И. Ш. Хабибуллин

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ВЫСОКОГО УРОВНЯ C/C++

- *Базовые компьютерные алгоритмы*
- *Структуры данных*
- *Принципы объектно-ориентированного программирования*
- *Жизненный цикл программного продукта*
- *Стандартная библиотека шаблонов STL*

УЧЕБНОЕ ПОСОБИЕ

bhv®

И. Ш. Хабибуллин

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ВЫСОКОГО УРОВНЯ C/C++

Рекомендовано УМО по университетскому политехническому образованию
в качестве учебного пособия для студентов высших учебных заведений,
обучающихся по направлению 654600
«Информатика и вычислительная техника»

Санкт-Петербург

«БХВ-Петербург»

2006

УДК 681.3.068+800.92(075.8)
ББК 32.973.26.018.1я73
Х12

Хабибуллин И. Ш.

Х12 Программирование на языке высокого уровня. С/С++. — СПб.: БХВ-Петербург, 2006. — 512 с.: ил.

ISBN 5-94157-559-9

Учебное пособие написано на основе одноименного учебного курса и посвящено технологии программирования на языках высокого уровня. Рассматриваются элементы современных языков программирования с примерами их реализации на языке С/С++. Большое внимание уделяется стилю программирования. Разобраны базовые алгоритмы и основные структуры данных, принципы объектно-ориентированного программирования, работа со стандартной библиотекой шаблонов STL, а также этапы и современные методы разработки надежного программного обеспечения. Приемы программирования и применения алгоритмов и структур данных иллюстрируются фрагментами программ. Может использоваться как справочник по языкам С и С++, так как содержит схемы, таблицы, описания стандартных библиотек функций и библиотеку шаблонов классов STL.

Для студентов технических вузов

УДК 681.3.068+800.92(075.8)
ББК 32.973.26.018.1я73

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Людмила Еремеевская*
Зав. редакцией	*Григорий Добин*
Редактор	*Константин Костенко*
Компьютерная верстка	*Татьяны Олоновой*
Корректор	*Наталия Першакова*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Рецензенты:

В. Г. Трусов, д.т.н., профессор, заведующий кафедрой «Программное обеспечение ЭВМ и информационные технологии» Московского государственного технического университета им. Н. Э. Баумана, заместитель председателя УМК по специальности 220400 «Программное обеспечение ЭВМ»;

Ф. М. Аблаев, д.ф.-м.н., профессор, член-корреспондент Академии наук Республики Татарстан, заведующий кафедрой теоретической кибернетики факультета вычислительной математики и кибернетики Казанского государственного университета.

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 26.05.06.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 41,28.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

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

ISBN 5-94157-559-9

Оглавление

Предисловие

Книга, которую вы держите в руках, посвящена введению в современное состояние алгоритмических языков и технологии программирования на языках высокого уровня, главным образом, на объектно-ориентированных языках. Она нацелена на студентов младших курсов технических специальностей, изучающих дисциплину "Алгоритмические языки и программирование", но может использоваться всеми программистами, не применявшими в своей работе языки C/C++ и объектно-ориентированное программирование, но желающими быстро и в полном объеме овладеть ими.

Ну и что, скажете вы? Таких книг море. Есть масса учебников для всех желающих. Книг, посвященных языку C/C++, в каждом магазине лежит добрый десяток. Среди них есть подробнейшие тысячестраничные "кирпичи" и для профессионалов, и для начинающих программистов. Зачем нужен еще один учебник по C/C++?

Во-первых, я сам с большим удивлением обнаружил, что среди великого многообразия книг по программированию нет учебника по курсу "Алгоритмические языки и программирование", отвечающего действующему ныне государственному образовательному стандарту. Информационные технологии развиваются очень быстро, и стандарты меняются раз в несколько лет. Выпуск учебников отстает даже от выпуска образовательных стандартов.

Во-вторых, книга не акцентирует внимание на синтаксических конструкциях языка программирования. В современные средства разработки встроена подробная интерактивная справка (Help), из которой всегда можно узнать синтаксические правила, да и компилятор укажет на неправильные конструкции в тексте программы. Язык C/C++ выбран только потому, что он остается основным алгоритмическим языком программирования в современных разработках программного обеспечения.

В-третьих, для чтения книги не нужны никакие предварительные познания в программировании, ее может читать любой человек, знакомый с компьютерами на уровне "продвинутого" пользователя и помнящий некоторые сведения из школьного курса основ информатики и вычислительной техники.

В-четвертых, сейчас все большее распространение получает свободно распространяемое программное обеспечение. Книга выводит читателя на уровень его разработки. Программист, прочитавший эту книгу, сможет сразу же принимать участие в разработке Linux, FreeBSD, XFree, OpenOffice и другого открытого программного обеспечения.

Эти отличительные особенности повлияли на манеру написания книги. Ее первые главы написаны очень просто, их материал может понять каждый логически мыслящий человек. По мере подачи материала изложение становится более сжатым. Это диктуется усложнением тематики последующих глав и конечной целью книги — вывести читателя на передний край современного программирования.

Основное внимание в книге уделяется стилю программирования. За время работы с компьютерами мне пришлось использовать множество языков, поработать во многих операционных системах и изучить множество технологий создания компьютерных программ. Каждая технология диктует свои правила программирования, но два правила остаются неизменными.

Первое правило заключается в неуклонном следовании принципу KISS. Эта аббревиатура расшифровывается так: "Keep It Simple, Stupid". Здесь в шутливой и грубоватой форме выражена рекомендация: программировать как можно проще, не применять вычурных конструкций и вынужденных трюков, которые затрудняют чтение текста программы и могут привести к трудноисправимым ошибкам в ней.

Второе правило гласит: программировать надо стильно. У каждого языка, у каждой технологии программирования есть свой стиль. Очень важно, знакомясь с языком, сразу же проникнуться его стилем и следовать ему в своей работе. Это предотвратит многие ошибки и даст возможность работать в любой команде профессионалов и свободно общаться с другими разработчиками.

В книге много упражнений и вопросов, взятых из жизни. С такими задачами и вопросами я сталкивался сам, изучая языки программирования. Такие задачи и вопросы ставили передо мной ученики. Если вы сумеете ответить на них, значит, вы хорошо усвоили материал раздела и можете смело переходить к следующему пункту.

В книге много листингов фрагментов реальных программ. Все они проверены компилятором GCC 3.4.2 с библиотекой функций и классов glibc 2.3.5 и выполнены в программной среде Linux 2.6.9, собранной в дистрибутиве Fedora Core 3. Все фрагменты программ написаны с использованием только стандартных средств языков C и C++, в них не применены никакие особенности конкретных компиляторов.

Для выполнения упражнений достаточно иметь под рукой текстовый редактор, компилятор с библиотекой функций и классов и программную среду, в которой будут выполняться готовые программы. Выбор компилятора обычно диктуется операционной системой, для которой пишется программа. Если вы пишете программу под UNIX, то используйте компилятор cc, имеющийся в среде разработки вашей системы. Если вы работаете в Linux или FreeBSD, то пользуйтесь компилятором GCC. Для .NET лучше всего подойдет Visual Studio for .NET. У профессионалов есть свои соображения и предпочтения, но к компилятору надо привыкнуть, ознакомиться с его особенностями и возможностями, а это требует некоторого времени.

Введение

История электронной вычислительной техники насчитывает уже более шестидесяти лет. Сначала электронные вычислительные машины собирались на триодах и пентодах, потом на транзисторах, затем на микросхемах. Микросхемы становились все более и более емкими и компактными. Сейчас микросхемы настолько усложнились, что изменилось даже их название, специалисты стали называть их чипами. Электронные устройства, похоже, исчерпали свои возможности, и, говоря о будущем вычислительной техники, ученые все увереннее называют другие физические принципы. Ожидаются оптические компьютеры, квантовые компьютеры, говорят о сверхпроводимости при низких температурах.

Поэтому, вводя понятие компьютера, сейчас не упоминают об электронике и вообще не говорят о его физическом устройстве. Компьютер определяется просто как *устройство для хранения, обработки и передачи информации.*

Элементная база машин часто меняется, но одно остается неизменным — электронные элементы всегда имеют два устойчивых состояния. Конечно, можно собрать электронные устройства с большим числом устойчивых состояний, но надежность таких устройств будет невелика, поскольку различать большое число состояний очень трудно.

У двух устойчивых состояний может быть самая разная физическая природа: магнитный сердечник может быть намагничен по часовой стрелке или против, на концах проводника может быть разность потенциалов или ее отсутствие, постоянный ток может идти в прямом или обратном направлении, конденсатор может быть заряжен или разряжен и т. д. Чтобы управлять этими состояниями, для них надо ввести какие-то обозначения. Можно было бы обозначить одно устойчивое состояние плюсом, а другое — минусом, можно было бы использовать диез и бемоль, "инь" и "янь", но специалисты выбрали ноль и единицу. Это было сделано не случайно, а для того чтобы в действиях с числами можно было применить двоичную систему счисления (*см. гл. 1*).

Итак, *физически* компьютер может быть собран из самых разных электронных элементов, но с точки зрения программирования, как говорят,

логически, компьютер состоит из огромного множества миниатюрных устройств, принимающих значения 0 и 1. Все сведения и все команды, которые мы хотим передать компьютеру, должны состоять только из нулей и единиц. Например, вот такая строка нулей и единиц хорошо понятна компьютеру:

```
00000001000011010000111000001010
```

Для некоторых типов машин это команда сложения двух целых чисел. Человеку такая запись ничего не говорит. Более того, человек не может ее запомнить, ему трудно даже переписать ее без ошибок. Что ж, очевидно, человек и компьютер говорят на совершенно разных языках.

У первых компьютеров на передней панели располагался ряд из 16 или 32, по числу разрядов машины, тумблеров. Включенное положение тумблера означало единицу, выключенное — ноль. Оператор машины набирал команду, состоящую из нулей и единиц, ставя в нужное положение соответствующие тумблеры, и нажимал кнопку ввода команды. После этого можно было набирать следующую команду, изменяя состояние тумблеров. После ввода всех команд оператор нажимал кнопку запуска, и программа начинала выполняться. Сейчас все эти манипуляции кажутся смешными и нелепыми, но в то время даже такая работа электронной вычислительной машины вызывала немалое восхищение.

Результат работы программы распечатывался на длинном рулоне бумаги, а промежуточные результаты высвечивались рядом небольших лампочек. Это было очень неудобно, но ученые были чрезвычайно довольны.

Составные части компьютера

Исходя из определения компьютера как устройства для хранения, обработки и передачи информации, следует заключить, что он состоит из трех основных частей:

- устройства для хранения информации или памяти (store);
- устройства для обработки информации — центрального процессора (Central Processor Unit, CPU);
- устройств для передачи информации — устройств ввода-вывода (Input/Output units, I/O).

Процессор называется центральным, потому что в каждом компьютере есть еще несколько небольших специализированных процессоров, не играющих самостоятельной роли. Они выполняют операции ввода-вывода, стандартные преобразования информации, другие рутинные действия, и совершенно незаметны программисту.

Логическое строение памяти компьютера

Память компьютера разделена на несколько уровней: регистры, кэш-память, оперативная память, внешняя память. Эти уровни различаются емкостью, быстродействием чтения и записи, ценой. Регистры встроены в процессор и используются им для хранения промежуточных результатов вычислений. Это самая быстрая память, но и самая дорогая. Кэш-память расположена между процессором и оперативной памятью и служит буфером между ними, ускоряющим обмен данными. Основным уровнем является оперативная память.

Оперативная память современного компьютера собрана на чипах, состоящих из миниатюрных элементов, имеющих, как говорилось выше, два устойчивых состояния. Эти элементы хранят *биты* (bit — сокращение слов binary digit) информации. Такое название, в буквальном переводе "двоичная цифра", выбрано потому, что в двоичной записи один бит — это один разряд двоичного числа: 0 или 1.

Бит — очень мелкая единица хранения информации. Допустим, нам надо хранить тексты в виде последовательностей букв, цифр и других печатных символов. Мы можем обозначить символ "А" нулем, символ "Б" — единицей и тем самым исчерпать возможности одного бита.

Два бита дают уже четыре варианта: 00, 01, 10, 11. Три бита дадут возможность хранения восьми символов, например, можно обозначить "А" — 000, "Б" — 001, "В" — 010, "Г" — "011", "Д" — 100, "Е" — 101, "Ё" — 110, "Ж" — 111. Вообще, N битами можно закодировать 2^N символов. Поскольку обычные тексты состоят из 150—200 символов, включая знаки препинания, математические обозначения и прочее, достаточно взять 8 битов, которыми можно закодировать 256 символов. Число 8 удобно еще и тем, что это третья степень двойки, а машина "любит" двоичный счет.

Набор из 8 битов называется *байтом* (byte). Правда, сначала байт состоял из семи битов, но это было очень давно. Во многих компьютерах к каждому байту добавляют девятый, контрольный, бит, но, поскольку он недоступен для программирования и не содержит полезной информации, его не включают в число битов, составляющих байт. Байт — это самая мелкая единица хранения информации в памяти компьютера. Все байты в ней пронумерованы, начиная от 0. К каждому байту можно обратиться по его номеру, как говорят, *адресу*, и прочитать хранящуюся в нем информацию или записать новую информацию.

Байт оказался слишком мелкой единицей хранения информации, ведь в нем можно записать всего один символ. Поэтому чаще используется *килобайт* — тысяча байтов, сокращенно KB, *мегабайт* — тысяча килобайтов, сокращенно MB, *гигабайт* — тысяча мегабайтов, GB, *терабайт* — тысяча гигабайтов, TB. Специалисты по сетевым технологиям, называющие байт

октетом (octet), предпочитают употреблять не байт, а бит. Они пользуются *килобитами*, сокращенно Kb, *мегабитами*, Mb и т. д.

В прошлом веке греческие приставки кило, мега, гига, тера необоснованно означали множитель, кратный не 1000, а $1024 = 2^{10}$, но в декабре 1998 г. Международной электротехнической комиссией (International Electrotechnical Commission, IEC) был принят стандарт, по которому этим приставкам вернули их исконный смысл. Единицы информации с множителем 1024 получили новые названия *кибибайт* (kibibyte, KiB), равный 1024 байтам, *мебибайт*, (mebibyte, MiB), равный 1024 кибибайтам, *гибибайт* (gibibyte, GiB), равный 1024 мебибайтам, *тебибайт* (tebibyte, TiB) и т. д. Новые названия еще не прижились, но старые названия с новым смыслом уже используются производителями. Например, если производитель жесткого диска утверждает, что его емкость составляет 160 Гбайт, то будьте уверены, что это именно 160 000 000 000 байтов и ни одним байтом больше.

Кроме этих точных единиц измерения объема памяти, существует еще понятие ячейки памяти. *Ячейка памяти* — это некоторое количество байтов: 2, 4, 8, 100, необходимое для хранения одного числа, одного слова, одной машинной команды. *Адресом* ячейки считается адрес ее первого байта, а *длиной* — количество байтов, ее составляющих. Говорят, что "для хранения целого числа выделяется ячейка длиной в 4 байта", что "переменная `x` хранится в ячейке с адресом 2048" и т. д.

Машинные команды

Машинные команды, исполняемые центральным процессором — это повелительные утверждения вида: "Прибавить к содержимому ячейки 2020 содержимое ячейки 2024", "Перенести содержимое ячейки 4000 в ячейку 4800", "Сравнить на равенство содержимое ячеек 1600 и 1604". Они состоят из двух частей: кода операции "прибавить", "перенести", "сравнить" и адресов операндов. В некоторых случаях команды содержат еще дополнительные информационные биты.

Обычно процессоры могут выполнять около сотни команд, поэтому для записи кода операции достаточно выделить один байт. Чаще всего так и делается, хотя иногда код операции занимает больше или меньше места в памяти. Адреса операндов занимают от одного до десяти байтов. Для каждого типа процессоров определяется самая распространенная длина команды. Она называется *машинным словом* или просто *словом* (word). У большинства популярных современных процессоров машинное слово составляет 4 байта, 32 двоичных разряда, но сейчас идет активный переход к 64-разрядным процессорам, у которых машинное слово равно 8 байтам.

Величина машинного слова определяет все остальные характеристики процессора. Он обрабатывает информацию порциями величиной в одно ма-

шинное слово, поэтому шины, которые связывают оперативную память и процессор, имеют число проводников, совпадающее с машинным словом. Как уже говорилось выше, внутри процессора есть небольшое количество ячеек очень быстрой памяти, называемых *регистрами*. Обычно их число колеблется от 16 до 64 ячеек. Регистры используются процессором для хранения промежуточных значений. Длина большинства регистров также совпадает с машинным словом.

Машинные команды хранятся в оперативной памяти, так же, как и те данные, которые они обрабатывают. Процессор по очереди извлекает команды из оперативной памяти, переносит их в свои регистры, там по коду операции узнает, что ему надо делать, и выполняет команду. В процессе выполнения команды процессор извлекает из оперативной памяти нужные операнды, а после выполнения заносит в оперативную память результаты команды.

Какие же команды можно давать процессору? Хотя их довольно много, они составляют всего несколько групп:

- команды пересылки информации из оперативной памяти в регистры процессора и обратно;
- арифметические команды сложения, вычитания, умножения и деления;
- команды сравнения по величине двоичных чисел;
- логические команды конъюнкции, дизъюнкции, отрицания, инверсии;
- команды сдвига двоичных разрядов;
- прочие команды, в числе которых команды передачи управления и несколько команд ввода-вывода.

Как видите, машинные команды очень примитивны, они выполняют мелкие действия над отдельными ячейками. Представьте себе, сколько нужно команд типа "загрузить содержимое ячейки с адресом 2048 в регистр 05" и "сдвинуть на два разряда влево содержимое регистра 12", чтобы запрограммировать задачу "найти в телефонном справочнике номер абонента с какой-то лошадиной фамилией, переехавшего в этот район в ноябре или декабре прошлого года". Представили? Тогда вы можете оценить гигантский труд программистов, проделанный ими всего за каких-нибудь полвека.

Поколения языков программирования

Двоичная запись информации крайне неудобна для человека, поэтому довольно быстро от двоичной записи перешли к восьмеричной записи, а затем к шестнадцатеричной. О шестнадцатеричной системе счисления мы поговорим позже (*см. гл. 2*), а пока я приведу шестнадцатеричную запись той же машинной команды сложения, которая была показана в двоичной записи:

```
010D0E0A
```

Как видите, запись команды осталась такой же непонятной, но, по крайней мере, в ней труднее ошибиться. Главное же заключается в том, что такая запись непонятна и машине! Для нее это просто ничего не значащий набор символов.

Пришлось написать небольшую программу, переводящую шестнадцатеричную запись в двоичную. После этого процесс выполнения программы усложнился. В машину сначала вводится непонятный для нее текст программы, написанный в шестнадцатеричных кодах. Потом этот текст обрабатывается программой, преобразующей каждый символ в двоичный набор нулей и единиц. И только после этого преобразованная программа начинает выполняться.

Шестнадцатеричная или восьмеричная запись команд стала первым языком программирования, требующим перевода на машинный язык. Такие языки были названы языками программирования *первого поколения*, *1GL* (1 Generation Language).

Следующим шагом стал переход от шестнадцатеричной записи информации к символьной. Раз уж машина не понимает вводимый в нее текст, то почему бы не сделать его хотя бы понятным человеку? Договорились обозначать команды сокращенной записью выполняемых ими действий, а ячейки памяти — какими-нибудь именами. Например, команду сложения целых чисел обозначили `ADD` — английским глаголом "прибавить". В этих обозначениях предыдущий пример выглядит так:

```
ADD A, B, C
```

Такая запись уже приемлема для человека. Разумеется, она совершенно непонятна машине, и понадобилось написать программу, переводящую эту запись в двоичный код. Такая программа была названа *ассемблером*, а язык — *языком ассемблера*. Языки ассемблера составили *второе поколение* языков программирования, *2GL*.

Наконец, через 10 лет развития вычислительной техники удалось разработать языки, позволяющие записывать команды машине почти как математические формулы. Например, предыдущую команду на таком языке можно записать в виде

```
C = A + B
```

Это был революционный прорыв. Теперь программисту не надо было думать о том, какие машинные команды будут выполняться в его программе. Он может вообще не знать систему команд компьютера. Более того, он может даже не знать тип компьютера, для которого пишет программу. Такими языками (*языками третьего поколения*, *3GL*) мы пользуемся до сих пор. Первым языком третьего поколения был язык вычисления формул FORTRAN, потом появился Algol и множество других, среди которых ныне

здравствующие — Pascal, C, C++, Java, C#. Программа, переводящая запись на машинный язык, называется *компилятором*. Конечно, компиляторы гораздо сложнее ассемблеров, это большие и сложные программы, создаваемые целыми коллективами программистов.

Для каждого языка третьего поколения создан не один компилятор. Во-первых, разные компиляторы создаются разными фирмами, конкурирующими между собой. Во-вторых, различаются отладочные и оптимизирующие компиляторы, создающие разный, черновой или окончательный, машинный код. В-третьих, для каждого типа компьютеров разрабатываются свои компиляторы, ведь набор машинных команд может сильно различаться. Выбор подходящего компилятора — это одна из важнейших задач программиста, приступающего к реализации своего проекта.

На этом счет поколениям языков программирования не остановился. Были созданы *декларативные* языки, составившие *четвертое поколение* языков программирования, *4GL*. Основная идея декларативных языков заключается в том, чтобы записывать не действия, приводящие к результату выполнения программы, а указания машине, по которым она сама проделает необходимые операции. Языки четвертого поколения говорят, *что* надо сделать, а не то, *как* это сделать.

Самым ярким примером языка четвертого поколения стал язык SQL (Structured Query Language), на котором происходит обращение к базам данных. На этом языке мы можем записать команду, гласящую: "Выбери из базы данных сведения обо всех Петровых, родившихся в январе 1985 года и не получивших среднее образование". Компилятор сам сгенерирует необходимые для выполнения этой команды действия и запишет их на машинном языке.

Попытки создать языки пятого поколения, основанные на голосовом вводе обычных для человека указаний типа: "Реши-ка мне дифференциальное уравнение", оказались неудачными и незаметно были сняты с повестки дня. Неожиданные новшества в создании новых языков прекратились. Теперь языки программирования развиваются постепенно, без заметных скачков.

Итак, на современных языках программирования записываются не машинные команды, а *операторы*, требующие предварительного перевода на машинный язык. Каждый оператор преобразуется компилятором в несколько машинных команд. По типу и составу операторов языки программирования делятся на алгоритмические или процедурные, функциональные и логические.

Алгоритмические языки

Программа, записанная на алгоритмическом языке программирования, представляет собой запись алгоритма решения задачи, ради которого написана программа. Она состоит из совокупности последовательных действий,

приводящих, в конце концов, к определенному результату. Такие языки программирования и являются предметом всей данной книги, поэтому здесь мы долго говорить о них не будем.

К алгоритмическим языкам относятся и объектно-ориентированные языки программирования. Алгоритмические языки программирования часто называют *процедурными* языками, но после появления объектно-ориентированных языков это название может ввести в заблуждение, ведь на объектно-ориентированных языках пишут не процедуры, а классы и входящие в них методы.

Функциональные языки

Функциональные языки программирования, такие как Lisp, Рефал, APL, Haskell, SMI, Scheme и другие, подходят к решению поставленных перед ними задач совершенно иначе, чем алгоритмические языки. Они представляют каждое выполняемое в программе действие в виде функции. Например, операция суммирования двух чисел записывается как функция `sum(a, b)` от двух переменных-слагаемых. Более сложные выражения записываются в виде набора функций, вызывающих друг друга. Например, выражение $\sqrt{a^2 + b^2}$ будет на функциональном языке выглядеть примерно так: `sqrt(sum(sqr(a), sqr(b)))`.

Функциональные языки не признают присваиваний и передач управления. Циклы вычисляются с помощью рекурсии, разветвления реализуются логическими функциями. Такая запись удобна при обработке символьной информации, длинных списков, ее выгодно применять для поиска и замены некоторых участков текста. Поэтому на функциональных языках часто пишут компиляторы, программы поиска, программы автоматического доказательства теорем.

Логические языки

Логические языки программирования, такие как Prolog, Mandala и другие, смотрят на результат программы как на некоторое утверждение, которое надо вывести из исходных данных. Программа рассматривается как доказательство этого утверждения. Доказательство проводится по правилам математической логики, откуда и взято название этих языков.

В программе, написанной на логическом языке, вообще нет действий. В ней задаются исходные данные как аксиомы и правила вывода истинных утверждений из этих аксиом. Программа по этим правилам проводит логические рассуждения и делает выводы, приводящие к нужному результату.

При этом для ускорения и упрощения работы программа использует накопленную ранее базу данных, содержащую уже сделанные логические выводы.

Логические языки программирования наиболее удобны для решения задач искусственного интеллекта, медицинской диагностики, машинного перевода — везде, где нужно делать логические заключения. Они применяются и для составления сложных расписаний, "добычи знаний" из больших баз данных, прогнозирования в экономике.

Языки низкого и высокого уровня

Еще одно популярное деление языков программирования — это деление на низкоуровневые и высокоуровневые языки программирования. Низкоуровневые языки близки к машинным языкам, они содержат набор машинных команд, записанных по правилам этого языка, а также директивы, макросы и другие элементы. Это, прежде всего, языки ассемблера и различные псевдокоды.

Программы, написанные на языках низкого уровня, наиболее близки к машине. На них можно в точности написать последовательность машинных команд, составляющих программу. Компиляторы низкоуровневых языков, чаще всего это ассемблеры, сравнительно невелики, работают быстро и надежно. Но программа, написанная на языке низкого уровня, получается очень длинной, трудной для понимания и выявления ошибок.

Высокоуровневые языки программирования, такие как языки третьего поколения, напротив, далеки от машинных команд. Программа получается относительно короткой, понятной и удобной для восприятия. Но компиляторы высокоуровневых языков велики, громоздки, работают медленно и зачастую неэффективно.

Особняком стоит язык С, который мы будем изучать в первой части книги. По своему синтаксису это типичный язык высокого уровня, но в его состав входят операции, позволяющие работать с отдельными байтами и битами, как это делают языки низкого уровня. Поэтому многие специалисты относят его к языкам низкого уровня. Иногда его называют даже "переносимым ассемблером".

Компилируемые и интерпретируемые языки

До сих пор мы говорили о том, что программу, написанную на языке высокого уровня, перед выполнением следует откомпилировать, т. е. перевести на машинный язык. Компилятор — это тоже компьютерная программа, на

вход которой подается файл с исходным текстом, написанным на языке высокого уровня. Этот файл во многих операционных системах называется *исходным модулем* (source module). Компилятор переводит программу на машинный язык и записывает ее в другой файл, называемый *объектным модулем* (object module).

Содержимое объектного модуля в самых простых случаях можно загрузить в оперативную память и выполнить. Но чаще всего объектный модуль перед выполнением надо еще *скомпоновать* (link) с другими объектными модулями, содержащими дополнительные модули программы, системные функции, служебные сведения и прочие данные, необходимые для выполнения программы.

Компоновку выполняет специальная программа, так и называемая *компоновщиком* (linker) или *редактором связей*. На ее вход подаются файлы с объектными модулями, а на выходе получается *исполнимый модуль* (executable module) — файл с полностью готовой к выполнению программой. Этот файл загружается в оперативную память и выполняется.

Очень часто компиляцию и компоновку объединяют в одно действие. Для этого компилятору дается специальное указание, и он сразу же после своей работы сам вызывает компоновщик. Программисту все это представляется единым процессом, только на экране дисплея появляются сообщения об окончании компиляции и начале компоновки.

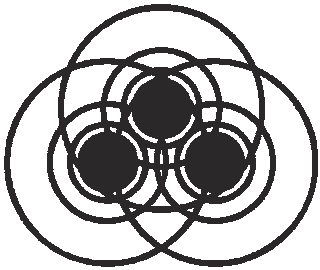
Все описанное выше характерно для *компилируемых* языков программирования. Другой класс языков составляют *интерпретируемые* языки программирования, не требующие компиляции и компоновки. Это тоже языки высокого уровня, поэтому программы, написанные на них, часто называемые *скриптами* (script) или *сценариями*, требуют перевода на машинный язык. Для этого создается программа-интерпретатор операторов, записанных в программе. В отличие от компилятора программа-интерпретатор переводит на машинный язык не весь текст целиком, а каждый оператор по отдельности, и тут же выполняет полученный машинный код.

Интерпретация программы происходит медленнее, чем выполнение исполнимого модуля, из-за того, что каждый оператор сначала преобразуется в машинные коды. Кроме того, в процессе интерпретации могут неожиданно возникнуть ошибки, ведь исходный текст не просматривается заранее в поисках ошибок. Тем не менее, интерпретируемые языки широко распространены из-за удобства создания программ. Программу можно написать и сразу выполнять без предварительных преобразований. Если работа программы кажется неудовлетворительной, то ее текст можно изменить и тут же выполнить программу заново без всякой перекомпиляции.

В современном программировании разница между компилируемыми и интерпретируемыми языками стирается. Это слияние происходит по двум на-

правлениям. С одной стороны, для интерпретируемых языков создаются компиляторы. Классическим интерпретируемым языком всегда считался язык BASIC, но сейчас для него написано много компиляторов. У программиста есть выбор. Программу, написанную на BASIC, можно интерпретировать и сразу выполнить, а можно предварительно откомпилировать в исполнимый модуль и выполнить в другое, более удобное время.

С другой стороны, интерпретаторы научились сохранять машинный код уже проинтерпретированных и выполненных операторов. При повторном выполнении этих операторов, например, в циклах, интерпретатор использует готовые машинные команды, что значительно ускоряет работу. Такие интерпретаторы называются *JIT-интерпретаторами* (Just-In-Time). Они работают значительно быстрее классических интерпретаторов и поэтому приобретают все большее распространение.



Глава 1

Алгоритмы и их запись

Компьютер работает очень быстро. Он выполняет сотни миллионов операций в секунду. Человек, даже тренированный, способен сделать за секунду только несколько осмысленных действий. Если изменить масштаб времени так, чтобы компьютер выполнял не больше десяти команд в секунду, то человек в этом масштабе будет совершать одно движение в 10—20 лет! Ясно, что при таком соотношении скоростей нельзя управлять компьютером как автомобилем, во время его работы. В таком случае компьютер практически все время будет простаивать, ожидая следующего действия пользователя. Надо дать компьютеру все инструкции или большинство инструкций до начала его работы, и он будет выполнять их со своей сумасшедшей скоростью. Вся программа действий компьютера должна быть заранее записана и введена в его оперативную память.

Допустим, мы выбрали один из языков программирования высокого уровня и решили написать на нем программу для компьютера. Что же она будет содержать? А что вообще содержат всякие инструкции и указания, которыми мы руководствуемся в жизни? Каждый из вас легко приведет множество примеров.

- Кулинарный рецепт: "Довести до кипения 5 стаканов подсоленного молока, тонкой струйкой всыпать 1 стакан манной крупы и, непрерывно помешивая, варить на слабом огне 10—15 минут до загустения. После этого положить 1 столовую ложку сахара и размешать".
- Совет прохожего: "Идите прямо два квартала, на перекрестке поверните направо, дойдите до магазина и войдите во двор. Зайдите во второй подъезд и поднимитесь на третий этаж".

- Дозировка и применение лекарства: "Принимать по одной таблетке 3—4 раза в день после еды в течение недели. При необходимости повторить курс лечения через 2—3 недели".
- Инструкция к стиральной машине: "Загрузите белье в машину, распределив равномерно крупные и мелкие вещи, и плотно закройте дверцу. Засыпьте стиральный порошок в выдвижной ящик. Нажмите кнопку Питание, чтобы включить машину. Выберите необходимые условия работы, поворачивая программатор по шкале и нажимая на каждую кнопку. Нажмите кнопку Старт".

Эти инструкции может выполнить любой человек, и не только человек, но даже машина. Другие инструкции требуют специальной подготовки.

- Из инструкции по окраске полов: "Профилированные галтели олифят. После высыхания шпаклюют резиновой пластинкой на сдир и шлифуют". Даже чтобы понять эти фразы, нужны соответствующие знания.
- Ремонт велосипеда: "Установить ротор на цапфу коленчатого вала, а кулачок на ротор так, чтобы штифты вошли в прорези ротора. Поставить и закрепить наковальню и конденсатор двумя винтами, при этом щеточка должна быть заведена за кулачок". Да, тут без инструктора не обойдешься.

Инструкции могут быть записаны специальными обозначениями, понятными только посвященным людям.

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

Известная со школы формула — это инструкция по вычислению корней квадратного уравнения. Для человека, не изучавшего математику, это какая-то непонятная шифровка, бессмысленный набор символов.

Аналогичные указания следует дать компьютеру. При этом надо учесть, что компьютер не обладает никакими предварительными знаниями, позволяющими человеку додумывать то, что не написано явно в инструкции. Компьютер — это "быстродействующий идиот", выполняющий то, что ему предписано. Даже людьми самые подробные инструкции понимаются поразному. Это иногда приводит к ошибкам, что хорошо знают военные, стремящиеся доводить выполнение команд до автоматизма. Что уж говорить о компьютере. Команды для него надо писать подробно и точно. Компьютеру дают не инструкцию, а алгоритм решения поставленной задачи.

Понятие алгоритма

Алгоритм или, по старинке, алгорифм — очень общее понятие и, как всякое общее понятие, его трудно определить. Попробуйте в двух-трех фразах

объяснить, что такое метод, наука, культура. Существует целая теория алгоритмов, которая вводит не одно, а даже несколько определений, доказывает их эквивалентность и детально изучает алгоритмы исходя из строгого определения. Теория алгоритмов доказала много интересных теорем. Оказывается, например, что область их применения ограничена — есть алгоритмически неразрешимые классы задач. У нас другой учебный курс, и нам не нужно точное определение. Для нас достаточно описать основные свойства: массовость, конечность, однозначность.

Массовость

Ценность алгоритма возрастает, если его можно применить к решению не одной, а целого класса задач. Такое свойство алгоритма называется его *массовостью*. Известные из арифметики правила сложения и умножения чисел "столбиком" — массовые алгоритмы, применимые к любым слагаемым или сомножителям.

Приведенная выше формула нахождения корней квадратного уравнения массовая, она применима к любому квадратному уравнению, все они могут быть решены одинаково, по одному алгоритму. Единственное ограничение — коэффициент a при второй степени неизвестного не должен обращаться в ноль. Но тогда это уже не квадратное, а линейное уравнение. Представьте себе, как трудно было бы делать расчеты, если бы для каждого квадратного уравнения пришлось бы придумывать свой метод решения! Впрочем, так приходится делать для большинства нелинейных уравнений, а если это не удается, применять приближенные методы нахождения их корней.

Кулинарные рецепты и инструкции по применению того или иного товара, как правило, не обладают массовостью, они составляются для каждого конкретного случая. Однако компьютерный алгоритм, разработанный для решения только одной частной задачи, чаще всего бесполезен, потому что за время, потраченное на его создание, кодирование и ввод в компьютер, можно было бы решить эту задачу вручную.

Разработчики алгоритмов стремятся расширить область их применения, сделать их как можно более массовыми. Идеально было бы найти универсальный алгоритм решения всех задач, но это, к счастью, невозможно, хотя бы потому, что есть алгоритмически неразрешимые классы задач. К счастью — ибо программисты никогда не останутся без работы.

С другой стороны, массовые алгоритмы, в силу своей общности, не учитывают конкретные особенности каждой решаемой ими задачи. Поэтому для особо важных задач приходится разрабатывать специальные алгоритмы, использующие все характеристики конкретной задачи. Этим объясняется великое разнообразие алгоритмов решения компьютерных задач.