СРЕДНЕЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАНИЕ

И. Г. СЕМАКИН, А. П. ШЕСТАКОВ

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

УЧЕБНИК

Допущено
Министерством образования Российской Федерации
в качестве учебника для студентов
образовательных учреждений среднего профессионального образования

3-е издание, стереотипное

ACADEMIA

Москва
Издательский центр «Академия»
2012

УДК 681.3.06(075.32)
ББК 22.18я723
С30

Рецензенты:
преподаватель спецдисциплин Московского государственного колледжа информационных технологий, председатель цикловой комиссии *Н. В. Валиулина*;
преподаватель цикловой комиссии «Вычислительная техника» Московского колледжа железнодорожного транспорта *А. А. Янушевская*

Семакин И. Г.

С30 Основы алгоритмизации и программирования : учебник для студ. учреждений сред. проф. образования / И. Г. Семакин, А. П. Шестаков. — 3-е изд., стер. — М. : Издательский центр «Академия», 2012. — 400 с.

ISBN 978-5-7695-8957-7

Рассмотрены основы структурной методики построения алгоритмов и программирования на базе языка Паскаль (версия ТурбоПаскаль (7.0), а также основные понятия объектно-ориентированного программирования и его реализация на языке ТурбоПаскаль. Описана интегрированная среда программирования Delphi и рассмотрены визуальная технология создания графического интерфейса программ и разработка программных модулей в этой среде.

Для студентов учреждений среднего профессионального образования.

УДК 681.3.06(075.32)
ББК 22.18я723

ISBN 978-5-7695-8957-7

ПРЕДИСЛОВИЕ

Программирование все в большей степени становится занятием лишь для профессионалов. Объявленный в середине 1980-х гг. лозунг «Программирование — вторая грамотность» остался в прошлом. В понятие «компьютерная грамотность» сегодня входит, прежде всего, навык использования многообразных средств информационных технологий. При решении той или иной информационной задачи сначала следует попытаться подобрать адекватное программное средство (электронные таблицы, системы управления базами данных, математические пакеты и др.), и только если эти средства не позволяют решить поставленную задачу, использовать универсальные языки программирования.

Различают программистов двух категорий: прикладных и системных. Системные программисты — это разработчики базовых программных средств ЭВМ (операционных систем, трансляторов, сервисных средств и т.д.), являющиеся профессионалами высочайшего уровня. Прикладные программисты разрабатывают средства программного обеспечения ЭВМ, предназначенные для решения задач в отдельных областях деятельности (науке, технике, производстве, сфере обслуживания, обучении и т.д.). Требования к качеству прикладных программ, так же высоки, как и к качеству системных. Любая программа должна не только правильно решать задачу, но и иметь современный интерфейс, быть высоконадежной, дружественной к пользователю и т.д. Только такие программы могут выдержать конкуренцию на мировом рынке программных продуктов.

По мере развития компьютерной техники развивались методика и технология программирования. Сначала возникли командное и операторное программирование, в 1960-х гг. бурно развивались структурное программирование, линии логического и функционального программирования, а в настоящее время широко распространяются объектно-ориентированное и визуальное программирование.

Задача, которую следует ставить в начале изучения программирования, — это освоение основ его структурной методики с помощью языка Паскаль, который его автор — швейцарский профессор Никлаус Вирт — создавал именно для этого. Структурная методика до настоящего времени остается основой программистской культуры. Не освоив ее, человек, взявшийся изучать программирование, не имеет никаких шансов стать профессионалом.

Реализации языка Паскаль в версиях фирмы Borland для IBM, известные под названием TurboPascal, значительно расширили язык по сравнению с вариантом Н. Вирта. Начиная с версии 5.5 TurboPascal стал также и языком объектного программирования.

Содержание главы 2 настоящего учебника ориентировано на глубокое усвоение учащимися базовых понятий языков программирования высокого уровня в их реализации на языке Паскаль, что значительно облегчит изучение других языков программирования. В главе 4 излагаются основы объектно-ориентированного программирования на примере их реализации на языке Паскаль. Здесь же рассматривается язык программирования Delphi, являющийся объектно-ориентированным расширением языка Паскаль с реализацией технологии визуального программирования.

При подготовке к изучению данного курса желательно усвоение учащимися основ алгоритмизации в рамках школьного базового курса информатики. Обычно в школе алгоритмизация изучается с использованием учебных исполнителей, позволяющих успешно освоить основы структурной методики:

- построение алгоритмов из базовых структур;
- применение метода последовательной детализации.

Желательно также иметь представление об архитектуре ЭВМ на уровне машинных команд (достаточно на модельных примерах учебных компьютеров, изучаемых в школьной информатике, т.е. не обязательно знание реальных языков команд или ассемблера). Это позволяет усвоить основные понятия программирования (переменной, присваивания); «входить в положение транслятора» и не делать ошибок, даже не помня каких-то деталей синтаксиса языка; предвидеть те «подводные камни», на которые может «напороться» программа в процессе выполнения. По существу все эти качества и отличают профессионального программиста от дилетанта.

Еще одно качество профессионала — это способность воспринимать красоту программы, т.е. получать эстетическое удовольствие от хорошо написанной программы. Нередко это чувство помогает интуитивно отличить неправильную программу от правильной. Однако основным критерием оценки программы должна быть, безусловно, не интуиция, а грамотно организованное тестирование.

Процесс изучения и практического освоения программирования подразделяется на три части:

- изучение методов построения алгоритмов;
- изучение языка программирования;
- изучение и практическое освоение определенной системы программирования.

Решению задач первой части посвящены главы 1 и 3 данного учебника. В главе 1 даются основные, базовые, понятия и принципы построения алгоритмов работы с величинами. В главе 3 излага-

ются некоторые известные методики полного построения алгоритмов, рассматриваются проблемы тестирования программ и оценки сложности алгоритмов.

Языки программирования ТурбоПаскаль и Delphi излагаются соответственно в главах 2 и 4 учебника. Однако подчеркнем, что данная книга — это, прежде всего, учебник по программированию, а не по языкам Паскаль и Delphi, поэтому исчерпывающего описания данных языков вы здесь не найдете, они излагаются в объеме, необходимом для начального курса программирования. Более подробное описание этих языков можно найти в книгах, указанных в списке литературы.

В данном учебнике нет инструкций по работе с конкретными системами программирования для изучаемых языков, с ними студенты должны познакомиться в процессе выполнения практических работ на ЭВМ, используя специальную литературу.

В главе 5 учебника содержатся задачи по программированию, которые можно использовать для организации практических и лабораторных занятий на любом из изучаемых языков.

ГЛАВА 1

ОСНОВНЫЕ ПРИНЦИПЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

1.1. Алгоритмы и величины

Этапы решения задачи на ЭВМ. Работа по решению любой задачи с использованием компьютера включает в себя следующие шесть этапов:

1. Постановка задачи.
2. Формализация задачи.
3. Построение алгоритма.
4. Составление программы на языке программирования.
5. Отладка и тестирование программы.
6. Проведение расчетов и анализ полученных результатов.

Часто эту последовательность называют *технологической цепочкой решения задачи на ЭВМ* (непосредственно к программированию из этого списка относятся п. 3...5).

На этапе постановки задачи следует четко определить, *что дано* и *что требуется найти*. Важно описать полный набор исходных данных, необходимых для решения задачи.

На этапе формализации чаще всего задача переводится на язык математических формул, уравнений и отношений. Если решение задачи требует математического описания какого-то реального объекта, явления или процесса, то ее формализация равносильна получению соответствующей математической модели.

Третий этап — это построение алгоритма. Опытные программисты часто сразу пишут программы на определенном языке, не прибегая к каким-либо специальным средствам описания алгоритмов (блок-схемам, псевдокодам), однако в учебных целях полезно сначала использовать эти средства, а затем переводить полученный алгоритм на язык программирования.

Первые три этапа — это работа без компьютера. Последующие два этапа — это собственно программирование на определенном языке в определенной системе программирования. На последнем — шестом — этапе разработанная программа уже используется в практических целях.

Таким образом, программист должен уметь строить алгоритмы, знать языки программирования, уметь работать в соответствующей системе программирования.

Основой профессиональной грамотности программиста является развитое алгоритмическое мышление.

Понятие алгоритма. Одним из фундаментальных понятий в информатике является понятие алгоритма. Сам термин «алгоритм», заимствованный из математики, происходит от *лат.* Algorithmi — написание имени Мухамеда аль-Хорезми (787 — 850), выдающегося математика средневекового Востока. В XII в. был осуществлен латинский перевод его математического трактата, из которого европейцы узнали о десятичной позиционной системе счисления и правилах арифметики многозначных чисел. Именно эти правила в то время называли алгоритмами. Сложение, вычитание, умножение «столбиком», деление «уголком» многозначных чисел — это первые алгоритмы в математике. Правила алгебраических преобразований и вычисление корней уравнений также можно отнести к математическим алгоритмам.

В наше время понятие алгоритма трактуется шире. *Алгоритм* — это последовательность команд управления каким-либо исполнителем. В школьном курсе информатики с понятием алгоритма и методами построения алгоритмов ученики знакомятся на примерах учебных исполнителей: Робота, Черепахи, Чертежника и др. Эти исполнители ничего не вычисляют. Они создают рисунки на экране, перемещаются в лабиринтах, перетаскивают предметы с места на место. Таких исполнителей принято называть *исполнителями*, *работающими в обстановке*.

В разделе «Программирование» информатики изучаются методы программного управления работой ЭВМ, т. е. в качестве исполнителя выступает компьютер. Компьютер работает с величинами — различными информационными объектами: числами, символами, кодами и др., поэтому алгоритмы, предназначенные для управления компьютером, называются *алгоритмами работы с величинами*.

Данные и величины. Совокупность величин, с которыми работает компьютер, принято называть *данными*. По отношению к программе различают *исходные*, *окончательные* (*результаты*) и *промежуточные* данные, которые получают в процессе вычислений (рис. 1.1).

Например, при решении квадратного уравнения $ax^2 + bx + c = 0$ исходными данными являются коэффициенты a, b, c, результатами — корни уравнения x_1, x_2, а промежуточными данными — дискриминант уравнения $D = b^2 - 4ac$.

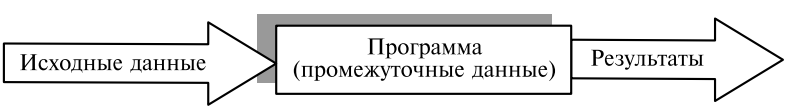

Рис. 1.1. Уровни данных относительно программы

Для успешного освоения программирования необходимо усвоить следующее правило: *всякая величина занимает свое определенное место в памяти ЭВМ*, иногда говорят — ячейку памяти. Термин «ячейка» для архитектуры современных ЭВМ несколько устарел, однако в учебных целях его удобно использовать.

Любая величина имеет три основных свойства: *имя*, *значение* и *тип*. На уровне команд процессора величина идентифицируется адресом ячейки памяти, в которой она хранится. В алгоритмах и языках программирования величины подразделяются на *константы* и *переменные*. Константа — неизменная величина, и в алгоритме она представляется собственным значением, например: 15, 34.7, k, True и др. Переменные величины могут изменять свои значения

Таблица 1.1

Основные типы данных

Тип	Значения	Операции	Внутреннее представление
Целые	Целые положительные и отрицательные числа в некотором диапазоне, например: 23, –12, 387	Арифметические операции с целыми числами: сложение, вычитание, умножение, целое деление и деление с остатком. Операции отношений (<, >, = и др.)	Формат с фиксированной точкой
Вещественные	Любые (целые и дробные) числа в некотором диапазоне, например: 2.5, –0.01, 45.0, 3.6×10^9	Арифметические операции. Операции отношений	Формат с плавающей точкой
Логические	True (истина) False (ложь)	Логические операции: И (and), ИЛИ (or), НЕТ (not).Операции отношений	1 — True; 0 — False
Символьные	Любые символы компьютерного алфавита, например: a, 5, +, $	Операции отношений	Коды таблицы символьной кодировки, например: ASCII — один символ — 1 байт; Unicode — один символ — 2 байт

в ходе выполнения программы и представляются в алгоритме символическими именами — идентификаторами, например: X, S2, cod15 и др. Любые константы и переменные занимают ячейку памяти, а значения этих величин определяются двоичным кодом в этой ячейке.

Теперь о типах величин — *типах данных* — понятии, которое встречается при изучении в курсе информатики баз данных и электронных таблиц. Это понятие является фундаментальным в программировании.

В каждом языке программирования существует своя концепция и своя система типов данных. Однако в любой язык входит минимально необходимый набор основных типов данных: *целые*, *вещественные*, *логические* и *символьные*. С типом величины связаны три ее свойства: множество допустимых значений, множество допустимых операций, форма внутреннего представления. В табл. 1.1 представлены свойства основных типов данных.

Типы констант определяются по контексту (т.е. по форме записи в тексте), а типы переменных устанавливаются в описаниях переменных.

По структуре данные подразделяются на *простые* и *структурированные*. Для простых величин, называемых также скалярными, справедливо утверждение *одна величина — одно значение*, а для структурированных — *одна величина — множество значений*. К структурированным величинам относятся массивы, строки, множества и др.

ЭВМ — исполнитель алгоритмов. Как известно, каждый алгоритм (программа) составляется для конкретного исполнителя, т.е. в рамках его системы команд. О каком же исполнителе идет речь при изучении темы «Программирование для ЭВМ»? Ответ очевиден: исполнителем здесь является компьютер, а точнее говоря, комплекс ЭВМ + система программирования (СП). Программист составляет программу на том языке, на который ориентирована СП. Схематически это изображено на рис. 1.2, где *входным языком* исполнителя является язык программирования Паскаль.

Независимо от того, на каком языке программирования будет написана программа, алгоритм решения любой задачи на ЭВМ

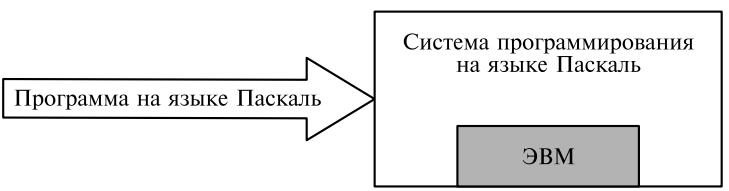

Рис. 1.2. Компьютер как исполнитель программ на языке Паскаль

может быть составлен из следующих команд: присваивания, ввода, вывода, обращения к вспомогательному алгоритму, цикла, ветвления.

Далее для описания алгоритмов будут использоваться блок-схемы и учебный алгоритмический язык (АЯ), применяемый в школьном курсе информатики.

1.2. Линейные вычислительные алгоритмы

Основным элементарным действием в вычислительных алгоритмах является *присваивание значения переменной величине*.

Если значение константы определено видом ее записи, то переменная величина получает конкретное значение только в результате присваивания, которое может осуществляться двумя способами: с помощью команды присваивания и с помощью команды ввода.

Рассмотрим пример. В школьном учебнике математики правило деления двух обыкновенных дробей описано следующим образом:

1) числитель первой дроби умножить на знаменатель второй дроби;

2) знаменатель первой дроби умножить на числитель второй дроби;

3) записать дробь, числитель которой есть результат выполнения п. 1, а знаменатель — результат выполнения п. 2.

Алгебраическая форма записи этого примера следующая:

$$\frac{a}{b} : \frac{c}{d} = \frac{a \cdot d}{b \cdot c} = \frac{m}{n}$$

Теперь построим алгоритм деления дробей для ЭВМ, сохранив те же обозначения переменных, которые были использованы в алгебраическом выражении.

Исходными данными здесь являются целочисленные переменные *a*, *b*, *c*, *d*, а результатом — целые величины *m* и *n*. Блок-схема алгоритма деления дробей приведена на рис. 1.3. Данный алгоритм на учебном алгоритмическом языке будет иметь следующий вид:

```
алг деление дробей
нач
    цел a, b, c, d, m, n
    ввод a, b, c, d
    m := a ×d
    n := b ×c
    вывод m, n
кон
```

Формат команды присваивания следующий:

```
переменная := выражение
```

Знак «:=» следует читать как «присвоить».

Команда присваивания означает следующие действия, выполняемые компьютером:

1) вычисляется *выражение*;

2) полученное значение присваивается *переменной*.

В приведенном алгоритме присутствуют две команды присваивания. В блок-схемах команды присваивания изображаются в прямоугольниках. Такой блок называется вычислительным.

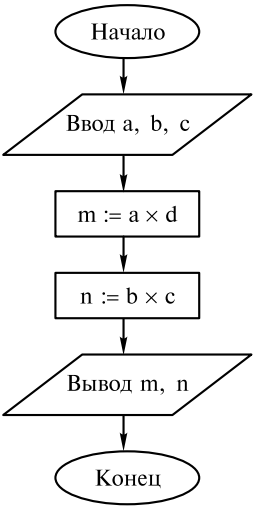

Рис. 1.3. Блок-схема алгоритма деления дробей

При описании алгоритмов не обязательно соблюдать строгие правила записи выражений, это можно делать в обычной математической форме, так как это еще не язык программирования со строгим синтаксисом.

В рассматриваемом алгоритме имеется команда ввода:

```
ввод a, b, c, d
```

В блок-схемах команда ввода записывается в параллелограмме — блоке ввода-вывода. При выполнении этой команды процессор прерывает работу и ожидает действий пользователя. Пользователь должен набрать на устройстве ввода (клавиатуре) значения вводимых переменных и нажать клавишу ввода <Enter>. Значения следует вводить в том же порядке, в каком эти переменные расположены в списке ввода. Обычно с помощью команды ввода присваиваются значения исходных данных, а команда присваивания используется для получения промежуточных и конечных величин.

Полученные компьютером результаты решения задачи должны быть сообщены пользователю, для чего и предназначена команда вывода:

```
вывод m, n
```

С помощью этой команды результаты выводятся на экран или через устройство печати на бумагу.

Поскольку присваивание является важнейшей операцией в вычислительных алгоритмах, обсудим ее более подробно.

Рассмотрим последовательное выполнение четырех команд присваивания, в которых участвуют две переменные величины — a, b. В табл. 1.2 для каждой команды присваивания указаны значения переменных, которые устанавливаются после ее выполнения.

Этот пример иллюстрирует три основных свойства присваивания:

- пока переменной не присвоено значение, она остается неопределенной;
- значение, присвоенное переменной, сохраняется вплоть до выполнения следующего присваивания этой переменной;
- новое значение, присваиваемое переменной, заменяет ее предыдущее значение.

Рассмотрим алгоритм, который часто используется при программировании. Даны две величины: X и Y. Требуется произвести между ними обмен значениями. Например, если сначала было X = 1, Y = 2, то после обмена должно стать X = 2, Y = 1.

Этой задаче аналогична следующая ситуация. Имеются два стакана: один — с молоком, другой — с водой. Требуется произвести между ними обмен содержимым. Ясно, что в этом случае необходим третий стакан — пустой. Последовательность действий при обмене будет следующей:

1) перелить молоко из 1-го стакана в 3-й;
2) воду из 2-го стакана в 1-й;
3) молоко из 3-го стакана во 2-й.

Аналогично для обмена значениями двух переменных требуется третья дополнительная переменная. Назовем ее Z. Тогда задачу обмена значениями можно решить последовательным выполнением трех команд присваивания (табл. 1.3).

Пример со стаканами не совсем точно характеризует ситуацию обмена между переменными, так как при переливании жидкостей один из стаканов становится пустым. В результате же выполнения команды присваивания (X := Y) переменная, стоящая справа (Y), сохраняет свое значение.

Таблица 1.2

Изменение значений переменных при выполнении команды присваивания

Команда	a	b
a := 1	1	—
b := 2 × a	1	2
a := b	2	2
b := a + b	2	4

Таблица 1.3

Обмен значениями между переменными

Команда	X	Y	Z
ввод X, Y	1	2	—
Z := X	1	2	1
X := Y	2	2	1
Y := Z	2	1	1

Алгоритм деления дробей имеет линейную структуру, т. е. в нем все команды выполняются в строго определенной последовательности, каждая по одному разу. Линейный алгоритм состоит из команд присваивания, ввода, вывода и обращения к вспомогательным алгоритмам (что будет рассмотрено далее).

При описании алгоритмов с помощью блок-схем типы данных, как правило, не указываются (но подразумеваются). В учебных алгоритмах (на АЯ) для всех переменных типы данных указываются явно, и их описание производится сразу после заголовка алгоритма. При этом используются следующие обозначения: **цел** — целые, **вещ** — вещественные, **лит** — символьные (литерные), **лог** — логические. В алгоритме деления дробей все переменные целого типа.

1.3. Ветвления и циклы в вычислительных алгоритмах

Составим алгоритм решения квадратного уравнения

$$ax^2 + bx + c = 0.$$

Эта задача хорошо знакома из математики. Исходными данными здесь являются коэффициенты a, b, c, а решением в общем случае — два корня (x_1 и x_2), которые вычисляются по формуле

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Все используемые в этой программе величины вещественного типа.

Алгоритм решения кваратного уравнения будет иметь следующий вид:

```
алг корни квадратного уравнения
вещ a, b, c, d, x1, x2
нач ввод a, b, c
    d  := b^2 − 4ac
    x1 := (−b + √d)/(2a)
    x2 := (−b - √d)/(2a)
    вывод x1, x2
кон
```

Недостаток такого алгоритма виден «невооруженным глазом». Он не обладает важнейшим свойством, предъявляемым к качественным алгоритмам: универсальностью по отношению к исходным данным. *Какими бы ни были значения исходных данных, алгоритм должен приводить к получению определенного результата и*

выполняться до конца. Результатом может быть числовой ответ, но может быть и сообщение о том, что при таких данных задача решения не имеет. Недопустимы остановки в середине алгоритма из-за невозможности выполнения какой-либо операции. Данное свойство в литературе по программированию называют результативностью алгоритма (получение какого-то результата в любом случае).

Для построения универсального алгоритма сначала требуется тщательно проанализировать математическое содержание задачи.

Решение квадратного уравнения зависит от значений коэффициентов *a*, *b*, *c*.

Проанализируем эту задачу, ограничиваясь поиском только вещественных корней:

если a = 0, b = 0, c = 0, любое значение x — решение уравнения;

если a = 0, b = 0, c ≠ 0, уравнение решений не имеет;

если a = 0, b ≠ 0, это линейное уравнение, имеющее одно решение x = –c/b;

если a ≠ 0 и $d = b^2 - 4ac \geq 0$, уравнение имеет два вещественных корня x_1, x_2;

если a ≠ 0 и d < 0, уравнение не имеет вещественных корней.

Блок-схема алгоритма решения квадратного уравнения показана на рис. 1.4.

Этот же алгоритм на алгоритмическом языке будет иметь следующий вид:

```
алг корни квадратного уравнения
вещ a, b, c, d, x1, x2
нач ввод a, b, c
    если a = 0
    то   если  b = 0
         то    если c = 0
               то вывод «Любое x — решение»
               иначе вывод «Нет решений»
               кв
         иначе x := –c/b
               вывод x
         кв
    иначе d := b^2 – 4ac
          если d < 0
          то вывод «Нет вещественных корней»
    иначе x1 := (–b + √d)/(2a); x2 := (–b - √d)/(2a)
          вывод «x1 =», x1, «x2 =», x2
    кв
кв
кон
```

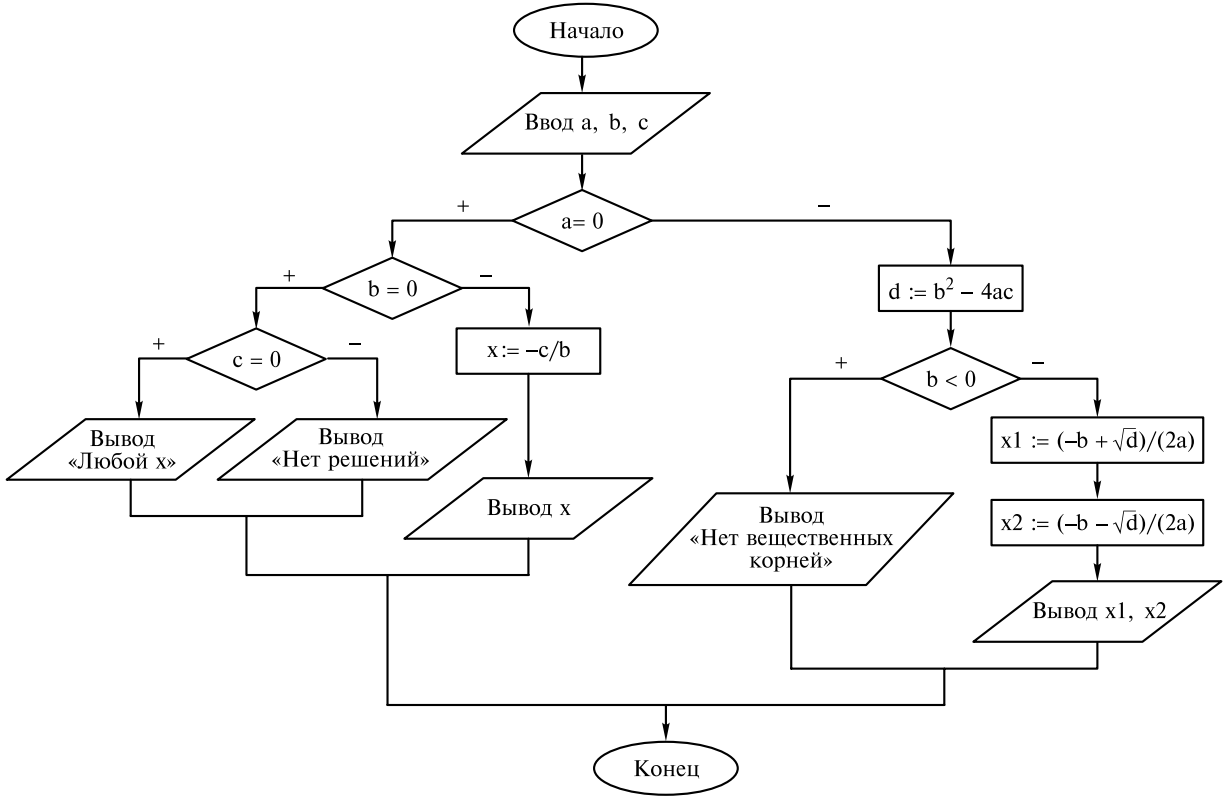

Рис. 1.4. Блок-схема алгоритма решения квадратного уравнения

В данном алгоритме многократно использована *структурная команда ветвления*, общий вид которой в виде блок-схемы показан на рис. 1.5. На алгоритмическом языке команду ветвления можно записать в следующем виде:

```
если условие
то серия 1
иначе серия 2
кв
```

В соответствии с приведенной блок-схемой команды ветвления сначала проверяется условие (вычисляется логическое выражение). Если условие истинно, то выполняется последовательность команд, на которую указывает стрелка с надписью «Да» (положительная ветвь) — «Серия 1». В противном случае выполняется отрицательная ветвь команд — «Серия 2».

На АЯ условие записывается после служебного слова «если», положительная ветвь — после слова «то», а отрицательная — после слова «иначе». Буквами «кв» в такой записи обозначают конец ветвления.

Если на ветвях одного ветвления содержатся другие ветвления, значит, алгоритм имеет структуру *вложенных ветвлений*. Именно такую структуру имеет алгоритм решения квадратного уравнения (см. рис. 1.4), в котором для краткости вместо слов «Да» и «Нет» использованы соответственно знаки «+» и «–».

Рассмотрим следующую задачу: дано целое положительное число n. Требуется вычислить $n!$ (n-факториал). Вспомним определение факториала:

$$n! = \begin{cases} 1, & \text{если } n = 0; \\ 1 \times 2 \times \ldots \times n, & \text{если } n \geq 1. \end{cases}$$

На рис. 1.6 приведена блок-схема алгоритма вычисления $n!$ с тремя переменными целого типа: n — аргумент, i — промежуточная переменная, F — результат. Для проверки правильности указанного алгоритма построим трассировочную табл. 1.4, в которой для конкретных значений исходных данных по шагам прослеживается изменение переменных, входящих в алгоритм. Данная таблица составлена для случая $n = 3$.

Приведенная трассировка доказывает правильность рассматриваемого алгоритма. Теперь запишем этот алгоритм на алгоритмическом языке:

```
алг факториал
цел n, i, F
нач ввод n
```

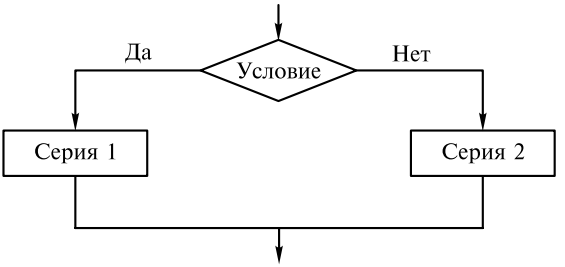

Рис. 1.5. Блок-схема команды ветвления

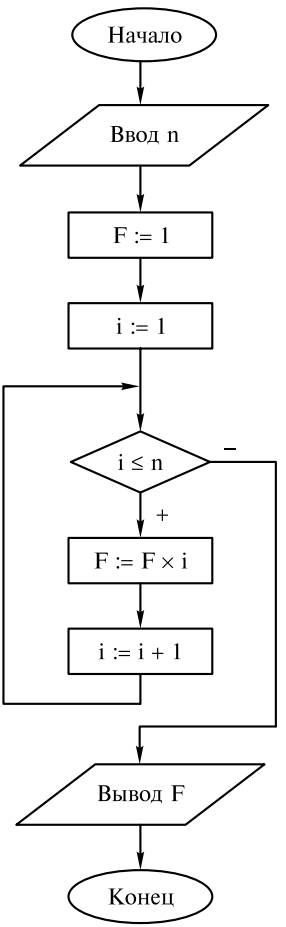

Рис. 1.6. Блок-схема алгоритма вычисления *n*!

Таблица 1.4

Трассировка алгоритма вычисления *n*!

Шаг	*n*	*F*	*i*	Условие
1	3			
2		1		
3			1	
4				1 ≤ 3, да
5		1		
6			2	
7				2 ≤ 3, да
8		2		
9			3	
10				3 ≤ 3, да
11		6		
12			4	
13				4 ≤ 3, нет
14		вывод		

```
    F := 1; i := 1
    пока i ≤ n, повторять
    нц   F := F × i
         i := i + 1
    кц
    вывод F
кон
```

Данный алгоритм имеет циклическую структуру. В нем использована структурная команда цикл «Пока», или цикла с предусловием. Блок-схема команды цикла «Пока» показана на рис. 1.7. На АЯ она имеет следующий вид:

```
пока условие, повторять
нц
     серия
кц
```

Выполнение серии команд (тела цикла) повторяется пока условие цикла истинно. Когда условие становится ложным, выпол-

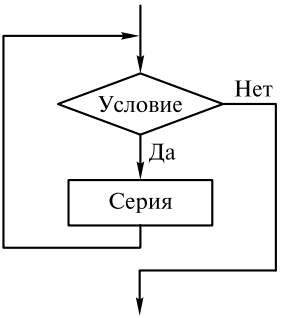

Рис. 1.7. Блок-схема команды цикла «Пока»

нение цикла заканчивается. Служебные слова «нц» и «кц» обозначают соответственно начало и конец цикла.

Цикл с предусловием — это основная, но не единственная форма организации циклических алгоритмов: существует цикл с постусловием.

Вернемся к алгоритму решения квадратного уравнения, рассмотрев его со следующей позиции: если a = 0 — это уже не квадратное уравнение и его можно не решать. В данном случае будем считать, что пользователь ошибся при вводе данных, и ввод следует повторить (рис. 1.8). Иначе говоря, в алгоритме будет предусмотрен контроль достоверности исходных данных с предоставлением пользователю возможности исправления ошибки. Наличие такого контроля — еще один признак хорошего качества программы.

На АЯ алгоритм решения квадратного уравнения с контролем ввода данных будет иметь следующий вид:

```
алг квадратное уравнение
вещ a, b, c, d, x1, x2
нач
    повторять
       ввод a, b, c
    до a ≠ 0
       d := b^2 – 4ac
    если d ≥ 0
    то x1 := (–b + √d)/(2a)
       x2 := (–b - √d)/(2a)
       вывод x1, x2
    иначе
       вывод «Нет вещественных корней»
    кв
кон
```

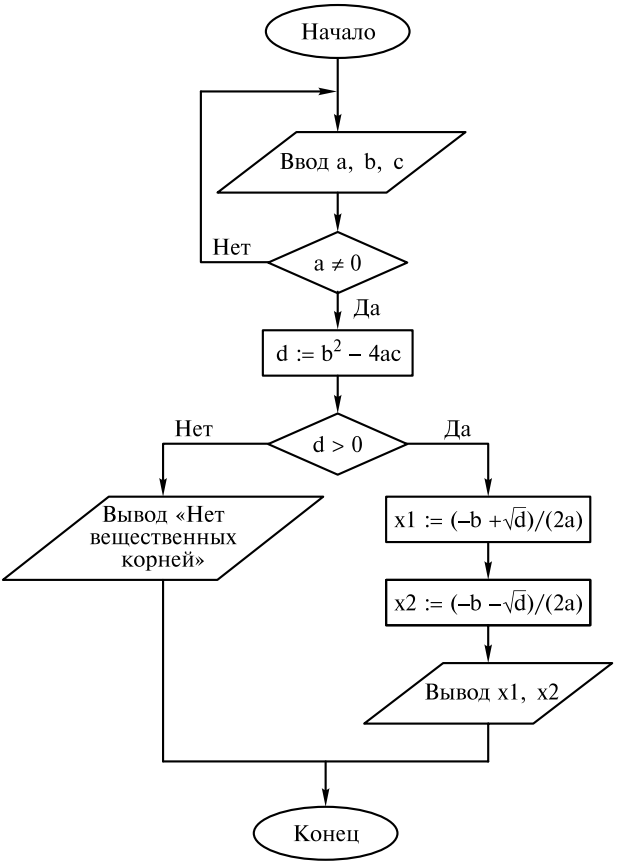

Рис. 1.8. Блок-схема алгоритма решения квадратного уравнения с контролем ввода данных

Блок-схема структурной команды цикла с постусловием цикла «До» показана на рис. 1.9. На АЯ данную команду можно записать следующим образом:

```
повторять
   серия
до условие
```

Здесь используется условие окончания цикла, т. е. когда оно становится истинным, цикл заканчивает работу.

Составим алгоритм решения следующей задачи: даны два натуральных числа M и N. Требуется вычислить их наибольший общий делитель — НОД (M, N).

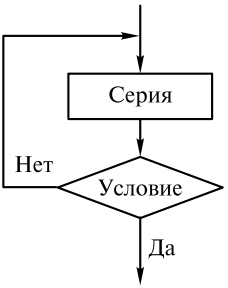

Рис. 1.9. Блок-схема команды цикла «До»

Эта задача решается с помощью метода, известного под названием алгоритма Евклида. Идея этого метода основана на утверждении, что если M > N, то НОД (M, N) = НОД (M – N, N). Попробуйте доказать это самостоятельно. Другое утверждение,

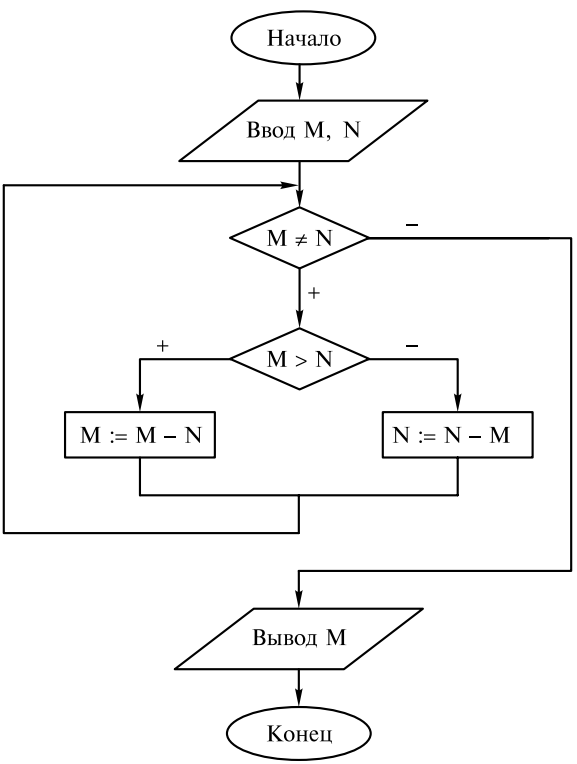

Рис. 1.10. Блок-схема алгоритма Евклида

лежащее в основе решения данного алгоритма, очевидно: НОД (M, M) = M. Для «ручного» выполнения этот алгоритм можно описать в форме следующей инструкции:

1) если числа равны, взять их общее значение в качестве ответа. В противном случае продолжить выполнение алгоритма;

2) определить большее из чисел;

3) заменить большее число разностью большего и меньшего значений;

4) вернуться к выполнению п. 1.

Блок-схема алгоритма Евклида приведена на рис. 1.10. На АЯ данный алгоритм можно записать следующим образом:

```
алг Евклид
цел M, N
нач ввод M, N
    пока M ≠ N, повторять
    нц   если M > N
         то M := M − N
         иначе N := N − M
         кв
    кц
кон
```

Алгоритм Евклида имеет структуру цикла с вложенным ветвлением. Выполните самостоятельно трассировку этого алгоритма для случая $M = 18$, $N = 12$. В результате должен получиться НОД = 6.

1.4. Логические основы алгоритмизации

Прямое отношение к программированию имеет дисциплина, называемая математической логикой, основу которой составляет алгебра логики, или исчисление высказываний. Под высказыванием понимается любое утверждение, в отношении которого можно однозначно сказать, истинно оно или ложно. Например, утверждение «Луна — спутник Земли» — истинно, «5 > 3» — истинно, «Москва — столица Китая» — ложно, «1 = 0» — ложно. ИСТИНА и ЛОЖЬ являются логическими значениями. Логические значения приведенных утверждений однозначно определены. Другими словами, их значения являются *логическими константами*.

Логическое значение неравенства $x < 0$, где x — переменная, является переменным, т. е. в зависимости от значения x оно может быть либо истинным, либо ложным. Таким образом вводится понятие *логической переменной*. В описаниях алгоритмов, а также в программах на различных языках программирования, логические переменные могут обозначаться символическими именами, кото-

рым соответствует логический тип данных. Иначе говоря, если известно, что *A*, *B*, *X*, *Y* и др. — переменные логического типа, это означает, что они могут принимать только значение ИСТИНА или ЛОЖЬ.

Основы формального аппарата математической логики создал в середине XIX в. английский математик Джордж Буль. В его честь исчисление высказываний называется булевой алгеброй, а логические величины — булевскими.

Логическое выражение (*логическая формула*) — это простое или сложное высказывание. Сложное высказывание строится из простых с помощью *логических операций* (связок).

Имеются три основных логических операции: отрицание, конъюнкция (логическое умножение) и дизъюнкция (логическое сложение).

Отрицание обозначается в математической логике знаком «¬» и читается как НЕ. Это одноместная операция. Например, запись $\neg(x = y)$ читается следующим образом: НЕ (*x* равно *y*), т.е. значение будет истинным, если *x* не равно *y*, и ложным, если *x* равно *y*. Отрицание изменяет значение логической величины на противоположное.

Конъюнкция обозначается знаком & и читается как И. Это двухместная операция. Например, запись $(x > 0)$ & $(x < 1)$ означает, что данная логическая формула примет значение ИСТИНА, если $x \in [0, 1]$, и значение ЛОЖЬ — в противном случае. Следовательно, результатом конъюнкции является ИСТИНА, если истинны оба операнда.

Дизъюнкция обозначается знаком $\vee$, который читается как ИЛИ. Например, запись $(x = 0) \vee (x = 1)$ означает, что формула принимает истинное значение, если *x* — двоичная цифра (0 или 1). Следовательно, результатом дизъюнкции является ИСТИНА, если хотя бы один операнд имеет значение ИСТИНА.

Правила выполнения рассмотренных логических операций отражены в табл. 1.5, называемой таблицей истинности (здесь *A* и *B* — логические величины, а буквами «И» и «Л» обозначены соответственно значения ИСТИНА и ЛОЖЬ).

Таблица 1.5

Таблица истинности

№ п/п	*A*	*B*	НЕ *A*	*A* И *B*	*A* ИЛИ *B*
1	И	И	Л	И	И
2	И	Л	Л	Л	И
3	Л	И	И	Л	И
4	Л	Л	И	Л	Л