Основы построения программ

В этой части...

Глава 1. Пока не включен компьютер
Глава 2. Элементы языка Турбо Паскаль
Глава 3. Описание одно — процессы разные
Глава 4. Цикл-ленд
Глава 5. Модули и абстрактные типы данных
Глава 6. Элементы технологии программирования
Глава 7. Еще раз о подпрограммах
Глава 8. Рекурсивные определения

В этой части представлены “основы основ”. В главе 1 определены понятия алгоритма и программы, описаны общее устройство компьютера и основы представления данных в нем. Она также вкратце освещает историю развития языков программирования.
В главе 2 представлены начало работы с системой Турбо Паскаль и основы языка программирования — лексика, структура программы, базовые типы данных, константы, выражения, переменные и операторы присваивания, подпрограммы ввода-вывода и определение собственных типов. Глава 3 знакомит с управлением и структурированием программы — операторами ветвления и подпрограммами. Представлено понятие области видимости имени. Глава 4 посвящена программированию циклических вычислений.
В главах 5 и 6 описываются элементы технологии программирования. В главе 5 представлены модули и их использование, а также понятия инкапсуляции и абстрактного типа данных. В главе 6 — спецификации задач и программ, нисходящее проектирование программ и структурное программирование, элементы стиля, основы отладки и тестирования программ, а также понятия сложности алгоритма и сложности задачи.
В главе 7 представлено углубленное описание подпрограмм. Здесь рассматривается использование локальных статических переменных и подпрограмм в качестве параметров. Глава 8 посвящена рекурсии, в частности, рекурсивным подпрограммам. Представлены также рекурсивные средства для описания структуры конструкций в языках программирования.

Глава 1

Пока не включен компьютер

В этой главе...

- Понятия алгоритма и процесса
- Два простейших классических алгоритма
- Общая структура персонального компьютера
- Отличия языков программирования высокого уровня от машинных
- Краткий обзор этапов создания программы
- Понятия трансляции, интерпретации, системы программирования
- Позиционные системы счисления и представление чисел в компьютере

Целые числа создал Господь Бог.
Остальное — дело рук человеческих.
Л. Кронекер

1.1. Примеры алгоритмов

Рассмотрим два алгоритма, известных всем, кто помнит школьную математику.

Пример 1.1. Действительные корни квадратного уравнения $ax^2 + bx + c = 0$, заданного коэффициентами a, b, c вычисляются следующим образом (при условии, что $a \neq 0$).

Прочитать коэффициенты a, b, c.

Вычислить дискриминант $d = b^2 - 4ac$.

Если $d > 0$, вычислить $x_1 = \dfrac{-b - \sqrt{d}}{2a}$, $x_2 = \dfrac{-b + \sqrt{d}}{2a}$ и написать эти числа, иначе, если $d = 0$, вычислить $x = \dfrac{-b}{2a}$ и написать это число, иначе написать "действительных корней нет".

Итак, у нас есть *задача*, т.е. требование "вычислить действительные корни квадратного уравнения, заданного коэффициентами", и описание решения задачи, или *алгоритм*. По алгоритму выполняется соответствующая последовательность действий, или *процесс* решения задачи.

В данном случае процессов может быть три.

1. Прочитать коэффициенты, вычислить d, проверить, что $d > 0$ (и это так), вычислить x_1, x_2 и написать эти числа.
2. Прочитать коэффициенты, вычислить d, проверить, что $d > 0$ (и это не так), проверить, что $d = 0$ (и это так), вычислить x и написать это число.
3. Прочитать коэффициенты, вычислить d, проверить, что $d > 0$ (и это не так), проверить, что $d = 0$ (и это не так), и написать, что действительных корней нет.

Алгоритм вычисления корней можно представить в виде *блок-схемы*. Блок-схемы строятся из элементов нескольких типов; основные из них — прямоугольники, ромбы и стрелки. В ромбах указываются условия, которые могут быть истинными или ложными, в зависимости от чего нужно выбирать дальнейший путь выполнения процесса. В алгоритме вычисления корней такими были условия $d > 0$ и $d = 0$. В прямоугольниках указываются действия, не связанные с выбором пути, например, что-то вычислить или написать. Стрелки соединяют прямоугольники и ромбы, указывая возможные последовательности действий при выполнении алгоритма. Из прямоугольника выходит одна стрелка, а из ромба — две; одна из них соответствует тому, что условие истинно, другая — тому, что ложно. Например, на рис. 1.1 представлена блок-схема алгоритма вычисления корней.

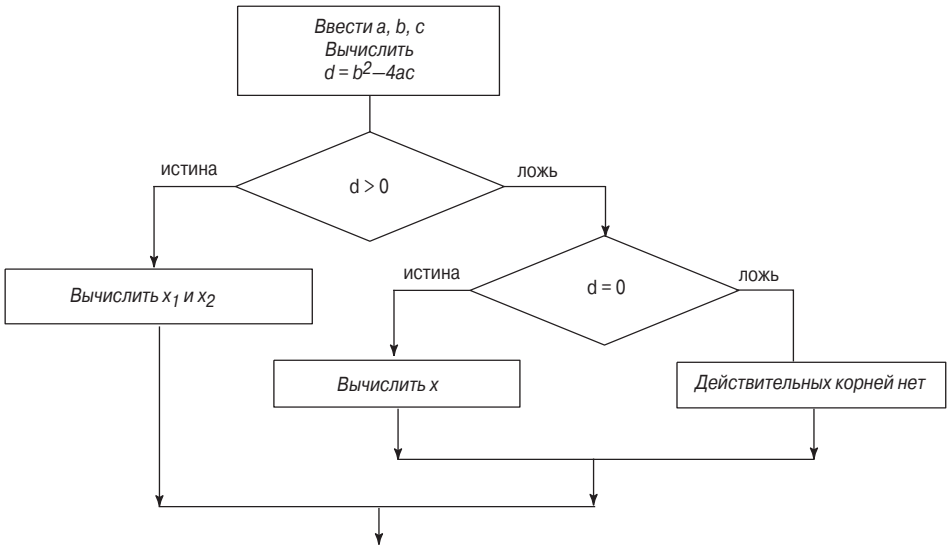

Рис. 1.1. Блок-схема вычисления действительных корней квадратного уравнения

□

Пример 1.2. Рассмотрим *алгоритм Евклида* для вычисления наибольшего общего делителя двух натуральных чисел. Обозначим наибольший общий делитель чисел a и b через НОД(a, b), а остаток от деления a на b — через $a \bmod b$. Алгоритм Евклида основан на том, что

НОД(a, b) = НОД(b, $a \bmod b$), если $b \neq 0$,
НОД(a, b) = a, если $b = 0$.

Например, НОД(12, 5) = НОД(5, 2) = НОД(2, 1) = НОД(1, 0) = 1. Запишем алгоритм в следующем виде.

```
Прочитать натуральные числа a и b.
Пока b > 0, выполнять следующие действия:
  начало действий
    вычислить c = a mod b;
    значение a заменить значением b;
    значение b заменить значением c;
  конец действий.
Написать значение a.
```

По этому алгоритму мы прочитаем числа a и b, например, 12 и 5. Затем, проверив условие $b > 0$, увидим, что оно истинно, поэтому вычислим $c = 12 \bmod 5 = 2$, вместо зна-

чения a, т.е. 12, запишем значение b, т.е. 5, а вместо значения b — значение c, т.е. 2. Итак, значениями a и b стали 5 и 2. Поскольку в алгоритме написано "пока", мы вернемся к проверке условия $b > 0$ и повторим описанные действия, но уже с числами 5 и 2. Поскольку $a \bmod b < b$, значение b каждый раз уменьшается и в конце концов станет равным 0. Тогда при очередной проверке условие $b > 0$ будет ложным, и мы напишем значение a (в данном случае 1).

Алгоритм Евклида представлен в виде блок-схемы на рис. 1.2.

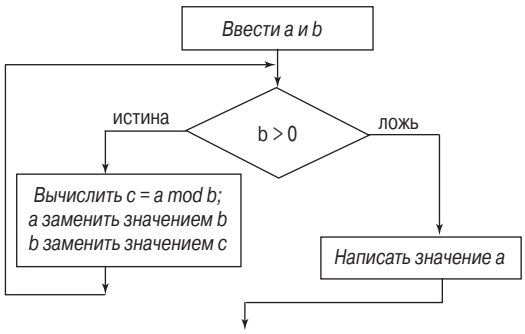

Рис. 1.2. Блок-схема алгоритма Евклида

□

Описание действий, которые нужно выполнить для решения задачи, называется *алгоритмом решения задачи*. Алгоритм, предназначенный для выполнения на компьютере, обычно называется *программой*, а последовательность действий — *процессом* решения задачи.

В примерах 1.1 и 1.2 представлены две формы записи алгоритмов, понятные человеку. Однако для компьютера они записываются иначе. Познакомимся с устройством компьютера и видом программ для него.

1.2. Заглянем в компьютер

1.2.1. Структура компьютера

Общий вид *компьютера*, или *вычислительной машины*, представлен на рис. 1.3. На его *материнской плате* располагаются *центральный процессор* и *оперативная память* (ОП); к плате могут подключаться другие платы, предназначенные для управления *внешними устройствами*. К ним относятся экран (терминал), клавиатура, манипулятор "мышь", дисководы и другие, например, сканер, плоттер для рисования или модем.

Компьютер "умеет" делать только одно — выполнять программы. Программа — это последовательность *команд*, задающих обработку *значений*, или *данных*. Выполняемая программа и ее данные находятся в ОП. Центральный процессор читает команды из памяти и выполняет их. Команды задают чтение значений (числовых и других) из памяти, создание новых значений и запись их в память. Данные читаются и записываются с помощью *системной шины*, которая входит в состав материнской платы и обеспечивает взаимодействие всех устройств компьютера.

Основные части процессора — операционное и управляющее устройства, а также собственная память для хранения и обработки данных. Операционное устройство выполняет команды и порождает новые значения. Память образуется специальными запоминающими элементами — *регистрами*. Управляющее устройство обеспечивает обмен значениями между операционным устройством и регистрами. Этот обмен происходит гораздо быстрее, чем обмен с ОП.

У процессора есть еще *кэш-память*. Обмен с ней происходит медленнее, чем с регистровой, но быстрее, чем с оперативной. Часть выполняемой программы и данных записывается в кэш-память. Это избавляет от необходимости обращаться к оперативной памяти за каждой командой или значением и ускоряет выполнение программы.

Кроме оперативной и кэш-памяти, в компьютере есть *внешняя память* — на *внешних носителях* данных, например, магнитных дисках. Носители размещаются на специальных *устройствах обмена* данными с "внешним миром" (внешних устройствах, или *устройствах ввода-вывода* — УВВ). К ним относятся, например, дисководы, экран, клавиатура, мышь. У всех УВВ есть свои процессоры, которые устроены проще, чем центральный, и выполняют другие наборы команд. Процессоры УВВ могут переносить данные с внешних носителей в оперативную память (чтение из "внешнего мира") или наоборот (запись данных во "внешний мир").

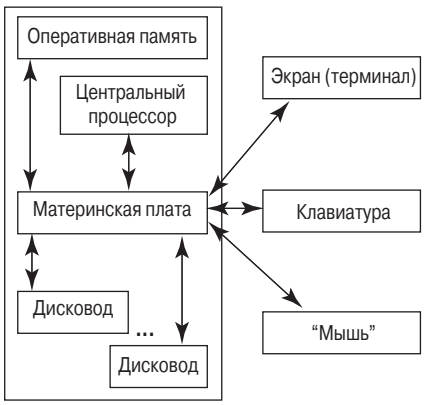

Рис. 1.3. Общая схема компьютера

Каждому устройству обмена выделен особый участок оперативной памяти — *порт*. Из него устройство берет данные для внешнего носителя, записывая их на диск или экран компьютера; в порт также записываются данные, например, от клавиатуры и дисковода.

Компьютеры, как правило, имеют несколько дисководов для работы с дисками различных типов (жесткими, гибкими, компактными и другими).

Данные на внешнем носителе существуют в виде *файлов*. Каждый файл организован по определенным правилам, которые называются *форматом файла*.

Существуют сотни различных форматов файлов — одни для текстов, другие для картинок, третьи для машинных программ и т.д.

1.2.2. Данные и программы

Человек привык записывать числа в десятичной системе счисления (по основанию 10), используя для записи 10 знаков (*цифр*) от 0 до 9. Каждая цифра задает число, которое зависит от ее *позиции* в записи. Например, 123 означает: 1 *сотня*, 2 *десятка* и 3 *единицы*, а 312 — 3 *сотни*, 1 *десяток* и 2 *единицы*. Если справа от числа единиц записана запятая или точка, а за ней снова цифры, то они обозначают число дробных частей единицы — десятых, сотых и т.д..

В компьютере числа представляются в *двоичной записи* (по основанию 2) с цифрами 0 и 1. Цифры отвечают двум различным устойчивым состояниям элемента памяти, который называется *бит* (от англ. *bit* — *bi*nary digi*t*, т.е. двоичная цифра). Состояния двух последовательных битов отвечают четырем сочетаниям цифр 00, 01, 10, 11, задающим целые числа 0, 1, 2, 3. Аналогично 3 бит задают восемь чисел от 0 до 7, 4 бит — 16 чисел от 0 до 15 и т.д.

Восемь последовательных битов образуют *байт*. Он может иметь $2^8 = 256$ различных состояний и представлять, например, целые числа от 0 до 255. Эти же состояния байта могут рассматриваться как числа от –128 до 127 (их ведь тоже 256), символы или что-нибудь еще.

Оперативная память представляет собой последовательность байтов, в которой каждый байт имеет свой номер — *адрес*.

Числовое значение в памяти обычно занимает несколько соседних байтов и указывается адресом первого из них. Для целых чисел обычно используют 1, 2 или 4 байт,

для нецелых (*вещественных*, или *действительных*) — 4, 6, 8 или 10 байт. Представление чисел и других значений подробнее рассмотрено в разделе 1.4.

Байт является единицей измерения объема памяти и передачи данных. Здесь используются такие единицы, как Кбайт ("кейбайт", $2^{10} = 1024$ байт, или не совсем точно "килобайт"), Мбайт ("мегабайт", $2^{20} = 1\,048\,576$ байт) или Гбайт ("гигабайт", $2^{30} = 1\,073\,741\,824$ байт). Регистры процессора в зависимости от своего назначения могут состоять из 1-10 байт. В процессоре их обычно несколько десятков. Объем кэш-памяти — десятки и сотни Кбайт, а оперативной — десятки и сотни Мбайт.[1]

Машинные команды, как и числа, также записываются в ОП. Они представляют собой указания типа: "прочитать число по такому-то адресу памяти в такой-то регистр", "сложить два числа из таких-то регистров и запомнить сумму в таком-то регистре", "записать число из такого-то регистра по такому-то адресу памяти" или "выполнить команду, записанную по такому-то адресу памяти". Действия процессора ("прочитать", "сложить" и т.п.) задаются в машинных командах *кодами операций*.

Система команд, выполняемых процессором, называется *машинным языком*.

Пример 1.3. Посмотрим, на что похожи машинные команды. Предположим, что `8200`, `8204` и `8248` — адреса оперативной памяти, по которым записаны числа, `001` и `002` — номера регистров, операция "прочитать" задается кодом `08`, "записать" — `09`, "сложить" — `01`, "выполнить команду" — `22`. Чтобы сложить два числа, записанные по адресам `8200` и `8204`, записать сумму по адресу `8248` и после этого выполнить команду, записанную по адресу `15376`, процессор должен выполнить такую последовательность команд.

```
08 8200 001
08 8204 002
01 001 002 001
09 001 8248
22 15376
```

Человеку создавать и читать такую машинную программу не очень-то легко. Вдобавок, что-то подобное в действительности записывается в двоичной системе счисления, т.е. в виде более длинных последовательностей из `0` и `1`.

□

Из приведенного примера можно сделать следующие выводы.

- Программы задают обработку данных.
- Программы записываются по определенной системе, т.е. с помощью некоторого языка, "понятного" компьютеру.
- Программы записываются в память и тоже являются данными.
- Тем не менее принято различать команды программы и ее данные, считая, что программа состоит из команд и данных.

В памяти компьютера одновременно находятся, как правило, несколько программ. Среди них есть целая система программ и данных под общим названием *операционная система* (ОС на рис. 1.4). Названия операционных систем хорошо известны читателю — DOS, Windows, UNIX и многие другие. Основные задачи ОС — определять, команды какой из программ должны выполняться в тот или иной момент времени, и обеспечивать обмен данными с УВВ. Программа (или система программ), с помощью которой время работы процессора распределяется между программами, называется

[1] *По состоянию на 2003 год оперативная память некоторых персональных компьютеров состоит из нескольких Гбайт, и очень скоро размеры такого порядка станут обычными.*

диспетчером (*супервизором*, *планировщиком*, *ядром* ОС), а программы, с помощью которых ОС управляет работой УВВ, — *драйверами* (драйвер клавиатуры, драйвер экрана и т.д.).

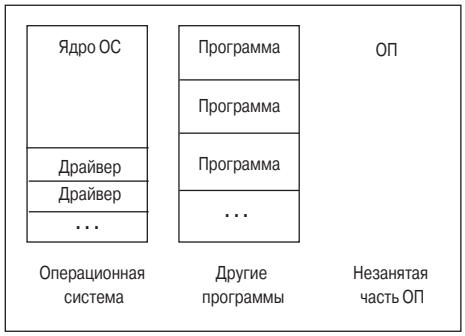

Рис. 1.4. Программы в памяти компьютера

Для выполнения программа переписывается в ОП с внешних носителей, как правило, с диска.[2] Это происходит при выполнении специальной программы из состава ОС — *загрузчика*.

Переписывание программы с внешнего носителя в оперативную память называется *загрузкой*.

После загрузки начинается *процесс выполнения* загруженной программы (рис. 1.5).

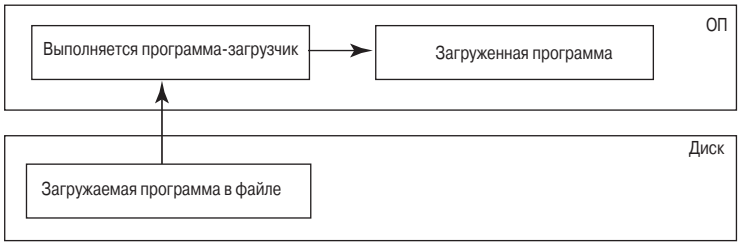

Рис. 1.5. Загрузка программы

Но откуда программа появляется на внешнем носителе? На этот вопрос попытаемся ответить в следующем разделе.

1.3. Языки высокого уровня и системы программирования

1.3.1. На пути к удобным языкам программирования

В первых вычислительных машинах программы записывались в виде последовательностей из `0` и `1` и вводились в машину с помощью специального щита, связанного с процессором, и штырьков, кодировавших эти `0` и `1`. По мере разви-

[2] *При включении компьютера в ОП записываются программы, которые обеспечивают взаимодействие процессора с оперативной памятью и другими устройствами. Без этих программ компьютер работать вообще не может, поэтому они записаны в постоянном запоминающем устройстве (ПЗУ). ПЗУ сохраняет свое содержимое и после выключения компьютера (в отличие от ОП).*

тия вычислительной техники изменялись и способы записи и ввода программ в оперативную память.

Скоро стало ясно, что человека можно освободить от записи программы в виде двоичных последовательностей. Вместо двоичных кодов можно использовать другую, символическую систему обозначений команд и данных, а перевод в машинные коды осуществить с помощью специальной программы, выполняемой компьютером. Так появились *языки ассемблерного типа*, или *ассемблеры* (от англ. *to assemble* — собирать).

Пример 1.4. Предположим, что адреса оперативной памяти `8200`, `8204`, `8248` и `15376` из примера 1.3 обозначены соответственно как `Пм1`, `Пм2`, `Пм3`, `Пм4`, регистры `001` и `002` — как `Рг1` и `Рг2`, а команды "прочитать", "записать", "сложить" и "выполнить команду" — как `Чит`, `Зап`, `Слож`, `ВыпК`. Тогда последовательность команд из примера 1.3 приобретет следующий вид.

```
Чит Пм1 Рг1
Чит Пм2 Рг2
Слож Рг1 Рг2 Рг1
Зап Рг1 Пм3
ВыпК Пм4
```

Однако и на ассемблерном языке даже простейшая программа — это длинная последовательность команд, по структуре совпадающих с машинными. Написать такую программу тоже нелегко, к тому же нужно знать множество подробностей устройства компьютера (например, для чего предназначены те или иные регистры, какие адреса памяти можно использовать, а какие нельзя, и т.п.). Поэтому программирование на ассемблерном языке — удел немногих.

□

В 1950-х годах начался поиск языков, призванных ускорить и облегчить процесс программирования. Были нужны языки, более близкие по форме как к математическому, так и к человеческому языку, свободные от подробностей машинного выполнения, но при этом позволяющие *переводить программы в соответствующие машинные с помощью самой машины*.

И такие языки вскоре были разработаны. Их стали называть *языками программирования высокого уровня*, поскольку действия компьютера в них представлены с высоким уровнем абстракции. В машинных же языках действия компьютера описаны подробно, т.е. с низким уровнем абстракции.

Первый язык высокого уровня появился в конце 1950-х годов и назывался *Фортран* (Fortran — это сокращение от англ. *for*mula *tran*slation, т.е. перевод формул). Например, действия, представленные в примерах 1.3 и 1.4, в этом языке можно выразить приблизительно так.

```
Z=X+Y
GO TO 315
```

Если предположить, что `X`, `Y`, `Z` обозначают участки памяти, в которые записываются числа, `315` — место в программе, которому соответствует команда по адресу `15376`, и учесть, что "go to" означает по-английски "перейти", то в этих двух строчках все понятно.

Вместе с языками разрабатывались программы перевода "высокоуровневых" программ в машинные — *трансляторы*, или *компиляторы*.

Транслятор читает программу, записанную на языке высокого уровня, распознает действия компьютера, которые она задает, и выражает их в виде соответствующей машинной программы.

Создание трансляторов поначалу было очень непростым делом. Однако с годами опыт накапливался, были разработаны соответствующие математические основы и инструментарий. Создание и реализация языков превратились в технологию, а число существующих языков программирования и трансляторов уже измеряется тысячами и постоянно растет.

1.3.2. Как создается программа

Сначала заказчик рассказывает программисту, что он хочет получить (часто не понимая, что именно). Программист (возможно, не один) пытается уяснить, что хочет заказчик. Для этого ему приходится *анализировать* и *уточнять* постановку задачи. В результате создается *спецификация задачи*, т.е. точное и однозначное ее описание. Иногда она соответствует тому, что хотел заказчик.

Чем выше квалификация программистов, тем больше шансов, что они правильно поймут заказчика и напишут правильную спецификацию задачи.

Из спецификации задачи становится ясно, какие действия нужно выполнить для ее решения. Их нужно описать, чтобы компьютер мог решать задачу.

Начинается *проектирование программы*. На основе спецификации задачи пишется *спецификация программы*, т.е. действия описываются в самом общем виде, далеком от того, что может выполнять компьютер. Этот алгоритм *уточняется несколько раз* до вида, по которому легко написать программу. Обычно в задаче можно выделить несколько *подзадач* и уточнить их решение по отдельности. Соответственно и алгоритм строится из связанных между собой частей, описывающих решение подзадач.

Написание программы (или ее частей) принято называть *кодированием*, или *разработкой*. Чаще всего программа пишется на каком-либо языке высокого уровня (иногда отдельные ее части на разных языках) и транслируется на машинный язык.

В процессе кодирования программисты могут наделать ошибок, которые обнаруживаются либо при переводе программы на машинный язык, либо при пробном выполнении программы или ее частей. Начинается процесс обнаружения и исправления ошибок — *отладка* программы.

Отладка включает в себя многократное выполнение программы с различными вариантами входных данных. Данные подбираются специально для того, чтобы выявить как можно больше ошибок, если они есть. Такая целенаправленная проверка работоспособности программы называется *тестированием*. Оно не гарантирует, что в программе нет ошибок, но позволяет выявить некоторые из них. Чем тщательнее проводится тестирование, тем больше ошибок обнаруживается и меньше остается.

Еще нужны *сопровождающие документы*, в которых описана *структура программы* и приводятся *инструкции по ее применению*. Назначение инструкций очевидно, а спецификация нужна для исправления программы уже во время ее использования и дальнейшего развития. В идеале, документы готовятся еще *в процессе проектирования и разработки*.

Однако может оказаться, что при проектировании программы был выбран не самый удачный алгоритм, из-за которого программа выполняется слишком медленно или расходует слишком много памяти. Начинаются изменения программы, требующие нового кодирования и отладки. Потом заказчика осеняют новые идеи, и приходится вообще заново ставить задачу и проектировать программу. И кажется, что конца этому нет.

Тем не менее в мире создаются программы, которые работают правильно и эффективно. И продуктивность работы программистов в целом возрастает.

Причин несколько. Одна из них — для каждого этапа создания программы, от анализа задачи до отладки, существуют и развиваются *технологии*, т.е. определенные системы методов, применение которых позволяет “не делать лишних движений”. Кроме того, постоянно совершенствуются *инструменты*, ускоряющие и облегчающие процесс разработки, — *системы программирования*. Кроме всего прочего, они реализуют очень важный принцип *повторного использования кода*. Дело в том, что во многих задачах возникают одни и те же подзадачи, для которых можно использовать одни

и те же программы (они называются *подпрограммами*), не создавая их заново. Такие подпрограммы собираются в специальные *библиотеки подпрограмм*. Чем богаче библиотека, тем меньше работы остается при разработке программы.

- Чем лучше программист владеет технологиями, инструментарием и библиотеками, тем его работа эффективнее.

1.3.3. Преобразования программы и система программирования

Рассмотрим, как программа на языке высокого уровня преобразуется в машинную. Программа (*исходный текст*) с помощью специальной программы (*текстовый редактор*) обычно записывается на диск в виде *исходного файла* (рис. 1.6). Возможно, программа состоит из нескольких исходных файлов — в больших программах их могут быть десятки и сотни.

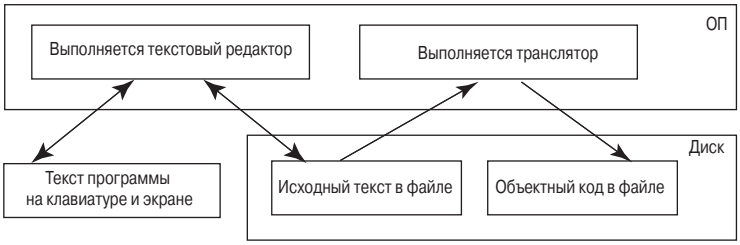

Рис. 1.6. Создание исходного файла

Затем запускается транслятор, который читает с диска исходный файл и строит его машинный эквивалент — *объектный код*. Процесс выполнения программы-транслятора называется *трансляцией* исходного текста.

Как правило, объектный код программы содержит далеко не все необходимые команды (программа может состоять из частей или включать в себя программы из библиотек), поэтому он обрабатывается еще одной программой — *редактором связей*, или *компоновщиком*. Компоновщик "собирает" (компонует) полный код программы и записывает его или в оперативную память (*загружает*), или на диск в виде *выполнимого файла* (рис. 1.7), который можно загрузить в дальнейшем.

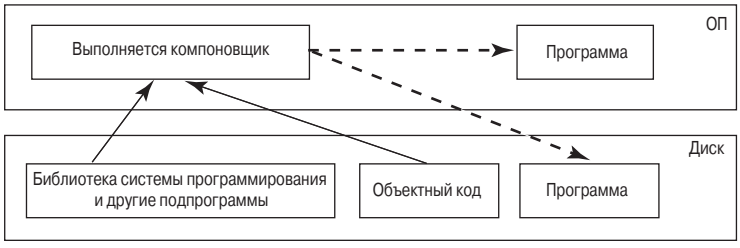

Рис. 1.7. Создание выполнимой машинной программы

Интерпретатор, в отличие от транслятора, не создает машинную программу. Входные данные для него — это программа на языке высокого уровня и данные, которые должны читаться при ее выполнении (рис. 1.8). Интерпретация программы состоит в том, что действия, ею заданные, сразу выполняются. Как правило, интерпре-

тация исходной программы происходит медленнее, чем выполнение соответствующей машинной программы.

Еще один способ обработки исходной программы сочетает трансляцию и интерпретацию. Программа транслируется не в машинные команды, а в некоторое *промежуточное представление*, которое в дальнейшем интерпретируется. Такая реализация облегчает *переносимость программ*, т.е. возможность их выполнения на разных типах компьютеров и на основе разных операционных систем. Так реализован, например, язык Java, быстро ставший популярным.

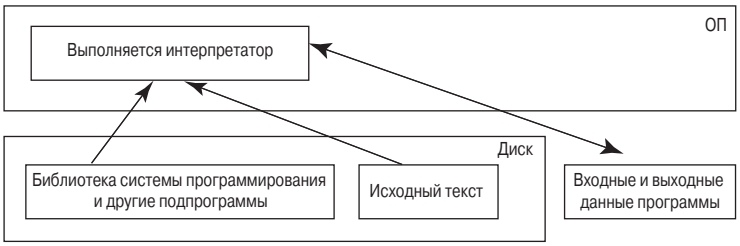

Рис. 1.8. Интерпретация высокоуровневой программы

Интерпретация программы используется также и в *отладчике*. Он позволяет интерпретировать исходную программу небольшими порциями и видеть результаты выполнения после каждой, облегчая поиск ошибок.

Описанные средства (текстовый редактор, транслятор и/или интерпретатор, компоновщик, загрузчик и отладчик) обычно собираются вместе и образуют *систему программирования*, или *интегрированную среду разработки* (Integrated Design Environment — IDE). Кроме них, в ее состав входит библиотека стандартных подпрограмм.

1.3.4. Язык Паскаль, его развитие и реализации

Вернемся к истории. Почти одновременно с языком Фортран был разработан и реализован *Алгол 60* (Algol — *Algo*rythmic *l*anguage, алгоритмический язык). Его конструкции были намного больше похожи на английские фразы, чем конструкции Фортрана, поэтому логика действий выражалась естественнее. В отличие от Фортрана, он имел средства для выражения так называемой *рекурсии*, присущей многим алгоритмам. Благодаря этим свойствам на Алголе стали записывать алгоритмы, предназначенные для изучения человеком. (Конечно же, существуют задачи, решение которых проще и лаконичнее всего выражается именно на Фортране, но их относительно немного.)

В середине 1960-х годов на основе Фортрана был создан язык *Бейсик* (BASIC). Его название можно перевести как “базовый”, но оно является сокращением от *B*eginner's *A*ll-purpose *S*ymbolic *I*nstruction *C*ode — универсальный символьный набор команд для начинающих. Он был проще Фортрана и позволял легко и быстро учиться писать простые программы. Однако для создания более сложных программ нужна определенная дисциплина программирования (*структурное программирование*), к освоению которой Бейсик не располагал.

Язык *Паскаль* был разработан швейцарским ученым Никлаусом Виртом также специально для обучения программированию. Описание языка было опубликовано в 1971 г. Он происходил от Алгола 60 и двух языков на его основе, созданных в 1960-х годах (Алгол W и Алгол 68). Язык Паскаль был проще своих предшественников, позволял осваивать структурное программирование и обладал рядом других полезных свойств. С середины 1970-х годов (в СССР несколько позже) он стал основным среди языков, применяемых в обучении.

Популярность языка Паскаль поддерживается его реализациями. Наиболее известна система программирования Turbo Pascal version 7.0, созданная фирмой Borland International еще в 1993 г. В ней реализован язык Турбо Паскаль 7.0, вобравший в себя многолетний опыт практического программирования.

На основе языка Турбо Паскаль 7.0 был разработан язык Object Pascal, реализованный в системе программирования *Turbo Vision*. Она была популярной до середины 1990-х годов, когда Object Pascal был реализован в системе программирования *Delphi*.

Система Delphi используется для профессиональной разработки реальных программ. Она является одной из наиболее популярных систем, реализующих так называемую *быструю разработку приложений* (программ), или *RAD* (*R*apid *A*pplication *D*esign). Такие системы содержат обширные библиотеки подпрограмм, которые обеспечивают отображение на экране (*визуализацию*) результатов работы программы и решение других задач, значительно повышая эффективность работы программистов.

Автор отдает себе отчет в том, что большинство реальных программ создаются и будут создаваться с помощью систем программирования, в основе которых лежит не язык Паскаль. Наиболее распространенным среди профессионалов языком является C++, продолжает набирать приверженцев Java, свои сферы применения имеют другие языки, например, Фортран, Бейсик или Кобол (в их современных версиях). Однако начинать освоение программирования, по мнению очень многих специалистов, лучше все-таки с помощью языка Паскаль.

Прежде, чем изучать собственно программирование и язык Паскаль, познакомимся подробнее с представлением данных в компьютере.

1.4. Представление чисел в компьютере

1.4.1. Позиционные системы счисления

Система счисления — это система обозначения чисел. В программировании, кроме десятичной и двоичной систем, используются также *восьмеричная* и *шестнадцатеричная*. Восьмеричная имеет восемь цифр 0, 1, 2, 3, 4, 5, 6, 7, а в шестнадцатеричной первые 10 цифр арабские, а следующие шесть — буквы *A*, *B*, *C*, *D*, *E*, *F*. Они обозначают числа, десятичная запись которых 10, 11, 12, 13, 14, 15 соответственно.

Все эти системы являются *позиционными* — число, обозначенное цифрой, зависит от ее места (позиции) в записи числа.

Позиционная система счисления с *основанием P* (*P*-ичная) имеет *P* цифр C_0, C_1, ..., C_{P-1}, обозначающих натуральные числа от 0 до *P*–1. Число *P* в *P*-ичной системе обозначается двухразрядной записью C_1C_0 ("*P*-ичная десятка"), $P+1$ — записью C_1C_1 и т.д., например, 10, 11, ..., 99 в десятичной системе, 10, 11 в двоичной, 10, 11, ..., 1*F*, 20, ..., *FF* в шестнадцатеричной. Число $P{\times}P$ обозначается уже тремя цифрами $C_1C_0C_0$ ("*P*-ичная сотня"), дальше идет $C_1C_0C_1$ и т.д.

Запись вида $(a_k a_{k-1}\ldots a_1 a_0)_P$ в *P*-ичной системе обозначает число, которое является значением полинома $a_k{\times}P^k + a_{k-1}{\times}P^{k-1} + \ldots + a_1{\times}P + a_0$. Правая (младшая) цифра записи задает количество единиц, левая (старшая) — количество чисел P^k.

Например, двоичная запись $(10011)_2$ обозначает число, которое в десятичной записи имеет вид 19 ($19 = 1{\times}2^4 + 0{\times}2^3 + 0{\times}2^2 + 1{\times}2^1 + 1{\times}2^0$). шестнадцатеричная запись $(1BC)_{16}$ обозначает десятичное 444, равное $1{\times}16^2 + 11{\times}16^1 + 12{\times}16^0$.

Очевиден следующий *алгоритм получения десятичного представления натурального числа N* по его *P*-ичному представлению $(a_k a_{k-1}\ldots a_1 a_0)_P$.

Представить в десятичной системе число P и цифры a_k, a_{k-1}, ..., a_1, a_0.
Вычислить значение полинома $a_k{\times}P^k + a_{k-1}{\times}P^{k-1} + \ldots + a_1{\times}P + a_0$,
применяя умножение и сложение в десятичной системе.

P-ичная запись чисел меньше 1 имеет вид $0,a_{-1}a_{-2}\ldots$, где a_{-i} — P-ичные цифры. Она обозначает действительное число меньше 1 — значение выражения $a_{-1}\times P^{-1} + a_{-2}\times P^{-2} + \ldots$ Например, так.

$(0,21)_3$ обозначает десятичное $2\times 3^{-1} + 1\times 3^{-2} = 0,777\ldots = 0,(7)$;

$(0,101)_2$ — десятичное $1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3} = 0,5 + 0,125 = 0,625$;

$(0,BC)_{16}$ — десятичное $11\times 16^{-1} + 12\times 16^{-2} = 0,734375$.

Чтобы получить десятичную запись числа меньше 1 по P-ичной записи $0,a_{-1}a_{-2}\ldots a_{-r}$, нужно также вычислить значение многочлена

$$a_{-1}\times P^{-1} + a_{-2}\times P^{-2} + \ldots + a_{-r}\times P^{-r},$$

в котором P-ичные цифры записаны в десятичной системе.

- Если основание P имеет простые делители, отличные от 2 и 5, то число с конечной P-ичной записью может изображаться бесконечной, но периодической десятичной дробью. Если же простыми делителями P являются только 2 и 5, то и десятичная дробь конечна.

1.4.2. Запись чисел в недесятичных системах

Рассмотрим, как по натуральному числу N получить цифры его P-ичного представления. Пусть $N = (a_k a_{k-1}\ldots a_1 a_0)_P$ с неизвестными цифрами в неизвестном количестве $k+1$. Запишем представление в следующем виде.

$$N = a_k\times P^k + a_{k-1}\times P^{k-1} + \ldots + a_1\times P + a_0 = (\ldots(a_k\times P + a_{k-1})\times P + \ldots + a_1)\times P + a_0.$$

Обозначим частное от деления N на P как N div P, а остаток — N mod P. Тогда $N \bmod P = a_0$, $N \operatorname{div} P = (\ldots(a_k\times P + a_{k-1})\times P + \ldots)\times P + a_1$. Отсюда очевидно, что

$$a_1 = (N \operatorname{div} P) \bmod P,$$

$$a_2 = ((N \operatorname{div} P) \operatorname{div} P) \bmod P, \text{ и т.д.}$$

Итак, P-ичная запись натурального числа N (в его десятичном представлении) получается по следующему алгоритму.

> Вначале частное T равно N, а представление пусто.
>
> Пока $T \geq P$, вычислить остаток R как T mod P и новое частное T как T div P (путем деления в столбик), представить R в P-ичной системе счисления и дописать к представлению слева.
>
> Представить T в P-ичной системе счисления и дописать к представлению слева.

Пример 1.5. По алгоритму получим двоичное представление числа 10, записывая остаток в скобках после частного: 10:2 = 5(**0**), 0 — количество единиц (самая младшая цифра); 5:2 = 2(**1**), 1 — число “двоичных десятков”; 2:2 = 1(**0**), 0 — число “двоичных сотен”, т.е. четверок. Поскольку 1 < 2, больше не делим, а дописываем **1** слева: **1010**.

Переведем десятичное 1022 в шестнадцатеричную систему. 1022:16 = 63(**14**), $(14)_{10} = (\boldsymbol{E})_{16}$, E — младшая цифра; 63 > 16, поэтому делим — 63:16 = 3(**15**), $(15)_{10} = (\boldsymbol{F})_{16}$, F — количество “десятков”. Поскольку 3 < 16, дописываем **3** (как шестнадцатеричную) слева и получаем **3*FE***.

□

Рассмотрим, как по действительному значению $V<1$ получаются цифры его P-ичного представления. Пусть $V=(0,a_{-1}a_{-2}\ldots)_P$, т.е.

$$V=P^{-1}\times(a_{-1}+P^{-1}\times(a_{-2}+\ldots)),$$

где цифры a_{-i} неизвестны. Умножим обе части равенства на P:

$$V\times P=a_{-1}+P^{-1}\times(a_{-2}+\ldots).$$

Отсюда $a_{-1}=\lfloor V\times P\rfloor$, а $P_{-1}\times(a_{-2}+\ \ldots)=\{V\times P\}$, где $\lfloor V\times P\rfloor$ и $\{V\times P\}$ обозначают целую и дробную части $V\times P$. Умножим $\{V\times P\}$ на P, возьмем целую часть, получим a_{-2} и т.д. Например, при $V=0{,}25$, $P=3$ получим

$$a_{-1}=\lfloor 0{,}25\times 3\rfloor=0,\ \{0{,}25\times 3\}=0{,}75,$$

$$a_{-2}=\lfloor 0{,}75\times 3\rfloor=2,\ \{0{,}75\times 3\}=0{,}25,\ \text{и т.д.}$$

Отсюда ясно, что троичной записью 0,25 будет бесконечная периодическая дробь $(0,(02))_3$.

Итак, по действительному числу V, $V<1$, можно получить R первых цифр его P-ичного представления, выполнив следующий алгоритм.

> Вначале представлением является "0.".
>
> Пока получено меньше, чем R дробных цифр, вычислить $V\times P$, d как $\lfloor V\times P\rfloor$ (целое число от 0 до $P-1$) и V как $\{V\times P\}$. Представить значение d в виде P-ичной цифры и дописать ее к представлению справа.

1.4.3. Связь двоичной, восьмеричной и шестнадцатеричной систем

Рассмотрим простой способ получения восьмеричных и шестнадцатеричных записей по двоичным и наоборот. Заметим, что $8=(1000)_2$, т.е. восьмеричным цифрам от 0 до 7 отвечают тройки двоичных (с незначащими нулями):

0 — 000, 1 — 001, 2 — 010, 3 — 011,

4 — 100, 5 — 101, 6 — 110, 7 — 111.

В двоичной записи достаточно, начиная с младшей цифры, заменить тройки двоичных цифр соответствующими восьмеричными (последняя тройка может быть неполной, т.е. без незначащих нулей). Например, 10101001 разобъем на 10 101 001; 10 изменим на 2, 101 — на 5, 001 — на 1 и получим 251_8. Наоборот, из 172_8 получаются тройки 001, 111, 010 и двоичная запись 1 111 010.

Аналогично $16=(10\,000)_2$, т.е. шестнадцатеричным цифрам соответствуют четверки двоичных:

0 — 0000, 1 — 0001, 2 — 0010, 3 — 0011,

4 — 0100, 5 — 0101, 6 — 0110, 7 — 0111,

8 — 1000, 9 — 1001, A — 1010, B — 1011,

C — 1100, D — 1101, E — 1110, F — 1111.

В двоичной записи достаточно, начиная с младшей цифры, заменить четверки двоичных цифр соответствующими шестнадцатеричными (последняя четверка может быть неполной). Например, 110101001 разбивается на 1 1010 1001; 1 превращается в 1, 1010 — в A, 1001 — в 9, и получается $1A9_{16}$. Наоборот, 703_{16} дает четверки 0111, 0000, 0011, т.е. двоичную запись 111 0000 0011.

Аналогичная связь между данными тремя системами проявляется и в записи дробных чисел. Нетрудно убедиться, что группы по три двоичные цифры, начиная со старших (первых после запятой) соответствуют восьмеричным цифрам, а группы по четыре — шестнадцатеричным. Последняя группа может быть неполной — к ней справа дописываются незначащие нули. Например, двоичная запись 0,010 111 110 1 разбивается на группы 010, 111, 110 и 1. Последняя дополняется до 100 и получается запись $0{,}2764_8$. Эта же двоичная запись разбивается на группы 0101, 1111, 01 (дополняется до 0100), откуда получаем $0{,}5F4_{16}$.

1.4.4. Сложение и умножение в двоичной системе

В десятичной системе 9 + 1 = 10, а в двоичной 1 + 1 = 10. При сложении в столбик это означает сумму 0 и перенос 1 в следующий, старший разряд. Запишем это в виде 1 + 1 = (1)0. Например, сложим 11_2 и 110_2 в столбик, указывая переносы в скобках. В младший разряд переноса нет, но искусственно допишем его как (0): (0) + 1 + 0 = (0)1. Далее (0) + 1 + 1 = (1)0, (1) + 0 + 1 = (1)0. Перенос 1 из старшего разряда становится четвертой цифрой. Запись в столбик имеет следующий вид (в скобках наверху указаны переносы).

```
 (1100)
+  11
  110
 ----
 1001
```

Когда число в десятичной системе умножается в столбик на число, записанное одной цифрой *X*, вычисляется произведение очередной цифры числа и *X*. К этому произведению прибавляется перенос от предыдущего разряда и получается сумма *S*. Младшая цифра *S*, т.е. *S* mod 10, записывается в результат, а старшая, *S* div 10, запоминается как перенос в следующий разряд. Так мы двигаемся от младшей цифры сомножителя к старшей. Например, при умножении 289 на 9 имеем: 9×9 = (8)1, (8) + 8×9 = (8)0, (8) + 2×9 = (2)6, т.е. 289×9 = 2601.

При умножении на число, в котором больше одной цифры, произведения, получаемые при умножении на количество десятков, сотен и так далее, сдвигаются влево и складываются.

В двоичной системе все так же, но проще. Результатом умножения числа *A* на 1 является *A*, на 10 — *A*0, на 100 — *A*00 и т.д. Дописыванию нулей справа соответствует сдвиг записи числа влево. Таким образом, достаточно записывать первый сомножитель, сдвигать его влево на нужное количество разрядов, учитывая только единицы во втором числе, и вычислять сумму получаемых слагаемых. Например, рассмотрим умножение 1101 на 1011.

```
  1101
×
  1011
 ------
  1101
 1101
1101
--------
10001111
```

Проверим: $1101_2 = 13$, $1011_2 = 11$, 13×11 = 143, $10001111_2 = 2^7 + 2^3 + 2^2 + 2^1 + 2^0 = 128 + 8 + 4 + 2 + 1 = 143$.

1.4.5. Внутреннее представление целых чисел

Целые числа представляются в компьютере в *беззнаковой и знаковой формах* (*кодах*). Далее числа отождествляются с их представлением, хотя с точки зрения математики это и неверно.

Беззнаковые коды занимают несколько байтов, обычно 1, 2 или 4. Байты нумеруются справа налево от 0 до N–1 (от младших к старшим).[3] Биты внутри байтов нумеруются от 0 до 7. N байт содержат $8N$ бит, поэтому есть еще сквозная нумерация битов — от 0 до $8N$–1 (от младших к старшим).

Последовательность битов $x_{8N-1}\ldots x_0$ представляет число $x_{8N-1}\times 2^{8N-1} + \ldots + x_1\times 2^1 + x_0$, например, 00…00 — число 0, 00…01 — 1, 00…10 — 2, 11…11 — 2^{8N-1}. Следовательно, диапазон (множество) чисел — от 0 до 2^{8N}–1. При N = 1, 2 и 4 самое большое число — соответственно 255, 65 535 и 4 294 967 295.

Знаковые числа занимают те же N, т.е. 1, 2 или 4 байт. Старший бит представляет знак числа: 0 — "+", 1 — "–". Положительные числа представляются так же, как и беззнаковые, только за счет знакового бита диапазон их меньше — от 0 до 2^{8N-1}–1. При N = 1, 2 или 4 это соответственно 127, 32 767 и 2 147 483 647. Такое представление называется *прямым кодом*. Например, прямой код максимального целого — 011…1.

Отрицательные числа представляются в так называемом *дополнительном коде*. Для отрицательного числа A он обозначается $D(A)$ и образуется так:

а) по прямому коду числа $|A|$ заменой всех 0 на 1 и всех 1 на 0 строится *обратный код* $R(A)$;

б) по $R(A)$ как беззнаковому целому вычисляется $D(A) = R(A) + 1$.

Очевидно, $D(A) = R(|A| - 1)$.

Построим, например, дополнительный код числа –144 в двух байтах. Прямой код для 144 —

0000 0000 1001 0000,

обратный —

1111 1111 0110 1111.

Добавление 1 дает дополнительный код числа –144:

$$\begin{array}{r} 1111\ 1111\ 0110\ 1111 \\ \underline{\qquad\qquad\qquad\quad 1} \\ 1111\ 1111\ 0111\ 0000 \end{array}$$

Заметим, что он является также обратным кодом числа –143.

По дополнительному коду отрицательное число "восстанавливается" в обратном порядке:

а) $D(A)$ считается беззнаковым целым; вычисляется $R(A) = D(A) - 1$;

б) код, обратный к $R(A)$, является прямым кодом числа $|A|$.

Результат будет тем же, если:

а) построить код $R(D(A))$, обратный к $D(A)$;

б) к $R(D(A))$ как к беззнаковому целому прибавить 1.

Соответствие целых чисел и их знаковых кодов приведено в табл. 1.1. При N = 1, 2 или 4 наименьшее число — соответственно –128, –32 768 и –2 147 483 648.

---

[3] *В действительности байты могут нумероваться в противоположном порядке.*

Таблица 1.1. Соответствие целых чисел и их знаковых кодов

Число	Код	Число	Код
$2^{8N-1}-1$	011…11	–1	111…11
$2^{8N-1}-2$	011…10	–2	111…10
…	…	…	…
1	000…01	$-2^{8N-1}-1$	100…01
0	000...00	-2^{8N-1}	100…00

1.4.6. Принципы представления действительных чисел

Действительные числа обычно занимают $N = 4$, 6, 8 или 10 байт, разделенных на *поля* (последовательности битов) <*знак*>, <*порядок*> и <*мантисса*>. Поле <*знак*> имеет длину 1 бит, длины двух других обозначим D и R соответственно; $1 + D + R = 8N$. Пусть соответственно s, e, m — значения этих полей как беззнаковых целых. Они представляют:

$s = 0$ — знак "+", $s = 1$ — знак "–";

e — истинный порядок числа $t = e - (2^{D-1} - 1)$;

m — мантиссу (дробную часть) $m_1 = m \times 2^{-R}$.

При значениях e, отличных от крайних (0 и 2^D-1), значения полей s, e, m задают число $(-1)^s \times (1 + m_1) \times 2^t$.

Поскольку $1 \le 1 + m_1 < 2$, говорят, что число представлено в *нормализованном виде*. Показатель t называется *истинным порядком* числа, а e — "*сдвинутым*" (он на $2^{D-1} - 1$ больше истинного). Таким образом, значения e от 1 до 2^D-2 задают истинные порядки t от $1 - (2^{D-1} - 1) = 2 - 2^{D-1}$ до $2^D - 2 - (2^{D-1} - 1) = 2^{D-1} - 1$.

Рассмотрим, например, представление при $D = 5$, $R = 10$ (в двух байтах). Сдвиг порядка $2^{5-1} - 1 = 2^4 - 1 = 15$. Рассмотрим изображение числа –12,375:

$$-12{,}375 = (-1100{,}011)_2 = (-1{,}100011)_2 \times 2^3,$$

т.е. $t = 3$, $m_1 = 0{,}100011$. Отсюда $s = 1$, $e = 3 + 15 = 18 = (10010)_2$, $m = 1\,000\,110\,000$, и число записывается в виде 1 10010 1000110000.

Последовательность битов 0 00001 0000000000 представляет минимальное положительное число, возможное при $D = 5$, $R = 10$: $(1 + 0) \times 2^{1-15} = 2^{-14}$. Следующее число записывается в виде 0 00001 0000000001; это $(1 + 2^{-10}) \times 2^{1-15} = 2^{-14} + 2^{-24}$. Как видим, разность между этими соседними числами — 2^{-24}.

Последовательность битов 0 11110 1111111111 представляет число, максимальное при $D = 5$, $R = 10$:

$$(1 + (2^{10} - 1) \times 2^{-10}) \times 2^{32-2-15} = (2 - 2^{-10}) \times 2^{15} = 2^{16} - 2^5 = 65\,504.$$

Предыдущее перед ним число имеет представление 0 11110 1111111110 и равно

$$(1 + (2^{10} - 2) \times 2^{-10}) \times 2^{32-2-15} = (2 - 2^{-9}) \times 2^{15} = 2^{16} - 2^6 = 65\,472.$$

Как видим, разность между двумя соседними числами изменилась от 2^{-24} до $2^5 = 32$.

При $e = 0$ независимо от s и m представляется число 0. При $e = 2^D - 1$ представление числа используется специальным образом и здесь не рассматривается.

Расположение, способ обработки и длины полей в представлении чисел зависят от конкретного типа компьютера и могут отличаться от указанных здесь.

- Действительные числа, представимые в компьютере, образуют конечное ограниченное подмножество рациональных чисел.

Резюме

- Описание решения задачи называется *алгоритмом*, а последовательность действий — *процессом*. Алгоритм записывается на определенном *языке* и, как правило, задает некоторое множество процессов его выполнения. Алгоритм, предназначенный для выполнения на компьютере, обычно называется *программой*.
- *Компьютер* — это система устройств, которые обрабатывают данные, выполняя программы.
- *Программа* — это последовательность команд для обработки числовых, символьных и других данных, записанная на языке, понятном компьютеру. Для выполнения программа загружается в оперативную память компьютера. Команды описывают обработку значений, или *данных*, порождение новых данных и запись их в память. Команды выполняются *арифметико-логическим устройством* процессора.
- *Регистры* — это наиболее быстродействующие элементы памяти. *Кэш-память* — это быстродействующая память, в которой временно хранится часть программы и ее данных. Ее использование освобождает процессор от необходимости обращаться к оперативной памяти за каждой командой или значением и ускоряет выполнение программы.
- Для продолжительного сохранения больших объемов информации используются *внешние носители*. Данные на внешних носителях называются *файлами*.
- В компьютере числа представляются в *двоичной системе* с цифрами 0 и 1, кодирующими два различимых устойчивых состояния элемента памяти, называемого *битом*. Восемь последовательных битов образуют *байт*. Байт является единицей измерения объема памяти и передачи данных.
- Оперативная память представляет собой *последовательность байтов*. Каждый байт имеет свой номер — *адрес*.
- *Языки высокого уровня* обычно содержат английские слова и общепринятую математическую нотацию.
- Перевод программы, написанной на языке высокого уровня, в машинный язык называется *трансляцией* (*компиляцией*) и происходит при выполнении программы-транслятора (*компилятора*). При этом высокоуровневая программа *читается*, *распознаются* заданные ею действия компьютера и *создается* их описание в виде машинной программы. *Интерпретатор* не транслирует программу в машинный код. Интерпретация состоит в том, что действия, заданные программой, сразу *выполняются*.
- *Редактор связей* (*компоновщик*) создает полный код программы и записывает его или в оперативную память (загружает), или на диск в виде файла, готового к выполнению.
- *Система программирования* (*интегрированная среда*) имеет в своем составе текстовый редактор, транслятор и/или интерпретатор, компоновщик, загрузчик и отладчик.
- *Структурное программирование* — это подход к созданию программ, облегчающий их проектирование, разработку, отладку и модификацию.
- Создание программы для решаемой задачи начинается с *уточнения* постановки задачи. *Анализ* задачи, как правило, позволяет выделить ее подзадачи.
- Структурное проектирование программы в значительной мере заключается в *разбиении* программы на части, описывающие решения подзадач, и их *согла-*

сование. Алгоритмы решения *уточняются* несколько раз до вида, по которому легко написать программу или ее часть. Написание собственно текста программы или ее частей называют *кодированием* или *разработкой*.

- Исправление ошибок в программе называется ее *отладкой*. Для выявления ошибок необходимо многократное выполнение программы со специально подобранными вариантами входных данных — *тестирование*.
- *Технологии программирования* определяют системы методов, применение которых ускоряет и облегчает создание надежных программ.
- *Система счисления* — это система обозначения чисел. В компьютере используется двоичная система счисления. В программировании применяются также десятичная, восьмеричная и шестнадцатеричная системы.
- Для представления целых чисел используются две формы — знаковая и беззнаковая. Действительные числа, представимые в компьютере, образуют конечное ограниченное подмножество рациональных чисел.

Контрольные вопросы

1. В чем отличие алгоритма от процесса решения задачи?
2. Укажите единицы памяти и количества состояний, в которых они могут находиться.
3. Что представляет собой машинная программа?
4. В чем заключается "высокий уровень" языка программирования?
5. Укажите основные этапы создания программы.
6. Чем отличается интерпретация программы от ее трансляции?
7. Какие системы счисления используются в программировании?

Задачи

1. Преобразовать десятичные числа 100, 255, 256, 640, 1024, 32767 в двоичную, восьмеричную и шестнадцатеричную системы счисления. Выполнить обратные преобразования.
2. Представить 36-ичные числа *ZY*, 100 в десятичной записи (36-ичные цифры *A*, *B*, …, *Y*, *Z* обозначают десятичные числа 10, 11, …, 34, 35 соответственно).
3. По указанным основаниям *P* и *P*-ичным записям дробей указать их десятичное представление:

   а) P =2; 0,0001; 0,1111;

   б) P = 3; 0,22; 0,(11);

   в) P = 16; 0,1; 0,*F*; 0,8; 0,(7).
4. Записать *P*-ичное представление десятичной дроби *d*, где:

   а) d = 0,5, P = 2, 3, 5, 8, 16, 20;

   б) d = 0,1, P = 2, 3, 5, 8, 16, 20.
5. Указать дополнительный код чисел –1, –8, –9, –32767, –32768 в двух байтах.
6. Предположим, что при сложении и вычитании целых чисел в знаковом коде перенос из старшего значащего разряда становится содержанием знакового раз-

ряда, а перенос из знакового теряется. Указать значения выражений $maxi + 1$ и $mini - 1$, где *maxi* и *mini* обозначают максимальное и минимальное целые числа, представимые в данном знаковом коде.

7. Выразить с помощью степеней числа 2 минимальное и максимальное по модулю действительные числа, представимые в 4 байт ($D = 8$, $R = 23$), в 8 байт ($D = 11$, $R = 52$) и в 10 байт ($D = 16$, $R = 63$). Выразить эти числа приблизительно в виде степеней числа 10, считая, что $2^{10} \approx 10^3$.