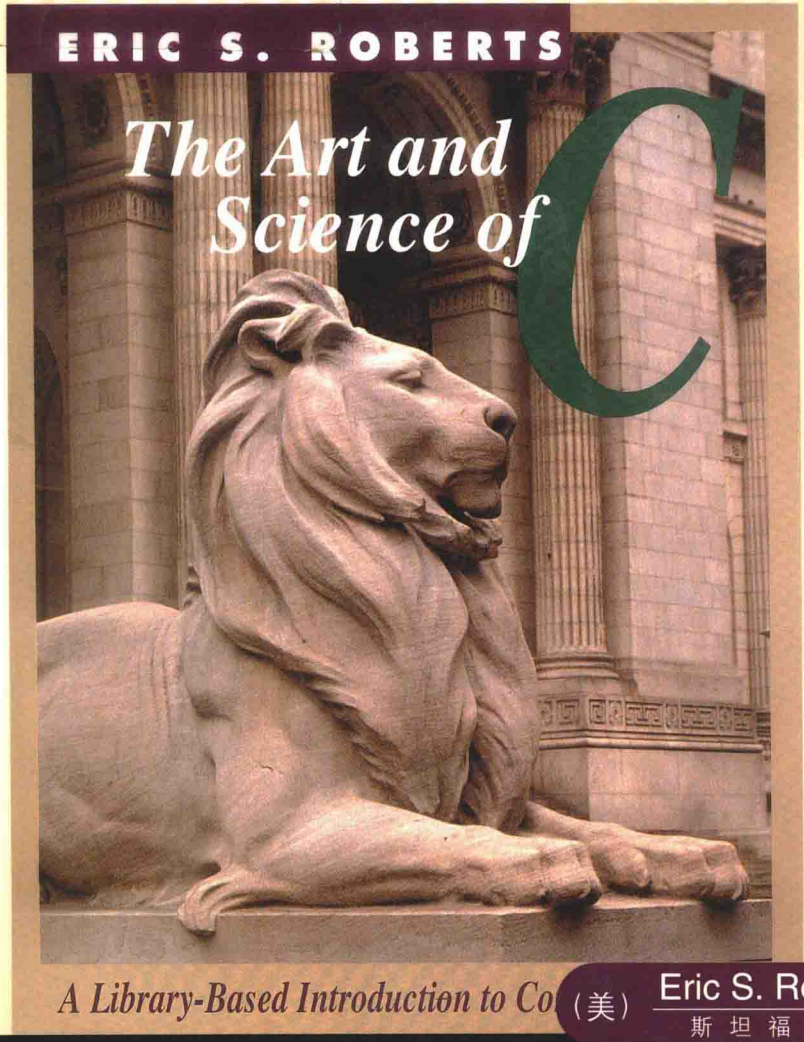
HZ BOOKS

经 典 原 版 书 库

PEARSON Addison Wesley

# C语言的科学和艺术

（英文版）




（美） Eric S. Roberts 著
斯坦福大学

机械工业出版社
China Machine Press

# C语言的科学和艺术 （英文版）

# The Art and Science of C
## A Library-Based Introduction to Computer Science

本书是一本计算机科学的经典教材，强调软件工程和优秀的程序设计风格。此外，学生还可以从书中学习到 ANSI C 的基础知识，这些内容已经成为计算行业的标准。作者的写作风格使得书中深奥的概念变得易于理解和引人入胜。

本书集中讨论库和抽象的用法，这是当代程序设计技术中最基本的知识。作者使用库来隐藏C语言的复杂性，更加突出主题，使学生可以较好地掌握每一个主题的精髓。然后，进一步给出每个库的底层实现，较好地展示了库自身的抽象威力。

本书从基础开始讲起，是C语言的入门教材。本书已经被美国斯坦福大学、哥伦比亚大学等多所大学和学院成功采用，是一本适合高等院校计算机及相关专业使用的优秀教材。

### 作者简介

**Eric S. Roberts** 是美国斯坦福大学计算机科学系教授，并担任系里主管教学事务的副主任，同时他还是工学院的 Charles Simonyi 讲席教授。他于1980年在哈佛大学应用数学系获得博士学位，并曾在DEC公司位于加州 Palo Alto 的系统研究中心做过5年的研究工作。作为一位获得高度评价的教育工作者，Roberts 还因其在本科生教学中的杰出贡献获得了1993年的Bing Award奖。他的另一本书《C程序设计的抽象思维》(Programming Abstractions in C: A Second Course in Computer Science) 的英文影印版即将由机械工业出版社引进出版。


PEARSON Addison Wesley
www.PearsonEd.com

ISBN 7-111-13991-7

封面设计 陈子平



网上购书：www.china-pub.com

北京市西城区百万庄南街1号 100037
读者服务热线：(010) 68995259 68995264
读者服务信箱：hzedu@hzbook.com
http://www.hzbook.com

PEARSON Education

ISBN 7-111-13991-7/TP · 3473
定价：60.00元

HZ BOOKS

PEARSON Addison Wesley

# C语言的科学和艺术

## The Art and Science of C

A Library-Based Introduction to Computer Science

(美) Eric S. Roberts 著

英文版

机械工业出版社
China Machine Press

经 典 原 版 书 库

# C语言的科学和艺术

（英文版）

# The Art and Science of C

## A Library-Based Introduction to Computer Science

（美）Eric S. Roberts 著
斯坦福大学

机械工业出版社
China Machine Press

**本书版权登记号：图字：01-2004-1204**

**图书在版编目（CIP）数据**

C语言的科学和艺术（英文版）/（美）罗伯茨（Roberts, E. S.）著. -北京：机械工业出版社，2004.4

（经典原版书库）

书名原文：The Art and Science of C: A Library-Based Introduction to Computer Science

ISBN 7-111-13991-7

Ⅰ.C… Ⅱ.罗… Ⅲ.C语言－程序设计－英文 Ⅳ.TP312

中国版本图书馆CIP数据核字（2004）第009771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

北京诚信伟业印刷有限公司印刷·新华书店北京发行所发行

2005年8月第1版第3次印刷

787mm × 1092mm 1/16 · 46印张

印数：5 001-7 000册

定价：60.00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，有的还专诚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：hzedu@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）

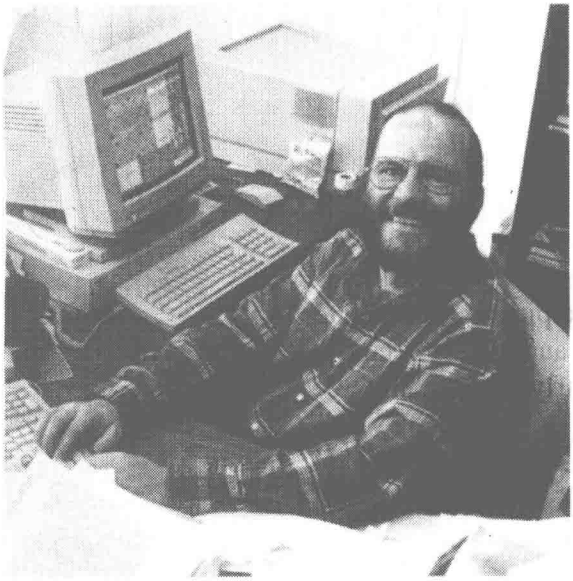
| | | | | |
|---|---|---|---|---|
| 尤晋元 | 王　珊 | 冯博琴 | 史忠植 | 史美林 |
| 石教英 | 吕　建 | 孙玉芳 | 吴世忠 | 吴时霖 |
| 张立昂 | 李伟琴 | 李师贤 | 李建中 | 杨冬青 |
| 邵维忠 | 陆丽娜 | 陆鑫达 | 陈向群 | 周伯生 |
| 周立柱 | 周克定 | 周傲英 | 孟小峰 | 岳丽华 |
| 范　明 | 郑国梁 | 施伯乐 | 钟玉琢 | 唐世渭 |
| 袁崇义 | 高传善 | 梅　宏 | 程　旭 | 程时端 |
| 谢希仁 | 裘宗燕 | 戴　葵 | | |

# 秘　书　组

武卫东　　温莉芳　　刘　江　　杨海玲

In loving memory of Grace E. Bloom
(1924–1994) for helping me appreciate
the value of ideas and the importance
of writing clearly about them.

# *About the Author*



I first began teaching introductory computer science more than 20 years ago while I was still a student at Harvard. Since receiving my Ph.D. in 1980, I have taught computer science at Harvard, Wellesley, and Stanford, where I am Associate Chair of the Computer Science Department. In that capacity, I am responsible for the undergraduate program in computer science. Although I have taught advanced courses in computer science and have also worked in the research industry, my greatest joy comes from opening up the enormous power of computers to students who are just beginning to learn about them. In their excitement, my own love for computer science is constantly renewed.

In addition to my teaching at Stanford, I have served since 1990 as the president of Computer Professionals for Social Responsibility, a public-interest association of computer professionals with 2000 members in 22 chapters throughout the United States. Computers affect our society in many different ways. Just as it is important to learn about the technology, it is critical that we also take the responsibility to ensure that computers are used for the benefit of all. If you have suggestions as to how I might make the presentation more clear, or you encounter errors in this text, please let me know. You can reach me by electronic mail at ericr@aw.com.

Eric S. Roberts
Department of Computer Science
Stanford University

# *To the Student*

Welcome! By picking up this book, you have taken a step into the world of computer science—a field of study that has grown from almost nothing half a century ago to become one of the most vibrant and active disciplines of our time.

Over that time, the computer has opened up extraordinary possibilities in almost every area of human endeavor. Business leaders today are able to manage global enterprises on an unprecedented scale because computers enable them to transfer information anywhere in a fraction of a second. Scientists can now solve problems that were beyond their reach until the computer made the necessary calculations possible. Filmmakers use computers to generate dramatic visual effects that are impossible to achieve without them. Doctors can determine much more accurately what is going on inside a patient because computers have enabled a massive transformation in the practice of medicine.

Computers are a profoundly empowering technology. The advances we have seen up to now are small compared to what we will see in the next century. Computers will play a major role in shaping that century, just as they have the last 50 years. Those of you who are students today will soon inherit the responsibility of guiding that progress. As you do so, knowing how to use computers can only help.

Like most skills that are worth knowing, learning how computers work and how to control their enormous power takes time. You will not understand it all at once. But you must start somewhere. Twenty-five centuries ago, the Chinese philosopher Lao-tzu observed that the longest journey begins with a single step. This book can be your beginning.

For many of you, however, the first step can be the hardest to take. Many students find computers overwhelming and imagine that computer science is beyond their reach. Learning the basics of programming, however, does not require advanced mathematics or a detailed understanding of electronics. What matters in programming is whether you can progress from the statement of a problem to its solution. To do so, you must be able to think logically. You must have the necessary discipline to express your logic in a form that the computer can understand. Perhaps most importantly, you must be able to see the task through to its completion without getting discouraged by difficulties and setbacks. If you stick with the process, you will discover that reaching the solution is so exhilarating that it more than makes up for any frustrations you encounter along the way.

This book is designed to teach you the fundamentals of programming and the basics of C, which is the dominant programming language in the computing industry today. It treats the *whys* of programming as well as the *hows,* to give you a feel for the programming process as a whole. It also includes several features that will help you focus on the essential points and avoid errors that slow you down. The next few pages summarize these features and explain how to use this book effectively as you begin your journey into the exciting world of computer science.

## For Chapter Review

Each chapter includes easily accessible material to guide your study and facilitate review of the central topics.

CHAPTER 4

# *Statement Forms*

The statements was interesting but tough.

— Mark Twain, *Adventures of Huckleberry Finn*, 1884

### *Objectives*

- To understand the relationship between statements and expressions.
- To recognize that the equal sign used for assignment is treated as a binary operator in C.
- To understand that statements can be collected into blocks.
- To recognize that control statements fall into two classes: conditional and iterative.
- To learn how to manipulate Boolean data and to appreciate its importance.
- To increase your familiarity with the relational operators: `=`, `!=`, `<`, `<=`, `>`, and `>=`.
- To understand the behavior of the `&&`, `||`, and `!` operators.
- To master the details of the `if`, `switch`, `while`, and `for` statements.



The list of **objectives** previews the key topics covered by the chapter. Because each objective identifies a concrete skill, the chapter objectives help you to assess your mastery of the essential material.

To make the best possible use of this textbook for learning the C language, be sure to take advantage of the tools it provides.



## Summary

In Chapter 3, you looked at the process of programming from a holistic perspective that emphasized problem solving. Along the way, you learned about several control statements in an informal way. In this chapter, you were able to investigate how those statements work in more detail. You were also introduced to a new type of data called *Boolean data*. Although this data type contains only two values—`TRUE` and `FALSE`—being able to use Boolean data effectively is extremely important to successful programming and is well worth a little extra practice.

This chapter also introduced several new operators, and at this point it is helpful to review the precedence relationships for all the operators you have seen so far. That information is summarized in Table 4-1 the operators are listed from highest to lowest precedence.

The important points introduced in this chapter include:

- *Simple statements* consist of an expression followed by a semicolon.
- The = used to specify assignment is an operator in C. Assignments are therefore legal expressions, which makes it possible to write *embedded* and *multiple assignments*.
- Individual statements can be collected into *compound statements*, more commonly called *blocks*.
- Control statements fall into two classes: *conditional* and *iterative*.
- The `genlib` library defines a data type called `bool` that is used to represent Boolean data. The type `bool` has only two values: `TRUE` and `FALSE`.
- You can generate Boolean values using the *relational operators* (`<`, `<=`, `>`, `>=`, `==`, and `!=`) and combine them using the *logical operators* (`&&`, `||`, and `!`).
- The logical operators `&&` and `||` are evaluated in left-to-right order in such a way that the evaluation stops as soon as the program can determine the result. This behavior is called *short-circuit evaluation*.

| Operator | Associativity |
|---|---|
| *unary* - ++ -- ! (*type cast*) | right-to-left |
| * / % | left-to-right |
| + - | left-to-right |
| < <= > >= | left-to-right |
| == != | left-to-right |
| && | left-to-right |
| \|\| | left-to-right |
| ?: | right-to-left |
| = *op*= | right-to-left |

TABLE 4-1 Precedence table for operators used through Chapter 4

The **Summary** describes, in more detail, what you should have learned in connection with the Objectives

# Learning to Program

Programming is both a science and an art. Learning to program well requires much more than memorizing a set of rules. You must learn through experience and by reading other programs. This text includes several features to aid in this process.

The final character in the string is a special character called *newline*, indicated by the sequence `\n`. When the `printf` function reaches the period at the end of the sentence, the cursor is sitting at the end of the text, just after the period. If you wanted to extend this program so that it wrote out more messages, you would probably want to start each new message on a new screen line. The **newline** character, defined for all modern computer systems, makes this possible. When the `printf` function processes the newline character, the cursor on the screen moves to the beginning of the next line, just as if you hit the Return key on the keyboard (this key is labeled Enter on some computers). In C, programs must include the newline character to mark the end of each screen line, or all the output will run together without any line breaks.

## 2.2 A program to add two numbers

To get a better picture of how a C program works, you need to consider a slightly more sophisticated example. The program `add2.c` shown in Figure 2-2 asks the user to enter two numbers, adds those numbers together, and then displays the sum.

The `add2.c` program incorporates several new programming concepts that were not part of `hello.c`. First, `add2.c` uses a new library called `simpio`, simplified

**FIGURE 2-2** add2.c

```
/*
 * File: add2.c
 * ------------
 * This program reads in two numbers, adds them together,
 * and prints their sum.
 */

#include <stdio.h>
#include "genlib.h"
#include "simpio.h"

main()
{
    int n1, n2, total;

    printf("This program adds two numbers.\n");
    printf("1st number? ");
    n1 = GetInteger();
    printf("2nd number? ");
    n2 = GetInteger();
    total = n1 + n2;
    printf("The total is %d.\n", total);
}
```

The text includes a large number of **Program examples** that illustrate how individual C complete are used to create complete programs. These examples also serve as models for your own programs; in many cases, you can solve a new programming problem by making simple modifications to a program from the text.

The `printf` function can display any number of data values as part of the output. For each integer value you want to appear as part of the output, you need to include the code `%d` in the string that is used as the first argument in the `printf` call. The actual values to be displayed are given as additional arguments to `printf`, listed in the order in which they should appear. For example, if you changed the last line of the `add2.c` program to

```
printf("%d + %d = %d\n", n1, n2, total);
```

the value of `n1` would be substituted in place of the first `%d`, the value of `n2` would appear in place of the second `%d`, and the value of `total` would appear in place of the third `%d`. The final image on the computer screen would be

```
This program adds two numbers.
1st number? 2↵
2nd number? 3↵
2 + 3 = 5
```

The `printf` function is discussed in more detail in Chapter 3.

### Cascading `if` statements

The syntax box on the right illustrates an important special case of the `if` statement that is useful for applications in which the number of possible cases is larger than two. The characteristic form is that the `else` part of a condition consists of yet another test to check for an alternative condition. Such statements are called **cascading if statements** and may involve any number of `else if` lines. For example, the program `signtest.c` in Figure 4-3 uses the cascading `if` statement to report whether a number is positive, zero, or negative. Note that there is no need to check explicitly for the `n < 0` condition. If the program reaches that last `else` clause, there is no other possibility, since the earlier tests have eliminated the positive and zero cases.

In many situations, the process of choosing between a set of independent cases can be handled more efficiently using the `switch` statement, which is described in a separate section later in this chapter.

**SYNTAX for *cascading* `if` statements**

```
if (condition1) {
    statements1
} else if (condition2) {          any
    statements2                   number
} else if (condition3) {          may
    statements3                   appear
} else {
    statements_none
}
```

where:
each $condition_i$ is a Boolean expression
each $statements_i$ is a block of statements to be executed if $condition_i$ is TRUE
$statements_{none}$ is the block of statements to be executed if every $condition_i$ is FALSE

### The `?:` operator (optional)

The C programming language provides another, more compact mechanism for onditional e

**Syntax boxes** summarize key rules of C syntax, for an at-a-glance review of key programming concepts.

# To Avoid Errors

All programmers, even the best ones, make mistakes. Finding these mistakes, or bugs, in your programs is a critically important skill. The following features will help you to build this skill.

**FIGURE 3-5** balance2.c (buggy version)

```
/*
 * File: balance2.c
 * ----------------
 * This file contains a buggy second attempt at a program to
 * balance a checkbook.
 */

#include <stdio.h>
#include "genlib.h"
#include "simpio.h"

main()
{
    double entry, balance;

    printf("This program helps you balance your checkbook.\n");
    printf("Enter each check and deposit during the month.\n");
    printf("To indicate a check, use a minus sign.\n");
    printf("Signal the end of the month with a 0 value.\n");
    printf("Enter the initial balance: ");
    balance = GetReal();
    while (TRUE) {
        printf("Enter check (-) or deposit: ");
        entry = GetReal();
        if (entry == 0) break;
        balance += entry;
        if (balance < 0) {
            printf("This check bounces.  $10 fee deducted.\n");
            balance -= 10;
        }
        printf("Current balance = %g\n", balance);
    }
    printf("Final balance = %g\n", balance);
}
```

To help you learn to recognize and correct bugs, this text includes several buggy programs that illustrate typical errors. To make sure you do not use these programs as models, such incorrect programs are marked with a superimposed image of a bug.

**Common Pitfalls** provide handy reminders about mistakes all beginning programmers are likely to make, and how to avoid them. Faulty lines of code are highlighted with a bug image and annotated in color.

**COMMON PITFALLS**

When writing programs that test for equality, be sure to use the == operator and not the single = operator, which signifies assignment. This error is extremely common and can lead to bugs that are very difficult to find, because the compiler cannot detect the error.

== Equal
!= Not equal

When you write programs that test for equality, be very careful to use the == operator, which is composed of two equal signs. A single equal sign is the assignment operator. Since the double equal sign violates conventional mathematical usage, replacing it with a single equal sign is a particularly common mistake. This mistake can also be very difficult to track down because the C compiler does not usually catch it as an error. A single equal sign usually turns the expression into an embedded assignment, which is perfectly legal in C; it just isn't at all what you want. For example, if you wanted to test whether the value of the variable x were equal to 0 and wrote the following conditional expression

```
if (x = 0) . . .                    This is incorrect.
```

the results would be confusing. This statement would not check to see if x were equal to 0. It would instead insist on this condition by assigning the value 0 to x, which C would then interpret (for reasons too arcane to describe at this point) as indicating a test result of FALSE. The correct test to determine whether the value of the variable x is equal to 0 is

```
if (x == 0) . . .
```

Be careful to avoid this error. A little extra care in entering your program can save a lot of debugging time later on