Начала программирования

Programming_title.indd 1 28.03.2011 17:18:04

Elements of Programming

Alexander Stepanov,
Paul McJones

Addison-Wesley

Programming_title.indd 2 28.03.2011 17:18:04

Начала программирования

Александр Степанов,
Пол Мак-Джоунс

Москва • Санкт-Петербург • Киев
2011

Programming_title.indd 3 28.03.2011 17:18:04

ББК 32.973.26–018.2.75
С79
УДК 681.3.07

Издательский дом "Вильямс"
Зав. редакцией *С. Н. Тригуб*
Перевод с английского и редакция *К. А. Птицына*
Под редакцией *К. А. Птицына* и *А. А. Степанова*

По общим вопросам обращайтесь в Издательский дом "Вильямс" по адресу:
info@williamspublishing.com, http://www.williamspublishing.com

Степанов, Александр Александрович, **Мак-Джонс**, Пол.

С79 Начала программирования. : Пер. с англ. — М. : ООО "И. Д. Вильямс", 2011. — 272 с. : ил. — Парал. тит. англ.

ISBN 978–5–8459–1708–9 (рус.)

ББК 32.973.26–018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Addison-Wesley Publishing Company, Inc.

Authorized translation from the English language edition published by Addison-Wesley Publishing Company, Inc, Copyright © 2009 Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Russian language edition published by Williams Publishing House according to the Agreement with R&I Enterprises International, Copyright © 2011

Научно-популярное издание
Александр Александрович Степанов, Пол Мак-Джонс
Начала программирования

Литературный редактор *И. А. Попова*
Верстка *А. Н. Полинчик*
Художественный редактор *В. Г. Павлютин*
Корректор *Л. А. Гордиенко*

Подписано в печать 08.04.2011. Формат 70×100/16
Гарнитура Times. Печать офсетная
Усл. печ. л. 21,1. Уч.-изд. л. 14
Тираж 1000 экз. Заказ №0000

Отпечатано по технологии CtP
в ОАО "Печатный двор" им. А. М. Горького
197110, Санкт-Петербург, Чкаловский пр., 15

ООО "И. Д. Вильямс", 127055, г. Москва, ул. Лесная, д. 43, стр. 1

ISBN 978–5–8459–1708–9 (рус.)
ISBN 978–0–321–63537–2 (англ.)

© Издательский дом "Вильямс", 2011
© Pearson Education, Inc., 2009

Оглавление

Содержание

Дорогой читатель!

Две идеи, на которых основана эта книга, пришли мне в голову в середине 1970-х годов, когда я еще жил в Москве. Первая идея состояла в том, чтобы разбивать программы на небольшие (5–15 строк) алгоритмические части, каждая из которых сама по себе могла бы использоваться в других программах. Вторая идея состояла в том, что каждый такой алгоритм должен работать на любых типах данных, к которым он применим. Я называю их *обобщенными алгоритмами*. Мне казалось, что эти две идеи самоочевидны. К моему удивлению, мне много лет не удавалось изложить их удовлетворительно для самого себя. Несколько лет назад Пол Мак-Джонс уговорил меня вспомнить мою математическую молодость, и мы вместе написали *Начала программирования*.

Эта книга не учит программировать, не объясняет новую программную технологию, не пропагандирует новый язык программирования. Некоторым читателям может показаться, что она содержит случайный набор алгоритмов. Это не так. Надеюсь, что у читателя будет достаточно интереса, чтобы разобраться в структуре книги, ибо цель книги именно в том, чтобы показать глубокую взаимосвязь отдельных алгоритмов.

Надеюсь, что русский перевод нашей книги позволит мне хотя бы в малой мере воздать за то, что я получил от русской науки. Посвящаю это издание всем замечательным российским математикам, меня учившим.

Александр Александрович Степанов

Предисловие

В настоящей книге применяется дедуктивный подход к программированию, основанный на объединении программ с абстрактными математическими теориями, которые обеспечивают их работу. Представлены вместе описания этих теорий, алгоритмы, записанные в терминах этих теорий, а также теоремы и леммы, описывающие их свойства. Реализация алгоритмов на реальном языке программирования является центральной темой книги. Теоретические описания адресованы людям, поэтому в них научная строгость должна сочетаться с некоторой неформальностью; код предназначен для компьютера, поэтому должен быть абсолютно точным, даже если он имеет общее назначение.

Как и в других областях науки, в программировании наиболее подходящей основой является дедуктивный метод. Он обеспечивает декомпозицию сложных систем на компоненты с математически обоснованным поведением. Создание таких компонентов, в свою очередь, служит необходимой предпосылкой разработки эффективного, надежного, безопасного и экономически эффективного программного обеспечения.

Эта книга предназначена для тех, кто стремится глубже понять суть программирования, будь то профессиональные программисты или ученые и инженеры, для которых программирование составляет важную часть их профессиональной деятельности.

Книга предназначена для чтения от начала и до конца. Читатели смогут достичь понимания материала, только изучая код, доказывая леммы и выполняя упражнения. Кроме того, мы предлагаем несколько проектов, причем решения нам не известны. Изложение в книге является кратким, но внимательный читатель в конечном итоге сумеет обнаружить связь между ее частями и понять, чем мы руководствовались при выборе материала. Читатель должен поставить перед собой цель — раскрыть принципы построения книги.

Мы предполагаем наличие способности выполнять элементарные алгебраические манипуляции[1]. Мы также предполагаем, что читатель знаком с основ-

[1]Для проведения переподготовки по элементарной алгебре рекомендуем книгу [Chrystal, 1904].

ной терминологией логики и теории множеств на уровне базовых университетских курсов по дискретной математике; в приложении A приведены сводные данные об используемой нами системе обозначений. Мы приводим определения некоторых понятий абстрактной алгебры, необходимых для формулировки алгоритмов. Предполагаем также наличие развитых навыков программирования и понимания архитектуры ЭВМ[2] и знакомство с элементарными алгоритмами и структурами данных[3].

Мы выбрали язык C++, поскольку в нем механизмы абстракции сочетаются с точным представлением существующей машины[4]. Мы используем небольшое подмножество языка и записываем требования как структурированные комментарии. Мы надеемся, что следовать за изложением в книге смогут и читатели, не знакомые с языком C++. Подмножество языка, используемое в книге, определено в приложении B[5]. Везде, где имеется различие между математической системой обозначений и языком C++, типографский шрифт и контекст определяют, приведено ли описание математического обоснования или кода C++. У многих концепций и программ из этой книги есть параллели в STL (Standard Template Library — стандартная библиотека шаблонов C++), но некоторые решения, принятые в книге, отличаются от таковых в STL. В книге также игнорируются проблемы, которые должны быть решены в реальной библиотеке, такой как STL: пространства имен, видимость, встроенные директивы и т. д.

В главе 1 приведено описание значений, объектов, типов, процедур и концепций. Главы 2–5 посвящены алгоритмам на алгебраических структурах, таких как полугруппы и полностью упорядоченные множества. В главах 6–11 описываются алгоритмы на абстрактных представлениях памяти. В главе 12 рассматриваются объекты, содержащие другие объекты. В послесловии представлены наши размышления о подходе, изложенном в книге.

Благодарности

Мы благодарны компании Adobe Systems и ее руководителям за поддержку курса “Foundations of Programming” и этой книги, которая выросла из него. В частности, Грег Гилли (Greg Gilley) был инициатором проведения курса и предложил написать книгу; Дэйв Стори (Dave Story), а затем Билл

[2]Рекомендуем книгу [Patterson and Hennessy, 2007].

[3]См. [Tarjan, 1983].

[4]Стандартным справочником является [Stroustrup, 2000].

[5]Код из книги компилируется и выполняется в версиях Microsoft Visual C++ 9 и g++ 4. Этот код может быть загружен с сайта www.elementsofprogramming.com вместе с несколькими простейшими макросами, которые обеспечивают его компиляцию, а также тестовыми модулями.

Хенслер (Bill Hensler) обеспечивали неуклонную поддержку. Наконец, книга не была бы возможна без грамотного руководства Шона Пэрента (Sean Parent) и непрерывного контроля качества кода и текста. Идеи, изложенные в книге, стали итогом нашего тесного, почти тридцатилетнего сотрудничества с Дэйвом Массером (Dave Musser). Бьярне Страуструп (Bjarne Stroustrup) специально доработал язык C++ для поддержки этих идей. И Дэйв и Бьярне были достаточно любезны, чтобы прибыть в Сан-Хосе и тщательно рассмотреть предварительный проект книги. Шон Пэрент и Бьярне Страуструп написали приложение, определяющее подмножество C++, которое используется в книге. Джон Брандт (Jon Brandt) просмотрел несколько черновых вариантов книги. Джон Уилкинсон (John Wilkinson) тщательно прочитал окончательную версию рукописи, предоставив неисчислимое количество ценных предложений.

Книга существенно выиграла благодаря усилиям редактора Питера Гордона (Peter Gordon), редактора проекта Элизабет Райан (Elizabeth Ryan), литературного редактора Эвелин Пайл (Evelyn Pyle) и редакционных рецензентов: Мэтта Аустерна (Matt Austern), Эндрю Кёнига (Andrew Koenig), Дэвида Массера (David Musser), Арчи Робисона (Arch Robison), Джерри Шварца (Jerry Schwarz), Джереми Сика (Jeremy Siek) и Джона Уилкинсона (John Wilkinson).

Мы благодарим всех студентов — участников курса, который проводился в компании Adobe, а также проводившегося ранее курса в компании SGI, за их предложения. Мы надеемся, что нам удалось связать материал этих курсов в единое целое. Мы благодарны за комментарии Дэйву Абрахэмсу (Dave Abrahams), Андрею Александреску (Andrei Alexandrescu), Константину Аркудасу (Konstantine Arkoudas), Джону Бэннингу (John Banning), Гансу Боему (Hans Boehm), Анджело Борсотти (Angelo Borsotti), Джиму Денерту (Jim Dehnert), Джону Детревиллу (John DeTreville), Борису Фомичеву (Boris Fomitchev), Кевлину Хенни (Kevlin Henney), Юсси Кетонену (Jussi Ketonen), Карлу Мальбрену (Karl Malbrain), Мэту Маркусу (Mat Marcus), Ларри Мэзинтеру (Larry Masinter), Дэйву Пэренту (Dave Parent), Дмитрию Полухину (Dmitry Polukhin), Джону Риду (Jon Reid), Марку Рузону (Mark Ruzon), Джеффу Скотту (Geoff Scott), Дэвиду Симонсу (David Simons), Анне Степановой (Anna Stepanov), Тони Ван Ирду (Tony Van Eerd), Уолтеру Ваннини (Walter Vannini), Тиму Винклеру (Tim Winkler) и Олегу Заблуде (Oleg Zabluda). Благодарим Джона Баннинга (John Banning), Боба Инглиша (Bob English), Стивена Граттона (Steven Gratton), Макса Гальперина (Max Hailperin), Евгения Кирпичова (Eugene Kirpichov), Алексея Некрасова (Alexei Nekrassov), Марка Рузона (Mark Ruzon) и Хао Сонга (Hao Song) за то, что выявили ошибки в первой редакции книги. Благодарим Фостера Брereтона (Foster Brereton), Габриэль Дос Реис (Gabriel Dos Reis), Райана Ернста (Ryan Ernst), Абрахама Себастьяна (Abraham Sebastian), Майка Спертуса (Mike Spertus), Хеннинга Тилеманна (Henning Thielemann) и Карлу Вилло-

рию Бургацци (Carla Villoria Burgazzi) за то, что выявили ошибки во второй редакции книги. Благодарим Синдзи Досака (Shinji Dosaka), Райана Ернста (Ryan Ernst) и Стивена Граттона (Steven Gratton) за то, что выявили ошибки во третьей редакции книги.[6]

Наконец, мы благодарны всем людям, которые учили нас посредством своих опубликованных работ или лично, а также университетам и компаниям, которые позволили нам углубить наше понимание программирования.

[6] С последними по времени замечаниями и исправлениями можно ознакомиться на сайте www.elementsofprogramming.com. (См. раздел “Errata”).

Об авторах

Александр Степанов (Alexander Stepanov) изучал математику в Московском государственном университете с 1967 по 1972 гг. Он работает в области программирования с 1972 года: сначала в Советском Союзе, а после эмиграции в 1977 году в Соединенных Штатах. Он занимался программированием операционных систем, инструментов программирования, компиляторов и библиотек. Его работа над началами программирования поддерживалась General Electric, Политехническим институтом Бруклина, AT&T, HP, SGI и Adobe. В 1995 году он получил премию "Excellence in Programming" *Dr. Dobb's Journal* за проект стандартной библиотеки шаблонов C++.

Пол Мак-Джонс (Paul McJones) изучал прикладную математику в Калифорнийском университете, Беркли, с 1967 до 1971 гг. С 1967 года он занимался программированием в областях операционных систем, сред программирования, систем обработки транзакций и приложений для промышленных предприятий и потребительского рынка. Он работал в Калифорнийском университете, IBM, Xerox, Tandem, DEC и в Adobe. В 1982 году он и его соавторы получили премию "ACM Programming Systems and Languages Paper" за статью "Диспетчер по восстановлению системы управления базами данных System R".

Глава 1
Вводные определения

Начиная с краткой таксономии идей, мы введем понятия *значения*, *объекта*, *типа*, *процедуры* и *концепции*, которые представляют различные категории идей в компьютере. Затем определим центральное понятие книги — *регулярность*. Для процедур регулярность означает, что процедуры возвращают равные результаты для равных аргументов. Для типов регулярность означает, что типы позволяют воспользоваться оператором проверки на равенство, а также сохраняющим равенство конструированием копии и присваиванием. Регулярность дает нам возможность заменять значения на другие, но равные им значения, для преобразования и оптимизации программ.

1.1. Категории идей: сущность, вид, род

Чтобы объяснить, в чем состоят объекты, типы и другие фундаментальные понятия программирования, полезно дать краткий обзор некоторых категорий идей, которые соответствуют этим понятиям.

Абстрактная сущность — это предмет, который является вечным и неизменным, тогда как *конкретная сущность* — это предмет, который может обрести свое существование в пространстве и времени или перестать существовать. *Атрибут* определяет соответствие между конкретной сущностью и абстрактной сущностью — он описывает некоторое свойство, измерение или качество конкретной сущности. *Идентичность*, одно из примитивных понятий нашего восприятия действительности, определяет одинаковость предмета, изменяющегося в течение времени. Атрибуты конкретной сущности могут изменяться, не затрагивая его идентичность. *Моментальный снимок* конкретной сущности — это полная коллекция его атрибутов в определенный момент времени. Конкретными сущностями являются не только физические объекты, но также и правовые, финансовые или политические сущности. Синий цвет и число 13 — это примеры абстрактных сущностей. Сократ и Соединенные

Штаты Америки — примеры конкретных сущностей. Цвет глаз Сократа и число американских штатов — это примеры атрибутов.

Абстрактный вид описывает общие свойства эквивалентных абстрактных сущностей. Примеры абстрактных видов — натуральное число и цвет. *Конкретные виды* описывают множество атрибутов почти полностью эквивалентных конкретных сущностей. Примеры конкретных видов — человек и штат США.

Функция — это правило, которое связывает одну или несколько абстрактных сущностей, называемых *аргументами*, из соответствующего вида с абстрактной сущностью, называемой *результатом*, из другого вида. Примерами функций являются функция определения преемника, которая связывает каждое натуральное число с непосредственно следующим за ним, и функция, которая связывает с двумя цветами результат их смешивания.

Абстрактный род описывает различные абстрактные виды, которые являются подобными в некотором отношении. Примеры абстрактных родов — число и бинарный оператор. *Конкретный род* описывает различные конкретные виды, подобные в некотором отношении. Примеры конкретных родов — млекопитающее и двуногое.

Сущность принадлежит к одному и только одному виду, который регламентирует правила для ее построения или существования. Сущность может принадлежать к нескольким родам, каждый из которых описывает определенные свойства.

Ниже мы покажем, что объекты и значения представляют сущности, типы представляют виды, а концепции представляют роды.

1.2. Значения

Если мы не знаем интерпретацию, то единственными предметами, которые мы видим в компьютере, являются нули и единицы. *Элемент данных* представляет собой конечную последовательность нулей и единиц.

Тип значения определяет соответствие между видом (абстрактным или конкретным) и множеством элементов данных. Элемент данных, соответствующий определенной сущности, называется *представлением* сущности; сущность называется *интерпретацией* элемента данных. Мы называем элемент данных вместе с его интерпретацией *значением*. Примерами значений являются целые числа, представленные в формате 32-битовых двоичных чисел в дополнительном коде с обратным порядком байтов, и рациональные числа, представленные как конкатенация двух 32-битовых последовательностей, интерпретируемых как целочисленный числитель и знаменатель, которые пред-

ставлены как значения двоичных чисел в дополнительном коде с обратным порядком байтов.

Элемент данных является *полностью сформированным* относительно типа значения, если и только если этот элемент данных представляет абстрактную сущность. Например, каждая последовательность из 32 битов является полностью сформированной, будучи интерпретируемой как целое число в дополнительном двоичном коде; значение NaN (Not a Number — не число) с плавающей запятой по стандарту IEEE 754 не является полностью сформированным, будучи интерпретируемым как вещественное число.

Тип значений является *собственно частичным*, если его значения представляют собственное подмножество абстрактной сущности в соответствующем виде; в противном случае он является *полным*. Например, тип `int` — собственно частичный, тогда как тип `bool` — полный.

Тип значений является *уникально представленным*, если и только если каждой абстрактной сущности соответствует не больше чем одно значение. Например, тип, представляющий истинностное значение как байт, который интерпретирует нулевое значение как ложь и отличное от нуля значение как истину, не является уникально представленным. Тип, представляющий целое число как знаковый бит и величину без знака, не обеспечивает уникальное представление нуля. Тип, представляющий целое число как двоичное число в дополнительном коде, является уникально представленным.

Тип значений является *неоднозначным*, если и только если значение типа имеет больше чем одну интерпретацию. Отрицание неоднозначного является *однозначным*. Например, неоднозначен тип, представляющий календарный год за период дольше чем одно столетие как две десятичные цифры.

Два значения из типа значений являются *равными*, если и только если они представляют одну и ту же абстрактную сущность. Они являются *представительно равными*, если и только если их элементы данных представляют собой идентичные последовательности нулей и единиц.

Лемма 1.1. Если тип значений является уникально представленным, то из равенства следует представительное равенство.

Лемма 1.2. Если тип значений не является неоднозначным, то из представительного равенства следует равенство.

Если тип значений уникально представлен, мы осуществляем проверку на равенство, проверяя, не являются ли обе последовательности нулей и единиц одинаковыми. В противном случае мы должны осуществлять проверку на равенство таким способом, который сохраняет его согласованность с интерпретациями его аргументов. Неуникальные представления выбираются, если проверка на равенство осуществляется менее часто, чем операции, вырабатывающие новые значения, и если есть возможность ускорить выработку новых

значений за счет замедления проверки на равенство. Например, два рациональных числа, представленные как пары целых чисел, равны, если они приводятся к одному к тому же несократимому виду. Два конечных множества, представленные как неотсортированные последовательности, равны, если после сортировки и устранения дубликатов равны их соответствующие элементы.

Иногда осуществление истинной *поведенческой* проверки на равенство становится слишком дорогим или даже невозможным, как в примере с типом кодировок вычислимых функций. В этих случаях мы вынуждены согласиться на более слабую *представительную* проверку на равенство, которая показывает, что два значения представляют собой одну и ту же последовательность нулей и единиц.

Компьютеры *реализуют* функции на абстрактных сущностях как функции на значениях. Безусловно, значения находятся в памяти, но функция на значениях, реализованная должным образом, не зависит от конкретных адресов памяти: она реализует отображение из значений в значения.

Функция, определенная применительно к данному типу значений, является *регулярной*, если и только если она соблюдает проверку на равенство: подстановка равного значения для аргумента дает равный результат. Большинство числовых функций являются регулярными. Примером числовой функции, не являющейся регулярной, служит функция, которая возвращает числитель рационального числа, представленного как пара целых чисел, поскольку $\frac{1}{2} = \frac{2}{4}$, но $\mathrm{numerator}(\frac{1}{2}) \neq \mathrm{numerator}(\frac{2}{4})$. Регулярные функции позволяют провести *рассуждение, основанное на равенствах*: подстановку равных вместо равных.

Нерегулярная функция зависит от представления, а не только от интерпретации ее аргумента. При проектировании представления для типа значений приходится одновременно выполнять две задачи: осуществление проверки на равенство и принятие решения о том, какие функции должны быть регулярными.

1.3. Объекты

Память — это множество слов, каждое из которых имеет *адрес* и *содержание*. Адрес — это значение фиксированного размера, называемого *длиной адреса*. Содержание — это значение другого фиксированного размера, называемого *длиной слова*. Получение содержания по адресу осуществляется в операции *загрузки*. Изменение связи содержания с адресом осуществляется в операции *сохранения*. Примерами реализации памяти являются байты в оперативной памяти и блоки на диске.

Объект — это представление конкретной сущности как значения в памяти. Объект имеет *состояние*, которое представляет собой значение, относящееся

к некоторому типу значений. Состояние объекта является изменяемым. Если некоторый объект соответствует конкретной сущности, то его состояние соответствует моментальному снимку этой сущности. Объект владеет множеством *ресурсов*, таких как слова в памяти или записи в файле, предназначенных для хранения его состояния.

В то время как значение объекта представляет собой непрерывную последовательность нулей и единиц, ресурсы, в которых хранятся эти нули и единицы, не обязательно расположены непрерывно. В этом состоит интерпретация, которая придает целостность объекту. Например, два значения `double` могут интерпретироваться как одно комплексное число, даже если они не являются смежными. Ресурсы объекта могут даже находиться не в одной памяти. Однако в настоящей книге рассматриваются только объекты, находящиеся в единственной памяти с одним адресным пространством. Каждый объект имеет уникальный *начальный адрес*, из которого могут быть достигнуты все его ресурсы.

Тип объектов — это шаблон для хранения и изменения значений в памяти. Каждому типу объектов соответствует тип значений, описывающий состояния объектов этого типа. Каждый объект принадлежит к некоторому типу объектов. Примером типа объектов могут служить целые числа, представленные в формате 32-битового двоичного числа в дополнительном коде с прямым порядком байтов, выровненные к 4-байтовой границе адреса.

Значения и объекты играют взаимно дополняющие роли. Значения являются неизменными и независимыми от любой конкретной реализации в компьютере. Объекты являются изменяемыми и имеют реализации, зависимые от компьютера. Состояние объекта в любой точке во времени может быть описано значением, которое можно в принципе записать на бумаге (создание моментального снимка) или *сериализовать* (преобразовать в последовательную форму) и отправить по линии связи. Описание состояний объектов в терминах значений позволяет нам абстрагироваться от определенных реализаций объектов, когда речь идет о равенстве. Функциональное программирование имеет дело со значениями, а императивное — с объектами.

Мы используем значения для представления сущностей. Так как значения являются неизменными, они могут представить абстрактную сущность. Последовательности значений могут также представлять последовательности моментальных снимков конкретных сущностей. Объекты хранят значения, представляющие сущности. Так как объекты являются изменяемыми, они могут представлять конкретные сущности, принимая новое значение для представления изменения в сущности. Объекты могут также представить абстрактную сущность, оставаясь постоянными или принимая различные приближения к абстрактному.

Мы используем объекты в компьютере по трем причинам.

1. Объекты моделируют изменяемые конкретные сущности, такие как карточки служащих в приложении учета заработной платы.
2. Объекты обеспечивают удобный способ реализации функции на значениях, таких как процедура, извлекающая квадратный корень числа с плавающей запятой с использованием итеративного алгоритма.
3. Компьютеры с памятью составляют единственную доступную реализацию универсального вычислительного устройства.

Некоторые свойства типов значений переносятся на типы объектов. Объект является *полностью сформированным*, если и только если полностью сформировано его состояние. Тип объектов является *собственно частичным*, если и только если его тип значений собственно частичен; в противном случае он является *полным*. Тип объектов является *уникально представленным*, если и только если уникально представлен его тип значений.

Так как конкретные сущности имеют идентичности, представляющие их объекты нуждаются в соответствующем понятии идентичности. *Маркер идентичности* — это уникальное значение, выражающее идентичность объекта, которое вычисляется из значения объекта и адреса его ресурсов. Примерами маркеров идентичности могут служит адрес объекта, индекс в массиве, в котором хранится объект, и номер служащего в картотеке персонала. Проверка маркеров идентичности на равенство соответствует проверке на идентичность. В течение времени существования приложения каждый конкретный объект может использовать различные маркеры идентичности в ходе перемещения либо в пределах структуры данных, либо из одной структуры данных в другую.

Два объекта одного и того же типа *равны*, если и только если равны их состояния. Если два объекта равны, мы говорим, что каждый из них является *копией* другого. Внесение изменения в объект не затрагивает ни одной его копии.

В настоящей книге используется язык программирования, в котором отсутствует какой-либо способ описания значений и типов значений как отдельных от объектов и типов объектов. Поэтому с данного момента будем предполагать, что если речь идет о типах, без уточнения, то подразумеваются типы объектов.

1.4. Процедуры

Процедура — это последовательность команд, которая изменяет состояние некоторых объектов; процедура может также создавать или уничтожать объекты.