СРЕДНЕЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАНИЕ

И.Г.СЕМАКИН, А.П.ШЕСТАКОВ

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

УЧЕБНИК

Рекомендовано
Федеральным государственным автономным учреждением «Федеральный институт развития образования» (ФГАУ «ФИРО»)
в качестве учебника для использования в учебном процессе образовательных учреждений, реализующих программы среднего профессионального образования по специальностям «Компьютерные системы и комплексы», «Информационные системы (по отраслям)», учебная дисциплина «Основы алгоритмизации и программирования» укрупненной группы специальностей «Информатика и вычислительная техника»

Регистрационный номер рецензии 308
от 25 июня 2012 г. ФГАУ «ФИРО»

ACADEMIA

Москва
Издательский центр «Академия»
2013

УДК 681.3.06(075.32)
ББК 22.18я723
С30

Рецензенты:
канд. физ.-мат. наук, преподаватель ГАОУ СПО «Колледж предпринимательства № 11» *Е. И. Ночка*;
старший научный сотрудник ГНУ ГОСНИТИ Россельхозакадемии *А. А. Соломашкин*

Семакин И. Г.

С30 Основы алгоритмизации и программирования : учебник для студ. учреждений сред. проф. образования / И. Г. Семакин, А. П. Шестаков. — М. : Издательский центр «Академия», 2013. — 304 с.

ISBN 978-5-7695-9537-0

Рассмотрены основы принципы алгоритмизации и программирования на базе языка Паскаль (версия Турбо Паскаль 7.0). Даны основные понятия объектно-ориентированного программирования и его реализация на языке Турбо Паскаль. Описана интегрированная среда программирования Delphi и визуальная технология создания графического интерфейса программ. Показана разработка программных модулей в этой среде.

Учебник может быть использован при изучении общепрофессиональной дисциплины ОП «Основы алгоритмизации и программирования» в соответствии с требованиями ФГОС СПО для специальностей 230113 «Компьютерные системы и комплексы» и 230401 «Информационные системы (по отраслям)» укрупненной группы специальностей 230000 «Информатика и вычислительная техника».

Для студентов учреждений среднего профессионального образования.

УДК 681.3.06(075.32)
ББК 22.18я723

ISBN 978-5-7695-9537-0

Уважаемый читатель!

Данное издание является частью учебно-методического комплекта по специальностям 230113 «Компьютерные системы и комплексы» и 230401 «Информационные системы (по отраслям)» укрупненной группы специальностей 230000 «Информатика и вычислительная техника».

Учебник предназначен для изучения общепрофессиональной дисциплины «Основы алгоритмизации и программирования».

Учебно-методические комплекты нового поколения включают традиционные и инновационные учебные материалы, позволяющие обеспечить изучение общеобразовательных и общепрофессиональных дисциплин и профессиональных модулей. Каждый комплект содержит в себе учебники и учебные пособия, средства обучения и контроля, необходимые для освоения общих и профессиональных компетенций, в том числе и с учетом требований работодателя.

Учебные издания дополняются электронными образовательными ресурсами. Электронные ресурсы содержат теоретические и практические модули с интерактивными упражнениями и тренажерами, мультимедийные объекты, ссылки на дополнительные материалы и ресурсы в Интернете. В них включен терминологический словарь и электронный журнал, в котором фиксируются основные параметры учебного процесса: время работы, результат выполнения контрольных и практических заданий. Электронные ресурсы легко встраиваются в учебный процесс и могут быть адаптированы к различным учебным программам.

Учебно-методический комплект разработан на основании Федерального государственного образовательного стандарта среднего профессионального образования с учетом его профиля.

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: системных и прикладных. 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 при первоначальном изучении программирования, — это освоение основ его структурной мето-

дики. Структурная методика до настоящего времени остается основой программистской культуры. Не освоив ее, человек, взявшийся изучать программирование, не имеет никаких шансов стать профессионалом.

В 1969 г. швейцарским профессором Никлаусом Виртом был создан язык программирования Паскаль, предназначавшийся для обучения студентов структурной методике программирования. Язык получил свое название в честь Блеза Паскаля — изобретателя первого вычислительного механического устройства. Позднее фирма Borland International, Inc (США) разработала систему программирования Турбо Паскаль для персональных компьютеров, которая вышла за рамки учебного применения и стала использоваться для научных и производственных целей. В Турбо Паскаль были внесены значительные дополнения к базовому стандарту Паскаля, описанного Н. Виртом. Со временем язык развивался. Начиная с версии 5.5, в Турбо Паскаль вводятся средства поддержки объектно-ориентированного программирования. В дальнейшем это привело к созданию Object Pascal — языка с возможностями объектно-ориентированного программирования. В начале 1990-х гг. объединение элементов объектно-ориентированного программирования в Object Pascal с визуальной технологией программирования привело к созданию системы программирования Delphi.

В *главе 1* настоящего учебника рассматриваются базовые понятия, относящиеся к любому процедурному языку программирования высокого уровня. Основное внимание уделяется структурной методике построения алгоритмов: использованию базовых алгоритмических структур, выделению в решаемой задаче подзадач и составлению вспомогательных алгоритмов. На этой основе можно переходить к изучению любого процедурного языка, поддерживающего структурную методику.

В *главе 2* дается описание языка Паскаль в варианте Турбо Паскаль (версия 7.0). В языках Object Pascal и Delphi сохраняется преемственность к Турбо Паскалю.

Содержание *главы 3* ориентировано на глубокое освоение учащимися базовых понятий языков программирования высокого уровня на примере Паскаля. Такая подготовка облегчает изучение других языков программирования в будущем.

В *главе 4* излагаются основы объектно-ориентированного программирования на примере их реализации в Object Pascal. 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 излагаются некоторые известные методики полного построения алгоритмов, рассматриваются проблемы тестирования программ и оценки сложности алгоритмов.

Языки программирования Турбо Паскаль 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 ознакомиться в процессе выполнения практических работ на ЭВМ, используя специальную литературу.

Глава 1

ОСНОВНЫЕ ПРИНЦИПЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Из данной главы вы узнаете:

- последовательность этапов решения задачи на компьютере средствами программирования;
- что такое алгоритм решения задачи на компьютере;
- с какими данными работает компьютер;
- с помощью какого набора команд (системы команд) можно построить любой алгоритм решения задачи на компьютере;
- правила описания алгоритмов на блок-схемах и на учебном алгоритмическом языке;
- три основные алгоритмические структуры — следование, ветвление, цикл;
- что такое трассировка алгоритма;
- логические основы программирования; сущность понятий «логическая величина», «логическое выражение», «логические операции»;
- правила выполнения логических операций и вычисления логических выражений;
- что такое вспомогательный алгоритм и как он описывается на алгоритмическом языке в форме процедуры;
- как происходит обращение к процедуре в основном алгоритме;
- в чем состоят принципы структурного программирования;
- историю развития языков и технологий программирования;
- классификацию языков программирования;

- что такое трансляция и какие существуют способы трансляции;
- состав и структуру процедурных языков программирования высокого уровня;
- способы описания синтаксиса языков программирования.

Вы научитесь:

- описывать алгоритмы различной структуры (линейные, ветвящиеся, циклические) в виде блок-схем и на учебном алгоритмическом языке;
- записывать логические выражения в качестве условий в ветвлениях и циклах;
- выполнять трассировку алгоритмов;
- описывать процедуры на алгоритмическом языке и обращения к ним из основного алгоритма.

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, сим-

Рис. 1.1. Уровни данных относительно программы

волами, 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 $D = b^2 - 4ac$.

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 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*. Типы величин характеризуются множеством допустимых значений, множеством допустимых операций, формой внутреннего представления (табл. 1.1).

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*.

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

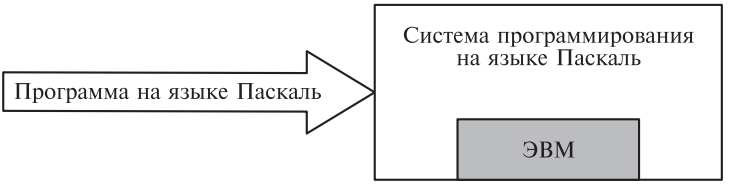


Рис. 1.2. Компьютер как исполнитель программ на языке Паскаль

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 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
```

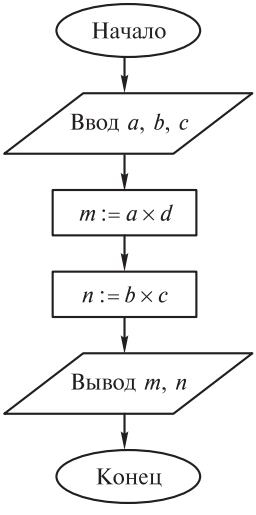


Рис. 1.3. Блок-схема алгоритма деления дробей

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*.

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*. Для каждой команды присваивания в табл. 1.2 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.

Таблица 1.2. Изменение значений переменных при выполнении команды присваивания

Команда	a	b
$a := 1$	1	—
$b := 2 \times a$	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

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 такой:

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.

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 А Я) 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 $a = 0$, $b = 0$, $c \neq 0$, уравнение решений не имеет;

если $a = 0$, $b \neq 0$, это линейное уравнение, имеющее одно решение $x = -c/b$;

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

если $a \neq 0$ 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
```

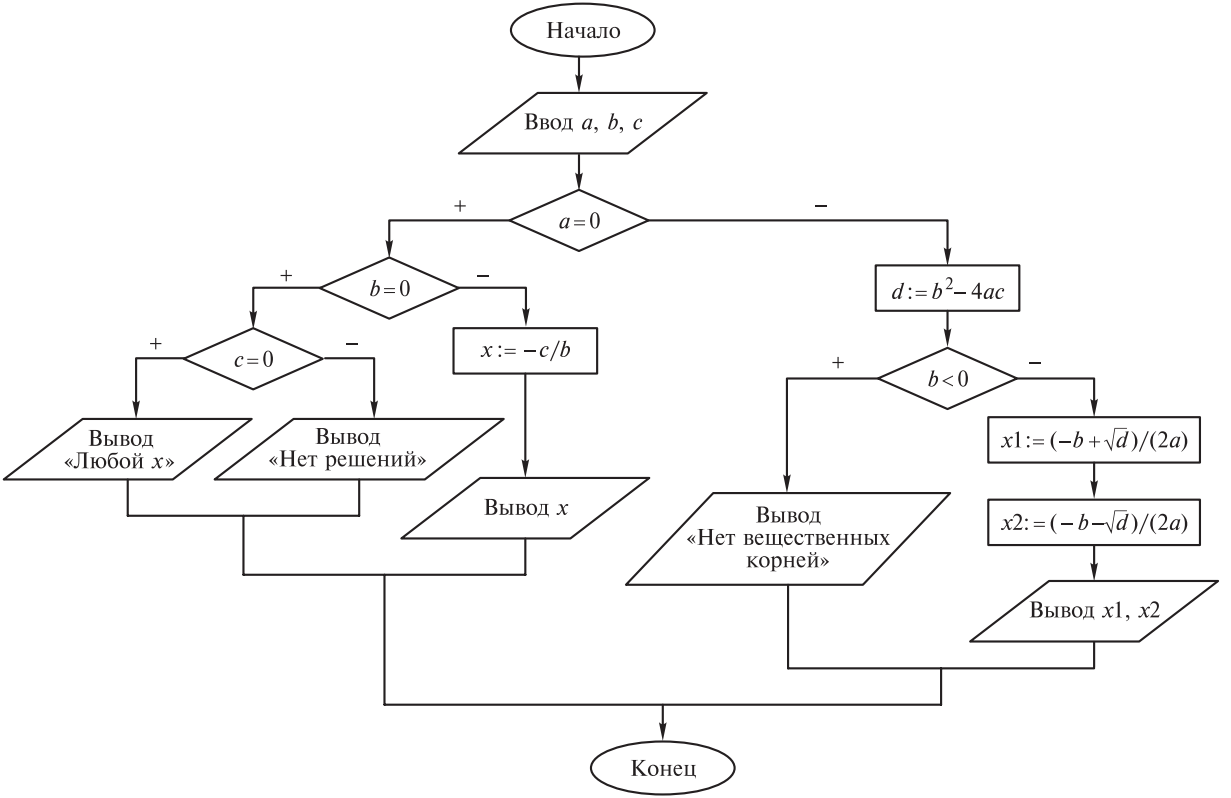


Рис. 1.4. Блок-схема алгоритма решения квадратного уравнения

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:

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

Блок-схема алгоритма вычисления $n!$ с тремя переменными целого типа: n — аргумент, i — промежуточная переменная, F — результат приведена на рис. 1.6. 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:

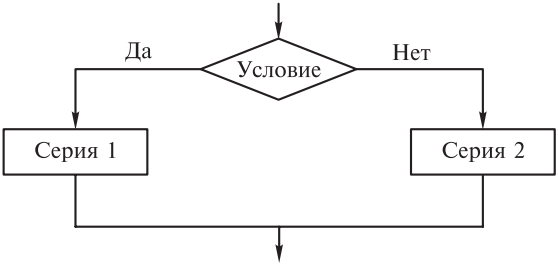


Рис. 1.5. Блок-схема команды ветвления

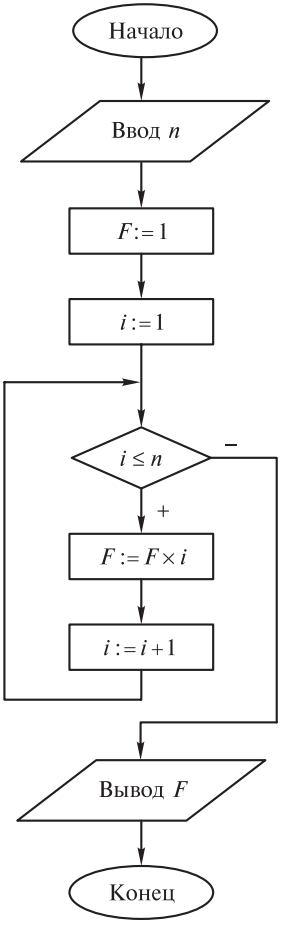


Рис. 1.6. Блок-схема алгоритма вычисления *n*!

Таблица 1.4. Трассировка алгоритма вычисления *n*!

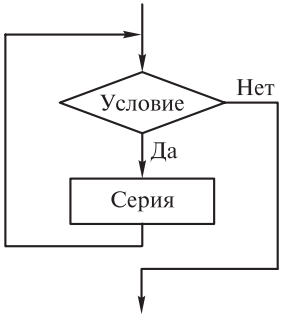
Шаг	*n*	*F*	*i*	Условие
1	3			
2		1		
3			1	
4				1 ≤ 3, да
5		1		
6			2	
7				2 ≤ 3, да
8		2		
9			3	
10				3 ≤ 3, да
11		6		
12			4	
13				4 ≤ 3, нет
14		Вывод		

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

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



Выполнение серии команд (тела цикла) повторяется пока условие цикла истинно. Когда условие становится ложным, выполнение 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, поэтому 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
```

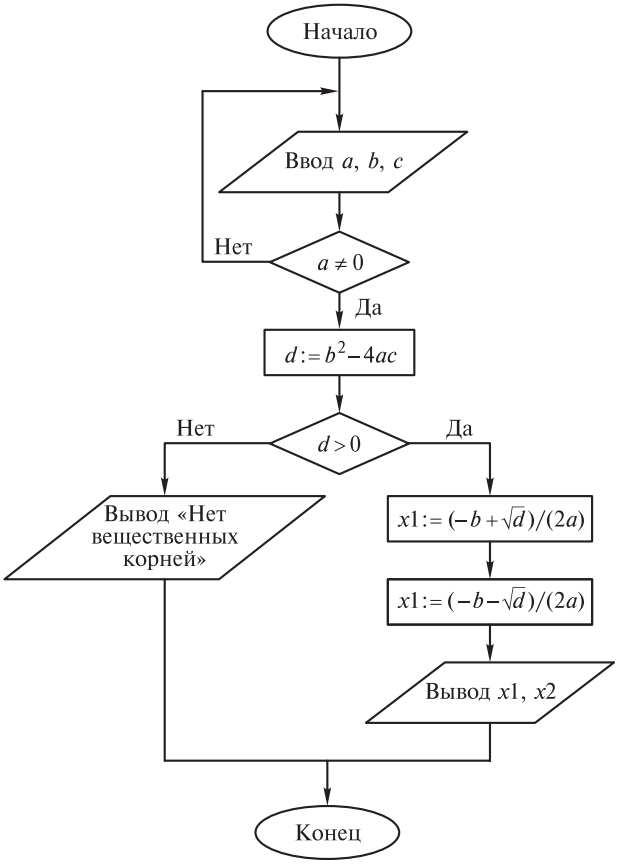


Рис. 1.8. Блок-схема алгоритма решения квадратного уравнения с контролем ввода данных

Блок-схема структурной команды цикла с постусловием или 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*).