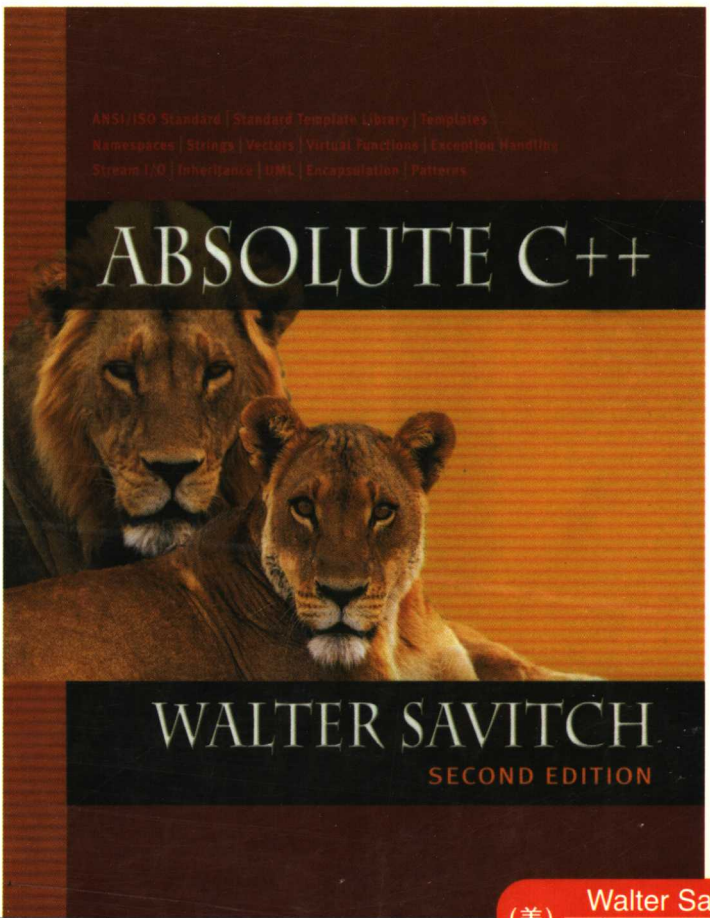
经 典 原 版 书 库

PEARSON Addison Wesley

Absolute C++

（英文版 · 第2版）



(美) Walter Savitch 著
加州大学圣迭戈分校

机械工业出版社
China Machine Press

经 典 原 版 书 库

Absolute C++

（英文版 · 第2版）

(Second Edition)

(美) Walter Savitch
加州大学圣迭戈分校 著

机械工业出版社
China Machine Press

本书版权登记号：图字：01-2006-1933

图书在版编目（CIP）数据

Absolute C++（英文版·第2版）/（美）萨维弛（Savitch, W.）著. －北京：机械工业出版社，2006.4

（经典原版书库）

书名原文：Absolute C++, Second Edition

ISBN 7-111-18829-2

Ⅰ. A…　Ⅱ. 萨…　Ⅲ. C语言－程序设计－英文　Ⅳ. TP312

中国版本图书馆CIP数据核字（2006）第029827号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）

责任编辑：迟振春

北京牛山世兴印刷厂印刷·新华书店北京发行所发行

2006年4月第1版第1次印刷

880mm×1230mm1/32·30.625印张

定价：56.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，

为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”；同时，引进全美通行的教学辅导书“Schaum's Outlines”系列组成“全美经典学习指导系列”。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业的教学度身订造的。其中许多教材均已为M. I. T., Stanford, U.C. Berkeley, C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

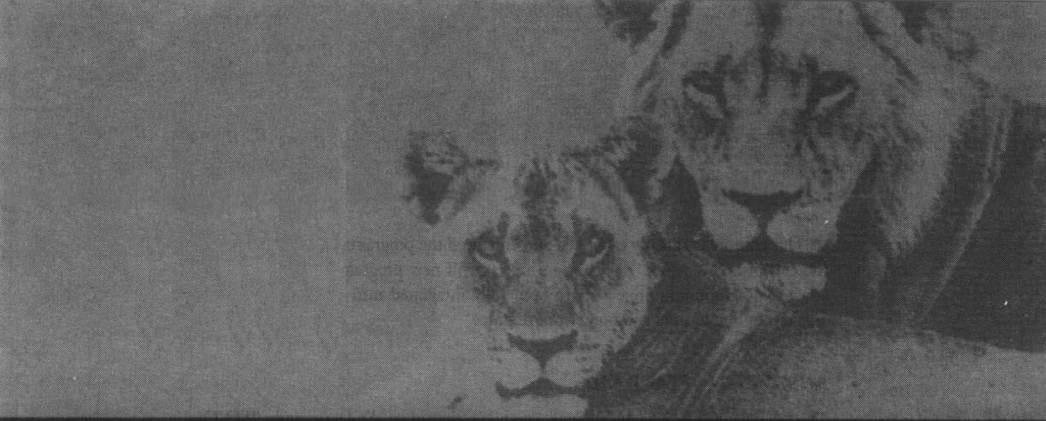
权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzjsj@hzbook.com
联系电话：(010) 68995264
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周克定	周傲英	孟小峰	岳丽华	范　明
郑国梁	施伯乐	钟玉琢	唐世渭	袁崇义
高传善	梅　宏	程　旭	程时端	谢希仁
裘宗燕	戴　葵			



Preface

This book is designed to be a textbook and reference for programming in the C++ language. Although it does include programming techniques, it is organized around the features of the C++ language, rather than any particular curriculum of programming techniques. The main audience I had in mind were undergraduate students who had not had extensive programming experience with the C++ language. As such the book would be a suitable C++ text or reference for a wide range of users. The beginning chapters are written at a level that is accessible to beginners, while the boxed sections of those chapters serve to introduce more experienced programmers to basic C++ syntax. Later chapters are also accessible, but are written at a level suitable for students who have progressed to these more advanced topics. (For those who want a textbook with more pedagogical material and more on very basic programming technique, try Problem Solving with C++: The object of Programming, *Fifth Edition*, Addison-Wesley.) *Absolute C++* is also suitable for anyone learning the C++ language on their own.

The C++ coverage in this book goes well beyond what a beginner needs to know. In particular, it has extensive coverage of inheritance, polymorphism, exception handling, and the Standard Template Library (STL), as well as basic coverage of patterns and the unified modeling language (UML).

CHANGES IN THIS EDITION

This second edition presents the same topics in the same order as the first edition. If you are an instructor already using the first edition, you can continue to teach your course without change. This edition has been reworked to make the language and the code clearer, but the content is the same.

This edition has greatly expanded and enhanced the programming projects given at the end of each chapter. This edition has over 50 new programming projects. Also, selected programming projects have been fully integrated into CodeMate, Addison-Wesley's online tutorial and homework resource.

ANSI/ISO C++ STANDARD

This edition is fully compatible with compilers that meet the latest ANSI/ISO C++ standard.

STANDARD TEMPLATE LIBRARY

The Standard Template Library (STL) is an extensive collection of preprogrammed data structure classes and algorithms. The STL is perhaps as big a topic as the core C++ language, so I have included a substantial introduction to STL. There is a full chapter on the general topic of templates and a full chapter on the particulars of STL, as well as other material on, or related to, STL at other points in the text.

OBJECT-ORIENTED PROGRAMMING

This book is organized around the structure of C++. As such, the, early chapters cover aspects of C++ that are common to most high-level programming languages but are not particularly oriented toward object-oriented programming (OOP) programming. For a reference book, and for a book for learning a second language, this makes sense. However, I consider C++ to be an OOP language. If you are programming in C++ and not C, you must be using the OOP features of C++. This text offers extensive coverage of encapsulation, inheritance, and polymorphism as realized in the C++ language. The final chapter, on patterns and UML, gives additional coverage of OOP-related material.

FLEXIBILITY IN TOPIC ORDERING

This book allows instructors wide latitude in reordering the material. This is important if a book is to serve as a reference. This is also in keeping with my philosophy of accommodateing the instructor's style, rather than tieing the instructor to my own personal preference of topic ordering. Each chapter introduction explains what material must already have been covered before each section of the chapter can be covered.

ACCESSIBLE TO STUDENTS

It is not enough for a book to present the right topics in the right order. It is not even enough for it be clear and correct. The material also needs to be presented in a way that is accessible to the novice. Like my other textbooks, which proved to be very popular with students, this book was written to be friendly and accessible to the student.

SUMMARY BOXES

Each major point is summarized in a boxed section. These boxed sections are spread throughout each chapter. They serve as summaries of the material, as a quick reference source, and as a quick way to learn the C++ syntax for a feature you know about in general but for which you do not know the C++ particulars.

SELF-TEST EXERCISES

Each chapter contains numerous self-test exercises. Complete answers for all the self-test exercises are given at the end of each chapter.

OTHER FEATURES

Pitfall sections, programming technique sections, and examples of complete programs with sample input and output are given throughout each chapter. Each chapter ends with a summary and a collection of programming projects.

CODEMATE ONLINE TUTORIAL RESOURCE

CodeMate is an online resource that provides tutorial help and evaluation of student work on programming projects. The code displays and selected programming projects in this edition have been fully integrated into CodeMate. Using CodeMate, a student can get hints on programming projects, write and compile the project, and receive feedback on how to address compiler errors messages, and all this can be done over the Internet from any computer with Internet access. Instructors can track each student's progress in the course's programming projects. A complimentary subscription is offered when an access code is bundled with a new copy of this text. Subscriptions may also be purchased online. For more information on CodeMate, go to

```
http://www.aw-bc.com/codemate
```

SUPPORT MATERIAL

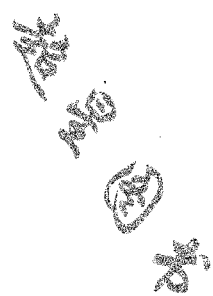
Support material is available to all users of this book; additional material is available to qualified instructors.

MATERIALS AVAILABLE TO ALL USERS

- Self-check quizzes
- Source code from the book
- PowerPoint slides

To access these student support materials, go to

```
http://www.aw-bc.com/savitch
```



RESOURCES AVAILABLE TO QUALIFIED INSTRUCTORS

The following supplements are available to qualified instructors. Please contact your local Addison-Wesley sales representative or send an e-mail to aw.cse@aw.com for information on how to access the instructor supplements.

- Instructor access to Addison-Wesley's CodeMate
- Instructor's Resource Guide—including chapter-by-chapter teaching hints, quiz questions with solutions, and solutions to many programming projects
- Test bank and test generator
- PowerPoint lectures, including programs and art from the text

E-MAIL CONTACT

I would very much like to hear your comments so that I can continue to improve this book and make it better suit your needs. Please send your comments to

```
wsavitch@ucsd.edu
```

I want to know how you like the book and I want suggestions for changes, but unfortunately I am not able to provide students with an e-mail consulting or tutoring service. My volume of e-mail has become too large for this. In particular, I cannot provide solutions to exercises in this book, or to other exercises provided by your instructor. At least as a partial consolation to those who desire such help, this book does include complete answers to all of the self-test exercises. The instructor's guide provides some answers to the programming projects, but that material is only available to instructors who adopt the book, and it cannot be given out to students.

ACKNOWLEDGMENTS

Numerous individuals have contributed invaluable help and support to making this book happen. Frank Ruggirello and Susan Hartman at Addison-Wesley first conceived the idea for this book and supported the first edition; for which I owe them a debt of gratitude, along with Matt Goldstein, the editor of second edition. I also want to thank Michelle Brown, Katherine Harutunian, Joyce Wells, and all the other people at Addison-Wesley for their wonderful support and encouragement.

A special thanks to Patty Mahtani at Addison-Wesley for bringing the book through production, for her wonderful support and encouragement, and for being Patty. Thanks also to Daniel Rausch and Meghan James and the great people at Argosy Publishing for their superb work on the typesetting and production of this book.

Kenrick Mock receives special thanks for writing the superb new programming projects and for updating the instructor's guide. David Teague deserves special acknowledgment for his careful reviewing and researching for the first edition of this book. I thank my good friend Mario Lopez for the many helpful conversations we had about C++.

The following reviewers provided suggestions for this edition. I thank them all for their hard work and helpful comments. Victoria Rayskin, University of Central Los Angeles; Jerry K. Bilbrey, Jr, Francis Marion University; Albert M. K. Cheng, University of Houston; Tim Lin, California Polytechnical Institute of Pomona; Ron DiNapoli, Cornell University; R. M. Lowe, Clemson University; Martin Dulberg, North Carolina State University; and Jeffrey L. Popyack, Drexel University.

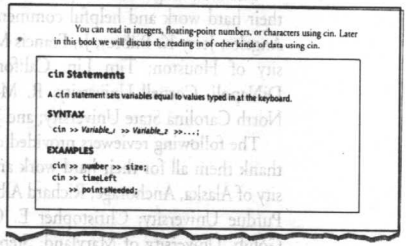
The following reviewers provided corrections and suggestions to the first edition. I thank them all for their hard work and helpful commments. Kenrick Mock, University of Alaska, Anchorage; Richard Albright, University of Delaware; H. E. Dunsmore, Purdue University; Christopher E. Cramer; Drue Coles, Boston University; Evan Golub, University of Maryland; Stephen Corbesero, Moravian College; Fredrick H. Colclough, Colorado Technical University; Joel Weinstein, Northeastern University; Stephen P. Leach, Florida State University; Alvin S. Lim, Auburn University; and Martin Dulberg, North Carolina State University.

W.S.
http://www-cse.ucsd.edu/users/savitch/
wsavitch@ucsd.edu

Feature Walkthrough

Summary Boxes

These boxes provide a brief synopsis of major points in each chapter, both highlighting and reinforcing core concepts throughout the book. Readers will find them to be a handy, quick reference for C++ syntax and features.



You can read in integers, floating-point numbers, or characters using cin. Later in this book we will discuss the reading in of other kinds of data using cin.

cin Statements

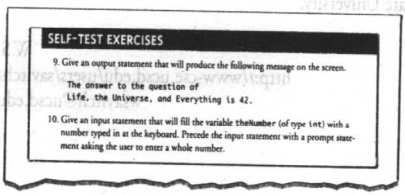
A cin statement sets variables equal to values typed in at the keyboard.

SYNTAX

```
cin >> Variable_1 >> Variable_2 >>...;
```

EXAMPLES

```
cin >> number >> size;
cin >> timeLeft
    >> pointsNeeded;
```



SELF-TEST EXERCISES

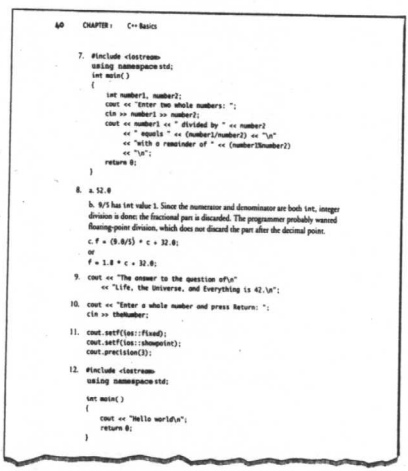
9. Give an output statement that will produce the following message on the screen.

```
The answer to the question of
Life, the Universe, and Everything is 42.
```

10. Give an input statement that will fill the variable `theNumber` (of type `int`) with a number typed in at the keyboard. Precede the input statement with a prompt statement asking the user to enter a whole number.

Self-Test Exercises and Answers

Strategically placed within each chapter, Self-Test Exercises offer readers an opportunity to assess their mastery of key topics.



40 CHAPTER 1: C++ Basics

7.
```
#include <iostream>
using namespace std;
int main( )
{
    int number1, number2;
    cout << "Enter two whole numbers: ";
    cin >> number1 >> number2;
    cout << number1 << " divided by " << number2
         << " equals " << (number1/number2) << "\n"
         << "with a remainder of " << (number1%number2)
         << "\n";
    return 0;
}
```

8. a. 52.0

b. 9/5 has int value 1. Since the numerator and denominator are both int, integer division is done: the fractional part is discarded. The programmer probably wanted floating-point division, which does not discard the part after the decimal point.

c.
```
f = (9.0/5) * c + 32.0;
```
or
```
f = 1.8 * c + 32.0;
```

9.
```
cout << "The answer to the question of\n"
     << "Life, the Universe, and Everything is 42.\n";
```

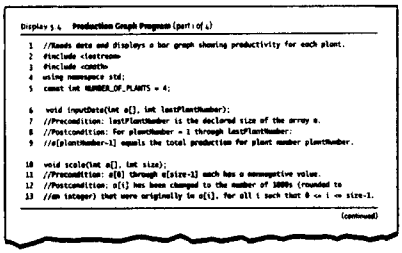
10.
```
cout << "Enter a whole number and press Return: ";
cin >> theNumber;
```

11.
```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(3);
```

12.
```
#include <iostream>
using namespace std;

int main( )
{
    cout << "Hello world\n";
    return 0;
}
```

Detailed answers are provided at the end of the chapter.



Display 5.4 Production Graph Program (part 1 of 4)

```
1  //Reads data and displays a bar graph showing productivity for each plant.
2  #include <iostream>
3  #include <cmath>
4  using namespace std;
5  const int NUMBER_OF_PLANTS = 4;

6  void inputData(int a[], int lastPlantNumber);
7  //Precondition: lastPlantNumber is the declared size of the array a.
8  //Postcondition: For plantNumber = 1 through lastPlantNumber:
9  //a[plantNumber-1] equals the total production for plant number plantNumber.

10 void scale(int a[], int size);
11 //Precondition: a[0] through a[size-1] each has a nonnegative value.
12 //Postcondition: a[i] has been changed to the number of 1000s (rounded to
13 //an integer) that were originally in a[i], for all i such that 0 <= i <= size-1.
```

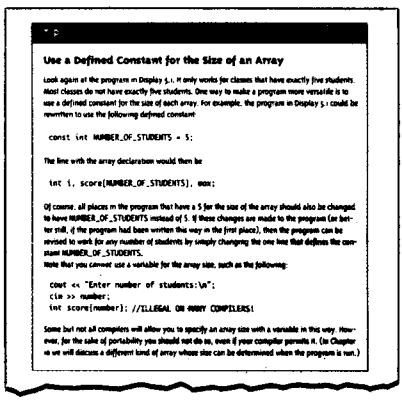
(continued)

Code Displays

There are abundant code listings throughout the text. Informal comments that explain potentially confusing or difficult portions appear alongside the code.

Tips

These helpful hints instruct readers on best programming practices. The author explains the rationale behind these practices and includes suggestions on how to execute them effectively.



Use a Defined Constant for the Size of an Array

Look again at the program in Display 5.1. It only works for classes that have exactly five students. Most classes do not have exactly five students. One way to make a program more versatile is to use a defined constant for the size of each array. For example, the program in Display 5.1 could be rewritten to use the following defined constant

```
const int NUMBER_OF_STUDENTS = 5;
```

The line with the array declaration would then be

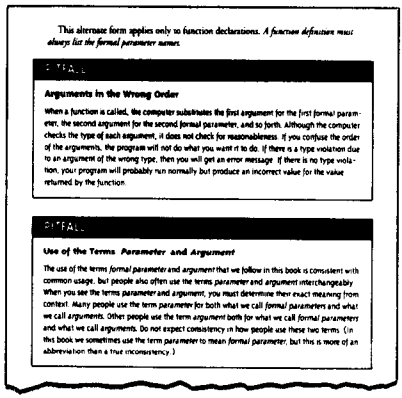
```
int i, score[NUMBER_OF_STUDENTS], max;
```

Of course, all places in the program that have a 5 for the size of the array should also be changed to have NUMBER_OF_STUDENTS instead of 5. If these changes are made to the program (or better still, if the program had been written this way in the first place), then the program can be revised to work for any number of students by simply changing the one line that defines the constant NUMBER_OF_STUDENTS.

Note that you cannot use a variable for the array size, such as the following:

```
cout << "Enter number of students:\n";
cin >> number;
int score[number]; //ILLEGAL OR HARMY COMPILERS!
```

Some but not all compilers will allow you to specify an array size with a variable in this way. However, for the sake of portability you should not do so, even if your compiler permits it. (In Chapter 10 we will discuss a different kind of array whose size can be determined when the program is run.)



This alternate form applies only to function declarations. *A function definition must always list the formal parameter names.*

PITFALL

Arguments in the Wrong Order

When a function is called, the computer substitutes the first argument for the first formal parameter, the second argument for the second formal parameter, and so forth. Although the computer checks the type of each argument, it does not check for reasonableness. If you confuse the order of the arguments, the program will not do what you want it to do. If there is a type violation due to an argument of the wrong type, then you will get an error message. If there is no type violation, your program will probably run normally but produce an incorrect value for the value returned by the function.

PITFALL

Use of the Terms *Parameter* and *Argument*

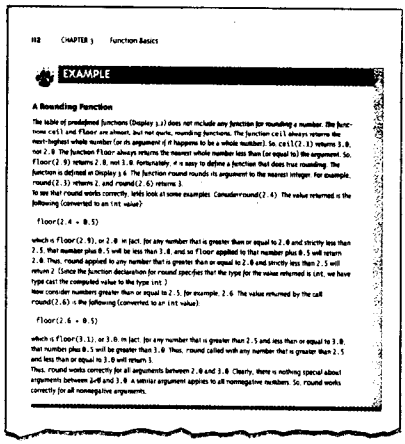
The use of the terms *formal parameter* and *argument* that we follow in this book is consistent with common usage, but people also often use the terms *parameter* and *argument* interchangeably. When you see the terms *parameter* and *argument*, you must determine their exact meaning from context. Many people use the term *parameter* for both what we call *formal parameters* and what we call *arguments*. Other people use the term *argument* both for what we call *formal parameters* and what we call *arguments*. Do not expect consistency in how people use these two terms. (In this book we sometimes use the term *parameter* to mean *formal parameter*, but this is more of an abbreviation than a true inconsistency.)

Pitfalls

These sections warn of common mistakes that can trip up beginning programmers and offer advice on how to avoid them.

Examples

These sections usually feature a complete program that solves a specific problem. The code examples are lengthier than in the standard code displays and highlight useful features of C++.



112 CHAPTER 3 Function Basics

EXAMPLE

A Rounding Function

The table of predefined functions (Display 3.1) does not include any function for rounding a number. The functions `ceil` and `floor` are almost, but not quite, rounding functions. The function `ceil` always returns the next-highest whole number for its argument if it happens to be a whole number). So `ceil(2.1)` returns `3.0`, not `2.0`. The function `floor` always returns the nearest whole number less than (or equal to) the argument. So, `floor(2.9)` returns `2.0`, not `3.0`. Fortunately, it is easy to define a function that does true rounding. The function is defined in Display 3.6. The function `round` rounds its argument to the nearest integer. For example, `round(2.3)` returns `2`, and `round(2.6)` returns `3`.

To see that `round` works correctly, let's look at some examples. Consider `round(2.4)`. The value returned is the following (converted to an `int` value):

```
floor(2.4 + 0.5)
```

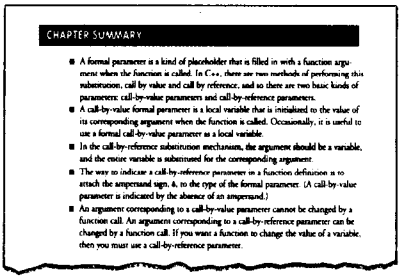
which is `floor(2.9)`, or `2.0` in fact. For any number that is greater than or equal to `2.0` and strictly less than `2.5`, that number plus `0.5` will be less than `3.0`, and so `floor` applied to that number plus `0.5` will return `2.0`. Thus, `round` applied to any number that is greater than or equal to `2.0` and strictly less than `2.5` will return `2`. (Since the function declaration for `round` specifies that the type for the value returned is `int`, we have type cast the computed value to the type `int`.)

Now consider numbers greater than or equal to `2.5`, for example, `2.6`. The value returned by the call `round(2.6)` is the following (converted to an `int` value):

```
floor(2.6 + 0.5)
```

which is `floor(3.1)`, or `3.0` in fact. For any number that is greater than `2.5` and less than or equal to `3.0`, that number plus `0.5` will be greater than `3.0`. Thus, `round` called with any number that is greater than `2.5` and less than or equal to `3.0` will return `3`.

Thus, `round` works correctly for all arguments between `2.0` and `3.0`. Clearly, there is nothing special about arguments between `2.0` and `3.0`. A similar argument applies to all nonnegative numbers. So, `round` works correctly for all nonnegative arguments.



CHAPTER SUMMARY

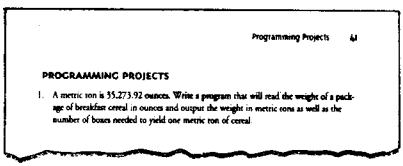
- A formal parameter is a kind of placeholder that is filled in with a function argument when the function is called. In C++, there are two methods of performing this substitution, call by value and call by reference, and so there are two basic kinds of parameters: call-by-value parameters and call-by-reference parameters.
- A call-by-value formal parameter is a local variable that is initialized to the value of its corresponding argument when the function is called. Occasionally, it is useful to use a formal call-by-value parameter as a local variable.
- In the call-by-reference substitution mechanism, the argument should be a variable, and the entire variable is substituted for the corresponding argument.
- The way to indicate a call-by-reference parameter in a function definition is to attach the ampersand sign, &, to the type of the formal parameter. (A call-by-value parameter is indicated by the absence of an ampersand.)
- An argument corresponding to a call-by-value parameter cannot be changed by a function call. An argument corresponding to a call-by-reference parameter can be changed by a function call. If you want a function to change the value of a variable, then you must use a call-by-reference parameter.

Chapter Summaries

This end-of-chapter tool provides a concise overview of the fundamental concepts presented in the chapter.

Programming Projects

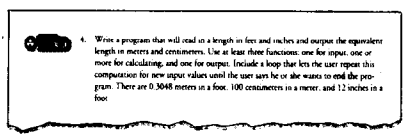
Found at the end of each chapter, Programming Projects challenge readers to design and implement a C++ program to solve a problem. Example solutions for all programming projects are available to instructors.



Programming Projects 41

PROGRAMMING PROJECTS

1. A metric ton is 35,273.92 ounces. Write a program that will read the weight of a package of breakfast cereal in ounces and output the weight in metric tons as well as the number of boxes needed to yield one metric ton of cereal.

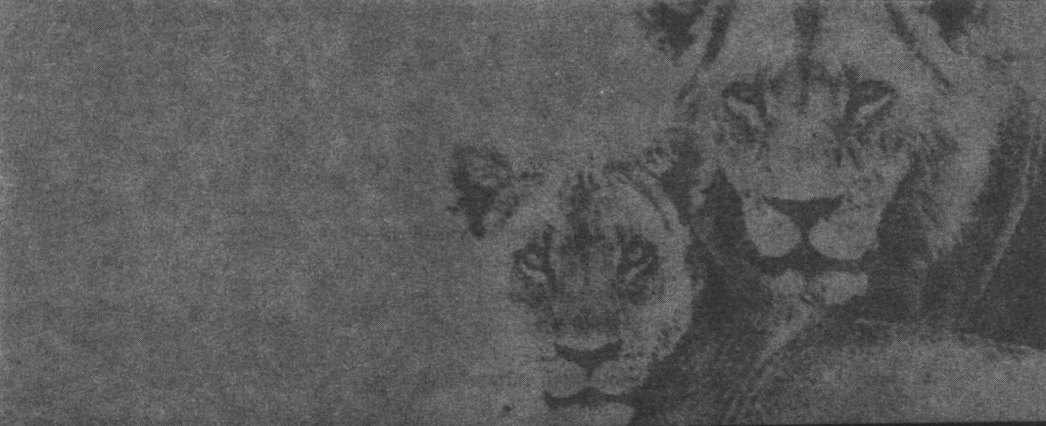


4. Write a program that will read in a length in feet and inches and output the equivalent length in meters and centimeters. Use at least three functions: one for input, one or more for calculating, and one for output. Include a loop that lets the user repeat this computation for new input values until the user says he or she wants to end the program. There are 0.3048 meters in a foot, 100 centimeters in a meter, and 12 inches in a foot.

CodeMate

CodeMate brings end-of-chapter programming projects to life. Working online, students can view, compile, run, and edit select programming problems as well as all code listings from the textbook. Best of all, CodeMate's tutorial feedback helps students work through common programming errors, improving their programming skills. An automated gradebook allows instructors to assign CodeMate problems and track student progress online.

Brief Contents



Contents