HZ BOOKS
华章教育

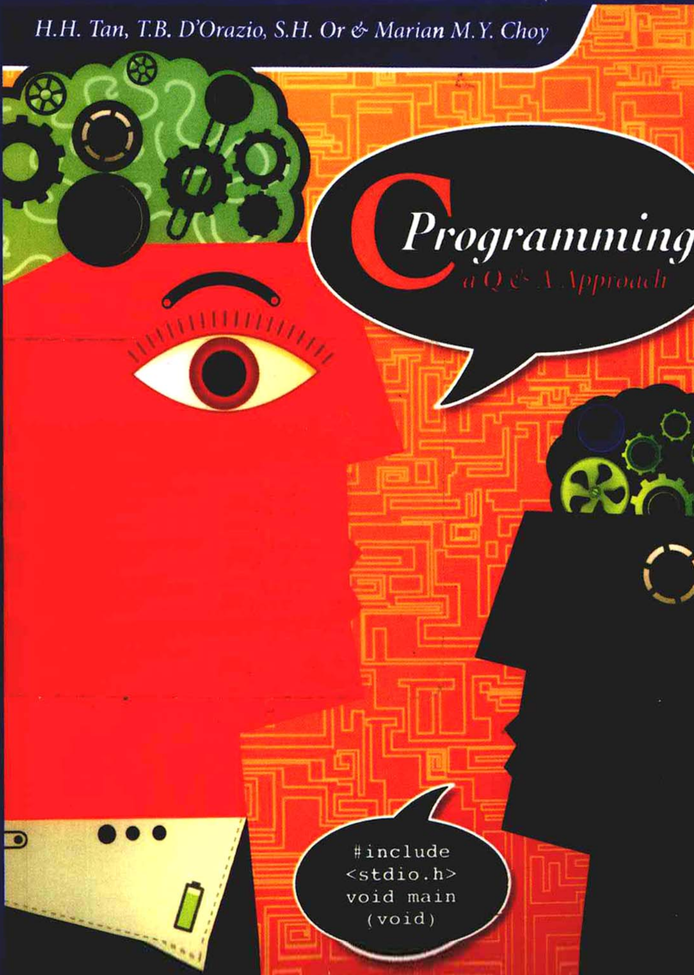
McGraw Hill Education

经典原版书库

C语言程序设计教程

H. H. Tan　T. B. D'Orazio　S. H. Or　Marian M. Y. Choy 著

（英文版）

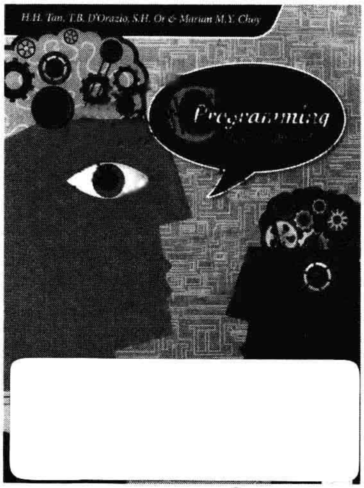


机械工业出版社
China Machine Press

经 典 原 版 书 库

C语言程序设计教程

C Programming
a Q & A Approach



H. H. Tan T. B. D'Orazio S. H. Or Marian M. Y. Choy 著

机 械 工 业 出 版 社
China Machine Press

图书在版编目（CIP）数据

C 语言程序设计教程（英文版）/（美）坦（H. H. Tan）等著．—北京：机械工业出版社，2012.12
（经典原版书库）
书名原文：C Programming: a Q & A Approach

ISBN 978-7-111-40432-3

I. C··· II. 坦··· III. C 语言－程序设计－教材－英文 IV. TP312

中国版本图书馆 CIP 数据核字（2012）第 273677 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：李　荣
北京京师印务有限公司印刷
2013 年 1 月第 1 版第 1 次印刷
150mm × 214mm · 17.75 印张
标准书号：ISBN 978-7-111-40432-3
定　　价：49.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991；88361066　　　投稿热线：（010）88379604
购书热线：（010）68326294；88379649；68995259　　　读者信箱：hzjsj@hzbook.com

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，还揭示了学术的源变，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和发展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到积极的推动作用，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公司较早意识到“出版要为教育服务”。自1998年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与Pearson，McGraw-Hill，Elsevier，MIT，John Wiley & Sons，Cengage等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出Andrew S. Tanenbaum，Bjarne Stroustrup，Brain W. Kernighan，Dennis Ritchie，Jim Gray，Afred V. Aho，John E. Hopcroft，Jeffrey D. Ullman，Abraham Silberschatz，William Stallings，Donald E. Knuth，John L. Hennessy，Larry L. Peterson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、研究及珍藏。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”已经出版了近两百个品种，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。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

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。随着计算机科学与技术专业学科建设的不断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将步入一个新的阶段，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。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：

华章网站：www.hzbook.com
电子邮件：hzjsj@hzbook.com
联系电话：(010) 88379604
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

HZ BOOKS
华章教育

华章科技图书出版中心

Preface

We developed this book to address the difficulty beginning students often find reading computer language texts. We felt that if we could get students involved in a text, hold their interest, and get them thinking about the meaning and uses of C code, we could make the process of learning a first language easier and fun. To accomplish this, we use a question and answer (Q&A) style. The reader's thought processes are stimulated by the same questions about code that students themselves often ask. By answering them directly and clearly, we focus the reader's attention on the important issues of C programming.

We also observed that most computer language texts have very few figures. Because visual images are so useful in teaching, we have made a concerted effort to create figures that are accurate yet easy to follow. These figures provide students reinforcement and clarification of concepts illustrated by the actions of the programs. In particular, we have three-dimensional sketches of the actions of loops and if control structures. Students can look at these figures and quickly grasp the flow of these structures. We believe that these figures are an improvement over the standard flow charts. We recognise that one of the most difficult topics for beginning C programmers is pointers. The foundation for our pointer figures is established early in the text with a table of the names, types, addresses and values of variables. Throughout the text, we use this table with arrows indicating how information at one location in memory relates to that in another location.

Many texts have numerous pages of code that are not adequately explained, and most beginning students are not capable or willing to independently decipher even relatively simple programs without guidance. In this book, we guide students through the code so that they understand both the operations and the thought processes that created the code. The goal is to make students realise what aspects of programming require extra thought and care and the importance of getting the details correct.

The unique style has been enthusiastically accepted among the students who have used early drafts of the book. Faculty unknown and unrelated to the authors have presented this text and asked the students' opinions. When compared with other texts, students have overwhelmingly preferred ours. Given this acceptance, we believe that you will find this book to be a valuable teaching and learning tool.

Text Organisation

The first chapter is an introduction to computing that assumes students have no basic knowledge other than using a computer for simple word processing. It introduces the concept of programming languages, describes hardware, the way information is stored in memory, computer languages, compilers, and software engineering. The purpose of this chapter is to introduce students to the way that computers work and the concepts behind software design.

Chapters 2 through 4 cover the fundamental aspects of a procedural programming language, basic syntax and control structures. C library functions are described and their use illustrated in these chapters. In Chapter 5, user-defined functions are covered, emphasising the concepts of modularity and reusable code. Pointers are gently introduced, being integrated with the development of functions that use addresses as arguments in function calls. After this chapter, the effects of using C features with user-defined functions are described.

Chapter 6 focuses on numerical arrays.

Chapter 7 describes both strings and pointers. Because strings are most commonly manipulated using addresses, this chapter is well suited to describing working with pointers to modify memory. Chapter 8 covers structures in C and their uses in creating linked lists, stacks, queues and binary trees. Also, large program design is covered in this chapter. This is included because engineering programs can quickly become very large. The importance of using C features to handle large programs is considered fundamental to preparing students for employment with firms that develop commercial software products in engineering.

We have called Chapter 9 (available at www.mheducation.asia/olc/cprogramming) An Introduction to C++, but it is actually much more. Because of the thorough coverage we have given C we are able to describe many of the core issues of object oriented programming with C++. Classes, encapsulation and polymorphism are described in simple terms. This chapter is richly illustrated. The simple language and illustrations provide students the background to use many of the fundamental C++ features.

Most chapters are divided into two parts, the Lessons and the Application Programs. The Lessons teach syntax, form and basic constructs. The Application Programs illustrate how what is taught in the lessons can be used to solve real engineering and computer science problems. The Application Programs show the thought processes a developer goes through to create a program. The goal of the Application Programs is to give students the ability to follow a structured methodology in developing their own programs.

Features

1. The book uses a simple question and answer approach that students find more friendly and accessible than standard narrative. The value of this approach lies in the authors' ability to craft the questions based on what students often ask and tailoring the answers in a manner that is easily understood.
2. Each Lesson begins with a single sample program: the source code accompanied by guided observations. Students gain understanding of C because they are compelled to follow the details of this code. Next, the output is revealed and following that, in the Explanation portion of the Lesson, a series of questions and answers are used to explain what the program is doing.
3. The Application Programs given in the second part of some chapters illustrate the usefulness of the C language for solving engineering and computer science problems. They are comprehensively explained. The examples focus on program design, software engineering, modularity and creating reusable code.
4. There are numerous figures illustrating programming concepts. Many of the figures are unique and give students an ability to grasp concepts quickly and easily.
5. A structured four-step method (becoming five steps after introducing strings and more complex data structures) of program development is illustrated in describing the Application Programs. The method includes creating structure charts and data flow diagrams.
6. Numerical method examples, which can be used in courses that combine programming and numerical methods, are included in the Application Programs.
7. The Lessons have annotated code that assists students in understanding program details and flow. The annotations help students focus on the code and highlight the important points that the code is demonstrating.
8. We realise that students typically will not follow multiple pages of code on their own. Therefore, the Application Programs, each of which can cover two or three pages, are explained completely. There are no multiple pages of unexplained code.
9. We know that students struggle with the concept of pointers. We also have found that for students to understand pointers, the figures representing them should be rooted to something that the students can visualize. It is not enough to have a box with an arrow pointing to another box. Using tables and grid-like sketches of memory, we have taken much of the mystery out of pointers. We have found that after reading our text, students "get" the concept of pointers.

10. Modification exercises after the Application Programs can be used for courses that have a laboratory. Instructors can tell the students in advance of the laboratory session to read a particular Application Program. During the lab, students can be guided through some of the changes that need to be made. Further changes can be assigned as home exercises.
11. Beginning students struggle with debugging because the process is new and foreign to them. Students often become frustrated because they must debug their very first program. Recognising this, we have included a detailed example of debugging very early in the text (Chapter 1). Beginners also find debugging loops difficult. In this text, we focus on loops and illustrate how values change as loops are executed. Students learn to trace loops and find errors. In addition, common beginners' errors are noted at appropriate locations throughout the text.
12. The True-False questions at the end of each Lesson (with solutions) allow students to quickly assess their progress in grasping the basics.
13. The Application Exercises at the end of most chapters can be used by an instructor for home assignments.
14. All of the programs in the book are available at www.mheducation.asia/olc/cprogramming. Students can modify and execute these programs to gain insight into how they operate.
15. We have included the last chapter, An Introduction to C++, as an online chapter available at www.mheducation.asia/olc/cprogramming. It actually covers more than just the basics. After reading this chapter, students will be able to use many of the fundamental aspects of object oriented programming.
16. Many of the Application Programs give students an introduction to numerical methods.

How To Use This Book

Student

In Chapter 1, you should focus on understanding what can be stored in memory, how a compiler works, and the steps in software engineering, and most importantly, your first C program. The rest of the chapters cover programming in C. Each lesson in these chapters begins with a short introduction to the lesson's source code. Use the introduction to guide you through the important points of the code. Then read the code and the annotations in the boxes. You can even try executing the code and observe how the program behaves. After doing this, make sure that you begin to understand the major topics being covered in the lesson. Then read the Explanation and do the True-False and short answer exercises. If you do not do well on the exercises, re-visit the lesson to clear any queries.

When you feel comfortable with the lessons in a chapter, begin the Application Programs. The purpose of these is to illustrate the thought processes that you would typically go through when you write your programs and to show practical uses of C. You will find as you write your own programs that you will be addressing many of the same issues raised in the Application Programs. In these, focus on learning the methodology and understanding the logic of each program. Remember, the logical flow is very important in programming. A statement can be correct in terms of syntax but totally wrong logically. Grasping the why and how of each Application Program, gives you the confidence to write your own program. Do not just read, but try out the programs, make changes to experiment variations, these help to crystallise what you have gained from the lessons. You will be able to explain the various behaviours of a program. Use this knowledge in writing the programs assigned by your instructor.

Instructors

For a full semester course aiming to equip students for advance programming in later courses, for example, in C programming with an introduction to C++, we recommend that you cover the entire text in the order presented. However, it is possible to cover the material in an order different from that given. For instance, Lesson 3.2 (Single Character Data) can be covered immediately before the first lesson of Chapter 7. Also, Lessons 8.7 (Creating Header Files), 8.8 (Use of Multiple Source Code Files and Storage Classes) and Function-like Macros and Conditional Inclusion which are available on www.mheducation.asia/olc/cprogramming, can be covered with the material in Chapter 5 (Functions) if so desired.

It is also possible to postpone some of the last lessons in Chapters 7, 8 and 9 and cover them as time permits. For instance, Lesson 7.9 (Pointer Notation versus Array Notation) can be postponed until immediately before advanced topics on pointers (Additional materials for Chapter 8, Pointers to Functions and Functions Returning Pointers are available on www.mheducation.asia/olc/cprogramming).

For a full semester course aiming to build a foundation in programming, we recommend that you cover through Lesson 7.8 and Lessons 7.10, 8.1, 8.2, 8.3, 8.4, and 8.5.

For a short course aiming to give students a general hands-on programming experience, students will be able to write valuable and sophisticated C programs if you cover just the first six chapters.

The book is filled with exercises. After each lesson are True-False and short answer exercises. The students should do these on their own. Some simpler programs are also assigned at the end of some of the lessons. These can be assigned as homework. One week is probably a sufficient amount of time for students to complete one of these programs.

The Modification Exercises at the end of the Application Programs can be used for courses that have a laboratory. Students should prepare for the laboratory by studying the pertinent Application Program. During the lab, students can be guided to make the modifications required by the exercises. Some of these exercises are straightforward while others are difficult. The ones that are difficult can be assigned as home exercises.

At the end of most of the chapters are Application Exercises. These tend to be the most challenging exercises in the book, and therefore are most appropriate as home assignments. Two to four weeks is an appropriate amount of time for doing them depending on the difficulty level. McGraw-Hill will furnish a solutions manual to selected exercises.

In addition, the book can be used as a reference for much of ANSI C. Reference tables are distributed throughout the book and some are given in the appendices.

Instructor Supplements

Instructors can access the following supplements at www.mheducation.asia/olc/cprogramming.

- Solutions Manual
- Powerpoint slides
- Test Bank
- Additional exercises
- Additional reading material

Acknowledgements

We would like to thank Eric Munson and Holly Stark of McGraw-Hill for their interest, support, encouragement, and insightful comments regarding this project. It has been a real pleasure working with them. We would also like to thank Byron Gottfried, the consulting editor of the BEST series, for both his support and valuable criticism, and Alisa Watson and her production team at McGraw-Hill for putting together what we believe is a very nice-looking text.

We also had a number of very thoughtful reviewers. Betty Barr, University of Houston; Raymond Bell, University of Texas, El Paso; Tat W. Chan, Fayetteville State University; Bart Childs, Texas A&M University; Chris J. Dovolis, University of Minnesota; Janet Hartman, Illinois State University; Elden W. Heiden, New Mexico State University; Elias Houstis, Purdue University; Joseph Konstan, University of Minnesota; Jandelyn Plane, University of Maryland; and Matthew Ward, WPI, all made quite helpful comments and suggestions. Their input is greatly appreciated.

Our early computing careers began at the University of Michigan and U. C. Berkeley. We would like to thank Professors J. M. Duncan, John Lysmer, and Raymond Canale (the instructor in T.B. D'Orazio's first freshman year computing class), for inspiring (and pushing) us to tackle creating some very complex programs. The satisfaction in developing these programs motivated us to write this book.

We would also like to thank Suzanne Lacasse and Kaare Hoeg, Director and former Director of the Norwegian Geotechnical Institute, respectively. Their confidence in us and their financial support in writing geotechnical application programs formed the basis for extending our computing backgrounds and established the foundation for further developing our programming skills. We thank Hui Xian Liu, the previous director of the Institute of Mechanical Engineering in Harbin, China, for his unending support, guidance, and encouragement of H.H. Tan.

Lastly, we would like to thank our families. T.B. D'Orazio's wife, Elizabeth, although very busy herself, found the time to be encouraging and supportive during the entire process. With her, this endeavour has been a joy. H.H. Tan's wife, Wei Huang, has been wonderful with her warmth, tolerance, and intelligence. Sijing Tan, H.H. Tan's daughter, drew most of the figures in the first four chapters. We very much appreciate her effort and skill in doing these.

H.H. Tan
T.B. D'Orazio

In this revision, we would like to first thank Gerald Bok and Chris Cheung from McGraw-Hill. Gerald, who was the main thrust behind this project, and taking care of a lot of tedious matters so that we can just focus on our editing work. Also Chris did a lot of interfacing and is very supportive throughout the project. We are also grateful to Maureen Tan at McGraw-Hill for her excellent support in this project.

Dr. Or would also like to thank Professor John Lui of the Chinese University of Hong Kong for his support to this project during his period of service as the Chairman of the Computer Science & Engineering Department. Thanks are also due to Dr. Or's wife Pui-yee Lau, as without her taking care of the children, this book cannot be a reality.

Marian would like to thank Tim Lambert (her favourite lecturer in the University of New South Wales), for inspiring her to see the fun and creativity in programming. Thanks are also due to Professor Horace Ip who gave her the first teaching contract in the City University of Hong Kong. She is also grateful to Professor W.C. Chew and Professor George L. Tham in the Faculty of Engineering in the University of Hong Kong. Their confidence in her provides her with a free platform to explore and experiment with various teaching styles.

She would like to thank all the students who took her programming course over the past years. She would like to extend sincere gratitude to her family, cell group members, brothers and sisters from church, friends and colleagues, who support her unfailingly and most of all, put up with her stubbornness and insistence on adhering to quality teaching. Most important of all, may God, the idea provider, inspire you.

S.H. Or
Marian M.Y. Choy

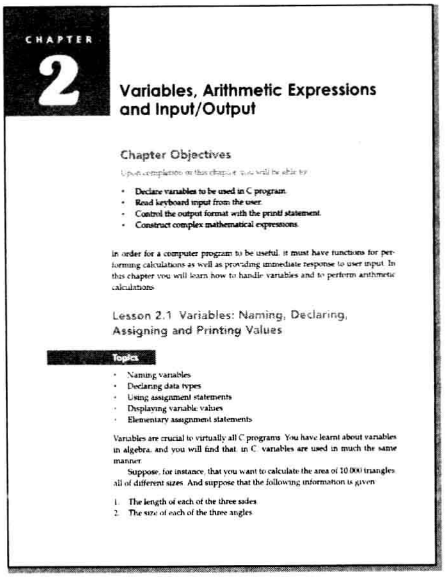
About the Authors

Andrew Tan H.H. from Morrison Knudsen Corporation and **Timothy D'Orazio,** currently head of the Civil Engineering program at San Francisco State University are the original authors of the book, entitled *C Programming for Engineering and Computer Science,* on which *C Programming: A Q&A Approach* is based. The new co-authors who developed *C Programming: A Q&A Approach* are Dr. Or Siu Hang and Dr. Marian Choy.

Or Siu Hang is the founder and Project Lead of the Computer Game Technology Center in the Department of Computer Science and Engineering at the Chinese University of Hong Kong. He received his PhD from the Chinese University of Hong Kong in 1998. He has over 19 years of teaching experience in programming courses aimed at first-year university students. His research interests include computer graphics, computer vision, multimedia and development of computer games. He is a member of the Institute of Electrical and Electronics Engineers (IEEE) and a member of the Association for Computing Machinery (ACM).

Marian Choy Ming Yan is a teaching consultant in the Faculty of Engineering at the University of Hong Kong. She has a passion for teaching and is eager to spend time enhancing teaching and learning activities. She obtained her undergraduate degree in computer engineering, followed by her PhD from the University of New South Wales, Australia. She has been in academia for more than 10 years and amongst her research interests are computers in education, adaptive technology and image processing.

Chapter Objectives, Concept Recap and Chapter Review that help students to quickly grasp key concepts at strategic points in the book



CHAPTER 2

Variables, Arithmetic Expressions and Input/Output

Chapter Objectives

Upon completion of this chapter you will be able to:

- Declare variables to be used in C program.
- Read keyboard input from the user.
- Control the output format with the printf statement.
- Construct complex mathematical expressions.

In order for a computer program to be useful, it must have functions for performing calculations as well as providing immediate response to user input. In this chapter you will learn how to handle variables and to perform arithmetic calculations.

Lesson 2.1 Variables: Naming, Declaring, Assigning and Printing Values

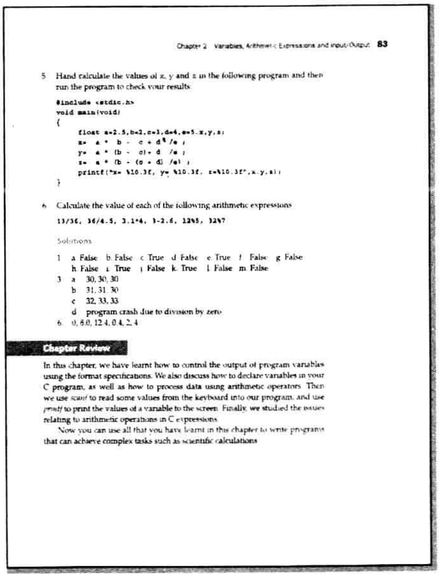
Topics

- Naming variables
- Declaring data types
- Using assignment statements
- Displaying variable values
- Elementary assignment statements

Variables are crucial to virtually all C programs. You have learnt about variables in algebra, and you will find that, in C, variables are used in much the same manner.

Suppose, for instance, that you want to calculate the area of 10,000 triangles, all of different sizes. And suppose that the following information is given:

1. The length of each of the three sides
2. The size of each of the three angles



Chapter 2 Variables, Arithmetic Expressions and Input/Output **83**

5 Hand calculate the values of x, y and z in the following program and then run the program to check your results.

```
#include <stdio.h>
void main(void)
{
      float a=2.5,b=2,c=1,d=4,e=5,x,y,z;
      x=  a *  b -  c + d % /e ;
      y=  a * (b -  c)+ d  /e ;
      z=  a * (b - (c + d) /e) ;
      printf("x= %10.3f, y= %10.3f, z=%10.3f",x,y,z);
}
```

6 Calculate the value of each of the following arithmetic expressions:

```
13/36, 36/4.5, 3.1*4, 3-2.6, 12%5, 32%7
```

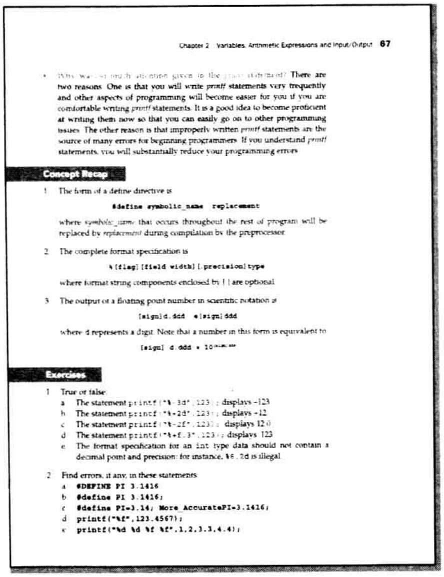
Solutions

1 a False b. False c True d False e True f False g False
h False i True j False k True l False m False
3 a 30, 30, 30
b 31, 31, 30
c 32, 33, 33
d program crash due to division by zero
6 0, 8.0, 12.4, 0.4, 2, 4

Chapter Review

In this chapter, we have learnt how to control the output of program variables using the format specifications. We also discuss how to declare variables in your C program, as well as how to process data using arithmetic operators. Then we use *scanf* to read some values from the keyboard into our program, and use *printf* to print the values of a variable to the screen. Finally, we studied the issues relating to arithmetic operations in C expressions.

Now you can use all that you have learnt in this chapter to write programs that can achieve complex tasks such as scientific calculations.



Chapter 2 Variables, Arithmetic Expressions and Input/Output **67**

- Why was so much attention given to the printf statement? There are two reasons. One is that you will write *printf* statements very frequently and other aspects of programming will become easier for you if you are comfortable writing *printf* statements. It is a good idea to become proficient at writing them now so that you can easily go on to other programming issues. The other reason is that improperly written *printf* statements are the source of many errors for beginning programmers. If you understand *printf* statements, you will substantially reduce your programming errors.

Concept Recap

1 The form of a define directive is

```
#define symbolic_name  replacement
```

where *symbolic_name* that occurs throughout the rest of program will be replaced by *replacement* during compilation by the preprocessor.

2 The complete format specification is

```
%[flag][field width][.precision]type
```

where format string components enclosed by [ ] are optional.

3 The output of a floating point number in scientific notation is

```
[sign]d.ddd  e[sign]ddd
```

where d represents a digit. Note that a number in this form is equivalent to

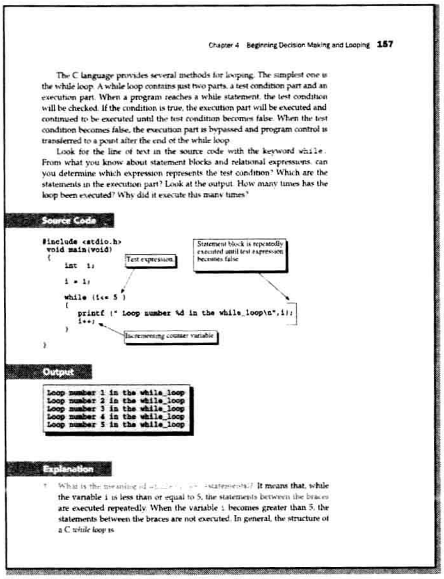
```
[sign] d.ddd * 10[sign]ddd
```

Exercises

1 True or false:
a The statement `printf("%-3d",123);` displays -123
b The statement `printf("%-2d",123);` displays -12
c The statement `printf("%-2f",123);` displays 12.0
d The statement `printf("%+f.3",123);` displays 123
e The format specification for an `int` type data should not contain a decimal point and precision: for instance, %6.2d is illegal

2 Find errors, if any, in these statements:
a `#DEFINE PI 3.1416`
b `#define PI 3.1416;`
c `#define PI=3.14; More_AccuratePI=3.1416;`
d `printf("%f",123.4567);`
e `printf("%d %d %f %f",1,2,3.3,4.4);`

Simple sample programs consisting of source code accompanied by guided observations, and output



Chapter 4 Beginning Decision Making and Looping 157

The C language provides several methods for looping. The simplest one is the while loop. A while loop contains just two parts, a test condition part and an execution part. When a program reaches a while statement, the test condition will be checked. If the condition is true, the execution part will be executed and continued to be executed until the test condition becomes false. When the test condition becomes false, the execution part is bypassed and program control is transferred to a point after the end of the while loop.

Look for the line of text in the source code with the keyword `while`. From what you know about statement blocks and relational expressions, can you determine which expression represents the test condition? Which are the statements in the execution part? Look at the output. How many times has the loop been executed? Why did it execute this many times?

Source Code

```
#include <stdio.h>
 void main(void)
 {
     int  i;

     i = 1;

     while (i<= 5 )
     {
        printf (" Loop number %d in the while_loop\n",i);
        i++;
     }
}
```

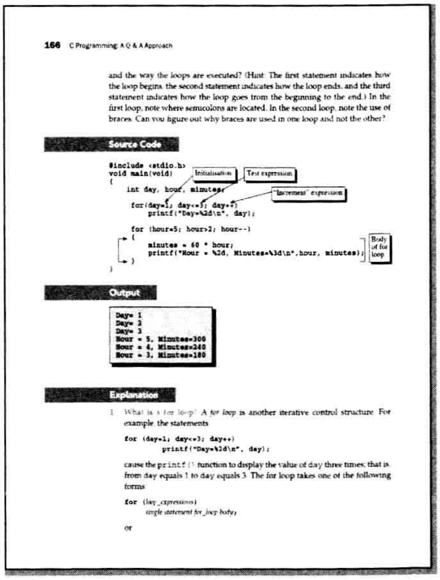
Statement block is repeatedly executed until test expression becomes false
Test expression
Incrementing counter variable

Output

```
Loop number 1 in the while_loop
Loop number 2 in the while_loop
Loop number 3 in the while_loop
Loop number 4 in the while_loop
Loop number 5 in the while_loop
```

Explanation

1 What is the meaning of `while (i <= 5)` statements? It means that, while the variable `i` is less than or equal to 5, the statements between the braces are executed repeatedly. When the variable `i` becomes greater than 5, the statements between the braces are not executed. In general, the structure of a C *while loop* is



166 C Programming: A Q & A Approach

and the way the loops are executed? (Hint: The first statement indicates how the loop begins, the second statement indicates how the loop ends, and the third statement indicates how the loop goes from the beginning to the end.) In the first loop, note where semicolons are located. In the second loop, note the use of braces. Can you figure out why braces are used in one loop and not the other?

Source Code

```
#include <stdio.h>
void main(void)
{
    int day, hour, minutes;

     for(day=1; day<=3; day++)
         printf("Day=%2d\n", day);

     for (hour=5; hour>2; hour--)
     {
         minutes = 60 * hour;
         printf("Hour = %2d, Minutes=%3d\n",hour, minutes);
     }
}
```

Output

```
Day= 1
Day= 2
Day= 3
Hour = 5, Minutes=300
Hour = 4, Minutes=240
Hour = 3, Minutes=180
```

Explanation

1 What is a *for loop*? A *for loop* is another iterative control structure. For example, the statements

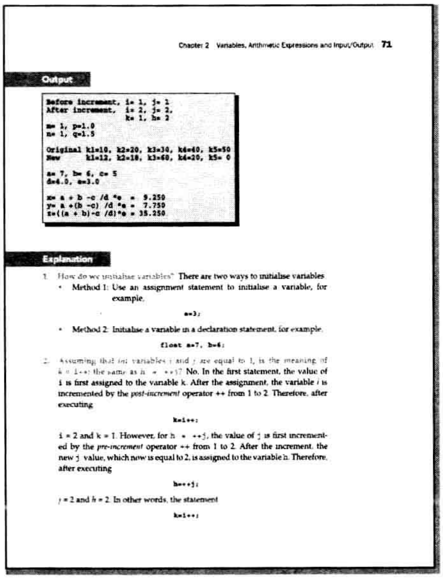
```
for (day=1; day<=3; day++)
        printf("Day=%2d\n", day);
```

cause the `printf()` function to display the value of day three times; that is, from day equals 1 to day equals 3. The for loop takes one of the following forms

for (*loop_expressions*)
    single statement for_loop body;

or

Explanation of code clearly presented in question and answer format



Output

```
Before increment, i= 1, j= 1
After increment,  i= 2, j= 2,
                  k= 1, h= 2
m= 1, p=1.0
n= 1, q=1.5

Original k1=10, k2=20, k3=30, k4=40, k5=50
New      k1=12, k2=18, k3=60, k4=20, k5= 0

a= 7, b= 6, c= 5
d=4.0, e=3.0

x= a + b -c /d *e  =  9.250
y= a +(b -c) /d *e =  7.750
z=((a + b)-c /d)*e = 35.250
```

Explanation

1. How do we initialise variables? There are two ways to initialise variables
   - Method 1: Use an assignment statement to initialise a variable, for example,

     `a=3;`

   - Method 2: Initialise a variable in a declaration statement, for example,

     `float a=7, b=6;`

2. Assuming that int variables i and j are equal to 1, is the meaning of k = i++; the same as h = ++j? No. In the first statement, the value of i is first assigned to the variable k. After the assignment, the variable *i* is incremented by the *post-increment* operator ++ from 1 to 2. Therefore, after executing

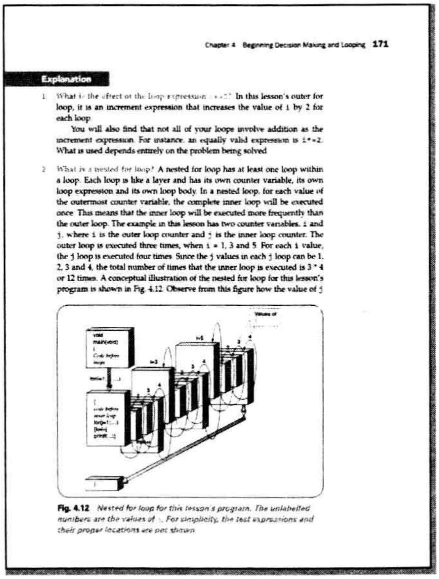
   `k=i++;`

   i = 2 and k = 1. However, for h = ++j, the value of j is first incremented by the *pre-increment* operator ++ from 1 to 2. After the increment, the new j value, which now is equal to 2, is assigned to the variable h. Therefore, after executing

   `h=++j;`

   j = 2 and *h* = 2. In other words, the statement

   `k=i++;`



Explanation

1. What is the effect of the loop expression i+=2? In this lesson's outer for loop, it is an increment expression that increases the value of i by 2 for each loop.

   You will also find that not all of your loops involve addition as the increment expression. For instance, an equally valid expression is i*=2. What is used depends entirely on the problem being solved.

2. What is a nested for loop? A nested for loop has at least one loop within a loop. Each loop is like a layer and has its own counter variable, its own loop expression and its own loop body. In a nested loop, for each value of the outermost counter variable, the complete inner loop will be executed once. This means that the inner loop will be executed more frequently than the outer loop. The example in this lesson has two counter variables, i and j, where i is the outer loop counter and j is the inner loop counter. The outer loop is executed three times, when i = 1, 3 and 5. For each i value, the j loop is executed four times. Since the j values in each j loop can be 1, 2, 3 and 4, the total number of times that the inner loop is executed is 3 * 4 or 12 times. A conceptual illustration of the nested for loop for this lesson's program is shown in Fig. 4.12. Observe from this figure how the value of j

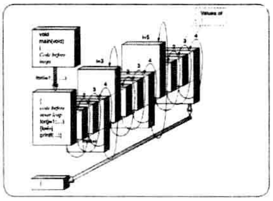


Fig. 4.12 *Nested for loop for this lesson's program. The unlabelled numbers are the values of j. For simplicity, the test expressions and their proper locations are not shown.*