大学计算机教育丛书（影印版）

SECOND EDITION

THE C ANSWER BOOK

Solutions to the Exercises in
The C Programming Language, second edition
by Brian W. Kernighan & Dennis M. Ritchie

C程序设计语言(第二版)习题解答

(第二版)

CLOVIS L. TONDO   SCOTT E. GIMPEL

清华大学出版社 · PRENTICE HALL

The C Answer Book

Second Edition

Solutions to the Exercises in
The C Programming Language, Second Edition
by Brian W. Kernighan and Dennis M. Ritchie

C 程序设计语言（第二版）习题解答

（第二版）

Clovis L. Tondo
Scott E. Gimpel

清 华 大 学 出 版 社
Prentice-Hall International, Inc.

(京)新登字 158 号

The C answer book: solutions to the exercises in The C programming language, second edition, by Brian W. Kernighan and Dennis M. Ritchie/ Colvis L. Tondo, Scott E. Gimpel. —2nd ed.

图书在版编目(CIP)数据

C 程序设计语言(第二版)习题解答:英文/(美)汤多(Tondo,C. L.),(美)金贝尔(Gimpel,S. E.)著.—影印版.—北京:清华大学出版社,1997.11
(大学计算机教育丛书)
ISBN 7-302-02728-5

Ⅰ.C… Ⅱ.①汤… ②金… Ⅲ.C 语言-程序设计-解题 Ⅳ.TP312-44

中国版本图书馆 CIP 数据核字(97)第 23936 号

出版者: 清华大学出版社(北京清华大学校内,邮编 100084)
因特网地址: www.tup.tsinghua.edu.cn
印刷者: 清华大学印刷厂
发行者: 新华书店总店北京科技发行所
开　本: 850×1168 1/32　印张: 6.75
版　次: 1997 年 11 月第 1 版　1998 年 7 月第 2 次印刷
书　号: ISBN 7-302-02728-5/TP・1414
印　数: 5001～10000
定　价: 12.00 元

出版前言

我们的大学生、研究生毕业后，面临的将是一个国际化的信息时代。他们将需要随时查阅大量的外文资料；会有更多的机会参加国际性学术交流活动；接待外国学者；走上国际会议的讲坛。作为科技工作者，他们不仅应有与国外同行进行口头和书面交流的能力，更为重要的是，他们必须具备极强的查阅外文资料获取信息的能力。有鉴于此，在国家教委所颁布的"大学英语教学大纲"中有一条规定：专业阅读应作为必修课程开设。同时，在大纲中还规定了这门课程的学时和教学要求。有些高校除开设"专业阅读"课之外，还在某些专业课拟进行英语授课。但教、学双方都苦于没有一定数量的合适的英文原版教材作为教学参考书。为满足这方面的需要，我们挑选了 7 本计算机科学方面最新版本的教材，进行影印出版。首批影印出版的 6 本书受到广大读者的热情欢迎，我们深受鼓舞，今后还将陆续推出新书。希望读者继续给予大力支持。Prentice Hall 公司和清华大学出版社这次合作将国际先进水平的教材引入我国高等学校，为师生们提供了教学用书，相信会对高校教材改革产生积极的影响。

清华大学出版社

Prentice Hall 公司

1997.11

Preface

This is an ANSWER BOOK. It provides solutions to all the exercises in *The C Programming Language*, second edition, by Brian W. Kernighan and Dennis M. Ritchie (Prentice Hall, 1988)*.

The American National Standards Institute (ANSI) produced the ANSI standard for C and K&R modified the first edition of *The C Programming Language*. We have rewritten the solutions to conform to both the ANSI standard and the second edition of K&R.

Careful study of *The C Answer Book*, second edition, used in conjunction with K&R, will help you understand C and teach you good C programming skills. Use K&R to learn C, work the exercises, then study the solutions presented here. We built our solutions using the language constructions known at the time the exercises appear in K&R. The intent is to follow the pace of K&R. Later, when you learn more about the C language, you will be able to provide possibly better solutions. For example, until the statement

```
if (expression)
    statement-1
else
    statement-2
```

is explained on page 21 of K&R, we do not use it. However, you could improve the solutions to Exercises 1-8, 1-9, and 1-10 (page 20 K&R) by using it. At times we also present unconstrained solutions.

We explain the solutions. We presume you have read the material in K&R up to the exercise. We try not to repeat K&R, but describe the highlights of each solution.

You cannot learn a programming language by only reading the language constructions. It also requires programming—writing your own code and study-

*Hereafter referred to as K&R.

ing that of others. We use good features of the language, modularize our code, make extensive use of library routines, and format our programs to help you see the logical flow. We hope this book helps you become proficient in C.

We thank the friends that helped us to produce this second edition: Brian Kernighan, Don Kostuch, Bruce Leung, Steve Mackey, Joan Magrabi, Julia Mistrello, Rosemary Morrissey, Andrew Nathanson, Sophie Papanikolaou, Dave Perlin, Carlos Tondo, John Wait, and Eden Yount.

Clovis L. Tondo

Contents

CHAPTER 1 **A Tutorial Introduction**

Exercise 1-1: (page 8 K&R)

Run the "`hello, world`" program on your system. Experiment with leaving out parts of the program to see what error messages you get.

```
#include  <stdio.h>

main()
{
      printf("hello, world");
}
```

In this example the newline character (`\n`) is missing. This leaves the cursor at the end of the line.

```
#include <stdio.h>

main()
{
      printf("hello, world\n")
}
```

In the second example the semicolon is missing after `printf()`. Individual C statements are terminated by semicolons (page 10 K&R). The compiler should recognize that the semicolon is missing and print the appropriate message.

```
#include <stdio.h>

main()
{
      printf("hello, world\n');
}
```

In the third example the double quote " after \n is mistyped as a single quote. The single quote, along with the right parenthesis and the semicolon, is taken as part of the string. The compiler should recognize this as an error and complain that a double quote is missing, that a right parenthesis is missing before a right brace, the string is too long, or that there is a newline character in a string.

Exercise 1-2: (page 8 K&R)

Experiment to find out what happens when `printf`'s argument string contains \c, where *c* is some character not listed above.

```
#include <stdio.h>

main()
{
      printf("hello, world\y");
      printf("hello, world\7");
      printf("hello, world\?");
}
```

The Reference Manual (Appendix A, page 193 K&R) states

> If the character following the \ is not one of those specified, the behavior is undefined.

The result of this experiment is compiler dependent. One possible result might be

```
hello, worldyhello, world<BELL>hello, world?
```

where `<BELL>` is a short beep produced by ASCII 7. It is possible to have a \ followed by up to three octal digits (page 37 K&R) to represent a character. \7 is specified to be a short beep in the ASCII character set.

Exercise 1-3: (page 13 K&R)

Modify the temperature conversion program to print a heading above the table.

```
#include <stdio.h>

/* print Fahrenheit-Celsius table
    for fahr = 0, 20, . . ., 300; floating-point version */
main()
{
     float fahr, celsius;
     int lower, upper, step;

     lower = 0;    /* lower limit of temperature table    */
     upper = 300; /* upper limit                          */
     step  = 20;  /* step size                            */

     printf("Fahr Celsius\n");
     fahr = lower;
     while (fahr <= upper) {
          celsius = (5.0/9.0) * (fahr-32.0);
          printf("%3.0f    %6.1f\n", fahr, celsius);
          fahr = fahr + step;
     }
}
```

The addition of

```
printf("Fahr Celsius\n");
```

before the loop produces a heading above the appropriate columns. We also added two spaces between `%3.0f` and `%6.1f` to align the output with the heading. The remainder of the program is the same as on page 12 K&R.

Exercise 1-4: (page 13 K&R)

Write a program to print the corresponding Celsius to Fahrenheit table.

```
#include <stdio.h>

/* print Celsius-Fahrenheit table
    for celsius = 0, 20, ..., 300; floating-point version   */
main()
{
      float fahr, celsius;
      int lower, upper, step;

      lower = 0;        /* lower limit of temperature table */
      upper = 300;      /* upper limit                      */
      step  = 20;       /* step size                        */

      printf("Celsius  Fahr\n");
      celsius = lower;
      while (celsius <= upper) {
            fahr = (9.0*celsius) / 5.0 + 32.0;
            printf("%3.0f  %6.1f\n", celsius, fahr);
            celsius = celsius + step;
      }
}
```

The program produces a table containing temperatures in degrees Celsius (0–300) and their equivalent Fahrenheit values. Degrees Fahrenheit are calculated using the statement:

```
fahr = (9.0*celsius) / 5.0 + 32.0
```

The solution follows the same logic as used in the program that prints the Fahrenheit-Celsius table (page 12 K&R). The integer variables `lower`, `upper`, and `step` refer to the lower limit, upper limit, and step size of the variable `celsius`, respectively. The variable `celsius` is initialized to the lower limit, and inside the `while` loop the equivalent Fahrenheit temperature is calculated. The program prints Celsius and Fahrenheit and increments the variable `celsius` by the step size. The `while` loop repeats until the variable `celsius` exceeds its upper limit.

Exercise 1-5: (page 14 K&R)

Modify the temperature conversion program to print the table in reverse order, that is, from 300 degrees to 0.

```
#include <stdio.h>

/* print Fahrenheit-Celsius table in reverse order        */
main()
{
      int fahr;

      for (fahr = 300; fahr >= 0; fahr = fahr - 20)
           printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
}
```

The only modification is:

```
for (fahr = 300; fahr >= 0; fahr = fahr - 20)
```

The first part of the `for` statement,

```
fahr = 300
```

initializes the Fahrenheit variable (`fahr`) to its upper limit. The second part, or the condition that controls the `for` loop,

```
fahr >= 0
```

tests whether the Fahrenheit variable exceeds or meets its lower limit. The `for` loop continues as long as the statement is true. The step expression,

```
fahr = fahr - 20
```

decrements the Fahrenheit variable by the step size.

Exercise 1-6: (page 17 K&R)

Verify that the expression `getchar() != EOF` is 0 or 1.

```
#include <stdio.h>

main()
{
	int c;

	while (c = getchar() != EOF)
		printf("%d\n", c);
	printf("%d - at EOF\n", c);
}
```

The expression

```
c = getchar() != EOF
```

is equivalent to

```
c = (getchar() != EOF)
```

(page 17 K&R). The program reads characters from the standard input and uses the expression above. While `getchar` has a character to read it does not return the end of file and

```
getchar() != EOF
```

is true. So 1 is assigned to `c`. When the program encounters the end of file, the expression is false. Then 0 is assigned to `c` and the loop terminates.

Exercise 1-7: (page 17 K&R)

Write a program to print the value of EOF.

```
#include <stdio.h>

main()
{
	printf("EOF is %d\n", EOF);
}
```

The symbolic constant EOF is defined in <stdio.h>. The EOF outside the double quotes in printf() is replaced by whatever text follows

```
#define EOF
```

in the include file. In our system EOF is -1, but it may vary from system to system. That's why standard symbolic constants like EOF help make your program portable.

Exercise 1-8: (page 20 K&R)

Write a program to count blanks, tabs, and newlines.

```
#include <stdio.h>

/* count blanks, tabs, and newlines                          */
main()
{
    int c, nb, nt, nl;

    nb = 0;                         /* number of blanks      */
    nt = 0;                         /* number of tabs        */
    nl = 0;                         /* number of newlines    */
    while ((c = getchar()) != EOF) {
        if (c == ' ')
            ++nb;
        if (c == '\t')
            ++nt;
        if (c == '\n')
            ++nl;
    }
    printf("%d %d %d\n", nb, nt, nl);
}
```

The integer variables `nb`, `nt`, and `nl` are used to count the number of blanks, tabs, and newlines, respectively. Initially, these three variables are set equal to 0.

Inside the body of the `while` loop, the occurrence of each blank, tab, and newline from input is recorded. All `if` statements are executed each time through the loop. If the character received is anything but a blank, tab, or newline, then no action is taken. If it is one of these three, then the appropriate counter is incremented. The program prints the results when the `while` loop terminates (`getchar` returns `EOF`).

The `if-else` statement is not presented until page 21 K&R. With that knowledge the solution could be:

```
#include <stdio.h>

/* count blanks, tabs, and newlines                              */
main()
{
     int c, nb, nt, nl;

     nb = 0;                        /* number of blanks          */
     nt = 0;                        /* number of tabs            */
     nl = 0;                        /* number of newlines        */
     while ((c = getchar()) != EOF)
          if (c == ' ')
               ++nb;
          else if (c == '\t')
               ++nt;
          else if (c == '\n')
               ++nl;
     printf("%d %d %d\n", nb, nt, nl);
}
```