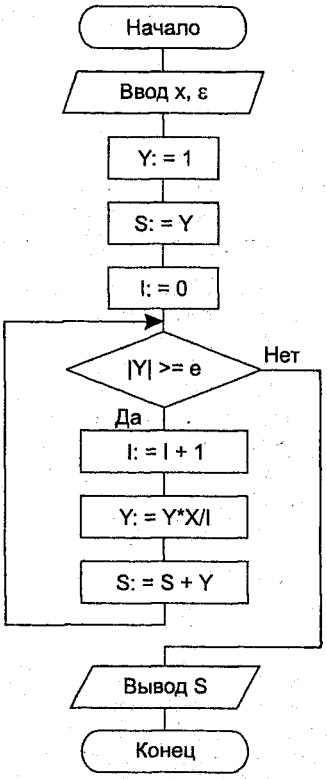


Рис. 7.2. Блок-схема алгоритма приближенного вычисления суммы ряда (пример 7.1)

Пример 7.1. Приближенное вычисление суммы бесконечно убывающего ряда

```
Program Summer2;
var
   Eps:real;     { Заданное число "эпсилон" }

   X: real;      { Основание степени в числителе дроби }

   S: real;      { В этой переменной будем
                   накапливать сумму }

   Y: real;      { Для хранения очередного слагаемого }

   i: integer;   { Счетчик числа шагов }
begin
   write('Введите X и Epsilon:');
   readln(X, Eps);

   Y:=1;         { Первое слагаемое }

   S:=Y;         { Положим в сумматор первое слагаемое }

   i:=0;         { Обнулим счетчик шагов }

   while abs(Y)>=Eps do { Пока добавленное слагаемое
                          не меньше "эпсилон",
                          считаем сумму.
                          Если "эпсилон" сразу
                          не меньше 1,
                          цикл не выполнится ни разу! }

   begin         { Началось тело цикла }

      i:=i+1;    { Вычислили номер текущего шага }

      Y:=Y*X/i;  { Посчитали новое слагаемое }

      S:=S+Y     { Увеличили сумму на текущее слагаемое }

   end;          { Тело цикла завершилось.
                   После этой строки компьютер перейдет
                   на оператор while для сравнения
                   переменной "эпсилон" с только
                   что добавленным слагаемым }
```

продолжение ⇩

Пример 7.1 (*продолжение*)

```
    { Теперь выведем результат на экран }
    writeln('Сумма чисел=', S:6:4);
    readln
end.
```

Задание 7.1. Вычислить сумму ряда:

$$\frac{1}{1^2}+\frac{1}{3^2}+\frac{1}{5^2}+\frac{1}{7^2}+\cdots$$

Вычисления прекращаются по тому же условию, что и в примере 7.1.

Полученная сумма должна быть близка к числу $\pi^2/8$.

Возведение числа в указанную целую степень

Рассмотрим задачу: возвести число *a*, введенное с клавиатуры, в степень *n*.

Задачу будем выполнять за $n + 1$ шаг.

Например, возведем число 2 в степень 3 (2^3):

0 шаг: $2^0 = 1$
1 шаг: $2^1 = 2^0 \cdot 2$ $(1 \cdot 2)$
2 шаг: $2^2 = 2^1 \cdot 2$ $(2 \cdot 2)$
3 шаг: $2^3 = 2^2 \cdot 2$ $(4 \cdot 2)$

Пример 7.2. Возведение числа a, введенного с клавиатуры, в степень n

```
Program Stp;
var P: real;     { Переменная, которая хранит результат
                   очередного шага }

   N: integer;   { Показатель степени }

   i: integer;   { Счетчик числа шагов }

   A: real;      { Основание степени }
begin
   write('Введите основание степени:');
   readln(A);
   write('Введите показатель степени:');
```

```
readln(N);

i:=0;          { 0-й шаг }

P:=1;          { 2^0=1 }

while i<abs(N) do   { Показатель может быть
                      отрицательным, поэтому
                      используем для анализа
                      его абсолютную величину.
                      Если показатель N=0,
                      то в тело цикла
                      не попадаем ни разу,
                      так как 0-й шаг уже сделан }

begin
    i:=i+1;    { Увеличиваем i на 1,
                 то есть i теперь равно номеру
                 текущего шага }

   P:=P*A      { Получаем результат i-го шага,
                 то есть A^i }
end;

{ В переменной P на данный момент получен результат
  для положительного N }

if N<0 then  { Если показатель N – отрицательный, }

   P:=1/P;   { то результат должен иметь
               обратную величину }

writeln('Результат=',P:6:3);
readln
end.
```

Задание 7.2. Используя цикл с предусловием, написать программу вычисления $N!$.

Задание 7.3. Выполните задачу из предыдущей темы (задание 6.8), но используйте для этого цикл с предусловием. Блок-схема алгоритма вывода звездочек в верхней (2-й) строке с 3-го столбца (координата x) до 75-го столбца приведена на рис. 7.3. Продолжите блок-схему. Будьте внимательны с условиями!

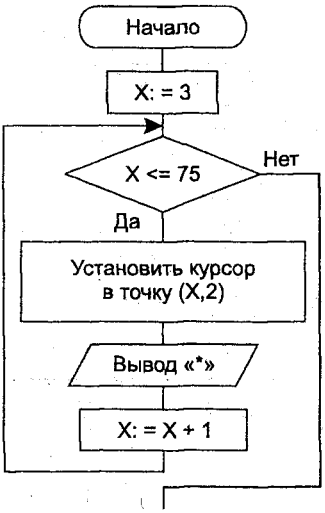


Рис. 7.3. Блок-схема алгоритма вывода звездочек во 2-й строке экрана с 3-й по 75-ю позицию

Обратите внимание: нам понадобилось самим устанавливать значение x до входа в цикл и увеличивать x на 1 в теле цикла! В цикле со счетчиком это все делалось за нас в самой конструкции цикла.

Задание 7.4. Измените в задании 7.3 в теле цикла шаг счетчика, сделав его равным 3.

Задание 7.5. Проведите звездочки по диагонали из нижнего левого угла в верхний правый. Сначала заполните блок-схему алгоритма (рис. 7.4).

Пояснение: координата x изменяется быстрее, чем y, поскольку экран прямоугольный.

Задание 7.6. Выведите в центр экрана с задержкой друг относительно друга следующие числа: 1, 2, 4, 8, 16, 32, 64, 128, 256.

Вероятно, вы поняли, что это степени числа 2.

Заполните блок-схему алгоритма (рис. 7.5), затем напишите программу.

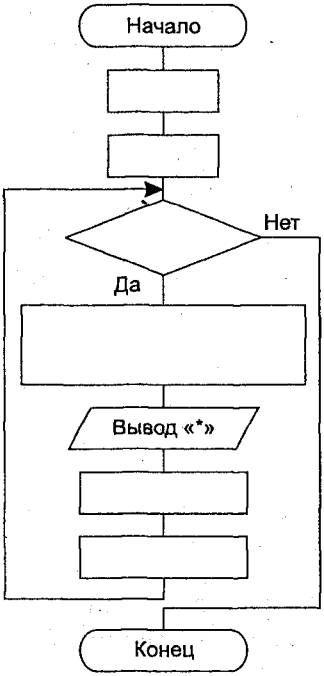


Рис. 7.4. «Слепая» блок-схема алгоритма вывода звездочек по диагонали из левого нижнего в правый верхний угол

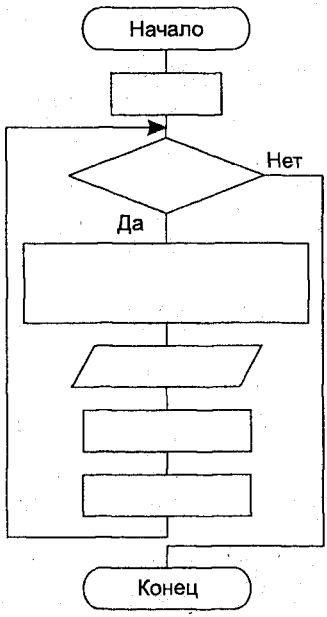


Рис. 7.5. «Слепая» блок-схема алгоритма вывода в центр экрана степеней числа 2 с временной задержкой

Задание 7.7. Введите два числа (например, A = 5 и B = 8) и найдите их произведение, используя только операцию сложения. Заполните блок-схему алгоритма (рис. 7.6).

Задание 7.8. Введите два числа (например, A = 45 и B = 8) и найдите частное от деления нацело первого числа на второе (A на B) (в переменной k) и остаток от деления нацело (в переменной A), используя только операцию вычитания. Заполните блок-схему алгоритма (рис. 7.7).

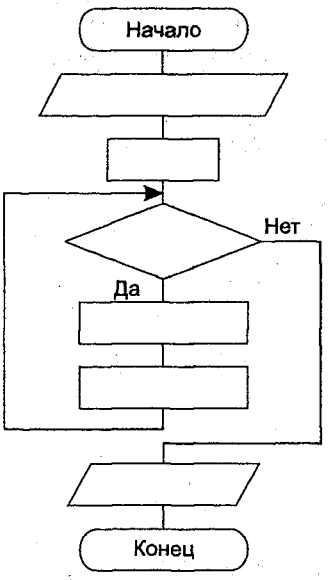


Рис. 7.6. «Слепая» блок-схема алгоритма вычисления произведения двух чисел с использованием только операции сложения

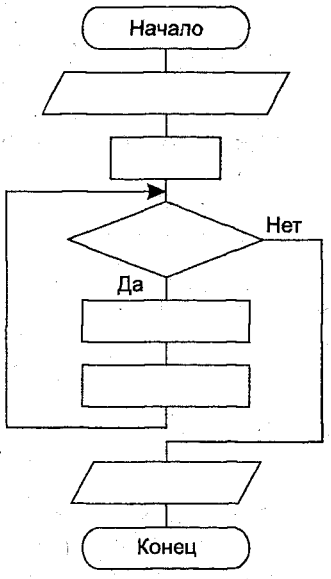


Рис. 7.7. «Слепая» блок-схема алгоритма вычисления целого частного и остатка от деления одного числа на другое с использованием только операции вычитания

Пояснение: в переменной k подсчитывайте, сколько раз сделана операция вычитания, то есть сколько раз число B содержится в числе A.

Урок 7.2. Цикл с постусловием

Вторая разновидность цикла проверяет условие после выполнения тела цикла. Поэтому правильно будет назвать это условие *условием окончания цикла*. Цикл такого вида называется *циклом с постусловием*.

Цикл будет повторяться до тех пор, пока проверка этого условия будет давать результат «ложь» (false),

то есть пока условие не выполнено. Даже если условие сразу окажется истинным, цикл выполнится хотя бы один раз.

Описание цикла с постусловием

Блок-схема в общем виде выглядит так (рис. 7.8):

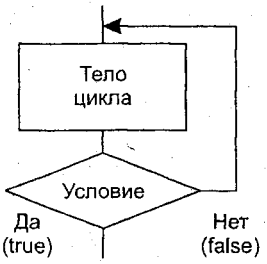


Рис. 7.8. Блок-схема цикла с постусловием

Выполнение цикла продолжается, если проверка логического условия дает результат «ложь». Если логическое условие выполняется, то происходит выход из цикла. Иными словами, если в цикле while проверялось условие продолжения цикла, то в цикле repeat ... until — условие окончания.

На языке Паскаль этот тип цикла реализуется так:

```
repeat
  <тело цикла>
{ операторы begin ... end не требуются! }
until <логическое условие>
```

Использование циклов repeat и while

Рассмотрим задачу, в которой требуется вводить с клавиатуры числа и подсчитывать их сумму. Сумму необходимо подсчитывать до первого введенного отрицательного числа. Блок-схема алгоритма приведена на рис. 7.9.

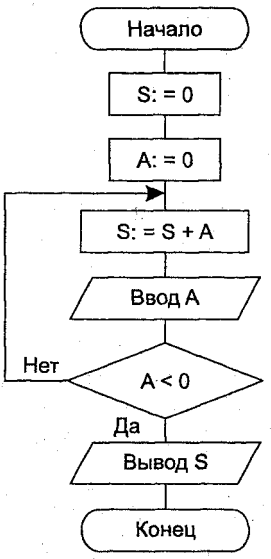


Рис. 7.9. Блок-схема алгоритма подсчета суммы вводимых элементов до первого отрицательного числа на базе цикла с постусловием

Пример 7.3. Использование цикла repeat для подсчета суммы вводимых чисел до первого отрицательного числа

```
Program Summer1;
var
   sum, a: real;   { sum – для накопления суммы,
                     a – для очередного числа }
begin
   sum:=0;         { Обнуляем сумму }

   a:=0;           { Это тактическая хитрость
                     (см. замечание к примеру) }

   repeat

      sum:=sum+a;  { Добавляем введенное число к сумме }

      write('Введите число:'); { Ввод очередного числа }

      readln(a)
```

```
    until a<0;        { Проверяем введенное число
                        на отрицательность }

    { При выходе из цикла выполняется этот оператор: }
    writeln('Сумма чисел=', sum:5:3);
    readln
end.
```

Использование оператора repeat ... until оправдано тогда, когда нужны повторяющиеся действия, от выполнения которых зависит дальнейшее продолжение цикла. Так, в приведенном примере продолжение цикла зависит от введенного числа. Если сразу введено отрицательное число, его не нужно добавлять к сумме. Если число неотрицательное, то нужно добавить его к сумме и продолжить выполнение цикла. Если перевести вышесказанное буквально на язык блок-схем, алгоритм должен выглядеть так (рис. 7.10):

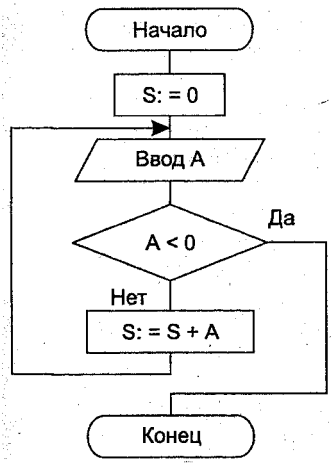


Рис. 7.10. Блок-схема алгоритма подсчета суммы вводимых элементов до первого отрицательного числа с выходом из цикла в середине тела цикла

Однако этот пример цикла нельзя отнести к циклам с пред- или постусловием. Здесь условие окончания

цикла находится в середине. Хитрость примера 7.3 состоит в том, чтобы выполнить оператор S:=S+A до проверки условия окончания цикла и не нарушить правильности алгоритма. Ведь если поставить S:=S+A после ввода переменной A, то введенное отрицательное число добавится к сумме, чего быть не должно. А если поставить его перед вводом переменной A, то что же тогда прибавится к сумме во время первого шага цикла? Ведь переменная A еще не введена! Изначально присвоив переменной A нулевое значение, мы решаем эту проблему.

Вы скажете: если repeat здесь использовать неудобно, не лучше ли использовать while?

Давайте посмотрим, какую программу придется написать в этом случае, и сравним получившиеся два варианта:

Пример 7.4. Использование цикла while для подсчета суммы вводимых чисел до первого отрицательного числа

```
var
   sum, a: real;
begin
   sum:=0;                    { Обнуляем сумму }

   write('Введите число:');   { Ввод первого числа }

   readln(a);                 { Так как мы собираемся
                                проверять переменную A
                                до начала цикла,
                                ей необходимо присвоить
                                начальное значение }

   while a>=0 do              { Проверяем введенное число
                                на отрицательность }

   begin
      sum:=sum+a;  { Добавляем введенное число к сумме }

      write('Введите число:'); { Ввод очередного числа }

      readln(a)
```

```
    end;

    { При выходе из цикла выполняется этот оператор: }
    writeln('Сумма чисел=', sum:5:3);
    readln
end.
```

Необходимость задать начальное значение переменной A вынуждает нас повторить операторы ввода переменной A дважды — до цикла и внутри него. С этой точки зрения использование while оказывается менее удобным.

Относительность выбора операторов while и repeat

Со временем вы поймете, что проверка условия окончания цикла до или после тела цикла — это вопрос исключительно личных предпочтений. В данном случае, например, можно написать эту программу через while и при этом не повторять оператор ввода:

Пример 7.5. Использование цикла while для подсчета суммы вводимых чисел до первого отрицательного числа без дублирования оператора ввода

```
var
   sum, a: real;
begin
   sum:=0;          { Обнуляем сумму }

   a:=1;            { Используем ту же хитрость,
                      что и в примере 7.3.
                      Это нужно, чтобы
                      значение A удовлетворяло
                      условию while }

   while a>=0 do    { Проверяем введенное число
                      на отрицательность }

   begin
      sum:=sum+a;
      write('Введите число:'); { Ввод очередного числа }
```

продолжение

Пример 7.5 (*продолжение*)

```
    readln(a)          { Добавляем введенное число к сумме }
    end;
    writeln('Сумма чисел=', sum:5:3);
    readln
end.
```

В приведенных примерах должно быть хорошо видно, насколько важен порядок выполнения действий внутри цикла. Достаточно переставить местами операторы — и программа начинает работать совершенно иначе. Для начинающих это самое трудное. Даже если понятно, какие операторы должны выполняться в теле цикла, — но как определить, в правильном ли порядке мы их расставили?

Мы рекомендуем не лениться и всегда использовать один волшебный метод — трассировку! Честно и аккуратно выполнив вручную несколько шагов цикла, можно понять, правильно ли написана программа.

Задание 7.9. Написать программу, которая подсчитывает произведение целых чисел, введенных с клавиатуры. Произведение подсчитывается до тех пор, пока вводятся числа в интервале от –10 до +10. Используйте цикл с постусловием.

Подсказка: в записи условия используйте логическую операцию.

Задание 7.10. Выполните ту же задачу, но с использованием цикла с предусловием.

Задание 7.11. Написать программу «Угадай-ка»:

С использованием датчика случайных чисел в программе загадывается число в диапазоне 0...100. На отгадывание числа дается 10 попыток. Играющий вводит каждый раз очередное число. После каждого ответа программа выводит на экран одно из сообщений — «больше», «меньше» или «угадано», в зависимости от числа, введенного пользователем. Цикл завершается при вы-

полнении одного из двух условий: либо число попыток достигло 10, либо дан правильный ответ.

Подсказки:

1. Каждый раз в цикле наращивается переменная k, содержащая счетчик попыток.
2. Для отслеживания правильного ответа введите логическую переменную flag, которой первоначально следует присвоить значение False. Если ответ верен, присвойте этой логической переменной значение True.
3. Цикл завершается, если значение счетчика попыток k равно 10 или если логическая переменная имеет значение True (использовать операцию логическое «или» — or).
4. При выходе из цикла надо сообщить, угадано ли число или же выход из цикла произошел по совершении 10 попыток. Для этого надо проверить значение логической переменной flag («истина» или «ложь»).

Сначала заполните блок-схему алгоритма (рис. 7.11).

Задание 7.12. Введите два числа (например, A = 5 и B = 8) и найдите их произведение, используя только операцию сложения. Нарисуйте блок-схему алгоритма, используя цикл с постусловием.

Задание 7.13. Введите два числа (например, A = 45 и B = 8) и найдите частное от деления нацело (в переменной k) и остаток от деления нацело (в переменной A), используя только операцию вычитания. Нарисуйте блок-схему алгоритма, используя цикл с постусловием.

Задание 7.14. Перед вами блок-схема алгоритма подсчета количества десятичных разрядов в заданном положительном числе N (рис. 7.12). Заполните таблицу трассировки и докажите, что в переменной k мы дей-

ствительно получаем количество разрядов (в нашем случае их 4).

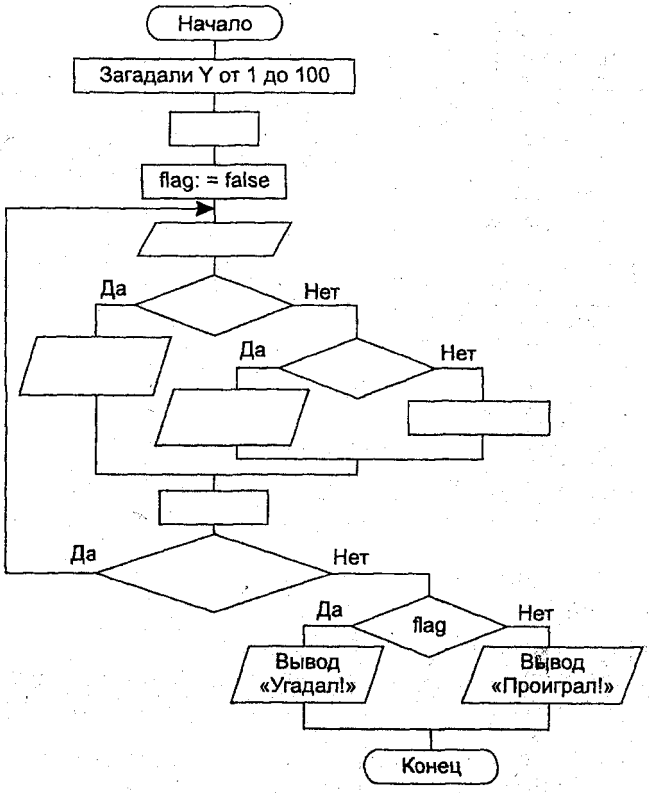


Рис. 7.11. «Слепая» блок-схема алгоритма «Угадайка» (см. задание 7.11)

Таблица 7.1. Таблица для трассировки алгоритма, приведенного на рис. 7.12 (задание 7.14)

Оператор	Условие	N	k	Примечание
Ввод N		1024		

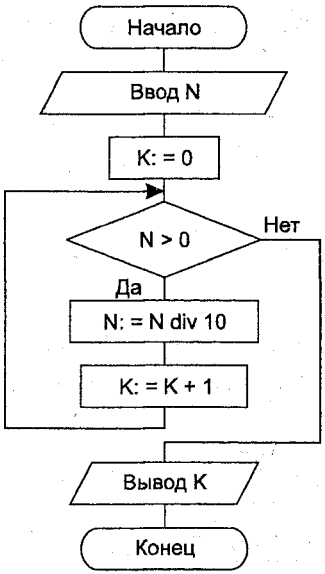


Рис. 7.12. Блок-схема алгоритма подсчета количества десятичных разрядов в заданном положительном числе N

Напишите программу и проверьте алгоритм для других значений N.

Задание 7.15. Перед вами блок-схема алгоритма подсчета суммы десятичных разрядов в заданном положительном числе N (рис. 7.13). Заполните таблицу трассировки и докажите, что в переменной S мы действительно получаем сумму разрядов (в нашем случае сумма равна 14).

Таблица 7.2. Таблица для трассировки алгоритма, приведенного на рис. 7.13 (задание 7.15)

Оператор	Условие	N	S	Примечание
Ввод N		4235		

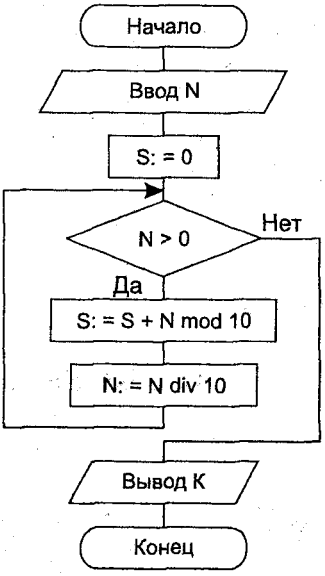


Рис. 7.13. Блок-схема алгоритма подсчета суммы десятичных разрядов в заданном положительном числе N

Напишите программу и проверьте алгоритм для других значений N.

Полезным примером цикла repeat ... until является использование библиотечной функции KeyPressed.

KeyPressed — библиотечная функция (из модуля CRT), которая, отработав, возвращает в программу результат в виде логического значения. Первоначально результат равен False. Как только будет нажата любая клавиша на клавиатуре, возвращаемый результат станет равным True. Таким образом, цикл вида repeat until KeyPressed будет выполняться, пока не будет нажата какая-либо клавиша.

Обычно это используется для получения задержки в программе — например, для просмотра результата на экране. Тело цикла пусто, цикл сводится только к проверке условия KeyPressed.

Выводы

1. Для организации многократно повторяющихся действий с неизвестным числом повторений используется оператор цикла с предусловием:
   while <логическое условие> do <оператор>
2. Выполнение цикла while прекращается, как только логическое условие примет значение false.
3. Цикл с предусловием (while) может не выполниться ни разу.
4. Для выполнения той же задачи служит цикл с постусловием:
   repeat
   <один или несколько операторов>
   until <логическое условие>
5. Выполнение цикла repeat прекращается, как только логическое условие примет значение true.
6. Цикл с постусловием (repeat) будет выполнен хотя бы один раз.
7. При использовании нескольких операторов в теле цикла repeat ... until операторная скобка (begin ... end) не нужна, так как пара repeat ... until сама является операторной скобкой.

Контрольные вопросы

1. В каких случаях предпочтительнее использовать оператор цикла for, а в каких — операторы цикла с условием?
2. Чем проверка условия выполнения цикла while отличается от проверки в цикле repeat ... until?
3. Что будет на экране в результате выполнения следующих фрагментов программ? (Переменные описаны для всех фрагментов.)

```
var
  k,x,i: integer;
begin

  { Фрагмент 1 }
  k:=256;
  while k<>1 do
    begin
      write(k:4);
      k:=k div 2
    end;

  { Фрагмент 2 }
  k:=5;
  repeat
    writeln(k);
    k:=k+5
  until k>30;

  { Фрагмент 3 }
  for i:=1 to 5 do
    writeln(i*5);

  { Фрагмент 4 }
  for i:=8 downto 4 do
    write(i:3);

  { Фрагмент 5 }
  for x:=1 to 79 do
  begin
    gotoxy(x,8);
    write('*');
    gotoxy(x,15);
    write('*')
  end;
end.
```

ТЕМА 8

Массивы — структурированный тип данных

В предыдущих уроках мы с вами рассматривали задачи, в которых использовалось небольшое количество данных. Для каждого мы создавали отдельную ячейку памяти с уникальным именем. Однако зачастую приходится работать с большим количеством однотипных данных. Им такие требуются отдельные ячейки памяти — это понятно и естественно (два литра воды не поместятся в один стакан). А вот указывать для каждой ячейки отдельное имя — неудобно. Как быть?

Один из методов решения такой: выделим для этих данных область последовательных ячеек памяти и назовем всю область общим именем. А для того, чтобы к каждой ячейке можно было обратиться, пронумеруем ячейки по порядку. Таким образом, для обращения к определенной ячейке нужно указать название всей конструкции и номер ячейки в ней.

Урок 8.1. Хранение однотипных данных в виде таблицы

Массив — совокупность однотипных данных, хранящихся в последовательных ячейках памяти и имеющих общее имя. Ячейки называются *элементами массива*. Все элементы пронумерованы по порядку, и этот номер называется *индексом элемента массива*.

Все элементы массива имеют один и тот же тип. Сам массив при этом имеет имя — одно для всех элементов. Для обращения к конкретному элементу массива необходимо указать имя массива и (в квадратных скобках) индекс элемента.

Простейший вид массива — одномерный массив (рис. 8.1).

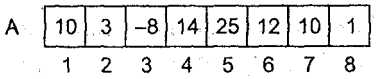


Рис. 8.1. Условное зображение одномерного массива в виде строки

A — имя массива, числа в клетках таблицы — элементы массива.

Рассмотрим запись A[3] = -8. В этой записи:

- A — имя массива,
- 3 — номер элемента массива (индекс),
- A[3] — обозначение 3-го элемента массива,
- -8 — значение 3-го элемента массива.

Основные действия по работе с массивами

Нам предстоит научиться выполнять ряд наиболее распространенных действий с массивами:

- описание;
- заполнение массива случайными числами;
- заполнение массива с клавиатуры;
- вывод на экран;
- поиск максимального элемента;
- вычисление суммы всех элементов массива;
- вычисление количества положительных элементов.

Описание массива на языке Паскаль

<Имя массива>: array [<тип индекса>] of <тип компонентов>;

Здесь <тип компонентов> — это тип данных, который имеет каждый элемент массива, а <тип индекса> — границы изменения индекса.

Например:

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

Здесь тип индекса — интервальный, изменяется в интервале от 1 до 10, тип данных (элементов массива) — целый.

Заполнение массива случайными числами и вывод массива на экран

Рассмотрим задачу, в которой требуется с помощью датчика случайных чисел создать одномерный массив и вывести его на экран. Блок-схема алгоритма показана на рис. 8.2.

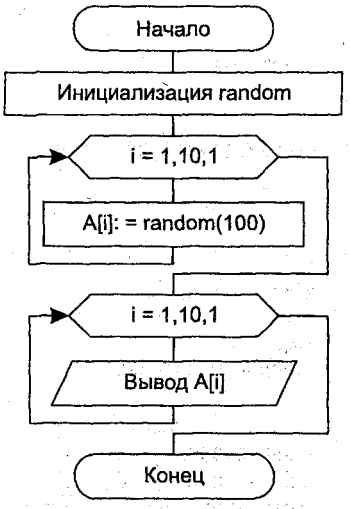


Рис. 8.2. Блок-схема алгоритма заполнения одномерного массива случайными числами и вывода массива на экран

Пример 8.1. Основные действия по работе с массивами

```
Program Massiv1;
uses Crt;
const       { Раздел описания констант,
              то есть постоянных величин,
              определяемых в программе заранее
```

```
                и не изменяющихся
                по ходу выполнения программы }

   N=10;    { Имена констант не используются для
              имен переменных величин (из раздела var) }
var
   A: array [1..N] of integer; { 1..N – тип индекса.
                                 Для индекса выбран
                                 интервальный тип,
                                 то есть интервал целых
                                 чисел от 1 до N,
                                 где N определено
                                 в разделе const }

   i:integer;    { Переменная, хранящая индекс элемента
                   массива, к которому идет обращение }
begin

{ II. Задание значений элементов массива
      как случайных чисел }

   Randomize; { Инициализация датчика случайных чисел }

   { Задание элементов массива: }

   for i:=1 to N do        { Переменная i изменяется
                             в цикле от 1 до N,
                             то есть мы по очереди
                             перебираем
                             все элементы массива }

      A[i]:=Random(100);   { В очередной элемент массива
                             A[i] записываем
                             случайное число от 0 до 99.
                             обратите внимание: i – номер
                             элемента массива (принято
                             говорить "индекс"), A[i] -
                             значение элемента массива }

{ III. Вывод элементов массива на экран в одну строку }
   ClrScr;
   writeln('Введенный массив:');
   for i:=1 to N do
```

продолжение ☞

Пример 8.1 (*продолжение*)

```
        write(A[i]:4);      { На каждый элемент массива
                              выделяется по 4 позиции
                              строки, чтобы они
                              не склеивались при выводе! }

    writeln;              { Этот "пустой" оператор вывода
                            отработает только один раз
                            и переведет курсор
                            на новую строку
                            для дальнейшей работы }
    readln
end.
```

В данном примере мы заполнили массив случайными числами от 0 до 99. Это обеспечила нам функция random(100). А если нам нужно получить случайные числа в другом диапазоне — например, не от нуля? Расчет нужно сделать такой: функция random(N) выдаст N различных чисел от 0 до N – 1. Если нам нужно, чтобы наименьшим числом диапазона было K, значит, необходимо прибавить это K к random(N). Наибольшее число, которое будет выдавать в этом случае формула random(N) + K, будет наибольшим числом диапазона.

Пусть, например, нам требуются случайные числа в диапазоне –100...+100. Считаем, сколько различных чисел в этом диапазоне: 100 положительных, 100 отрицательных и ноль. Итого 201. Формула тут проста: вычесть из большего меньшее и прибавить 1. Значит, N = 201, а K = –100. То есть получаем формулу random(201) - 100.

ЗАМЕЧАНИЕ

К сожалению, в таком виде формула работать не будет — при запуске программа «вылетит» с сообщением об ошибке. Это от «излишнего ума», который проявляет здесь среда Паскаль. Дело в том, что Паскаль считает тип этого выражения по функции random. *А она имеет*

тип word. *Иными словами, беззнаковый. При попытке вычесть 100 из числа, меньшего 100, получаем отрицательный результат, что Паскаль не устраивает. Самый простой способ обойти эту напасть — поменять местами уменьшаемое (*random*) и вычитаемое, то есть написать:* –100 + random(201). *Тогда Паскаль будет считать тип этого выражения как* integer *по первому числу (*-100*), и ошибки не возникнет.*

Задание 8.1. Оформите эту программу так, чтобы задание массива и вывод его элементов на экран выполнялись в одном цикле. Вам понадобится составной оператор для тела цикла begin ... end.

Задание 8.2. Добавьте в программу задания 8.1 новый цикл вывода элементов массива в обратном порядке (начиная с последнего). Попробуйте выполнить то же задание без введения нового цикла.

ЗАПОМНИТЕ!

*Массив — это множество ячеек памяти. Поэтому любое действие с массивом заключается в том, чтобы перебрать все эти ячейки или, по крайней мере, какую-то их часть. Это значит, что любое действие с массивами должно содержать в себе цикл, в котором перебираются элементы массива. Если вы пишете программу с массивом и не написали цикла (*for, while *или* repeat*) — значит, вы ошиблись.*

Создание пользовательского типа данных

В следующем примере массив описывается в новом разделе — разделе описания типов пользователя (type). Вы можете по-прежнему пользоваться описанием массива в разделе описания переменных (как в примере 8.1).

Вариант описания массива, приведенный в этом примере, в большей степени соответствует грамотному стилю оформления программы.

Пример 8.2. Ввод с клавиатуры одномерного массива целых чисел и вывод его элементов на экран с противоположным знаком

```
Program Massiv2;
const
   N=10;

type            { Раздел описания типов переменных.
                  Эти типы определяет сам пользователь,
                  то есть мы определяем тип одномерного
                  массива из n целых чисел }

   Mas=array [1..N] of integer;   { 1..N – тип индекса;
                  для индекса выбран интервальный тип,
                  то есть интервал целых чисел от 1 до N,
                  где N определено в разделе const }

var
   Line:Mas;  { Line – одномерный массив,
                  его тип определен нами как Mas }

   i:integer; { Переменная, хранящая индекс элемента
                  массива, к которому идет обращение }

begin

{ IV. Ввод массива с клавиатуры }

   for i:=1 to N do   { Обращение к элементам массива
                        происходит в цикле, по очереди }

   begin      { Начало цикла ввода элементов массива }

      write('Введите элемент с индексом ',i,':');

      readln(Line[i]) { Обращаемся к i-му
                        элементу массива
                        (Line[1],Line[2] и т. д.) }
```

```
  end;          { Конец цикла ввода элементов массива }

  { Вывод элементов происходит также в цикле:}

  for i:=1 to N do   { Перебираем все N элементов
                       массива }

     write(-Line[i]:5);   { 10 элементов выводятся
                            в строку. Выводим все
                            элементы массива
                            с противоположным знаком }

  writeln;  { После вывода массива элементов -
              переход на новую строку }
  readln
end.
```

Заметьте, в задании не требовалось, чтобы знак всех элементов массива менялся на противоположный. Требовалось лишь вывести их в таком виде на экран. Если бы требовалось изменить сам массив, обязательно нужно было бы сделать примерно следующее:

```
For i:=1 to N do
  Line[i]:=-Line[i];
```

Очень важно понимать: то, что мы видим на экране, не всегда есть то, что действительно хранится в памяти. Программист может играть здесь роль фокусника. Важно лишь, чтобы задание было точно выполнено.

То есть если в задании требуется изменить каким-либо образом данные и вывести результат на экран, а программист просто вывел данные на экран в нужном виде, то такое задание не может считаться выполненным.

Например, в задании требуется поставить все элементы массива в обратном порядке и вывести результат на экран. Программист решил, что проще всего будет просто вывести на экран все элементы, начиная с последнего. Такое решение не засчитывается!

Задание 8.3. Выполнить следующие действия:

1) создать одномерный массив A из 10 целых чисел (с помощью датчика случайных чисел);
2) вывести массив на экран в виде строки чисел;
3) подсчитать сумму элементов массива (блок-схема алгоритма показана на рис. 8.3);
4) вывести сумму на экран.

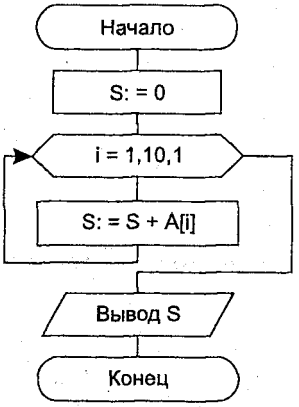


Рис. 8.3. Блок-схема алгоритма вычисления суммы элементов одномерного массива

Задание 8.4. Выполнить следующие действия:

1) создать одномерный массив A из 10 целых чисел;
2) вывести массив на экран в виде строки чисел;
3) поменять местами элементы массива (блок-схема алгоритма показана на рис. 8.4) следующим образом:
   1-й элемент — со 2-м,
   3-й — с 4-м,
   5-й — с 6-м,
   7-й — с 8-м,
   9-й — с 10-м;

(надо вспомнить, как идет обмен значениями двух переменных (рис. 2.4));

4) вывести измененный массив на экран.

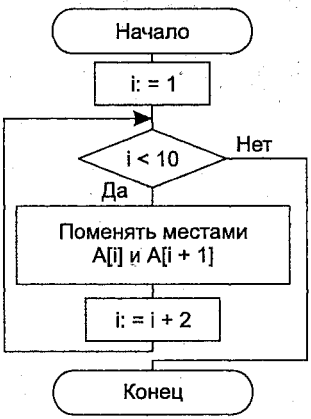


Рис. 8.4. Блок-схема алгоритма обмена соседних элементов одномерного массива

Поиск максимального элемента массива

Довольно-таки типичная задача для большого количества данных — поиск максимума. Например, в списке успеваемости учеников класса найти самого прилежного. Иначе говоря, требуется выбрать наибольшее значение среднего балла и указать фамилию ученика.

Пример 8.3. Программа поиска максимального элемента в массиве и его индекса (см. блок-схему алгоритма на рис. 8.5)

```
Program Maximum;
const
   N=10;
type
   Mas=array [1..N] of integer;
var
   A:Mas;
```

продолжение ☞

Пример 8.3 (*продолжение*)

```
    i   :integer;      { Счетчик цикла }

    Max :integer;      { Переменная для хранения
                         величины максимального элемента }

    Imax:integer;      { Переменная для хранения
                         индекса максимального элемента }

begin                  { Тело программы }

   { Заполним элементы массива значениями датчика
     случайных чисел и выведем весь полученный массив
     на экран в одном цикле }
   Randomize;
   for i := 1 to N do
   begin
      A[i]:=-50+Random(101);
      write(A[i]:5)
   end;
   writeln;

{ V. Поиск максимального элемента
     и его индекса в массиве }

   Imax:=1;            { Сначала считаем,
                         что первый элемент массива
                         и есть максимальный }

   Max:=A[1]; { Его индекс и величину
                записываем соответственно
                в переменные Imax и Max }

   for i := 2 to N do   { Сравним нашего кандидата
                          в максимумы со всеми
                          остальными элементами массива
                          (со второго до последнего) }

      if Max < A[i] then { Если наш кандидат
                           в максимумы оказался
                           меньше текущего элемента... }

      begin
         Max:=A[i];       { ... то будем считать теперь
                            кандидатом в максимумы
```

```
                                текущий элемент }
        Imax:=i                 { Запомним его значение
                                  и индекс
                                  в переменных Max и Imax }
      end;
    writeln('Максимальный элемент в массиве=',Max:5);
    writeln('Его индекс=',Imax:5);
    readln
end.
```

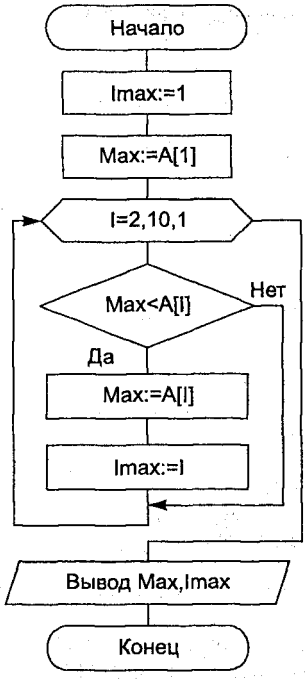


Рис. 8.5. Блок-схема алгоритма поиска максимального элемента массива и его индекса

Заметим, что в процессе поиска максимума не обязательно хранить обе величины — номер максимума и его значение. Достаточно хранить одну, в зависимости от поставленной задачи.

Если индекс максимума не нужно знать, достаточно будет переменной Max. Если, наоборот, нужен только номер — достаточно Imax. Тонкость состоит в том, что если нужно найти и то и другое, все равно достаточно найти только Imax, ведь значение максимума легко может быть получено по его индексу (A[Imax]).

Иными словами, нашу программу можно упростить следующим образом:

Пример 8.4. Программа поиска максимума, не хранящая значение максимума, а запоминающая только его номер

```
Program Maximum2;
const
   N=10;
type
   Mas=array [1..N] of integer;
var
   A       :Mas;
   i, imax  :integer;
begin
   Randomize;
   for i := 1 to N do
   begin
      A[i]:=-50+Random(101);
      write(A[i]:5)
   end;
   writeln;

{ V. Поиск индекса максимального элемента в массиве }
   imax:=1;
   for i := 2 to N do
      if A[imax] < A[i] then
         imax:=i;
   writeln('Максимальный элемент в массиве=',A[imax]:5);
   writeln('Его индекс=',imax:5);
   readln
end.
```

Задание 8.5. Выполните поиск максимального и минимального элемента в массиве за один цикл (блок-схема алгоритма показана на рис. 8.6).

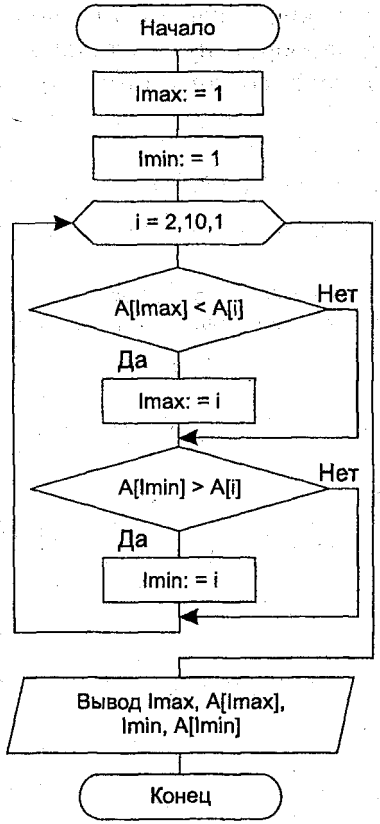


Рис. 8.6. Блок-схема алгоритма поиска индексов максимального и минимального элементов массива за один цикл

Задание 8.6. В одномерном массиве из 10 элементов определить местоположение минимального элемента. Обнулить элементы, стоящие до него, но не сам этот элемент. (Обнулить — значит записать 0 на место элемента, то есть выполнить A[i] := 0.) Измененный массив вывести на экран.

Задание 8.7. В одномерном массиве из 10 элементов определить местоположение минимального и мак-

симального элементов. Обнулить элементы, стоящие между ними, а также сами эти элементы.

Вычисление суммы и количества элементов массива с заданными свойствами

Еще одна задача — посчитать сумму элементов, которые удовлетворяют какому-то условию. Наверное, вы уже поняли общий принцип: перебираем все элементы массива (цикл for) и проверяем для каждого элемента выполнение условия (оператор if). Если условие выполнено, добавим элемент к сумме (S := S+A[i]).

Пример 8.5. Вычисление суммы положительных элементов массива

```
program PositivSumm;
const   N=10;
type    Mas=array [1..N] of integer;
var a:Mas;
    i:integer;      { Счетчик цикла }

    S:integer;      { Копилка – переменная
                      для суммирования
                      положительных элементов }
begin
   { Заполним массив случайными числами
     в диапазоне -100..+100 }
   randomize;
   for i:=1 to N do
   begin
      a[i]:=-100+random(201);
      write(a[i]:5)
   end;
   writeln;

   { Присвоим переменным начальные значения }

   S:=0;            { Переменная S – аккумулятор.
                      Она будет накапливать сумму
                      всех положительных элементов.
                      Нужно присвоить ей такое
                      начальное значение, чтобы оно
```

```
                не повлияло на результат
                суммирования. Таким числом
                является ноль }

  for i:=1 to N do { Перебираем все элементы массива }

     if A[i]>0 then   { Проверяем каждый элемент
                        на положительность }

        S:=S+A[i];    { Если элемент положительный,
                        добавляем значение элемента
                        к аккумулятору }

  { Выводим результат на экран: }
   writeln('Сумма положительных элементов=',S);
   readln
end.
```

Вот задача, очень похожая на предыдущую: посчитать количество элементов массива с указанными свойствами. Решается аналогично: переберем все элементы, проверим для каждого условие и, если оно выполнено, увеличим счетчик на единицу (k := k+1).

Пример 8.6. Вычисление количества четных элементов массива (блок-схема алгоритма показана на рис. 8.7)

```
program EvenCount;
const   N=10;
type    Mas=array [1..N] of integer;
var a:Mas;
    i:integer;     { Счетчик цикла }

    K:integer;     { Копилка – переменная для подсчета
                     количества четных элементов }
begin
   { Заполним массив случайными числами
     в диапазоне +10..+100 }
   randomize;
   for i:=1 to N do
   begin
      a[i]:=+10+random(91);
      write(a[i]:5)
   end;
```

продолжение ⇩

Пример 8.6 (*продолжение*)

```
    writeln;

    { Присвоим переменным начальные значения }

    K:=0;           { Переменная K – счетчик.
                      Она будет выполнять ту же функцию,
                      что и пальцы на руке при счете:
                      каждый раз, когда будет встречаться
                      четное число, будем загибать один
                      палец, то есть увеличивать
                      переменную K на единицу.
                      В начале нужно присвоить ей
                      такое значение, чтобы оно
                      не повлияло на результат
                      суммирования. Таким числом
                      является ноль }

    for i:=1 to N do  { Перебираем все элементы массива }

       if a[i] mod 2 = 0 then { Проверяем каждый элемент
                                на четность, то есть
                                проверяем, что остаток
                                от деления этого элемента
                                массива на 2 равен нулю }

          K:=K+1;  { Если элемент четный, увеличиваем
                     счетчик на единицу } 

    { Выводим результат на экран: }
     writeln('Количество четных элементов=',K);
     readln
end.
```

Урок 8.2. Поиск в массиве

Теперь рассмотрим еще ряд задач, которые приходится решать при работе с массивами, а именно задачи поиска. На примере задачи поиска отрицательного элемента мы рассмотрим несколько методов, применяемых для поиска в массиве.

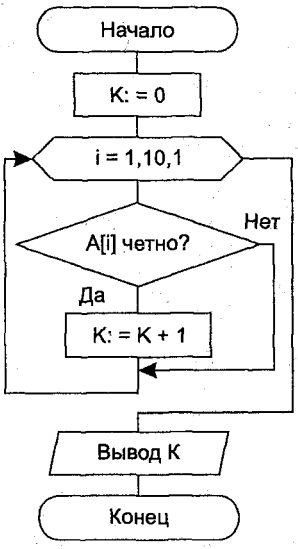


Рис. 8.7. Блок-схема алгоритма вычисления количества четных элементов массива

Определение наличия в массиве отрицательного элемента с использованием флажка

Рассмотрим задачу: определить, есть ли в массиве хотя бы один отрицательный элемент.

На первый взгляд задача кажется весьма простой: достаточно перебрать все элементы и проверить каждый на отрицательность. Это правильно. Но что же делать дальше? И как определить, что в массиве вообще нет отрицательных элементов? А если их несколько?

Данная задача может быть решена несколькими способами. Первый — самый простой — использовать флажок.

Пример 8.7. Определение наличия в массиве отрицательного элемента с использованием флажка

```
program Search1;
const  N=10;
```

продолжение ⇗

Пример 8.7 (*продолжение*)

```
type   mas=array [1..N] of integer;
var  a:mas;
     i:integer;
     fl:boolean;     { Флажок указывает
                       на успешность поиска }
begin
   { Заполним массив случайными числами }
   randomize;
   for i:=1 to N do
   begin
     a[i]:=-2+random(20);
     write(a[i]:4)
   end;
   writeln;

   { Начало блока поиска }
   { Первый вариант поиска отрицательного элемента –
     использование флажка }

   fl:=false;     { Изначально флажок = false,
                    так как мы еще ничего не нашли }
   for i:=1 to N do
      if a[i]<0 then { Проверяем каждый элемент
                       на отрицательность }

         fl:=true;   { Если нашли отрицательный,
                       устанавливаем флажок
                       в состояние "истина" }
   if fl then
      writeln('В массиве есть отрицательный элемент')
   else
      writeln('В массиве нет отрицательных элементов');

   { Конец блока поиска }

   readln
end.
```

Определение наличия в массиве отрицательных элементов путем вычисления их количества

Нетрудно заметить, что использование логического флажка сужает функциональность метода. Ведь мы опреде-

ляем только наличие отрицательного элемента, и ничего более. Если вместо флажка использовать целое число, оно сможет сообщить нам нечто большее о результатах поиска.

Пример 8.8. Определение наличия в массиве отрицательных элементов путем подсчета количества таких элементов

```
program Search2;
const  N=10;
type   mas=array [1..N] of integer;
var a:mas;
   i:integer;
   fl:integer; { Это уже не флажок, а счетчик количества
                 найденных отрицательных элементов }
begin
   { Заполним массив случайными числами }
   randomize;
   for i:=1 to N do
   begin
     a[i]:=-2+random(20);
     write(a[i]:4)
   end;
   writeln;

   { Начало блока поиска }
   { Второй вариант поиска отрицательного элемента -
     количество отрицательных элементов }

   fl:=0; { Изначально счетчик = 0,
            так как мы еще ничего не нашли }

   for i:=1 to N do
      if a[i]<0 then { Проверяем каждый элемент
                       на отрицательность }

         inc(fl); { Если нашли отрицательный -
                    увеличиваем счетчик на единицу }

   writeln('Количество отрицательных элементов=',fl);
   if fl>0 then
      writeln('В массиве есть отрицательный элемент')
   else
```

продолжение ⇩

Пример 8.8 (*продолжение*)

```
        writeln('В массиве нет отрицательных элементов');

    { Конец блока поиска }

    readln
end.
```

Нахождение номера отрицательного элемента массива

Мы только что рассмотрели перебор элементов массива в поисках элементов с какими-то свойствами (в нашем случае отрицательных). Можно также использовать этот метод для поиска номера отрицательного элемента.

Пример 8.9. Определение наличия в массиве отрицательного элемента путем вычисления его номера

```
program Search3;
const  N=10;
type   mas=array [1..N] of integer;
var a:mas;
    i:integer;
    fl:integer;      { Индекс найденного
                       отрицательного элемента }
begin
   { Заполним массив случайными числами }
   randomize;
   for i:=1 to N do
   begin
     a[i]:=-2+random(20);
     write(a[i]:4);
   end;
   writeln;

   { Начало блока поиска }
   { Третий вариант поиска отрицательного элемента -
     нахождение индекса отрицательного элемента }

   fl:=0; { Изначально флажок (индекс) = 0,
            так как мы еще ничего не нашли }
```

```
for i:=1 to N do

   if a[i]<0 then  { Проверяем каждый элемент
                     на отрицательность }

      fl:=i;       { Если нашли отрицательный,
                     запоминаем его номер }

{ Теперь по значению переменной fl можно определить,
  был ли в массиве хоть один отрицательный элемент.
  Если fl остался равен нулю, значит проверка на
  отрицательность ни разу не выполнилась }

if fl>0 then
   writeln('Индекс отрицательного элемента=',fl)
else
   writeln('В массиве нет отрицательных элементов');

{ Конец блока поиска }

readln
end.
```

Возможно, вы уже задаетесь вполне резонным вопросом: а если в массиве несколько отрицательных элементов, то какой из них мы нашли?

Так как при нахождении отрицательного элемента цикл не заканчивается, ответ очевиден: последний. Этот метод находит номер последнего в массиве отрицательного элемента.

А как нам найти первый отрицательный элемент?

Один из методов (на данный момент самый легкий) — исправить цикл так, чтобы он перебирал элементы с конца:

```
for k := N downto 1 do...
```

Теперь рассмотрим последнюю придирку к нашему алгоритму. Если нам нужно найти номер первого отрицательного элемента, то зачем перебирать все элементы? Достаточно остановиться, как только будет найден первый такой элемент.

Эти рассуждения подталкивают нас заменить цикл for, который перебирает все элементы, циклом while, который остановится в нужный нам момент:

```
k := 1;
while a[k] >= 0 do { Перебираем все не подходящие нам
                     (то есть неотрицательные) элементы }
 inc(k); { Берем номер следующего элемента }
```

Лирическое отступление. В только что приведенном примере мы использовали оператор inc, который раньше не упоминали. Он увеличивает значение указанной переменной на единицу. То есть оператор inc(k) аналогичен оператору k := k + 1. По аналогии с inc, в Паскале имеется еще оператор dec. Он уменьшает значение указанной переменной на единицу. Мы не привели эти операторы ранее, чтобы преждевременно не забивать ваши головы излишней информацией. Вы можете ими не пользоваться и продолжать писать k := k + 1 и k := k - 1, как и ранее, вместо inc(k) и dec(k).

Рассмотренный метод поиска первого отрицательного элемента обладает одним серьезным недостатком: он не останавливается, если в массиве вообще нет отрицательных элементов. Перебрав все элементы массива, цикл продолжит искать отрицательные элементы дальше. Это приведет или к зависанию компьютера, или к аварийному завершению программы в зависимости от настроек. Исправим ошибку. Для этого добавим еще одно условие окончания поиска — «если мы перебрали все элементы».

Пример 8.10. Определение наличия в массиве отрицательного элемента и вычисление его номера (если такой есть)

```
program Search4;
const  N=10;
type   mas=array [1..N] of integer;
var a:mas;
    i:integer;  { Здесь нам не нужен флажок –
                  счетчик цикла будет нашим флажком }
```

```
begin
  { Заполним массив случайными числами }
  randomize;
  for i:=1 to N do
  begin
    a[i]:=-2+random(20);
    write(a[i]:4)
  end;
  writeln;

  { Начало блока поиска }
  { Четвертый вариант поиска
    номера первого отрицательного элемента }

  i:=1;          { Так как используем цикл while вместо
                   for, приходится самим заботиться
                   о счетчике цикла }

  while (a[i]>=0) and (i<=N) do { Перебираем все
                                  не подходящие нам
                                (неотрицательные) элементы.
                                 Кроме того, проверяем
                                 выход за границы массива }

    inc(i);    { Берем номер следующего элемента }

  { Цикл закончен. Проверим, по какому из двух условий
    это произошло }
  if i>N then

    { Если в результате поиска
      вышли за границы массива -
      значит, отрицательного элемента нет }
    writeln('Индекс отрицательного элемента=',i)

  else
    writeln('В массиве нет отрицательных элементов');

  { Конец блока поиска }

  readln
end.
```

Задание 8.8. Определить, есть ли в массиве положительные четные элементы, и если есть, вывести номер последнего из них.

Урок 8.3. Двумерные массивы

На предыдущих уроках мы с вами рассмотрели одномерные массивы. Это означает, что массивы имеют одно измерение — количество элементов. Визуально такие массивы можно представить как строку элементов. Однако наш мир не ограничивается одним измерением. На этом уроке мы рассмотрим массивы, которые можно визуально представить как таблицу.

Двумерный массив — это таблица из однотипных элементов, организованная по строкам и столбцам. Местоположение каждого элемента двумерного массива (матрицы) определяется индексом (номером) строки и индексом (номером) столбца (рис. 8.8).

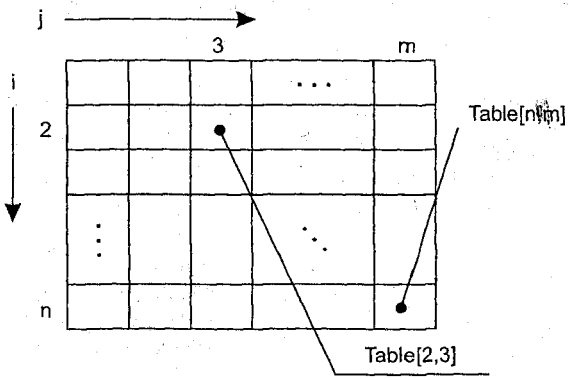


Рис. 8.8. Изображение двумерного массива в виде таблицы

В следующем примере мы создадим матрицу 5×4, задав значения ее элементов с помощью датчика случайных чисел, и выведем ее на экран по строкам. При

этом будем использовать вложенные циклы: внешний цикл будет проходить по строкам, а внутренний — по столбцам. Блок-схема алгоритма представлена на рис. 8.9.

Пример 8.11. Создание матрицы 5 × 4, вывод ее на экран по строкам

```
program Massiv_2;
const
   N = 5;    { Число строк }
   M = 4;    { Число столбцов }
var
   Table : array [1..N, 1..M] of integer; { Заказываем
               область памяти для хранения двумерного
               массива из N строк и M столбцов }

{ Вообще говоря, нигде не определено,
  что первый индекс – это номер строки,
  а второй – это номер столбца.
  Так как выводом на экран занимается программист,
  он сам решает, как ему удобнее.
  Нам удобнее считать, что номер строки – первый индекс,
  а номер столбца – второй }

   i, j : integer;   { Переменные для хранения
                       индексов строки и столбца }
begin
   { Заполнение массива датчиком случайных чисел:}
   randomize;
   for i:=1 to N do
      for j:=1 to M do

         Table[i,j]:=Random(100); { Запись
                                    случайного числа
                                    в массив на место
                                    с номером строки i и
                                    номером столбца j }

   { Вывод матрицы на экран по строкам: }
   for i:=1 to N do
   begin
      for j:=1 to M do
         write(Table[i,j]);
```

продолжение ⇩

Пример 8.11 (*продолжение*)

```
        writeln                { Переход на новую строку после
                                 вывода всех элементов строки i }
    end;
    readln
end.
```

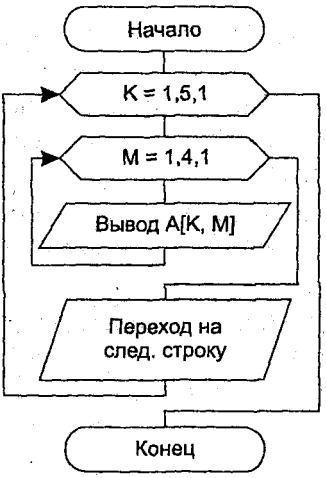


Рис. 8.9. Блок-схема алгоритма вывода двумерного массива 5 × 4 на экран по строкам

Задание 8.9. Написать программу, в которой:

- определить матрицу 3 × 5;
- вывести ее на экран;
- определить величину максимального элемента данной матрицы и вывести на экран его значение и его позицию в матрице.

Выводы

1. Для хранения однотипных данных используется структурированный тип данных — массивы (одномерные и многомерные).

2. Для описания массива необходимо указать его имя, тип данных (array), диапазон изменения индексов его элементов (в квадратных скобках) и тип элементов, из которых он состоит:

```
mas: array [1..20] of integer;
```

3. Обращение к каждому элементу массива идет по имени массива и по индексу (номеру) элемента в массиве.
4. При выполнении любых действий с массивами необходимо использовать циклы, в которых перебираются номера элементов массива.
5. При выполнении операций поиска в массиве нужно перебирать по очереди его элементы и проверять для каждого искомое условие.
6. Примером многомерного массива является двумерный массив (матрица). Обращение к элементам матрицы идет по ее имени, индексу строки и индексу столбца.

Контрольные вопросы

1. Какое условие должно выполняться, чтобы некоторое количество отдельных данных можно было объединить в один массив?
2. Что в записи A[4] = −12 является именем массива, что — индексом, а что — значением элемента?
3. Чем одномерный массив отличается от двумерного?
4. Какие необходимы действия, чтобы вывести на экран все отрицательные элементы массива?
5. Почему при поиске какого-либо элемента в массиве нельзя обойтись без цикла?

ТЕМА 9

Вспомогательные алгоритмы. Процедуры и функции. Структурное программирование

В этой теме мы с вами рассмотрим методы декомпозиции. То есть мы будем учиться разбивать одну большую задачу на несколько отдельных, помельче. Небольшую самостоятельную задачу обычно гораздо легче решить. В этой теме мы познакомимся со способом записи таких подзадач на языке Паскаль.

Урок 9.1. Конструирование алгоритма «сверху вниз»

При конструировании достаточно сложного алгоритма логично разбивать его на ряд более простых задач. Построение алгоритма идет «сверху вниз». Сначала строится основной алгоритм. В нем записываются обращения к вспомогательным алгоритмам, которые позволят решить отдельные, более простые задачи (подзадачи 1-го уровня). Если есть необходимость, осуществляют дальнейшую детализацию, и подзадача разбивается на еще более простые задачи (подзадачи 2-го уровня). Вспомогательные алгоритмы для решения подзадач последнего уровня не содержат обращений к другим вспомогательным алгоритмам (рис. 9.1).

Итак, *вспомогательным алгоритмом* называется алгоритм решения некоторой задачи, являющейся подчиненной по отношению к исходной (основной) задаче. При реализации таких алгоритмов на языке Паскаль их оформляют в виде *процедур* или *функций*.

Согласно концепции структурного программирования, вспомогательный алгоритм должен:

- иметь имя, по которому его можно вызвать из других алгоритмов;

- возвращать управление тому алгоритму, из которого он был вызван. После того как завершится выполнение вспомогательного алгоритма, вызвавший его алгоритм должен продолжить работу с той точки, в которой он был прерван;
- иметь возможность вызывать другие алгоритмы;
- иметь достаточно малые размеры.

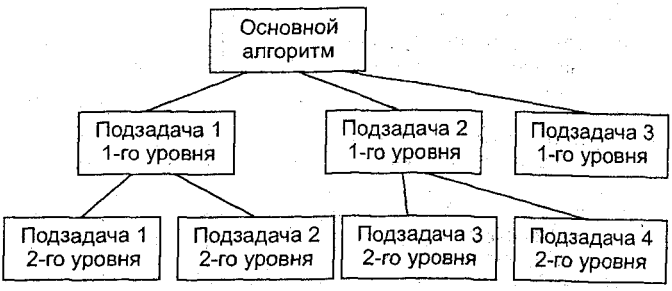


Рис. 9.1. Схема алгоритма, построенного по принципу «сверху вниз»

Практическая задача с использованием вспомогательных алгоритмов

Задача: выполнить с массивом действия, которые были предложены в заданиях 8.3, 8.4 (см. урок 8.1):

а) заполнить одномерный целочисленный массив из 10 элементов случайными числами от −20 до +20;

б) вывести массив на экран в виде строки чисел;

в) подсчитать сумму элементов массива;

г) поменять местами элементы массива следующим образом:

1-й элемент — со 2-м,

3-й — с 4-м,

5-й — с 6-м,

7-й — с 8-м,

9-й — с 10-м;

д) вывести измененный массив на экран.

Разобьем основной алгоритм на подзадачи в порядке их перечисления в задании (рис. 9.2).

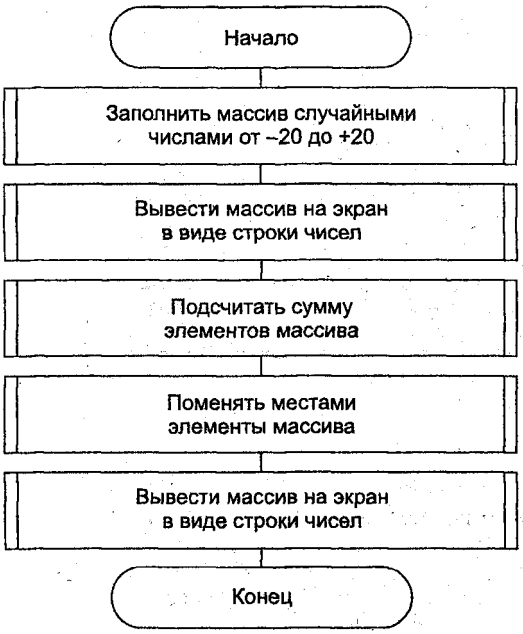


Рис. 9.2. Блок-схема алгоритма решения задачи из примера 9.1

Обратим внимание: пункты б) и д) выполняют одинаковое действие — вывод массива на экран. Поэтому оформим их в виде отдельного фрагмента программы под некоторым именем, а в нужных местах вызовем этот фрагмент по его имени.

Такой фрагмент называется *процедурой*.

Процедура оформляется по тем же правилам, что и программа.

Пункты а) и г) оформим также в виде процедур.

Пункт в) — вычисление суммы элементов массива — оформим отдельным фрагментом, но с возвращением результата вычисления. Такой фрагмент называется *функцией*. В отличие от процедуры, из функции всегда возвращается результат в виде переменной, тип которой указывается в заголовке функции (см. пример 9.1).

Такая организация программы по блокам вспомогательных алгоритмов (процедурам и функциям) придает ей логичную структуру и делает ее более удобной для дальнейшего использования.

Процедуры и функции в программе оформляют в виде отдельных блоков, каждый из которых начинается специальным словом (procedure или function соответственно). Эти блоки должны располагаться в тексте программы перед оператором begin, начинающим основную программу. В каждом блоке процедуры или функции может находиться свой раздел описания переменных (var) и должна быть пара операторов begin ... end, между которыми пишется текст процедуры (или функции).

Внимательно читайте комментарии к программе!

Пример 9.1. Демонстрация процедур и функций на примере работы с одномерным массивом

```
Program Massiv_1;
const
 N=10;
type
 Mas=array [1..N] of integer;

var { Раздел описания переменных.
      Все предыдущие разделы описаний, которые мы
      приводили, включая этот, называются глобальными.
      Константы, типы и переменные из этих разделов
      являются глобальными, то есть действуют в теле
      программы, а также во всех процедурах и функциях,
      описанных в этой программе }
```

```
   Line: Mas;     { Переменная Line – одномерный массив.
                    Его тип определен нами как Mas }

   Sm  :integer; { Переменная для вычисления
                   суммы элементов }

{ Процедура ввода одномерного массива.
  Внимание! После запуска программы управление
  передается первому исполняемому оператору
  из тела программы! Процедура Inp будет выполняться
  только после вызова ее из тела программы! }

Procedure Inp:   { Заголовок процедуры Inp }

var    { Раздел описания локальных переменных,
         то есть переменных процедуры Inp,
         действующих только в пределах этой процедуры }

   i:integer;    { Переменная, хранящая индекс
                   очередного элемента массива,
                   к которому идет обращение.
                   Эта переменная i не имеет
                   никакого отношения
                   к глобальной переменной i.
                   Хотя они имеют одинаковое название,
                   это совершенно разные ячейки памяти.
                   В данной процедуре (Inp)
                   обращение к переменной i
                   будет означать локальную переменную.
                   Это, в частности, значит,
                   что к глобальной переменной i
                   из процедуры обратиться нельзя.
                   В данном случае это и не нужно.
                   Если вы хотите иметь возможность
                   обращаться к обеим переменным,
                   нужно давать им разные имена }

begin             { Начало тела процедуры Inp }
   for i:=1 to N do
   begin
      write('Введите элемент с индексом ',i,':');
      readln(Line[i])
   end
end;              { Конец тела процедуры Inp }
```

продолжение ⇩

Пример 9.1 (*продолжение*)

```
{ Процедура вывода одномерного массива на экран }

Procedure Out;    { Заголовок процедуры Out }
var
   i:integer;     { Рекомендуется описывать локально
                    те переменные, которые используются
                    для промежуточных рабочих значений
                    в процедуре, чтобы избежать ошибок
                    в значениях глобальных переменных.
                    Например, рекомендуется локально
                    описывать счетчики циклов }
begin             { Начало тела процедуры Out }
   for i:=1 to N do
      write(Line[i]:5);
   writeln
end;              { Конец тела процедуры Out }

{ Процедура обмена местами соседних элементов }
Procedure Change;
var
   i:integer;
   X:integer;     { Переменная для промежуточного
                    хранения элемента массива
                    при обмене }

begin             { Начало тела процедуры Change }

   i:=1;          { Начальное значение индекса }

   { При обмене пар элементов индекс изменяется
     с шагом 2. Цикл for ... такой шаг не поддерживает,
     поэтому применяем цикл while ... do }

   while i<10 do { На каждом шаге
                   будем менять i-й и i+1-й элементы }
   begin
      X:=Line[i];          { Первый элемент из пары
                             сохраняем в X }

      Line[i]:=Line[i+1]; { На его место записываем
                            второй элемент из пары }
```

```
      Line[i+1]:=X;          { На место второго элемента
                               пишем сохраненный элемент
                               из переменной X }

      i:=i+2                 { Увеличение индекса на 2
                               для перехода
                               к следующей паре элементов }
   end
end;                  { Конец тела процедуры Change }

{ Функция вычисления суммы элементов массива }

Function Sum:integer;     { Заголовок функции.
                            Через двоеточие указывается
                            тип возвращаемого результата.
                            Результат возвращается
                            через имя функции.
                            В нашем случае возвращаемый
                            результат – сумма элементов
                            массива. Тип результата
                            соответствует типу элементов
                            массива }
var
   i:integer;
   S:integer;     { Переменная для накопления суммы }

begin             { Начало тела функции Sum }
   S := 0;
   for i := 1 to N do
      S := S + Line[i];

   Sum := S;      { Это обязательная строка!
                    Результат вычислений возвращается
                    в вызывающую программу
                    через имя функции Sum }

end;              { Конец тела функции Sum }

{ Начинается тело программы }

begin             { Именно это и есть главный оператор
                    begin – начало всей программы }
```

продолжение ☞

Пример 9.1 (*продолжение*)

```
    Inp;        { Вызов процедуры ввода элементов массива }

    Out;        { Вызов процедуры вывода массива на экран }

    Sm := Sum;{ Вызов функции вычисления суммы }

    writeln('Сумма элементов=',Sm);  { Вывод результата
                                       на экран }

    { Результат вычисления функции необходимо присвоить
      переменной того же типа (как в выражении Sm:=Sum)
      или вывести сразу на экран: }
    writeln('Сумма элементов:',Sum);

    Change:     { Вызов процедуры обмена элементов }

    Out;        { Вызов процедуры вывода массива на экран }

    readln
end. { Конец тела программы. Только здесь ставим точку }
```

Задание 9.1. Написать программу, в которой выполняется:

а) ввод одномерного массива A из 14 чисел (положительных и отрицательных);

б) вывод массива на экран;

в) сдвиг всех элементов массива на одну позицию влево (рис. 9.3); первый элемент встает на место последнего (см. блок-схему алгоритма на рис. 9.4);

г) вывод массива на экран;

д) подсчет количества положительных элементов.

Все пункты оформить как процедуры или функции.

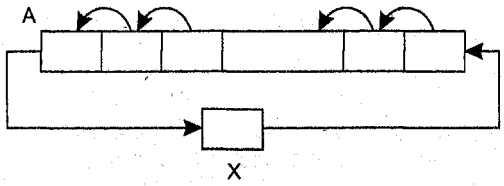


Рис. 9.3. Схема циклического сдвига одномерного массива влево

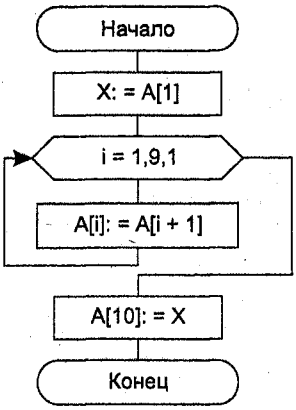


Рис. 9.4. Блок-схема алгоритма циклического сдвига массива влево

Урок 9.2. Пример работы с функцией: поиск максимального элемента

Пример 9.2. Программа поиска максимального элемента в массиве

```
Program Maximum;
const
   N=10;
type
   Mas=array [1..N] of integer;
var            { Раздел описания глобальных переменных }

   Line: Mas;  { Переменная Line – одномерный массив.
                 Его тип определен нами как  Mas }
   m,i : integer;

{ Функция определения индекса
  максимального элемента массива }
function Maxim:integer;

var            { Раздел описания локальных переменных }

   imax:integer;  { Переменная для хранения индекса
                    максимального элемента }

   i   :integer;  { Эта переменная i не совпадает
                    с глобальной переменной i }
```

продолжение ⇩

Пример 9.2 (*продолжение*)

```
begin           { Начало тела процедуры }

   imax:=1;     { За максимальный элемент
                  принимаем первый элемент массива }

   for i := 2 to N do   { Сравним текущего кандидата
                          в максимумы со всеми
                          остальными элементами массива
                          (то есть начиная со 2-го
                          и до последнего) }

      if Line[imax] < Line[i] then  { Если кандидат в
                                      максимумы оказался
                                      меньше i-го
                                      элемента... }

         imax:=i;        { ... то считаем i-й элемент
                           новым кандидатом в максимумы }

   Maxim:=imax { Возвращаем найденное значение,
                 присваивая имени функции
                 найденный индекс }

end;            { Конец тела функции }

begin           { Начало тела программы }

   { Заполнение элементов массива
     с использованием датчика случайных чисел }
   Randomize;
   for i := 1 to N do
      Line[i]:=Random(100);

   m:=Maxim;    { Вызов функции поиска индекса
                  максимального элемента массива }

   writeln('Максимальный элемент в массиве=',Line[m]:5);
   writeln('Его индекс=',m:5);
   readln
end.
```

Задание 9.2. Написать программу, в которой для массива из 20 элементов, заполненного случайными числами от –20 до +20, 20 раз выполняется следующая про-

цедура: слева направо по порядку сравниваются все соседние элементы и, если первый элемент в паре оказался больше второго, элементы меняются местами. В конце процедуры массив выводится на экран в строку. Таким образом, результатом программы должны быть 20 строк, в каждой из которых бóльшие элементы массива постепенно «сдвигаются» вправо, а меньшие — влево.

Выводы

1. При решении сложной задачи разумно ее разбивать на подзадачи. При реализации на языке Паскаль каждая такая подзадача (блок) оформляется в виде функции или процедуры.
2. Процедуры и функции оформляются вне тела программы. Они начинают работать только при вызове из тела программы или из другого блока (функции, процедуры).
3. Результат работы функции возвращается через ее имя.
4. Различают глобальные и локальные переменные.
5. Глобальные переменные действуют в теле программы и в любом из ее блоков (функциях, процедурах).
6. Локальные переменные действуют только внутри блока, в котором они описаны.

Контрольные вопросы

1. Зачем нужно создавать процедуры и функции?
2. Чем отличаются функции от процедур?
3. Зачем нужны локальные переменные?
4. Если локальная и глобальная переменные имеют одинаковые имена, то к какой переменной будет идти обращение?

ТЕМА 10

Как работать с символьными строками

Как известно, основной вид информации, которую хранит, получает и использует человек, — это текстовая информация. В эпоху активного использования вычислительной техники большая часть информации, обрабатываемой компьютерами, является текстовой. Для удобства ее обработки на компьютере придуманы специальные типы данных и операции над ними.

Урок 10.1. Работаем с цепочками символов: тип String

Для работы с цепочками символов (словами и предложениями) в Паскале введен специальный тип данных — String. Он чем-то похож на массив символов. Однако, в отличие от массива, со строками можно делать больше действий. Например, строки можно складывать.

Описание строковой переменной

Для работы с переменной типа string она должна быть описана в разделе var:

```
S : String;
```

В этом случае под строку S выделяется 255 символов, а в памяти, соответственно, она будет представлена 255 байтами. (На самом деле в памяти выделяется 256 байт, но это нам сейчас не важно.)

Если мы не планируем использовать такие большие строки, можно явно указать максимальный размер нужной вам строки. Например, запись S1 : string[40]; гово-

рит о том, что строка S1 может содержать от 0 до 40 символов.

Основные действия со строками

Рассмотрим операции, которые можно осуществлять с данными строкового типа (ввод-вывод, присваивание, сравнение).

Пример 10.1. Основные действия с символьными строками

```
Program Line_1;
var
  Name1: String[20]; { Под строку выделено 20 символов }
  Name2: String[20];
  Title: String[40]; { Под строку выделено 40 символов }
  Rez:   String[70];
begin
  Name1:='Д.Прайс_'; { Фактическое число символов
                       в строке должно быть < =
                       числу символов, объявленному
                       в разделе var
                       для этой переменной }

  Title:='Программирование на языке Паскаль';

  Rez:=Name1+Title;    { Строки можно складывать.
                         Это означает, что к
                         символам левой строки
                         будут справа приписаны
                         символы правой строки.
                         Если результат не поместится
                         в строковую переменную,
                         в которую мы его записали,
                         он будет обрезан }

  writeln(Rez);        { На экране имеем:
          Д.Прайс_Программирование на языке Паскаль }

  { К строковым переменным одинаковой длины
    можно применять операции отношения (сравнения): }
  writeln('Введите первое имя:');
  readln(Name1);
```

```
    writeln('Введите второе имя:');
    readln(Name2);
    if Name1 = Name2 then
       writeln('Имена одинаковы');
    if Name1 < Name2 then
       writeln('Первое имя в алфавитном списке раньше');
    readln
end.
```

ЗАМЕЧАНИЕ

Максимальная длина строки — 255 символов (255 байт). Фактическая длина строки хранится в нулевом байте этой строки (именно поэтому под строку реально выделяется на 1 байт больше).

Задание 10.1. Написать программу, которая выводит в алфавитном порядке три введенных пользователем имени.

Урок 10.2. Некоторые функции и процедуры Паскаля для работы со строками

Для удобства обработки строковых переменных в среде Паскаль предусмотрен ряд процедур и функций. Так как они хранятся в специальном файле-библиотеке подпрограмм среды Паскаль, эти подпрограммы принято называть *библиотечными*.

Использование библиотечных подпрограмм работы со строками

В следующем примере мы будем вводить символьную строку, в которой несколько раз встречается слово-образец. Задача состоит в том, чтобы удалить все вхождения этого слова, вставить вместо каждого слово-заменитель и вывести полученную строку на экран.

Пример 10.2. Поиск и замена

```
Program Line_2;
var
   Str:String;     { По умолчанию для строки
                     выделяется 255 байт }

   Word1,Word2:String[20];   { Переменные для хранения
                               слова-образца
                               и слова-заменителя }

   Len:byte;       { Переменная для хранения
                     длины слова-образца.
                     Тип byte - для хранения целых чисел
                     в диапазоне 0..255 }

   Position:byte; { Переменная для хранения позиции,
                    с которой начинается в строке
                    слово-образец
                    (его первое вхождение) }
begin
   writeln('Введите слово-образец:');
   readln(Word1);
   writeln('Введите слово-заменитель:');
   readln(Word2);
   writeln('Введите исходную строку:');
   readln(Str);
   Len:=Length(Word1); { Функция Length
                         возвращает длину строки Word1 }
   Position:=Pos(Word1,Str); { Pos(Word1,Str) возвращает
                               номер позиции в строке
                               Str, с которой начинается
                               первое вхождение слова-
                               образца Word1. Если слово
                               в строке отсутствует,
                               то Pos возвращает 0 }

   while Position<>0 do { Пока в строке Str
                          есть слово-образец }
   begin
      Delete(Str,Position,Len); { Процедура Delete
                                  удаляет Len символов
                                  из строки Str начиная
```

```
                                с позиции Position,
                                то есть слово-образец
                              }
        Insert(Word2,Str,Position); { Процедура Insert
                                вставляет строку Word2
                                в строку Str, начиная
                                с позиции Position.
                                Таким образом она
                                раздвигает строку }

        Position:=Pos(Word1,Str) { Вычисляем номер новой
                                позиции слова-образца
                                для следующего шага
                                цикла while }
    end;
    writeln(Str);
    readln
end.
```

Задание 10.2. Написать программу, в которой вводится строка из слов с некоторым количеством пробелов между ними. Удалить лишние пробелы, оставив по одному между словами.

Выводы

1. Для работы с массивом символов разумнее использовать тип данных String.
2. Со строками можно выполнять операции присваивания, сложения и сравнения.
3. Максимальное количество символов, которое можно хранить в строковой переменной, равно 255.
4. Для удобства работы с типом данных String рекомендуется использовать библиотечные функции и процедуры языка Turbo Pascal.
5. Библиотечные функции позволяют раздвигать строки, вычислять их длину, удалять в них подстроки и осуществлять поиск.

Контрольные вопросы

1. Если в строковой переменной не планируется хранить более 50 символов, как ее разумнее описать в программе?
2. Если к строковой переменной длиной 200 символов, описанной как string, «добавить» (+) строковую переменную длиной 100 символов, какова будет длина получившейся строковой переменной?
3. Как определить длину введенной с клавиатуры строки?
4. Как определить количество точек во введенной с клавиатуры строке?
5. Как увеличить строку вдвое, дописав рядом с каждым символом строки такой же (например, из строки «Вася» получить «ВВаассяя»)?

ТЕМА 11

Процедуры и функции с параметрами

Все вспомогательные алгоритмы, рассмотренные нами в теме 9, были «слепыми». То есть они осуществляли свои действия независимо от каких-либо значений. Однако нетрудно было заметить, что мы пользуемся массой встроенных в Паскаль вспомогательных алгоритмов, действия которых зависят от значений, указываемых нами в скобках. Эти значения называются *параметрами.*

Урок 11.1. Простые примеры использования подпрограмм с параметрами

Уже самый первый оператор, с которым мы познакомились, — writeln('Привет!') — является процедурой с параметром. Процедура в данном случае занимается выводом на экран, а параметр указывает, что именно должно появиться на экране.

Наша текущая задача — научиться самим создавать процедуры с параметрами.

Простейшие процедуры с параметрами

Пример 11.1. Использование процедур с параметрами для рисования на экране перекрестий и последовательностей звездочек

```
program Param1;

uses Crt; { Мы будем активно использовать функции
            рисования на экране }

{ Процедура Stars выводит на экран
  указанное количество звездочек }
```

```
procedure Stars(N:integer); { После имени процедуры
                              в скобках указываются
                              ее параметры и их типы.
                              В данном случае процедура
                              имеет один параметр N
                              (типа integer), который
                              указывает количество
                              звездочек, выводимых на
                              экран }

var i:integer; { Раздел описания локальных переменных }

begin
   for i:=1 to N do { Мы используем параметр N
                      для указания того,
                      сколько раз нужно
                      выводить на экран символ "*" }
      write('*')
end;

{ Процедура Cross выводит на экран
  перекрестье с центром в координатах (X,Y) }

procedure Cross(X,Y:integer); { Если параметры
                                однотипные, их имена
                                можно перечислить
                                через запятую }
begin
   gotoxy(x,y-1);    writeln('|');
   gotoxy(x-2,y);  writeln('--+--');
   gotoxy(x,y+1);    writeln('|')
end;
{ Начало основной программы }
begin
   clrscr;        { Очистим экран, чтобы лучше было видно
                    результаты работы }

   Stars(10);     { Вывели 10 звездочек
                    в начало первой строки }

   gotoxy(6,20);
```

продолжение ⇩

Пример 11.1 (*продолжение*)

```
    Stars(70);     { Заполнили звездочками
                     почти всю 20-ю строку }

    Cross(40,13); { Вывели перекрестье в центр экрана }

    Cross(70,5);  { Вывели перекрестье
                     в правый верхний угол экрана }
    readln
end.
```

Обратите внимание: в процедурах мы используем параметры так, как будто у нас есть переменные с соответствующими именами, и они имеют определенные значения. Это приблизительно так и есть. Эти «переменные» даже можно менять. То есть вполне можно написать N := N + 2, и параметр N в этом месте действительно увеличится на 2. Нужно только понимать, что «время жизни» этих «переменных» такое же, как у локальных переменных — по окончании работы процедуры они уничтожаются.

Формальные и фактические параметры

Если в качестве параметра при вызове процедуры подставить имя переменной, а внутри процедуры этот параметр изменить, то на саму переменную основной программы это никоим образом не повлияет.

Здесь мы сталкиваемся с понятиями формальных и фактических параметров. Параметры, имена которых используются в процедуре, называются *формальными*. Они могут совпадать или не совпадать по имени с переменными, которые мы подставляем при вызове процедуры.

В момент вызова процедуры значения *фактических* параметров (которые мы подставили в скобки при вызове) копируются в отдельные ячейки памяти, которые используются процедурой и после ее окончания осво-

бождаются. Фактические параметры при этом не изменяются.

Простейшие функции с параметрами

Использование собственных функций позволяет, например, расширить список стандартных функций Паскаля.

Пример 11.2. Применение пользовательских функций с параметрами для расширения набора математических функций языка Паскаль

```
program Param2;

{ Функция вычисления тангенса }
function tg(x:real):real; { Необходимо описать
                            не только тип параметра,
                            но и тип самой функции.
                            В данном случае они
                            совпадают, но, вообще
                            говоря, это не обязательно }
begin
   tg:=sin(x)/cos(x) { В данном случае функция
                       очень простая. Однако ее
                       использование делает программу
                       намного более удобной для чтения}
end;

{ Функция вычисления числа x,
  возведенного в целую положительную степень n }
function Stepen(x:real; n:integer):real;
                { Если параметры разных типов,
                  они перечисляются
                  через точку с запятой }

var
   i:integer;
   y:real; { Аккумулятор для накапливания произведения }
begin
   y:=1;   { Начальное значение
             произведения-аккумулятора
             всегда равно единице }
```

продолжение ⇩

Пример 11.2 (*продолжение*)

```
    for i:=1 to n do
      y:=y*x;
    Stepen:=y
end;

{ Начало основной программы }
begin
   writeln('Тангенс числа Пи/6=',tg(Pi/6):7:3);

   { Результат функции вещественный, поэтому
     для красивого вывода на экран применяем формат }
   writeln('Число 2.5 в степени 8 равно ',
           Stepen(2.5,8):7:3);

   { Заметьте, использование функций
     делает программу более наглядной }
   writeln('Тангенс Пи/3 в степени 5 равен ',
           Stepen(tg(Pi/3),5):7:3);
   readln
end.
```

Урок 11.2. Способы передачи параметров

Среди стандартных процедур Паскаля можно найти и такие, которые изменяют само значение параметра — например, inc(). Как самому создать такую процедуру?

Пример 11.3. Процедуры, изменяющие значения параметров

```
program Param3;
var a,b:integer;

{ Процедура обмена двух переменных местами }
procedure Change(var x,y:integer);
                { Для указания того, что значения
                  передаваемых переменных будут изменены
                  в процедуре, используют служебное
                  слово var. Это называется
                  передачей параметров по имени.
                  Параметр, имя которого используется
```

```
                для передаваемой переменной,
                называется формальным }

var z:integer; { Временная переменная для обмена }
begin
   z:=x;
   x:=y;
   y:=z
end;

{ Функция возведения числа x в целую положительную
  степень n }
function Stepen(x:real; n:integer):real;
              { Если параметры разных типов,
                они перечисляются
                через точку с запятой }
var
   i:integer;
   y:real; { Аккумулятор для накапливания произведения }
begin
   y:=1; { Начальное значение произведения-аккумулятора
           всегда равно единице }
   for i:=1 to n do
      y:=y*x;
   Stepen:=y
end;

{ Начало основной программы }
begin
   a:=5;
   b:=8;
   writeln('До обмена A=',a,' B=',b);

 Change(a,b);  { Передаваемая переменная,
                 имя которой указано в скобках
                 при вызове процедуры,
                 называется фактическим параметром.
                 Имена формальных и фактических
                 параметров могут не совпадать }

   writeln('После обмена A=',a,' B=',b);
   readln
end.
```

Названия *формальный параметр* и *фактический параметр* подчеркивают, что при передаче параметров таким образом (со служебным словом var) переменная для формального параметра не создается и память не выделяется. Это имя только формально используется для описания действий, которые будут совершены с данной переменной в подпрограмме. Фактически вместо этого имени используется имя переменной, подставленное при вызове подпрограммы.

Заметим, что при вызове процедуры, изменяющей значения параметров, в качестве фактического параметра нельзя использовать выражения. Фактическим параметром может быть только имя переменной! Это связано с тем, что в такую процедуру передается не значение, а адрес ячейки памяти, в которой хранится переменная-параметр. Если подставить выражение, то Паскаль сообщит об ошибке — ведь выражение не имеет адреса!

Способ передачи параметров, изменяющий их значения, называется *передачей параметров по адресу*. Обычный способ, при котором фактический параметр копируется в отдельную ячейку памяти и который позволяет в качестве параметра передавать значения выражения, называется *передачей параметров по значению*.

Один и тот же вспомогательный алгоритм может получать параметры обоими способами — часть по адресу, остальные — по значению.

СОВЕТ

Использование процедур с параметрами, передающимися по адресу, с одной стороны, добавляет дополнительные возможности для программирования, а с другой — часто приводит к ошибкам, которые очень трудно выловить. Изменение значений переменных при вызове процедур и функций не всегда бросается в глаза, особенно если программа большая и ее написание занимает много дней. Будьте внимательны!

Задание 11.1. Написать процедуру, которая получает целочисленную переменную и «разворачивает» ее цифры в обратном порядке — например, число 1234 преобразует в число 4321.

Задание 11.2. Написать процедуру, которая получает две вещественные переменные и возвращает вместо каждой из них отклонение от среднего арифметического. Например, числа 5 и 8 должны превратиться в –1,5 и 1,5, а числа 3,2 и 0,8 — в 1,2 и –1,2.

Выводы

1. Существует возможность создавать свои собственные процедуры и функции с параметрами.
2. Их использование повышает наглядность программы и добавляет ей универсальности.
3. Передаваемые параметры перечисляются в скобках после имени подпрограммы с указанием типа данных.
4. Для изменения значений передаваемых параметров используется передача по адресу.
5. Для указания того, что передача параметров происходит по адресу, используется служебное слово var.
6. Используя передачу параметров по адресу, будьте особенно внимательны!

Контрольные вопросы

1. Приведите пример, когда использование процедуры с параметрами сокращает текст программы втрое.
2. Какой из способов передачи параметров расходует больше памяти?

3. Можно ли передавать в качестве параметра произведение двух переменных при передаче параметров по значению? Где окажется значение произведения? Почему этого нельзя сделать при передаче по адресу?
4. Рассмотрим следующую процедуру:

```
procedure Maxim(var x,y:integer);
begin
  if x>y then y:=x
  else x:=y
end;
```

Тело программы выглядит так:

```
x:=2; y:=3;
Maxim(y,x);
```

Каковы будут значения переменных x и y?

5. Рассмотрим следующую процедуру:

```
procedure Mult(var x:integer;y:integer);
begin
  x:=y*2
end;
```

Тело программы выглядит так:

```
x:=2;
Mult(x,2*x);
```

Каково будет значение переменной x?

ТЕМА 12

Файлы: сохраняем результаты работы до следующего раза

Как вам должно быть известно из общего курса информатики, память, с которой работает Паскаль и в которой он хранит все свои данные (как и любая другая программа), называется *оперативной*. Она обладает одним неприятным свойством: ее содержимое стирается при выключении питания компьютера. Чтобы информация сохранялась при выключенном питании (принято говорить «для долговременного хранения информации»), используется *внешняя память*. Это разного рода диски, дискеты и другие виды носителей. Работе с внешней памятью из программы на Паскале и посвящен наш текущий разговор.

Урок 12.1. Как работать с текстовым файлом

Файлом называется порция данных, хранящаяся на диске и имеющая имя. Другими словами, все, что вы пытаетесь сохранить на диске, должно быть записано в виде файла. Для того чтобы работать с файлом в программе, необходимо ввести специальную переменную, которая называется *файловой*. Через нее мы будем записывать и читать информацию из файла.

Основным элементом текстового файла является символьная строка (ASCII). Можно работать как со строкой целиком, так и с каждым символом в отдельности. Обращение к символам, хранящимся в файле, происходит последовательно.

Открытие файла для чтения

Начиная работать с файлом, его *открывают*. При этом в памяти создается особая структура данных, частью

которой является *файловый указатель*. Это как бы «курсор», который указывает на позицию файла, с которой будет происходить следующая операция чтения (или записи). После чтения символа (или строки) из файла файловый указатель передвигается на следующий символ (строку). При записи в файл эта позиция всегда указывает на конец файла.

Мы начнем с самого простого — попытаемся открыть текстовый файл для чтения и выведем его содержимое на экран. Для того чтобы программе было что открывать, создайте в Блокноте или прямо в среде Turbo Pascal текстовый файл и назовите его work.txt. Этот файл должен быть сохранен в той же папке, что и рабочие (.pas) файлы с программой на Паскале. Содержимое файла нам не важно. Мы рекомендуем набрать несколько строчек текста, желательно латинскими символами.

Пример 12.1. Вывод на экран содержимого текстового файла work.txt

```
Program File_1;
var
   Fil: text; { Описание файловой переменной
                для работы с текстовым файлом }

   { Паскаль умеет работать с разными типами файлов.
     Мы ограничимся изучением самого распространенного
     и понятного типа – текстового. Такой тип данных
     в Паскале называется text. Удобство работы с
     текстовым файлом состоит в том, что его можно
     открыть в обычном текстовом редакторе
     (например, в Блокноте) и просмотреть или исправить }

   Str : string; { Переменная для чтения
                   строк из файла }
begin

   { Перед началом работы с файлом необходимо выполнить
     несколько предварительных действий: }
```

продолжение ⇩

Пример 12.1 (*продолжение*)

```
    Assign(Fil,'work.txt'); { Файловой переменной Fil
                              назначается конкретное
                              имя файла
                              (в данном случае
                              work.txt) }

    Reset(Fil);             { Файл открывается
                              с признаком "для чтения".
                              Файловый указатель при этом
                              устанавливается
                              в начало файла }

    { Читаем все строки по одной из файла work.txt
      и выводим их на экран. При чтении из файла work.txt
      для определения того, что файл прочитан целиком,
      используется признак "конец файла". Этот признак
      проверяется библиотечной функцией Eof (end of
      file). Пока файловый указатель не достигнет конца
      файла, этот признак, а также результат,
      возвращаемый функцией Eof, имеют значение False
      ("ложь"). Когда конец файла достигнут
      (то есть файл прочитан весь), функция Eof
      возвращает результат True ("истина") }

    while not Eof(Fil) do  { Чтение файла происходит
                             до тех пор, пока функция Eof
                             возвращает False. Чтобы условие
                             продолжения цикла выполнялось
                             пока не достигнут конец файла,
                             используется
                             логическое отрицание not }

    begin
       readln(Fil,Str);  { Чтение строки
                           из файла work.txt }

       writeln(Str)   { Вывод прочитанной строки
                        на экран }
    end;
    Close(Fil)        { После окончания работы
                        открытый файл
                        нужно закрыть }
end.
```

Открытие файла для записи

Из предыдущего примера вы, вероятно, поняли, что открытие файла для чтения происходит в результате процедуры reset. При этом файловый указатель устанавливается в начало файла, и процедурами чтения из файла readln и read можно построчно или посимвольно прочитать содержимое всего файла. Если же мы хотим не читать уже имеющийся файл, а записывать в файл свою информацию, мы должны открыть файл для записи. Это делает процедура rewrite. Если в текущей папке нет файла с именем, указанным в процедуре assign, то создается новый пустой файл. Если файл с таким именем есть, его содержимое очищается. Так или иначе, файловый указатель устанавливается в начало пустого файла, после чего в файл можно записывать информацию построчно (writeln) или посимвольно (write).

В следующем примере программа создает новый текстовый файл work.txt и копирует его содержимое в файл user.txt.

Пример 12.2. Создание и копирование файла

```
Program File_2;
var
   Fil_1, Fil_2: text; { Описание файловых переменных
                         для работы с текстовыми файлами }

   Str : string;   { Переменная для чтения
                     строк из файла и записи в него }

{ Процедура создания нового файла }
Procedure New_file;
var   I : integer;
begin
   Assign(Fil_1,'work.txt');
   Rewrite(Fil_1);      { Создается новый пустой файл
                          и открывается с признаком "для
```

продолжение

Пример 12.2 (*продолжение*)

```
                        записи". Файловый указатель
                        устанавливается в начало файла
                        (которое в данном случае
                        совпадает с концом).
                        Файл пуст }

    { Введем с клавиатуры три любых символьных строки
      и запишем их в файл work.txt }
    for I:=1 to 3 do
    begin
      writeln('Введите строку и нажмите Enter:');

      readln(Str);      { Введенная строка помещается
                          в переменную Str }

      writeln(Fil_1,Str) { Строка Str записывается
                           в файл. Файловый указатель
                           перемещается в начало
                           следующей строки, то есть
                           в то место, куда будет
                           внесена следующая запись }
    end;
    Close(Fil_1)        { Файл закрывается.
                          Это обязательно нужно делать
                          после окончания работы
                          с файлом или перед его новым
                          открытием }
end;

{ Чтение всех строк по одной из файла work.txt
  и запись их в файл user.txt }
Procedure Copy_file;
begin
   Assign(Fil_1,'work.txt');

   Reset(Fil_1);        { Файл открывается
                          с признаком "для чтения".
                          Файловый указатель при этом
                          устанавливается
                          в начало файла }
```

```
    Assign(Fil_2,'user.txt'); { Файловой переменной Fil_2
                                назначается имя файла,
                                в который будет
                                идти копирование }

    Rewrite(Fil_2);           { Создается новый файл
                                и открывается "для записи". }

    while not Eof(Fil_1) do  { Чтение файла происходит
                               до тех пор, пока функция Eof
                               возвращает False. Чтобы условие
                               продолжения цикла выполнялось
                               пока не достигнут конец файла,
                               используется
                               логическое отрицание not }
    begin

       readln(Fil_1,Str);  { Чтение строки
                             из файла work.txt }

       writeln(Fil_2,Str)  { Запись прочитанной строки
                             в файл user.txt }

    end;
    Close(Fil_1);
    Close(Fil_2)           { После окончания работы
                             все открытые файлы
                             положено закрывать }
end;
begin                      { Начало основной программы }

    New_File;  { Вызов процедуры создания нового файла }

    Copy_file  { Вызов процедуры копирования в файл }

end.
```

После работы программы откройте оба файла (это можно сделать в режиме Open в среде Turbo Pascal или в Блокноте) и убедитесь, что все получилось верно.

ЗАМЕЧАНИЕ

Если вы вводили эти три строки по-русски, то в Блокноте вы, скорее всего, увидите странную путаницу из русских букв. Это оттого, что Turbo Pascal – программа для MS-DOS, и символы, которые вы вводили, тоже были записаны в кодировке MS-DOS. Чтобы их нормально прочитать, нужно или открывать эти файлы из программы для MS-DOS (например, Norton Commander), или использовать преобразование формата (это умеет делать, например, MS Word).

Задание 12.1. Напишите программу, которая:

а) создает текстовый файл из четырех строк строчных латинских букв;

б) читает строки из созданного файла и преобразовывает их в строки заглавных латинских букв;

в) после преобразования каждую строку записывает в другой созданный текстовый файл.

Проверьте результаты работы путем чтения обоих файлов!

Урок 12.2. Сохранение двумерного массива чисел в текстовом файле

Числовые данные тоже можно сохранять в текстовых файлах. Так как текстовый файл можно будет читать не только программно, но и средствами обычного редактора, это очень удобно: можно сохранить в файле числовые результаты и вставить их потом в другую программу.

Сохранение числовых данных в текстовом файле

Пример 12.3. Сохранение чисел в текстовом файле

```
Program File_3;
var
```

```
   A,B:integer;
   Fil:text;
begin
   A:=3;
   B:=10;
   Assign(Fil,'prim.txt');
   Rewrite(Fil);

   write(Fil,A,' ');     { Файловый указатель в файле
                           prim.txt после этой операции
                           будет стоять за пробелом
                           после числа 3, без перехода
                           на следующую строку }

   write(Fil,B);         { При записи числа
                           в текстовый файл записывается
                           его символьный код }

   Close(Fil)
end.
```

Задание 12.2. Просмотреть полученный файл prim.txt в текстовом редакторе.

Сохранение массива чисел в текстовом файле

Рассмотрим, как можно записывать в текстовый файл числовые массивы. Это очень удобный способ обмена информацией: можно подготовить данные на одном компьютере, сохранить их в файл, перенести его на другой компьютер и там открыть в другой программе.

Пример 12.4. Запись матрицы вещественных чисел 5 × 4 в текстовый файл и чтение данных из файла

```
program File_4;
const
   M=5; N=4;
var
   Fil:text;
   A:real;
   S:char;
   I,J: integer;
begin
```

продолжение ☞

Пример 12.4 (*продолжение*)

```
assign(Fil,'matrix.txt');
rewrite(Fil);
randomize;

{ Запись матрицы в текстовый файл }
for I:=1 to M do
begin
   for J:=1 to N do
   begin

      A:=random(100);        { Запишем в элемент
                               матрицы случайное
                               число }

      write(Fil,A:5:3,' ') { Число A записывается
                             в файл в указанном
                             формате, за ним стоит
                             пробел }

   end;

   writeln(Fil)            { i-я строка массива
                             кончилась.
                             Файловый указатель
                             переводится на следующую
                             строку в файле }

end;
close(Fil);

{ Чтение файла и вывод матрицы на экран по строкам.
  Повторное использование процедуры assign(Fil)
  не требуется }
reset(Fil);

while not eof(Fil) do
begin
   while not eoln(Fil) do  { eoln(Fil) возвращает
                             статус конца строки.
                             Возвращаемый результат
                             имеет тип boolean.
```

```
                                  Пока не будет достигнут
                                  конец строки, цикл
                                  выполняется.
                                  Функция eoln работает
                                  только для текстового
                                  файла (для него имеет
                                  смысл понятие строки) }
      begin
         read(Fil,A);             { Чтение числа из текущей
                                    строки файла }

         write(A:5:3);            { Вывод числа на экран }

         read(Fil,S);             { Чтение пробела
                                    за числом.
                                    Пробел - это символ.
                                    Он не может быть
                                    считан, как число,
                                    в переменную A }

         write(S)                 { Вывод пробела на экран }
      end;
      writeln;                    { Перевод курсора
                                    на экране
                                    на следующую строку }

      readln(Fil)                 { Перевод
                                    файлового указателя
                                    на следующую строку }
   end;
   close(Fil);
   readln
end.
```

Пример 12.5. Чтение данных из текстового файла в двумерный массив

```
program ReadMatrix;
const
   M=5; N=4;
var
   Fil:text;
   A:array[1..M,1..N] of real;
```

продолжение ⇘

Пример 12.5 (*продолжение*)

```
    S:char;
    I,J: integer;
begin
    assign(Fil,'matrix.txt');

    reset(Fil);      { Открываем для чтения тот же файл,
                       в который записали матрицу }

  { Чтение матрицы из текстового файла }
    for I:=1 to M do
    begin
       for J:=1 to N do
       begin

          read(Fil,A[I,J],s);    { Читаем из файла
                                   элементы построчно.
                                   В переменную S
                                   считываем пробел
                                   между числами }

          write(A[I,J]:6:3,' ') { Выводим матрицу
                                   на экран
                                   в том же порядке }

       end;

       writeln;                  { Переходим на следующую
                                   строку на экране }

       readln(Fil)               { Переходим на следующую
                                   строку файла }
    end;
    closefile(Fil);
    readln
end.
```

Задание 12.3. Написать программу чтения файла prim.txt и вывода чисел из файла на экран монитора.

Задание 12.4. Используя файл matrix.txt, подсчитать сумму элементов в каждой строке и записать полученные результаты в новый файл rezult.txt.

Дописывание информации в конец файла

Процедура rewrite, как уже было сказано, очищает все содержимое файла, если он уже существует. Если же нужно дописать что-то к уже существующему файлу, используется процедура append. Она тоже открывает файл для чтения, но не очищает старое содержимое файла, а устанавливает файловый указатель в конец.

Пример 12.6. Дописывание информации в конец текстового файла

```
program File_6;
const
   M=5; N=4;
var
   Fil:text;
   append(Fil);                { Открытие файла
                                 для добавления текста
                                 в конец }

   writeln(Fil,'Конец матрицы'); { Допишем этот текст
                                 в конец записанной матрицы }

   close(Fil);
   readln
end.
```

ЗАМЕЧАНИЕ

*Вы, конечно, обратили внимание, что процедуры вывода на экран (*write *и* writeln*) имеют те же названия, что и процедуры записи в файл. То же касается и процедур* read *и* readln*. Это оттого, что экран и клавиатура с точки зрения Паскаля также являются файлами. Эти файлы называются* con *(консоль). То есть когда мы пользуемся процедурой вывода на экран, Паскаль на самом деле выводит информацию в файл* con*, под которым система понимает монитор. А когда мы читаем информацию с клавиатуры, Паскаль читает информацию из файла* con*,*

под которым система понимает клавиатуру. Вы можете убедиться в этом сами: присвойте файловой переменной имя файла con (assign(Fil,'con')) *и попробуйте открыть этот файл для записи* (rewrite(Fil)) *и записать в него какие-нибудь данные (например,* writeln('Hello, world!')). *Этим же методом можно выводить из Паскаля информацию на принтер. Нужно только знать, что имя файла-принтера* — lpt1.

Выводы

1. Для долговременного хранения полученной информации используются файлы.
2. Нами рассмотрена работа с текстовыми файлами.
3. Основным элементом файла типа text является строка символов ASCII.
4. Для работы с файлом вводится файловая переменная, через которую идет обращение к файлу.
5. При работе с файлом необходимо учитывать положение файлового указателя, который перемещается к концу файла по мере чтения.
6. Для работы с файлом необходимо назначить файловой переменной имя файла с помощью команды assign и открыть его для чтения (reset) или для записи (rewrite).
7. Запись в файл и чтение из файла производится процедурами write/writeln и read/readln с обязательным указанием в качестве первого параметра имени файловой переменной.
8. После окончания работы с файлом его необходимо закрыть командой close.

Контрольные вопросы

1. В какой памяти хранятся данные, которые обрабатывает программа? В какой памяти нужно сохранять эти данные, если они должны храниться долго?
2. На что указывает файловый указатель?
3. Какие действия необходимо совершить, чтобы открыть файл для чтения?
4. На диске хранился файл example.txt, в котором находилось 5 страниц текста (примерно 9 Кбайт). Программа выполнила следующие действия:

```
assign(f,'example.txt');
rewrite(f); writeln(f,'Конец');
close(f);
```

   Как изменится количество информации, содержащейся в файле?
5. Ответьте на предыдущий вопрос, если вместо операции rewrite поставить операцию append.
6. В файле хранится строка из 10 цифр (от 0 до 9), разделенных пробелами. Программа считывает их из файла следующим образом:

```
for i:=1 to 9 do
begin
  read(f,c); x:=ord(c)
end
```

   Чему будет в результате равна переменная x (i,x:integer; c:char;)?

ТЕМА 13

Графический режим работы. Модуль Graph

Мы выводили на экран узоры из звездочек, но, безусловно, это изображение не являлось графическим. Такие изображения из символов иногда называют псевдографикой. Монитор работал в текстовом режиме — 80 × 25 символов на экране (см. урок 1.2).

В этой теме мы познакомимся с возможностями среды Turbo Pascal для работы с графической информацией.

Урок 13.1. Включаем графический режим работы

Все предыдущие задачи красивого вывода на экран мы решали, используя текстовый режим работы и библиотеку Crt, обслуживающую этот режим.

Особенности работы с графикой

При работе в графическом режиме изображение на экране строится не из символов, а из точек — пикселов. Каждый пиксел имеет две координаты, x и y (рис. 13.1), и определенный цвет (по умолчанию белый).

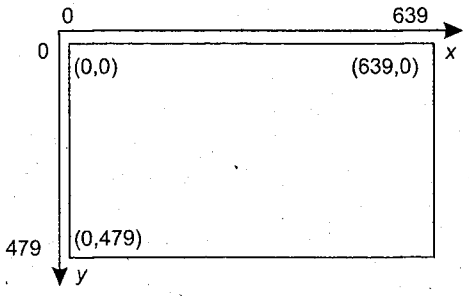


Рис. 13.1. Система координат пикселов в графическом режиме работы

При использовании модуля Graph, обслуживающего графический режим, Turbo Pascal умеет работать с разрешениями экрана до 640 × 480 пикселов.

ВНИМАНИЕ

Для работы программы в графическом режиме необходима специальная программа — драйвер графического режима. В обычной установке среды Паскаль такой драйвер хранится в файле egavga.bgi. Рекомендуется скопировать его в свой текущий каталог.

Местоположение этого файла можно найти с помощью режима Поиск или в каталоге BGI установки Паскаля.

Левый верхний пиксел имеет координаты (0,0). Количество пикселов зависит от типа дисплейного адаптера и режима его работы. Для современных компьютеров это разрешение (640 × 480) считается уже устаревшим. Но для работы с более высоким разрешением требуется современный драйвер экрана (не egavga.bgi, а svga.bgi, например). Он не входит в стандартную поставку Паскаля, поэтому мы не будем его рассматривать.

Переключение в графический режим видеоадаптера

Стандартное состояние компьютера при запуске среды Turbo Pascal соответствует работе экрана в текстовом режиме. Поэтому для использования графических средств надо инициировать графический режим работы дисплейного адаптера, то есть переключить экран в графический видеорежим. Для этого подключается графический драйвер — специальная программа, осуществляющая управление теми или иными техническими средствами (монитором и видеоадаптером).

Все стандартные процедуры и функции для работы в графическом режиме содержатся в библиотечном мо-

дуле Graph. Поэтому необходимо подключить его в разделе объявления дополнительных модулей.

Пример 13.1. Заполнение экрана разноцветными точками

```
uses
   Graph, Crt;     { Подключаем модули.
                     Crt нам также понадобится }
var
   Gd,Gm:Integer; { Переменная Gd определяет
                    тип драйвера адаптера }

   { Переменная Gm определяет режим работы адаптера;
     по умолчанию выбирается старший режим
     (с самым высоким разрешением) }

   Color : byte;

begin
   Gd := Detect;  { Тип драйвера адаптера
                    определяем автоматически }

   InitGraph(Gd,Gm, ''); { Инициализация графики.
                           В кавычках указывается путь
                           к программе-драйверу
                           с расширением bgi.
                           Сейчас предполагается,
                           что драйвер находится
                           в вашем текущем каталоге
                           (откуда вы запускали
                           Паскаль).
                           Если при запуске возникнет
                           ошибка, проверьте,
                           где на самом деле
                           хранится файл egavga.bgi,
                           и укажите этот путь
                           в качестве третьего параметра
                           InitGraph. Например
                     InitGraph(Gd,Gm,'c:\sys\tp71\bgi') }

   If GraphResult <> grOK then { Если инициализация
                                 не была успешной -
                                 остановка }

      Halt(1);                                    продолжение ⇩
```

Пример 13.1 (*продолжение*)

```
    Randomize;

    { На экран выводятся разноцветные точки
      внутри квадрата 100 X 100,
      пока вы не нажмете любую клавишу }
    repeat
       Color := Random(15);
       PutPixel(Random(100),Random(100),Color);
       { Процедура PutPixel(X,Y,C) перекрашивает пиксел
         с координатами (X,Y) в цвет C.
         Мы задаем координаты случайным образом! }

       Delay(10)
    until KeyPressed;

    CloseGraph      { Эта процедура выключает
                      графический режим
                      и снова делает экран
                      текстовым }

end.
```

Задание 13.1. Измените программу так, чтобы:

- пикселы светились по всему экрану (640 × 480);
- пикселы светились в верхней половине экрана;
- пикселы светились в нижней половине экрана (добавляйте постоянное приращение к случайной координате).

Урок 13.2. Продолжаем изучать возможности модуля Graph

Мы рассмотрели, как переключиться в графический режим работы и как выводить на экран точки определенного цвета. Но одними точками возможности модуля Graph не ограничиваются. Он умеет также рисовать простейшие геометрические фигуры — линии, прямоугольники и окружности.

Рисование линий средствами модуля Graph

Рассмотрим пример рисования линий. Цвет рисуемой линии устанавливается процедурой SetColor. Сами линии рисуются с помощью процедуры Line.

Пример 13.2. Вычерчивание линий в цикле

```
uses
   Graph, Crt;
var
   Gd,Gm:Integer;
   Color : byte;
begin
   Gd := Detect;
   InitGraph(Gd,Gm,'');
   If GraphResult <> grOK then
      Halt(1);
   Randomize;

   { На экран выводятся разноцветные линии (отрезки)
     внутри квадрата 200 X 200,
     пока вы не нажмете любую клавишу }

   { Процедура Line(X1,Y1,X2,Y2) рисует отрезок.
     Начало отрезка – координаты (X1,Y1),
     конец отрезка – координаты (X2,Y2).
     Координаты мы задаем случайным образом.
     Процедура SetColor(цвет) устанавливает цвет линий.
     Цвет мы тоже задаем случайно! }

   repeat
      SetColor(Random(15));
      Line(Random(200),Random(200),
           Random(200),Random(200));
      Delay(10)
   until KeyPressed;
   Readln;
   CloseGraph

   { Эта процедура выключает графический режим
     и снова переводит экран в текстовый вид.
     Именно поэтому важно было написать readln
     перед CloseGraph, иначе мы бы не увидели
     результата рисования (при переходе
```

продолжение ⇨

Пример 13.2 (*продолжение*)

```
        в текстовый режим вся картинка графического режима
        пропадает) }

end.
```

Задание 13.2. Измените программу так, чтобы линии выводились по всему экрану (640 × 480), все отрезки начинались бы в центре, а конец отрезков задавался бы случайно (рис. 13.2).

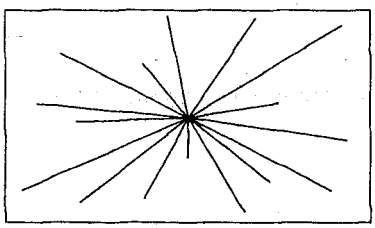


Рис. 13.2. Пояснительный рисунок к заданию 13.2

Задание 13.3. Измените программу так, чтобы на экране было два пучка отрезков: из левого нижнего угла и из правого верхнего (рис. 13.3).

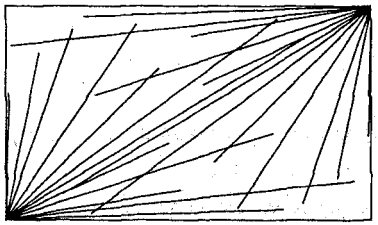


Рис. 13.3. Пояснительный рисунок к заданию 13.3

Рисование окружностей средствами модуля Graph

Теперь рассмотрим, как в графическом режиме можно рисовать окружности. Этим занимается процедура Circle.

Пример 13.3. Вычерчивание разноцветных концентрических окружностей

```
uses
   Graph, Crt;
var
   Gd,Gm,r:Integer;
   Color : byte;
begin
   Gd := Detect;
   InitGraph(Gd,Gm,'');
   If GraphResult <> grOK then
      Halt(1);
   Randomize;

   { На экран выводится 10 разноцветных окружностей }
   for r:=1 to 10 do
   begin
      SetColor(random(16));

      Circle(320,240,r*5) { Процедура Circle(X,Y,R)
                            рисует окружность
                            с центром в точке (X,Y)
                            и радиусом R }

   end;
   Readln;
   CloseGraph
end.
```

СОВЕТ

Если вы наберете в редакторе Паскаля слово Graph, *установите на него курсор и нажмете* Ctrl+F1, *то вы получите справку по процедурам и функциям модуля* Graph. *Постарайтесь добраться до готовых примеров, скопировать их в свою программу и поэкспериментировать.*

В модуле Graph имеется еще очень много процедур (например, ellipse, bar, rectangle, outtext), однако их мы предлагаем вам изучить самостоятельно.

Выводы

1. Работать с монитором можно в графическом режиме. Для этого надо в своем рабочем каталоге иметь файл с драйвером — в нашем случае egavga.bgi.
2. При работе в графическом режиме изображение строится из пикселов (640 × 480). Их количество (иначе говоря, разрешающая способность экрана) определяется типом адаптера монитора и его режимом работы.
3. Процедуры для работы в графическом режиме хранятся в библиотечном модуле Graph.
4. Для переключения в графический режим работы используется процедура InitGraph. Для возврата в текстовый режим — CloseGraph.
5. Библиотечный модуль Graph имеет очень много процедур для рисования различных геометрических фигур. Для получения справки по процедурам модуля Graph нужно установить курсор на слово Graph в программе и нажать Ctrl+F1.

Контрольные вопросы

1. Чем отличаются друг от друга графический и текстовый режимы работы?
2. Из каких элементов строится изображение в графическом режиме работы?
3. Какие координаты имеет точка, находящаяся в правом нижнем углу экрана? А в центре экрана?
4. Какие команды нужно написать, чтобы нарисовать две перекрещивающиеся линии, идущие по диагоналям экрана?
5. Какие команды нарисуют ряд концентрических окружностей, расходящихся из центра экрана на равном расстоянии друг от друга?

ТЕМА 14

Операторы, изменяющие естественный ход программы

Язык Паскаль задумывался как структурный язык. То есть любой алгоритм в нем можно описать в виде набора операторов условия и цикла, каждый из которых можно рассматривать как отдельный блок. В блоки «вкладываются» более мелкие блоки, и т. д. Поэтому реализация программы легко осуществляется в рамках структурного программирования.

Ряд языков программирования (таких, например, как Фортран и Бейсик) не удовлетворяют свойству структурности: в них для описания алгоритма приходится использовать, например, оператор безусловного перехода (goto).

Для первых языков программирования использование безусловного перехода являлось совершенно естественным, так как сама инструкция безусловного перехода используется в каждой программе, написанной на машинном коде. Без нее нельзя реализовать такие, например, конструкции, как if...else.

Первоначально языки программирования придумывались как средство более удобной записи машинных команд, поэтому в них оператор goto применялся очень широко.

Однако после была выработана идея, что использование в программе безусловного перехода сильно запутывает программу, особенно если этот переход осуществляется «наверх», то есть возвращает нас в программе к тем операторам, которые уже выполнялись. Тезис структурного программирования призывает вовсе не использовать в программе оператор goto. Для людей, которые, вероятно, первый раз о таком слышат, это кажется вполне естественным. А вот для программистов,

которые привыкли мыслить «в терминах goto», отказ от его использования был очень странен и вызывал много возражений.

Поэтому для облегчения перехода на Паскаль программистов старой школы и для тех редких случаев, когда использование goto оказывается более удобным, оператор был оставлен в языке.

В этой теме мы рассмотрим три оператора, изменяющие обычный ход течения программы, без использования которых вполне можно обойтись. Рассказываем мы о них для полноты изложения и из-за удобства их применения в ряде случаев.

Урок 14.1. Использование оператора безусловного перехода goto

Споры между противниками и сторонниками goto не утихают до сих пор. Мы приведем один из основных примеров оправданного использования goto.

Рассмотрим задачу проверки элементов массива на уникальность (определить, все ли элементы массива различны).

Проанализируем задание и методы его решения.

Что означает — все элементы различны? Это значит, что в массиве нет ни одной одинаковой пары элементов.

А как это проверить? Задачу проще решать от противного: постараемся найти в массиве два одинаковых элемента. Если таких не найдется, значит, все элементы различны.

Так как одинаковые элементы могут быть как угодно разбросаны по массиву, необходимо сравнить каждый элемент с каждым. То есть нужно сравнить первый элемент со всеми остальными (это цикл), затем второй элемент со всеми остальными (и это цикл), и так

перебрать все элементы. Это означает, что мы должны использовать вложенные циклы. Всего при этом у нас получится около N^2 сравнений (точнее, $(N^2-N)/2$).

Заметим, что если мы в какой-то момент найдем совпадающую пару элементов, перебирать все оставшиеся будет уже не обязательно. Значит, нужно выйти из обоих циклов. Вот для этого нам и понадобится оператор goto.

Пример 14.1. Проверка элементов массива на уникальность

```
program use_goto;

label konec;  { Чтобы указать, в какое место программы
                будет совершен переход оператором goto,
                это место необходимо пометить.
                В разделе Label описывается имя будущей
                метки, чтобы Паскаль не посчитал метку
                ошибочным оператором }

const N=10;     { Размер массива }

var a:array[1..N] of integer;

   i,j:integer; { Счетчики циклов }

   fl:boolean;
begin
   { Заполним массив и выведем его на экран }
   randomize;
   for i:=1 to N do
   begin
      a[i]:=random(15);
      write(a[i]:4)
   end;
   writeln;

   fl:=false;  { Переменная fl (флаг) нужна нам
                 для того, чтобы по выходе из цикла
                 определить, вышли мы по окончании
                 обоих циклов или раньше }

   for i:=1 to N-1 do
      for j:=i+1 to N do
```

```
        if a[i] = a[j] then
        begin
           fl:=true;

           goto konec   { переходим к метке "konec" }

        end;

konec:      { Это место программы мы и пометили
              как цель безусловного перехода }

  if fl then
     writeln('В массиве есть одинаковые элементы')
  else
     writeln('Все элементы массива уникальны');
  readln
end.
```

Итак, использование goto считается оправданным, если таким способом происходит выход из нескольких вложенных циклов вперед.

Справедливости ради приведем пример той же программы без использования goto.

Пример 14.2. Проверка элементов массива на уникальность без использования goto

```
program without_goto;

const N=10;      { Размер массива }

var a:array[1..N] of integer;

   i,j:integer;  { Счетчики циклов }

   fl:boolean;
begin
   { Заполним массив и выведем его на экран }
   randomize;
   for i:=1 to N do
   begin
      a[i]:=random(15);
      write(a[i]:4)
   end;
```

продолжение ⇩

Пример 14.2 (*продолжение*)

```
    writeln;

    fl:=false;    { Переменная fl (флаг) нужна нам
                    для того, чтобы по выходе из цикла
                    определить, вышли мы по окончании
                    обоих циклов или раньше }

    { Вместо for приходится использовать while,
      так как появляется еще одно условие окончания цикла
    }
    i:=1;
    while (j<N) and not fl do  { В каждом цикле мы
                                 проверяем два условия:
                                 выход за границы массива
                                 и флажок окончания
                                 поиска }
    begin
       j:=i+1;
       while (j<=N) and not fl do
       begin
          if a[i] = a[j] then
             fl:=true;
          j:=j+1
       end;
       i:=i+1
    end;
    if fl then
       writeln('В массиве есть одинаковые элементы')
    else
       writeln('Все элементы массива уникальны');
    readln
end.
```

Программа стала чуть сложнее из-за применения while вместо for, зато мы обошлись без goto.

Урок 14.2. Операторы, изменяющие ход выполнения цикла

В Паскале имеются еще два оператора, действие которых напоминает действие оператора безусловного пе-

рехода. Оба они применяются для изменения хода выполнения цикла (напоминаем, что в Паскале есть три вида циклов — for, while и repeat).

Оператор break

Оператор break прерывает действие текущего цикла и передает управление тому оператору программы, который должен выполниться после окончания цикла.

Разумеется, оператор break должен находиться внутри тела цикла.

Рассмотрим применение оператора break.

Пример 14.3. Вывод на экран первого по счету двузначного числа, сумма квадратов цифр которого делится на 17

```
var i:integer;
begin
   for i:=10 to 99 do
      if (sqr(i mod 10) + sqr(i div 10)) mod 17 = 0
      then break;
   writeln(i);
   readln
end.
```

Вместо применения оператора break можно использовать переменную-флажок, по значению которой определять, нужно ли выполнять цикл далее.

В данном случае (пример 14.3), так как факт наличия нужного числа считался заранее известным, проще было бы использовать цикл while, проверяющий только искомое условие.

Пример 14.4. Вывод на экран первого по счету двузначного числа, сумма квадратов цифр которого делится на 17 (без оператора break)

```
var i:integer;
begin
   i:=10;
   while (sqr(i mod 10) + sqr(i div 10)) mod 17 <> 0 do
      i:=i+1;
```

продолжение ⇨

Пример 14.4 (*продолжение*)

```
    writeln(i);
    readln
end.
```

Оператор continue

Оператор continue заканчивает выполнение текущего шага цикла. То есть после оператора continue сразу выполнится проверка условия окончания цикла. Если цикл еще должен продолжаться, начнет снова выполняться тело цикла.

Оператор continue, так же как и break, должен находиться внутри тела цикла.

Рассмотрим применение оператора continue.

Пример 14.5. Вывод на экран всех двузначных чисел, сумма квадратов цифр которых не делится на 17

```
var i:integer;
begin
   for i:=10 to 99 do
   begin
      if (sqr(i mod 10) + sqr(i div 10)) mod 17 = 0
      then continue;
      writeln(i)
   end;
   readln
end.
```

Как и в случае с break, использование оператора continue не является обязательным.

Выводы

1. Паскаль — структурный язык программирования. Любая программа на нем может быть составлена из блоков.
2. Для удобства работы программистов старой школы в языке сохранен оператор безусловного перехода

goto. Он позволяет выйти из блока операторов в другую точку программы.

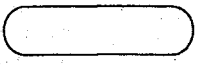
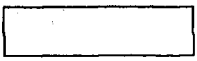
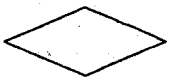
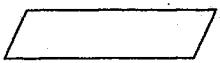
3. Для указания той точки, куда может выйти оператор goto, используется метка — произвольный идентификатор, после которого стоит двоеточие.
4. Идентификатор метки должен быть описан в разделе label.
5. На одну метку могут ссылаться несколько операторов goto.
6. Для досрочного завершения выполнения тела цикла можно использовать операторы break и continue.
7. Оператор break прекращает выполнение цикла. После его выполнения программа выполняет оператор, следующий после цикла.
8. Оператор continue прекращает выполнение текущего тела цикла. После его выполнения программа проверяет условие окончания цикла.

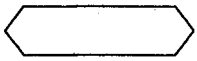
Контрольные вопросы

1. Какую функцию выполняет оператор безусловного перехода goto?
2. Почему нежелательно использовать в программе оператор goto?
3. Чем отличается действие оператора break от действия оператора continue?
4. В каком случае в программе удобно использовать оператор break? Можно ли без него обойтись?

Приложение 1. Элементы блок-схем

Таблица 1. Элементы блок-схем

Наименование	Обозначение	Описание
Начало/конец		Начало и конец любой блок-схемы
Действие		Любое простое (линейное) действие, кроме ввода-вывода, например, оператор присваивания (A := B + C)
Условие		Ветвление алгоритма. Имеет два выхода — «Да» и «Нет». Проверяется условие, записанное внутри блока. Если условие истинно — далее выполняется часть блок-схемы по метке «Да». Если ложно — по метке «Нет»
Ввод-вывод		Общение алгоритма с внешним миром. Первое слово внутри блока всегда — «Ввод» или «Вывод». Первое слово означает, что данные из внешнего мира попадают в алгоритм (ввод); второе — из алгоритма выдаются во внешний мир (вывод)
Процедура		Вызов алгоритма, написанного отдельным модулем (то есть имеющим свое «Начало» и «Конец»). Обычно это вспомогательный алгоритм

Наименование	Обозначение	Описание
Цикл с параметром		Внутри блока обычно указывается выражение вида I = 1, N. Это означает, что нижеследующие блоки алгоритма (заканчивающиеся возвратом к блоку «Цикл с параметром») будут выполняться N раз. То есть на каждом шаге цикла переменная I будет последовательно принимать значения от 1 до N с шагом 1

Несколько примечаний к таблице.

1. На языке блок-схем оператор присваивания принято обозначать стрелкой влево — A ← B + C. Мы используем обозначение присваивания из языка Паскаль, так как в этой книге нам оно кажется более уместным.
2. На языке блок-схем можно установить шаг цикла, не равный единице: I = 3, N, 2 — переменная I меняется от 3 до N с шагом 2.
3. Иногда в литературе обозначают параметры изменения цикла с параметром так: $i \leftarrow \overline{1, N}$.
4. Все блоки соединяются между собой тонкими линиями, которые должны быть горизонтальными или вертикальными (но не наклонными). Соединительная линия всегда должна входить в блок сверху посередине, а выходить снизу посередине. Исключение составляет блок «условие», у которого два выхода и они могут отходить влево, вправо или вниз.
5. Каждый блок должен иметь ровно один вход (сверху) и ровно один выход (снизу). Исключение составляют блоки начала-конца и блок условия.

Приложение 2. Домашние задания

При выполнении всех заданий рекомендуется оформлять в виде процедур:

- заполнение массива;
- вывод массива;
- обработку массива (в виде процедуры или функции).

Задания к главе 2

1. Вычислите по действиям значение выражения:
   1) 2–13 mod 7/3+sqr(4);
   2) sqr(9)–14 div 3*2+sqrt(4);
   3) 14–17 mod 6*2+abs(0.5–2.5);
   4) 12*3 div 5/3.5+sqr(1);
   5) sqrt(9)*2–19 mod 5 div 2;
   6) sqrt(19 div 2)/1.5–14 mod 7;
   7) sqr(21/7)+12 mod 7–2;
   8) abs(2–sqrt(9))–20 mod 7;
   9) sqr(2*sin(3–7 div 2));
   10) sqrt(19 mod 7+sqr(8 div 3));
   11) sqrt(1/0.25)+11 mod 6;
   12) abs(8-sqrt(2/0.5)+4 mod 3);
   13) sqr(4)/abs(39 mod 7–8).
2. Запишите на языке Паскаль:

   1) $y = \left| x^2 + \frac{x + 2,5}{3x} \right| - 3\sqrt{\sin 2x - \frac{2}{1 - x}}$;

   2) $y = 5\sqrt{\frac{2x - 3,5}{8x^2}} + e^{x+2} \cdot \cos 3x$;

3) $y=\dfrac{2\sqrt{x^2+\dfrac{2}{x}}}{2,1+|\sin x|}+\ln 2x$;

4) $y=\dfrac{|5x-6|}{\sqrt{x^2-3}+2,4}-3\ln\dfrac{2}{5x}$;

5) $y=4e^{\frac{x^2}{2}}-\dfrac{\cos 5\sqrt{x}}{1,2|3x-4|}$;

6) $y=\dfrac{\sin^2|3x|-1}{2x}-\dfrac{1}{\sqrt{3}}\ln\dfrac{1,7}{x+1}$;

7) $y=\dfrac{2-x}{3e^{|x|-1}}-\sqrt{4+\dfrac{x^2}{2,4}}$;

8) $y=\ln\dfrac{x^2-3}{4|x|}+3\sin\dfrac{2\sqrt{x}}{x-2,8}$;

9) $y=0,2e^{\frac{2x}{x^2-3}}-\dfrac{\sqrt{x-2}}{2\cos|x+1|}$;

10) $y=\dfrac{4\sqrt{x-2}}{\sin^2 2x}+\ln\dfrac{|x-3|}{2,6x}$;

11) $y=\dfrac{3e^{2|x|-1}}{4\sin 5x}-2\sqrt{x}\dfrac{3,5}{2-5x}$;

12) $y=\sin^2(2,5x-3)+3\sqrt{\dfrac{\cos^2 x}{|x+1|}}$;

13) $y=\dfrac{\cos^2|2x+1|}{3e^{5x-4}}+5\ln 4x$.

3. Запишите на языке Паскаль:

1) $y = 2|x-3|^{3x+5}$;

2) $y = \dfrac{5}{4\log_{x-2}|3+x|}$;

3) $y = \dfrac{2\,\mathrm{tg}^2\,2x - 1}{4}$;

4) $y = 7\,\mathrm{ctg}\,3x - \dfrac{1}{3x-5}$;

5) $y = (4-|x|)^{x-2} : (x+1)$;

6) $y = \dfrac{\log_{4x}(1-x)}{3|x|}$;

7) $y = \dfrac{3}{|x|}\,\mathrm{tg}^2\,5x$;

8) $y = \dfrac{x-2}{|\mathrm{ctg}^2\,2x - 1|}$;

9) $y = 2^{x^2 - \frac{1}{|x-2|}}$;

10) $y = \left|\log_{\frac{1}{2x}} 4x^2\right|$;

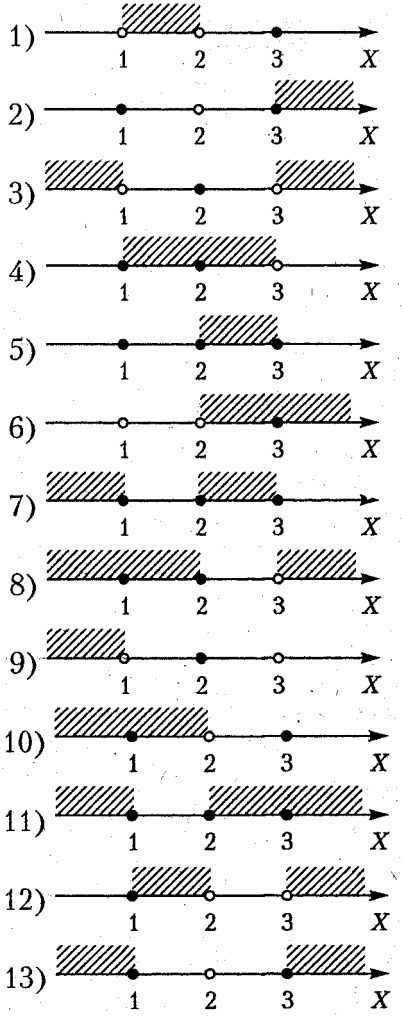
11) $y = 4|\mathrm{tg}\,3x^2 - 2|$;

12) $y = \dfrac{2-x}{\mathrm{ctg}|x^2-1|}$;

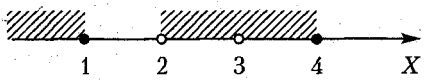
13) $y = \dfrac{7-x}{5x^{\sin^2 x}}$.

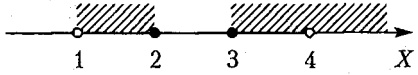
Задания к главе 4

1. Запишите на языке Паскаль выражение, которое истинно, если переменная x принадлежит заштрихованной области:

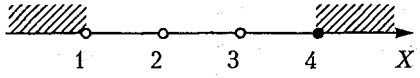


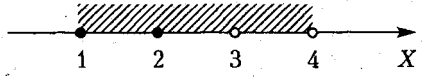
2. Запишите на языке Паскаль выражение, которое истинно, если переменная x принадлежит заштрихованной области:

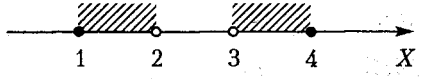
1) 

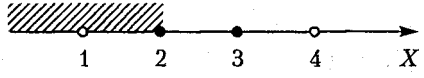
2) 

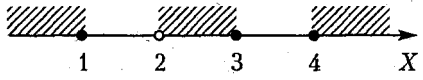
3) 

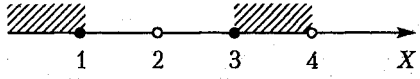
4) 

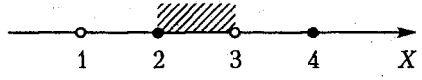
5) 

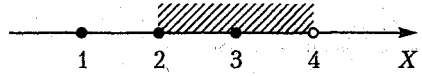
6) 

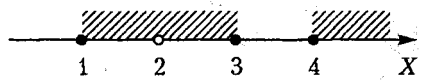
7) 

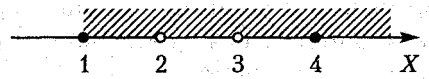
8) 

9) 

10) 

11) 

12) 

13) 

3. Запишите на языке Паскаль выражение, которое истинно, если переменная x:

1) принадлежит области $[-1;1]\cup[3;5]$;
2) не принадлежит области $[-1;1]\cup[3;5]$;
3) принадлежит области $[-\infty;1]\cup[2;4]\cup[5;+\infty]$;
4) не принадлежит области $[-\infty;1]\cup[2;4]\cup[5;+\infty]$;
5) принадлежит области $[-\infty;0]\cup[1;7]$;
6) не принадлежит области $[-\infty;0]\cup[1;7]$;
7) принадлежит области $[-4;4]\cup[8;+\infty]$;
8) не принадлежит области $[-4;4]\cup[8;+\infty]$;
9) принадлежит области $[-\infty;-5]\cup[-1;1]\cup[5;+\infty]$;
10) не принадлежит области $[-\infty;-5]\cup[-1;1]\cup[5;+\infty]$;
11) принадлежит области $[-3;2]\cup[4;8]$ и не равна ±1;
12) не принадлежит области $[-3;2]\cup[4;8]$ или равна 6;
13) принадлежит области $[-4;-3]\cup[1;2]\cup[3;4]$.

Задания к главам 6–7

Вариант 1

Цикл со счетчиком

С клавиатуры вводится n целых чисел. В ответ на экран выводится то же число, если оно меньше 7. В противном случае выводится число 7.

Цикл с предусловием

С клавиатуры вводятся числа до первого отрицательного. Определить, является ли последовательность чисел строго возрастающей.

Цикл с постусловием

Вычислите частичную сумму ряда

$$S = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots.$$

Вычисления прекратите, когда модуль очередного слагаемого станет меньше 0,001. Проверьте, насколько сумма приблизилась к значению 2.

Вариант 2

Цикл со счетчиком

С клавиатуры вводится n целых чисел $a_1, a_2, ..., a_n$. Если в последовательности чисел есть число, равное a_1, определите сумму всех чисел, следующих за первым таким числом. В противном случае выведите число –10.

Цикл с предусловием

Введите целое число и подсчитайте в нем количество разрядов.

Цикл с постусловием

Вычислите частичную сумму ряда

$$S = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \cdots.$$

Вычисления прекратите, когда модуль очередного слагаемого станет меньше 0,001. Значение x вводится с клавиатуры. Проверьте, насколько сумма приблизилась к значению sin (x), x измеряется в радианах.

Вариант 3

Цикл со счетчиком

С клавиатуры вводится n целых чисел $a_1, a_2, ..., a_n$. В процессе ввода чисел выводите на экран следующие суммы: $a_1 + a_2, a_2 + a_3, ..., a_{n-1} + a_n$.

Цикл с предусловием

Вычислите сумму всех чисел Фибоначчи, которые не превосходят заданного натурального M. Числа Фибоначчи определяются по формулам: $F_0 = F_1 + 1$; $F_i = F_{i-1} + F_{i-2}$.

Цикл с постусловием

Вычислите частичную сумму ряда

$$S = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots.$$

Вычисления прекратите, когда модуль очередного слагаемого станет меньше 0,001. Значение x вводится

с клавиатуры. Проверьте, насколько сумма приблизилась к значению cos (x), x измеряется в радианах.

Вариант 4

Цикл со счетчиком

С клавиатуры вводится n целых чисел $a_1, a_2, ..., a_n$. Выводите на экран в процессе ввода чисел: $a_1, 2a_2, 3a_3, ... na_n$.

Цикл с предусловием

С клавиатуры вводятся числа до тех пор, пока не введено 100. Определите, есть ли в последовательности чисел два подряд идущих нулевых числа.

Цикл с постусловием

Вычислите частичную сумму ряда

$$S = \frac{1}{1 \cdot 3} + \frac{1}{3 \cdot 5} + \frac{1}{5 \cdot 7} + \cdots.$$

Вычисления прекратите, когда модуль очередного слагаемого станет меньше 0,001. Проверьте, насколько сумма приблизилась к значению 1/2.

Вариант 5

Цикл со счетчиком

С клавиатуры вводится n вещественных чисел $a_1, a_2, ..., a_n$. Получите $(1 + r)/(1 + s)$, где r — сумма чисел последовательности, которые не превосходят 1, а s — сумма чисел, больших 1.

Цикл с предусловием

Вычислите частное от деления двух чисел нацело, используя операцию вычитания.

Цикл с постусловием

Вычислить частичную сумму ряда

$$S = \frac{1}{3 \cdot 5} + \frac{1}{7 \cdot 9} + \frac{1}{11 \cdot 13} + \cdots.$$

Вычисления прекратите, когда модуль очередного слагаемого станет меньше 0,001. Проверьте, насколько сумма приблизилась к значению $\pi/8$.

Вариант 6

Цикл со счетчиком

С клавиатуры вводится n вещественных чисел $a_1, a_2, ..., a_n$. Выводите на экран a_i, если абсолютная величина $a_i \le 2$, в противном случае выводите число 0,5.

Цикл с предусловием

Вычислите остаток от деления двух чисел нацело, используя операцию вычитания.

Цикл с постусловием

Вычислите частичную сумму ряда

$$S = \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \cdots.$$

Вычисления прекратите, когда модуль очередного слагаемого станет меньше 0,001. Проверьте, насколько сумма приблизилась к значению $\pi^2/6$.

Вариант 7

Цикл со счетчиком

С клавиатуры вводится n вещественных чисел $a_1, a_2, ..., a_n$. Определите число соседств двух чисел разного знака.

Цикл с предусловием

Вычислите произведение двух чисел, используя операцию сложения.

Цикл с постусловием

Вычислите частичную сумму ряда

$$S = 1 - \frac{1}{2} + \frac{1}{4} - \frac{1}{8} + \cdots.$$

Вычисления прекратите, когда модуль очередного слагаемого станет меньше 0,001. Проверьте, насколько сумма приблизилась к значению 2/3.

Вариант 8

Цикл со счетчиком

С клавиатуры вводятся последовательно 10 пар целых чисел x и y. В каждой паре определите максимальное число и выведите его на экран.

Цикл с предусловием

Вычислите частичную сумму ряда

$$S = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots.$$

Вычисления прекратите, когда модуль очередного слагаемого станет меньше 0,001. Проверьте, насколько сумма приблизилась к значению $\pi/4$.

Цикл с постусловием

С клавиатуры вводятся числа до первого положительного. Определите, является ли последовательность чисел строго убывающей.

Вариант 9

Цикл со счетчиком

С клавиатуры вводятся последовательно 10 пар целых чисел x и y. Выводите на экран только пары, имеющие противоположные знаки.

Цикл с предусловием

Вычислите частичную сумму ряда

$$S = -\frac{2}{1!} + \frac{3}{2!} - \frac{4}{3!} + \frac{5}{4!} \cdots.$$

Вычисления прекратите, когда модуль очередного слагаемого станет меньше 0,001.

Цикл с постусловием

С клавиатуры вводится число, которое определяет цвет следующей заливки экрана. Процесс повторяется до тех пор, пока не будет введен черный цвет.

Вариант 10

Цикл со счетчиком

Найдите сумму чисел, кратных 3, в диапазоне от 30 до 60.

Цикл с предусловием

Вычислите частичную сумму ряда

$$S = \frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \frac{1}{3 \cdot 4} + \cdots.$$

Вычисления прекратите, когда модуль очередного слагаемого станет меньше 0,001. Проверьте, насколько сумма приблизилась к значению 1/4.

Цикл с постусловием

С клавиатуры вводятся последовательно целые числа x и y. Пока не будут введены нулевые значения, в точку с этими координатами выводите на экран приветствие.

Вариант 11

Цикл со счетчиком

С клавиатуры вводится n вещественных чисел a_1, a_2, ..., a_n. Определите количество чисел, превышающих по модулю 100.

Цикл с предусловием

Вычислите частичную сумму ряда

$$S = \frac{1}{1 \cdot 2 \cdot 3} + \frac{1}{2 \cdot 3 \cdot 4} + \frac{1}{3 \cdot 4 \cdot 5} + \cdots.$$

Вычисления прекратите, когда модуль очередного слагаемого станет меньше 0,001.

Цикл с постусловием

Вычислите число ряда Фибоначчи с номером N. Числа Фибоначчи определяются по формулам: $F_0 = F_1 = 1$; $F_i = F_{i-1} + F_{i-2}$.

Вариант 12

Цикл со счетчиком

Вычислите

$$1 \cdot 2 + 2 \cdot 3 \cdot 4 + \ldots + n \cdot (n + 1) \cdot \ldots \cdot 2n.$$

Цикл с предусловием

С клавиатуры вводятся числа до первого отрицательного. Определите, есть ли в последовательности чисел три подряд идущих пятерки.

Цикл с постусловием

С клавиатуры вводятся последовательно целые числа x, y, m, n, которые определяют координаты левой

верхней и правой нижней вершин текстового окна. Цвет окна задается случайным образом. Процесс повторяется до тех пор, пока x не равен 0.

Вариант 13

Цикл со счетчиком

Вычислить $(2n)!!= 2 \cdot 4 \cdot 6 \cdot \ldots \cdot 2n$.

Цикл с предусловием

Возведите число n в квадрат, используя следующую схему:

$1^2 = 1$;

$2^2 = 1 + 3$;

$3^2 = 1 + 3 + 5$;

$4^2 = 1 + 3 + 5 + 7$...

Цикл с постусловием

Вычислите частичную сумму ряда

$$S = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots.$$

Вычисления прекратите, когда модуль очередного слагаемого станет меньше 0,001. Значение x вводится с клавиатуры.

Вариант 14

Цикл со счетчиком

С клавиатуры вводится n целых чисел a_1, a_2, ..., a_n. Найдите номер первого четного числа последовательности чисел. Если четных чисел нет, ответом должен быть нуль.

Цикл с предусловием

Возведите число n в куб, используя следующую схему:

$1^3 = 1$;

$2^3 = 3 + 5$;

$3^3 = 7 + 9 + 11$;

$4^3 = 13 + 15 + 17 + 19$;

$5^3 = 21 + 23 + 25 + 27 + 29$...

Цикл с постусловием

С клавиатуры вводятся целые числа до 100. Подсчитайте количество положительных и отрицательных чисел среди введенных.

Вариант 15

Цикл со счетчиком

С клавиатуры вводится n целых чисел a_1, a_2, ..., a_n. Выводите на экран в процессе ввода чисел: a_1, a_1a_2, $a_1a_2a_3$, ..., $a_1a_2a_3...a_n$.

Цикл с предусловием

Вычислить частичную сумму ряда

$$S = -\frac{1}{3 \cdot 1} + \frac{1}{5 \cdot 2} - \frac{1}{7 \cdot 3} + \frac{1}{9 \cdot 4} \cdots .$$

Вычисления прекратите, когда модуль очередного слагаемого станет меньше 0,001.

Цикл с постусловием

Введите число N. Разделите его нацело на 2, полученное частное также разделите на 2, и т. д. Процесс продолжайте до тех пор, пока очередное частное не станет меньше 2. На каждом шаге выводите на экран в строчку без пробелов промежуточные остатки от деления нацело и в завершение — последнее частное. На экране должно получиться перевернутое двоичное представление числа N.

Задания к главе 8

Вариант 1

1. Заполните одномерный массив из n элементов следующим образом:

Номера элементов массива	1	2	3	4	5	6	7	...
Значения элементов массива	1	4	9	16	25	36	49	...

Выведите результат на экран.

2. Заполните случайным образом одномерный массив из n элементов и обнулите элементы от 1-го до максимального включительно. Выведите оба массива на экран.
3. Заполните двумерный массив размерности $n \times n$ следующим образом:

0	0	0	0	0	...
2	2	2	2	2	...
0	0	0	0	0	...
4	4	4	4	4	...
0	0	0	0	0	...
...	...	...	...	...	...

Выведите результат на экран.

4. Заполните случайным образом двумерный массив размерности $n \times n$. На основании исходного массива создайте итоговый массив, в котором элементы исходного повернуты на 90° против часовой стрелки. Выведите оба массива на экран.

Пример:

Исходный массив 5 × 5

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Итоговый массив 5 × 5

5	10	15	20	25
4	9	14	19	24
3	8	13	18	23
2	7	12	17	22
1	6	11	16	21

Вариант 2

1. Заполните одномерный массив из n элементов следующим образом:

Номера элементов массива	1	2	3	4	5	6	7	...
Значения элементов массива	0	2	0	4	0	6	0	...

Выведите результат на экран.

2. Заполните случайным образом одномерный массив из n элементов и обнулите элементы от минимального до последнего включительно. Выведите оба массива на экран.
3. Заполните двумерный массив размерности $n \times n$ следующим образом:

1	1	1	1	1	...
2	2	2	2	2	...
3	3	3	3	3	...
4	4	4	4	4	...
5	5	5	5	5	...
...	...	...	...	...	...

Выведите результат на экран.

4. Заполните случайным образом двумерный массив размерности $n \times n$. На основании исходного массива создайте итоговый массив, в котором элементы исходного повернуты на 90° по часовой стрелке. Выведите оба массива на экран. Пример:

Исходный массив 5 × 5

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Итоговый массив 5 × 5

21	16	11	6	1
22	17	12	7	2
23	18	13	8	3
24	19	14	9	4
25	20	15	10	5

Вариант 3

1. Заполните одномерный массив из n элементов следующим образом:

Номера элементов массива	1	2	3	4	5	6	7	...
Значения элементов массива	2	4	6	8	10	12	14	...

   Выведите результат на экран.

2. Заполните случайным образом одномерный массив из n элементов и обменяйте последний и максимальный элементы местами. Выведите оба массива на экран.

3. Заполните двумерный массив размерности $n \times n$ следующим образом:

0	2	0	4	0	...
0	2	0	4	0	...
0	2	0	4	0	...
0	2	0	4	0	...
0	2	0	4	0	...
...	...	...	...	...	...

   Выведите результат на экран.

4. Заполните случайным образом двумерный массив размерностью $n \times n$. На основании исходного массива создайте итоговый массив, в котором элементы исходного отображены относительно главной диагонали. Выведите оба массива на экран.

   Пример:

Исходный массив 5 × 5

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Итоговый массив 5 × 5

1	6	11	16	21
2	7	12	17	22
3	8	13	18	23
4	9	14	19	24
5	10	15	20	25

Вариант 4

1. Заполните одномерный массив из n элементов следующим образом:

Номера элементов массива	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**
Значения элементов массива	1	0	4	0	25	0	49	...

   Выведите результат на экран.

2. Заполните случайным образом одномерный массив из n элементов и обменяйте первый и минимальный элементы местами. Выведите оба массива на экран.

   Заполните двумерный массив размерности $n \times n$ следующим образом:

1	2	3	4	5	...
1	2	3	4	5	...
1	2	3	4	5	...
1	2	3	4	5	...
1	2	3	4	5	...
...	...	...	...	...	...

   Выведите результат на экран.

   Заполните случайным образом двумерный массив размерности $n \times n$. На основании исходного массива создайте итоговый массив, в котором элементы исходного отображены относительно дополнительной диагонали. Выведите оба массива на экран.

   Пример:

Исходный массив 5 × 5

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Итоговый массив 5 × 5

25	20	15	10	5
24	19	14	9	4
23	18	13	8	3
22	17	12	7	2
21	16	11	6	1

Вариант 5

1. Заполните одномерный массив из n элементов следующим образом:

Номера элементов массива	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**
Значения элементов массива	1	8	27	64	125	216	343	...

   Выведите результат на экран.

2. Заполните случайным образом одномерный массив из n элементов и определите номер элемента с максимальной суммой соседей. Соседи 1-го элемента — последний и второй. Соседи последнего элемента — предпоследний и 1-й. Выведите результат на экран.

3. Заполните двумерный массив размерности $n \times n$ следующим образом:

1	0	3	0	5	...
1	0	3	0	5	...
1	0	3	0	5	...
1	0	3	0	5	...
1	0	3	0	5	...
...	...	...	...	...	...

   Выведите результат на экран.

4. Заполните случайным образом двумерный массив размерности $n \times n$. Обменяйте местами элементы следующих строк:

   1) 1-я строка — n-я строка

   2) 2-я строка — $(n-1)$-я строка

   3) 3-я строка — $(n-2)$-я строка

   4) ...

   Выведите оба массива на экран.

   Пример:

Исходный массив 5 × 5

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Итоговый массив 5 × 5

21	22	23	24	25
16	17	18	19	20
11	12	13	14	15
6	7	8	9	10
1	2	3	4	5

Вариант 6

1. Заполните одномерный массив из n элементов следующим образом (для $n = 8$):

Номера элементов массива	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Значения элементов массива	8	7	6	5	4	3	2	1

   Выведите результат на экран.

2. Заполните случайным образом одномерный массив из n элементов и определите количество четных элементов на нечетных местах. Выведите результат на экран.

3. Заполните двумерный массив размерности $n \times n$ следующим образом:

1	1	1	1	1	...
0	0	0	0	0	...
3	3	3	3	3	...
0	0	0	0	0	...
5	5	5	5	5	...
...	...	...	...	...	...

Выведите результат на экран.

4. Заполните случайным образом двумерный массив размерности $n \times n$. Поменяйте местами элементы следующих столбцов:

   1) 1-й столбец — n- й столбец

   2) 2-й столбец — $(n-1)$-й столбец

   3) 3-й столбец — $(n-2)$-й столбец

   4) ...

   Выведите оба массива на экран.

   Пример:

Исходный массив 5 × 5

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Итоговый массив 5 × 5

5	4	3	2	1
10	9	8	7	6
15	14	13	12	11
20	19	18	17	16
25	24	23	22	21

Вариант 7

1. Заполните одномерный массив из n элементов следующим образом (для $n = 8$):

Номера элементов массива	1	2	3	4	5	6	7	8
Значения элементов массива	2	1	4	3	6	5	8	7

Выведите результат на экран.

2. Заполните случайным образом одномерный массив из n элементов и определите минимальную сумму среди пар следующих элементов:

   1) 1-й элемент — n-й элемент

   2) 2-й элемент — (n–1)-й элемент

   3) 3-й элемент — (n–2)-й элемент

   4) ...

   Выведите результат на экран.

3. Заполните двумерный массив размерности $n \times n$ следующим образом (для $n = 6$):

0	0	0	0	0	0
1	0	0	0	0	0
1	2	0	0	0	0
1	2	3	0	0	0
1	2	3	4	0	0
1	2	3	4	5	0

Выведите результат на экран.

4. Заполните случайным образом двумерный массив размерности $n \times n$ (n — четно). Поменяйте местами элементы соседних столбцов:

   1) 1-й столбец — 2-й столбец

   2) 3-й столбец — 4-й столбец

   3) ...

   4) (n–1)-й столбец — n-й столбец

   5) ...

   Выведите оба массива на экран.

   Пример:

Исходный массив 4 × 4

1	2	3	4
6	7	8	9
11	12	13	14
16	17	18	19

Итоговый массив 4 × 4

2	1	4	3
7	6	9	8
12	11	14	13
17	16	19	18

Вариант 8

1. Заполните одномерный массив из n элементов следующим образом:

Номера элементов массива	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**
Значения элементов массива	3	6	9	12	15	18	21	...

   Выведите результат на экран.

2. Заполните случайным образом одномерный массив из n элементов и сдвиньте элементы циклически на 1 позицию вправо:

   Пример (для $n = 8$):

Исходный массив

Номера элементов массива	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Значения элементов массива	1	2	3	4	5	6	7	8

Итоговый массив

Номера элементов массива	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Значения элементов массива	8	1	2	3	4	5	6	7

   Выведите оба массива на экран.

3. Заполните двумерный массив размерности $n \times n$ следующим образом (для $n = 6$):

0	2	3	4	5	6
0	0	3	4	5	6
0	0	0	4	5	6
0	0	0	0	5	6
0	0	0	0	0	6
0	0	0	0	0	0

Выведите результат на экран.

4. Заполните случайным образом двумерный массив размерности $n \times n$. Определите позицию минимального элемента в массиве.

   Выведите на экран массив, выделив минимальный элемент цветом.

Вариант 9

1. Заполните одномерный массив из n элементов числами ряда Фибоначчи:

Номера элементов массива	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**
Значения элементов массива	1	1	2	3	5	8	13	...

   Выведите результат на экран.

2. Заполните случайным образом одномерный массив из n элементов и сдвиньте элементы циклически на 1 позицию влево.

   Пример (для $n = 8$).

Исходный массив

Номера элементов массива	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Значения элементов массива	1	2	3	4	5	6	7	8

Итоговый массив

Номера элементов массива	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Значения элементов массива	2	3	4	5	6	7	8	1

   Выведите оба массива на экран.

3. Заполните двумерный массив размерностью $n \times n$ следующим образом (для $n = 6$):

1	0	0	0	0	0
0	2	0	0	0	0
0	0	3	0	0	0
0	0	0	4	0	0
0	0	0	0	5	0
0	0	0	0	0	6

Выведите результат на экран.

4. Заполните случайным образом двумерный массив размерностью $n \times n$. Обнулите минимальные элементы в каждом столбце. Выведите оба массива на экран.

Вариант 10

1. Заполните одномерный массив из n элементов следующими числами:

Номера элементов массива	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**
Значения элементов массива	3	5	7	9	11	13	15	...

Выведите результат на экран.

2. Заполните случайным образом одномерный массив из n элементов и обменяйте следующие элементы местами:

   1) 1-й элемент — 2-й элемент

   2) 3-й элемент — 4-й элемент

   3) ...

   4) (n–1)-й элемент — n-й элемент

   Пример (для $n = 8$):

Исходный массив

Номера элементов массива	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Значения элементов массива	1	2	3	4	5	6	7	8

Итоговый массив

Номера элементов массива	1	2	3	4	5	6	7	...
	2	1	4	3	6	5	8	7

Выведите оба массива на экран.

3. Заполните двумерный массив размерности $n \times n$ следующим образом (для $n = 6$):

0	0	0	0	0	1
0	0	0	0	2	0
0	0	0	3	0	0
0	0	4	0	0	0
0	5	0	0	0	0
6	0	0	0	0	0

Выведите результат на экран.

4. Заполните случайным образом двумерный массив размерности $n \times n$. Обнулите максимальные элементы в каждой строке.

Выведите оба массива на экран.

Вариант 11

1. Заполните одномерный массив из n элементов следующими числами:

Номера элементов массива	1	2	3	4	5	6	7	...
Значения элементов массива	1	3	5	7	9	11	13	...

Выведите результат на экран.

2. Заполните случайным образом одномерный массив из n элементов и поменяйте следующие элементы местами:

   1) 1-й элемент — n-й элемент
   2) 2-й элемент — $(n-1)$-й элемент
   3) 3-й элемент — $(n-2)$-й элемент
   4) ...

Пример (для $n = 8$):

Исходный массив

Номера элементов массива	1	2	3	4	5	6	7	8
Значения элементов массива	1	2	3	4	5	6	7	8

Итоговый массив

Номера элементов массива	1	2	3	4	5	6	7	8
Значения элементов массива	8	7	6	5	4	3	2	1

Выведите оба массива на экран.

3. Заполните двумерный массив размерности $n \times n$ следующим образом (для $n = 6$):

0	0	0	0	0	6
0	0	0	0	5	6
0	0	0	4	5	6
0	0	3	4	5	6
0	2	3	4	5	6
1	2	3	4	5	6

Выведите результат на экран.

4. Заполните случайным образом двумерный массив размерности $n \times n$. Поменяйте местами максимальные элементы первого и последнего столбцов.

Выведите оба массива на экран.

Вариант 12

1. Заполните одномерный массив из n элементов следующими числами:

Номера элементов массива	1	2	3	4	5	6	7	...
Значения элементов массива	2	5	8	11	14	17	20	...

Выведите результат на экран.

2. Заполните случайным образом одномерный массив из n элементов (n — четно) и поменяйте следующие элементы местами:

1) 1-й элемент — $n/2$-й элемент
2) 2-й элемент — $(n/2-1)$-й элемент
3) ...
4) $(n/2+1)$-й элемент — n-й элемент
5) $(n/2+2)$-й элемент — $(n-1)$-й элемент
6) ...

Пример (для $n = 8$):

Исходный массив

Номера элементов массива	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Значения элементов массива	1	2	3	4	5	6	7	8

Итоговый массив

Номера элементов массива	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Значения элементов массива	4	3	2	1	8	7	6	5

Выведите оба массива на экран.

3. Заполните двумерный массив размерности $n \times n$ следующим образом (для $n = 6$):

1	2	3	4	5	6
1	2	3	4	5	0
1	2	3	4	0	0
1	2	3	0	0	0
1	2	0	0	0	0
1	0	0	0	0	0

Выведите результат на экран.

4. Заполните случайным образом двумерный массив размерностью $n \times n$. Поменяйте местами минимальные элементы 1-й и последней строки.

Выведите оба массива на экран.

Вариант 13

1. Заполните одномерный массив из n элементов (для $n = 8$) следующими числами:

Номера элементов массива	1	2	3	4	5	6	7	8
Значения элементов массива	800	700	600	500	400	300	200	100

Выведите результат на экран.

2. Заполните случайным образом одномерный массив из n элементов (n — четно) и поменяйте местами элементы правой и левой половины массива.

   Пример (для $n = 8$):

   Исходный массив

Номера элементов массива	1	2	3	4	5	6	7	8
Значения элементов массива	1	2	3	4	5	6	7	8

   Итоговый массив

Номера элементов массива	1	2	3	4	5	6	7	8
Значения элементов массива	5	6	7	8	1	2	3	4

   Выведите оба массива на экран.

3. Заполните двумерный массив размерности $n \times n$ следующим образом (для $n = 6$):

6	5	4	3	2	1
6	5	4	3	2	0
6	5	4	3	0	0
6	5	0	0	0	0
6	0	0	0	0	0
0	0	0	0	0	0

   Выведите результат на экран.

4. Заполните случайным образом двумерный массив размерности $n \times n$. Подсчитайте количество четных элементов в каждой строке. Выведите результат на экран.

Вариант 14

1. Заполните одномерный массив из n элементов (для $n = 8$) следующими числами:

Номера элементов массива	1	2	3	4	5	6	7	8
Значения элементов массива	4	7	10	13	16	19	22	25

Выведите результат на экран.

2. Заполните случайным образом одномерный массив из n элементов (n — четно) и выполните арифметический сдвиг вправо на k позиций ($k < n$).

   Пример (для $n = 8, k = 3$):

   Исходный массив

Номера элементов массива	1	2	3	4	5	6	7	8
Значения элементов массива	1	2	3	4	5	6	7	8

   Итоговый массив

Номера элементов массива	1	2	3	4	5	6	7	8
Значения элементов массива	0	0	0	1	2	3	4	5

   Выведите оба массива на экран.

3. Заполните двумерный массив размерности $n \times n$ следующим образом (для $n = 6$):

0	0	0	0	0	1
0	0	0	0	2	1
0	0	0	3	2	1
0	0	4	3	2	1
0	5	4	3	2	1
6	5	4	3	2	1

   Выведите результат на экран.

4. Заполните случайным образом двумерный массив размерности $n \times n$. Подсчитайте количество элементов, кратных 3, в каждом столбце. Выведите результат на экран.

Вариант 15

1. Заполните одномерный массив из n элементов (для $n = 8$) следующими числами:

Номера элементов массива	1	2	3	4	5	6	7	8
Значения элементов массива	800	700	600	500	400	300	200	100

Выведите результат на экран.

2. Заполните случайным образом одномерный массив из n элементов (n — четно) и выполните арифметический сдвиг влево на k позиций ($k < n$).

   Пример (для $n = 8$):

Исходный массив

Номера элементов массива	1	2	3	4	5	6	7	8
Значения элементов массива	1	2	3	4	5	6	7	8

Итоговый массив

Номера элементов массива	1	2	3	4	5	6	7	8
Значения элементов массива	4	5	6	7	8	0	0	0

Выведите оба массива на экран.

3. Заполните двумерный массив размерности $n \times n$ следующим образом (для $n = 6$):

1	0	0	0	0	1
0	2	0	0	2	0
0	0	3	3	0	0
0	0	4	4	0	0
0	5	0	0	5	0
6	0	0	0	0	6

Выведите результат на экран.

4. Заполните случайным образом двумерный массив размерности $n \times n$. Обнулите четные элементы в строках с четными номерами и нечетные элементы в строках с нечетными номерами. Выведите оба массива на экран.

Алфавитный указатель