第8章　C风格的控制台I/O

C++支持两种I/O系统。第一种是从C语言中继承来的，第二种是由C++定义的面向对象的I/O系统。本章和下一章讨论类C的I/O系统（第二部分讨论C++ I/O）。对大部分新项目，虽然你可能想使用C++ I/O系统，但是C风格的I/O仍然很常见，知道它的特征是全面理解C++的基础。

在C语言中，输入和输出是通过库函数完成的。既有控制台函数，也有文件I/O函数。从技术上讲，控制台I/O和文件I/O之间只有微小的差别。然而，从概念上讲，它们是两个截然不同的范畴。本章详细介绍控制台I/O函数。下一章将介绍文件I/O系统以及两种系统是如何相互作用的。

本章只讨论由标准C++定义的那些控制台I/O函数，但有一个例外。标准C++没有定义执行各种屏幕控制操作（如光标位置）或显示图像的任何函数，因为这些操作在不同的机器上变化很大。它也没有定义任何写到Windows下窗口或对话框的函数。相反，控制台I/O函数只完成基于TTY（电传打字机）的输出。然而，大多数编译器都在它们的库中包含了屏幕控制和图形函数以适应特殊的环境。当然，可以使用C++来编写Windows程序，但要记住，C++语言没有直接定义执行这些任务的函数。

标准C I/O函数都使用头文件stdio.h。C++程序也可以使用C++风格的头文件<cstdio>。

本章中提到的控制台I/O函数从键盘输入并输出到屏幕上。然而，这些函数实际上让系统的标准输入和标准输出作为它们I/O操作的目标和/或源。此外，标准输入和标准输出也可重定向到其他设备。这些概念将在第9章中讨论。

8.1　一个重要的应用说明

本书的第一部分使用了类C的I/O系统，因为它是为C++的C子集定义的惟一I/O系统。正如所提到的，C++也定义了它自己的面向对象的I/O系统。对大多数C++应用来说，你可能想使用C++专用的I/O系统，而不是本章描述的C I/O系统。然而，理解基于C的I/O系统是非常重要的，理由如下：

- 在你的工作生涯中，你可能被要求编写限于C子集的代码。此时，将需要使用类C的I/O函数。
- 今后几年，C和C++将共同存在。因而，多数程序将是C和C++代码的混合物。还有，将C语言程序“升级”成C++程序是很普遍的。因此，必须了解C和C++I/O系统的知识。例如，为了把类C的I/O函数改变成相应的面向对象的C++I/O函数，有必要掌握C和C++I/O系统是如何操作的。
- 掌握类C的I/O系统的基本原理是学习C++面向对象的I/O系统的基础（两者有很多共同的概念）。
- 在某些情况下（例如，在小的程序中），使用C的非面向对象的I/O方法比使用由C++定

义的面向对象的I/O方法要容易。

另外，有个默认的原则，就是任何C++程序员也必须是C程序员，因而若不掌握C语言的I/O系统，就会限制用户的专业水平。

8.2 读写字符

最简单的控制台I/O函数是getchar( )和putchar( )。前者从键盘读字符，后者把字符显示在屏幕上。函数getchar( )等待敲击键，然后返回其值。通常，getchar( )也可以自动“回送”敲入的字符到屏幕上。函数getchar( )在屏幕的当前光标位置上写字符。

getchar( )和putchar( )的原型如下：

```
int getchar(void);
int putchar(int c);
```

如原型所示，getchar( )函数返回一个整数值。也可以指定这个值为char变量，因为这个字符包含于低位字节中（高位字节通常为0）。如果有错，getchar( )返回EOF。

对putchar( )而言，虽然putchar( )带一个整型参数，通常可以用一个字符变元调用它，但只有其低位字节被实际输出到屏幕上。putchar( )函数返回被写入的字符，若操作失败，返回EOF（宏EOF定义于stdio.h中，通常其值为-1）。

下面的程序演示了getchar( )和putchar( ),它从键盘输入字符串并在屏幕上以与原来不同的形式显示，即把原来的大写字母打印成小写字母，把原来的小写字母打印成大写字母。当用户敲入一个‘.’时，该程序结束。

```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
  char ch;

  printf("Enter some text (type a period to quit).\n");
  do {
    ch = getchar();

    if(islower(ch)) ch = toupper(ch);
    else ch = tolower(ch);

    putchar(ch);
  } while (ch != '.');

  return 0;
}
```

8.2.1 getchar( )的问题

getchar( )有一个潜在的问题。正常情况下，getchar( )缓存输入，直到键入了回车键，这称为行缓冲输入，在键入的字符实际传送给程序以前都必须敲入一个回车键。还有，因为getchar( )每次调用时只输入一个字符，行缓冲可使一个或多个字符等在输入队列中，在交互式环境中，这是很恼人的。即使标准C/C++规定getchar( )可以用交互式函数来实现，但它很少见。因而，如果前面的程序有不期望的情况发生，也就不足为奇了。

8.2.2　getchar( )替代物

尽管 getchar( )在一个交互式环境中很有用，它可能不被你的编译器实现。在这种情况下，可以利用其他函数来完成从键盘中读字符的工作。虽然标准C++没有提供确保交互式输入的任何函数，但实际上所有的 C++ 编译器都含有交互式键盘输入函数。尽管这些函数没有为标准 C++ 所定义，但当 getchar( )不能满足程序员需要时，可以使用它们。

有两个最常用的交互式函数：getch( )和 getche( )，它们的原型如下：

```
int getch(void);
int getche(void);
```

对于大多数编译器，这些函数的原型都可在头文件 conio.h 中找到。对某些编译器来说，这些函数前面有一下划线。例如，在微软的 Visual C++ 中，它们称为 _getch( ) 和 _getche( )。

函数 getch( )等待敲键并立即返回其值，但字符并不回送到屏幕上。函数 getche( )和 getch( )的功能相同，但字符要回送到屏幕上。在本书中，有许多需要从键盘读字符的例子，我们在交互式程序中使用 getche( )或 getch( )而不是 getchar( )。然而，如果编译器不支持这种替代函数，或 getchar( )是以交互式实现的，那么必要时可用 getchar( )来替代。

例如，上节的程序可用 getch( )代替 getchar( )，如下所示。

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>

int main(void)
{
  char ch;

  printf("Enter some text (type a period to quit).\n");
  do {
    ch = getch();

    if(islower(ch)) ch = toupper(ch);
    else ch = tolower(ch);

    putchar(ch);
  } while (ch != '.');

  return 0;
}
```

运行这个程序时，每按一次键，它就被立即传送到程序中并以相反的大小写显示出来。输入不再是行缓冲的。虽然本书中的代码没有进一步使用getch( )或getche( )，在你编写的程序中，它们是很有用的。

在写本书时，笔者正在使用微软的Visual C++ 编译器，_getche( )和_getch( )与标准C/C++的输入函数，例如 scanf( )或 gets( )不兼容。你必须使用特殊版本的标准函数，如 cscanf( )或 cgets( )。关于详细情况，请参阅 Visual C++ 用户手册。

8.3 读写字符串

gets( )和puts( )是控制台I/O中更复杂、更高效的函数，它们使你能够读写字符串。

函数gets( )读取从键盘上输入的字符串并把它存放在由其变元所指的地址中，它从键盘读入字符，直到遇到回车键为止。回车键不属于串的一部分，相反，将空结束符放在串尾来代替，并且由gets( )返回。事实上，不能用gets( )来返回回车符（而getchar( )可以返回回车符）。通过在按回车键之前使用退格键，可以更正键入错误。gets( )的原型如下：

```
char *gets(char *str);
```

其中，str是接受用户键入字符的字符数组。gets( )也返回str。下面的程序将一字符串读到数组str中并打印其长度。

```
#include <stdio.h>
#include <string.h>

int main(void)
{
  char str[80];

  gets(str);
  printf("Length is %d", strlen(str));

  return 0;
}
```

当使用gets( )时，需要小心，因为它不对正在接受输入的数组执行边界检查。因此，用户可以键入比数组能够容纳的更多的字符。尽管gets( )对于你将使用的范例程序和简单实用工具是很好的，在商用代码中一般不使用它。它的一个替代物是fgets( )函数，这个函数允许你防止数组越界，我们将在下一章中描述它。

函数puts( )将它的字符串变元写到屏幕上，后跟一新行。其原型为：

```
int puts(const char *str);
```

函数puts( )同printf( )一样，也识别反斜杠代码，如制表符'\t'。对puts( )的调用比printf( )开销小，因为puts( )只能输入字符串，不能输出数字或进行格式转换，因而puts( )用的空间少且速度比printf( )快。因此，函数puts( )经常用于代码优化。如果操作失败，函数puts( )返回EOF，否则返回非负值。然而，当向控制台进行写操作时，通常假定不会发生错误，因此不必跟踪puts( )函数的返回值。下面的语句把hello写到屏幕上：

```
puts("hello");
```

表8.1是基本控制台I/O函数的小结。

表8.1 基本I/O函数

函数	操作
getchar( )	从键盘上读一个字符，等待键入回车键
getche( )	从键盘上读一字符并回送到屏幕上，不必等待键入回车键。它不是标准C/C++中定义的，但是有一个相同的扩展名
getch( )	从键盘上读一个字符且不回送到屏幕上，不必等待键入回车键。它不是标准C/C++定义的，但是有一个相同的扩展名

（续表）

函数	操作
putchar()	向屏幕上写入一个字符
gets()	从键盘读一个字符串
puts()	向屏幕上写一个字符串

下列程序揭示了几个基本控制台I/O函数。该程序是一个非常简单的计算机化字典。它首先提示用户输入一个单词，然后将键入值与机内数据库中的词库进行匹配。匹配成功则打印单词含义。要特别注意程序中的间接引用。记住，dic数组是指向字符串的指针数组。注意，列表由两个空值终止。

```
/* A simple dictionary. */
#include <stdio.h>
#include <string.h>
#include <ctype.h>

/* list of words and meanings */
char *dic[][40] = {
  "atlas", "A volume of maps.",
  "car", "A motorized vehicle.",
  "telephone", "A communication device.",
  "airplane", "A flying machine.",
  "", "" /* null terminate the list */
};

int main(void)
{
  char word[80], ch;
  char **p;

  do {
    puts("\nEnter word: ");
    scanf("%s", word);

    p = (char **)dic;

    /* find matching word and print its meaning */
    do {
      if(!strcmp(*p, word)) {
        puts("Meaning:");
        puts(*(p+1));
        break;
      }
      if(!strcmp(*p, word)) break;
      p = p + 2; /* advance through the list */
    } while(*p);
    if(!*p) puts("Word not in dictionary.");
    printf("Another? (y/n): ");
    scanf(" %c%*c", &ch);
  } while(toupper(ch) != 'N');

  return 0;
}
```

8.4 格式化的控制台I/O

printf( )和scanf( )是两个实现格式化输入输出的函数。它们在用户的控制下以不同格式读写数据。函数printf( )向控制台写数据，函数scanf( )从键盘读数据。两者都可以操作包括字符、字符串和数字在内的任何内嵌数据类型。

8.5 printf( )

函数printf( )的一般形式如下：

```
int printf(const char *control_string, ...);
```

printf( )函数返回写入字符的数目，如果出现一个错误，则返回一个负值。

control_string（控制串）由两种类型的项目组成。第一类由将打印在屏幕上的字符串组成；第二类包括定义其后变元显示方式的格式限定符。格式限定符以一个百分号开头，后跟格式代码。变元列表中的变元数与格式限定符数完全相等，格式限定符与变元按顺序从左到右匹配。例如，下面的printf( )调用：

```
printf("I like %c%s", 'C', "++ very much!");
```

显示

```
I like C++ very much!
```

函数printf( )接受许多种格式限定符，如表8.2所示。

表8.2 printf()的格式限定符

代码	格式
%c	字符
%d	有符号十进制整数
%i	有符号十进制整数
%e	科学表示（小写e）
%E	科学表示（大写E）
%f	十进制浮点数
%g	用%e或%f中较短一个
%G	用%E或%F中较短一个
%o	无符号八进制数
%s	字符串
%u	无符号十进制整数
%x	无符号十六进制数（小写）
%X	无符号十六进制数（大写）
%p	显示一个指针
%n	相关变元必须是整型指针，这个限定符引起至今写入的字符数被放到那个整数中
%%	打印一个百分号

8.5.1 打印字符

打印一个字符时使用%c，结果将匹配字符原样地显示在屏幕上。

打印一个字符串时使用%s。

8.5.2　打印数字

可以使用%d或%i来指示一个有符号十进制数，这两个格式限定符是等价的。由于历史原因，两者都受到支持。打印一个无符号值时使用%u，%f格式限定符显示浮点数。%e 和%E限定符告诉 printf( )以科学记数法显示一个双精度数，科学记数法显示数字的一般形式为：

```
x.ddddE+/-yy
```

如以大写方式显示 E，则用 %E，否则用 %e。

可利用%g或%G格式限定符告诉printf( )使用%f或%e。printf( )可以选择格式限定符以产生最短输出。当适用时，如果想以大写方式显示 E，则用 %G 格式，否则用 %g 格式。下列程序显示了 %g 格式限定符的用法：

```
#include <stdio.h>

int main(void)
{
  double f;

  for(f=1.0; f<1.0e+10; f=f*10)
    printf("%g ", f);

  return 0;
}
```

程序运行结果如下：

```
1 10 100 1000 10000 100000 1e+006 1e+007 1e+008 1e+009
```

用 %o 和 %x 格式可分别以八进制和十六进制方式显示无符号整数。因为十六进制数字系统使用 A-F 代表 10-15，可按小写或大写方式来显示这些字母。如要用大写显示，则用 %X 格式限定符，否则用 %x，如下所示：

```
#include <stdio.h>

int main(void)
{
  unsigned num;

  for(num=0; num<255; num++) {
    printf("%o ", num);
    printf("%x ", num);
    printf("%X\n", num);
  }

  return 0;
}
```

8.5.3　显示一个地址

如果希望显示一个地址，使用 %p。这个格式限定符使得 printf( )可以显示一个机器地址，其格式与计算机所用的寻址类型兼容。下列程序显示变量 sample 的地址。

```
#include <stdio.h>
```

```
int sample;

int main(void)
{
  printf("%p", &sample);

  return 0;
}
```

8.5.4 %n限定符

%n格式限定符不同于其他代码，它并不告诉printf( )显示什么内容，而是将已输出的字符个数放入变元指向的变量中。换句话说，对应于%n格式限定符的值必须是一个指向变量的指针。在printf( )调用返回后，这个变量将包含一个到遇到%n时字符输出的数目。下列程序可使读者更好地理解这个不常用的格式代码。

```
#include <stdio.h>

int main(void)
{
  int count;

  printf("this%n is a test\n", &count);
  printf("%d", count);

  return 0;
}
```

程序结果在显示this is a test后显示数字4。%n格式限定符的主要应用是使用户程序实现动态格式化。

8.5.5 格式修饰符

许多格式限定符都有修饰符用于稍微改变它们的含义。例如，可以规定最小域宽、小数位数及向左对齐。格式修饰符放在百分号与格式代码之间。下面讨论这些修饰符。

8.5.6 最小域宽限定符

百分号和格式代码之间的整数称为最小域宽限定符，它保证输出时用空格填充以达到最小域宽。如果串或数字长度比最小域宽长，它就被完全打印出来。默认填充值为空格。如果想用0来填充，那么在域宽限定符前放一个0。例如，%05d将对不是五位数的输出数字填充0以达到最小域宽。下列程序展示了最小域宽限定符的使用方法：

```
#include <stdio.h>

int main(void)
{
  double item;

  item = 10.12304;

  printf("%f\n", item);
  printf("%10f\n", item);
  printf("%012f\n", item);
```

```
  return 0;
}
```

这个程序生成如下输出：

```
10.123040
 10.123040
00010.123040
```

最小域宽限定符最广泛的应用是生成表，表中数据按列排列。例如，下面的程序范例产生1~19的平方及立方表：

```
#include <stdio.h>

int main(void)
{
  int i;

  /* display a table of squares and cubes */
  for(i=1; i<20; i++)
    printf("%8d %8d %8d\n", i, i*i, i*i*i);

  return 0;
}
```

程序输出如下：

```
       1        1        1
       2        4        8
       3        9       27
       4       16       64
       5       25      125
       6       36      216
       7       49      343
       8       64      512
       9       81      729
      10      100     1000
      11      121     1331
      12      144     1728
      13      169     2197
      14      196     2744
      15      225     3375
      16      256     4096
      17      289     4913
      18      324     5832
      19      361     6859
```

8.5.7 精度限定符

精度限定符跟在最小域宽限定符的后面（如果有一个），由一个点号后跟一个整数组成，其准确含义依赖于它所修饰的数据类型。

当使用%f，%e，%E限定符应用精度限定符于浮点数据时，它确定所显示的小数位数。例如，%10.4f显示一个数，该数至少10个字符宽，精确到小数点后面4位。

如果精度限定符为%g或%G，则表示有效位数。

当修饰字符串时，精度限定符表示最大域宽。例如，%5.7s显示一个最小5个字符但不超过7个字符的字符串。如果串大于最大域宽，则舍去多余的字符。

当修饰整型数时，精度限定符确定每个数字显示的最小位数。在这种情况下，用0来填充以达到所要求的位数。

下面的范例解释了精度限定符的用法：

```
#include <stdio.h>

int main(void)
{
  printf("%.4f\n", 123.1234567);
  printf("%3.8d\n", 1000);
  printf("%10.15s\n", "This is a simple test.");

  return 0;
}
```

程序运行产生如下结果：

```
123.1235
00001000
This is a simpl
```

8.5.8 对齐输出

默认时，所有输出均为右对齐。换句话说，如果域宽比要打印的数据长，那么数据按域的右边界对齐显示。但可以通过在百分号后加一个减号来迫使数据向域左边界对齐。例如%-10.2f表示在10个字符域宽内以左对齐方式显示一个小数点后有两位有效数字的浮点数。

下面的程序范例展示了左对齐的用法：

```
#include <stdio.h>

int main(void)
{
  printf("right-justified:%8d\n", 100);
  printf("left-justified:%-8d\n", 100);

  return 0;
}
```

8.5.9 处理其他数据类型

有两个格式修饰符允许printf( )显示短整型和长整型整数。这些修饰符可应用于d,i,o,u和x的类型限定符。l修饰符告诉printf( )显示长整型数据，例如%ld表示显示一个长整型整数。h修饰符指示printf( )显示短整型数据，例如%hn表示显示无符号短整型整数。

l和h修饰符也可应用于n限定符，表示相应的变元分别是一个指向长整型或短整型数的指针。

如果编译器用标准C++进行编译，那么可以把l修饰符和c格式一起使用，以表示一个宽字符。也可以把l修饰符和s格式一起使用，以表示一个宽字符串。

L 修饰符也可当作浮点限定符 e,f,g 的前缀，在这种情况下表示显示双精度数。

8.5.10　* 和 # 修饰符

除了它的格式限定符外，函数 printf( )支持两个附加的修饰符 * 和 #。

如果 g，G，f，E 或 e 前面有符号 #，就可以确保即使没有小数位也要显示小数点。如果 x 或 X 前面有符号 #，那么十六进制数将带 0x 前缀显示，在 o 前面加上 # 符号会使得打印的数字前补 0，# 不能用于其他任何格式限定符。

最小域宽和精度限定符可以通过变元而不是常数提供给 printf( )。为了实现这一点，我们用 * 作为占位符。当编译器扫描到格式串时，printf( )将 * 与变元串中的参数按顺序匹配。例如，在图 8.1 中，最小域宽是 10，精度是 4，将要显示的值是 123.3。

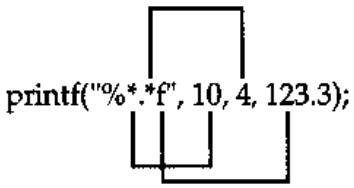

图 8.1　* 如何与其值匹配

下面的程序演示了 # 和 * 修饰符。

```
#include <stdio.h>

int main(void)
{
  printf("%x %#x\n", 10, 10);
  printf("%*.*f", 10, 4, 1234.34);

  return 0;
}
```

8.6　scanf( )

scanf( )是一个通用的控制台输入例程。它能读入各种内嵌类型的数据并自动将其转换为适当的格式。它很像 printf( )的逆操作。scanf( )的原型是：

int scanf(const char *control_string, ...);

scanf( )函数返回成功地赋予了一个值的数据项数。如果出现一个错误，scanf( )返回 EOF。control_string 确定值如何读到变元列表内所指的变量中。

控制串包含三类字符：

- 格式限定符
- 空白字符
- 非空白字符

下面我们分别讨论这三类字符串。

8.6.1 格式限定符

输入格式限定符以百分号开始，它告诉scanf( )下一步要读的数据是什么类型。这些格式代码列于表8.3中。格式限定符从左到右与变元列表中的变元相匹配，我们来看看几个范例。

表8.3 scanf( )格式限定符

代码	含义
%c	读一个字符
%d	读一个十进制整数
%i	读一个十进制、八进制或十六进制格式的整数
%e	读一个浮点数
%f	读一个浮点数
%g	读一个浮点数
%o	读一个八进制数
%s	读一个字符串
%x	读一个十六进制数
%p	读一个指针
%n	接收一个等于到目前为止输入的字符数目的整数
%u	读一个无符号整数
%[ ]	扫描一个字符集
%%	读一个百分号

8.6.2 输入数字

用%d或%i限定符可以读一个整字。用%e，%f或%g可以以标准记数或科学记数法读一个浮点数。

用%o或%x可以读八进制或十六进制的整数，%x有大小写两种方式。当读入十六进制数时，可以选择大小写两种方式来读字母A-Z。

下面的程序读入八进制和十六进制数。

```
#include <stdio.h>

int main(void)
{
  int i, j;

  scanf("%o%x", &i, &j);
  printf("%o %x", i, j);

  return 0;
}
```

当遇到第一个非数字字符时，scanf( )函数停止读数字。

8.6.3 输入无符号整数

用%u格式限定符可以读一个无符号整数。例如，

```
unsigned num;
scanf("%u", &num);
```

读入一个无符号整数并放入num中。

8.6.4 用scanf( )读单个字符

正如本章前面解释的，使用getchar( )或派生函数，可以读单个字符；也可以使用%c格式限定符通过scanf( )达到这一目的。然而，像getchar( )的大多数实现一样，当使用%c限定符时，scanf( )一般进行行缓冲输入方式。然而在交互式环境中，这会带来一些麻烦。

尽管在读其他类型的数据时，空格、制表符及换行符被用做域分隔符；当读单独一个字符时，它们与读其他字符是一样的。例如，若输入“xy,”，下面的代码：

```
scanf("%c%c%c", &a, &b, &c);
```

把x存入a，空格存入b，y存入c。

8.6.5 读字符串

用%s格式限定符，可以使用函数scanf( )从输入流中读入字符串。使用%s时，scanf( )读入字符直到键入空白字符。读入的字符被放入对应参数所指的字符数组中，并以空结束符终止串。当使用scanf( )时，空白符是空格、制表符或换行符、纵向制表符、换页。与以键入回车符终止串的gets( )不同，scanf( )在键入第一个空白符时终止读串。这意味着，不能用scanf( )去读“this is a test”那样的串，因为遇到第一个空白符时输入过程便终止了。下面的代码读入“hello there”，由此可了解%s限定符的用法。

```
#include <stdio.h>

int main(void)
{
  char str[80];

  printf("Enter a string: ");
  scanf("%s", str);
  printf("Here's your string: %s", str);

  return 0;
}
```

结果只显示出串的“hello”部分。

8.6.6 读入地址

使用%p格式限定符可以获得内存地址。这个限定符使得scanf( )读的是CPU使用的格式化地址。例如，下列程序读入一个地址并显示内存地址的内容。

```
#include <stdio.h>

int main(void)
{
  char *p;

  printf("Enter an address: ");
  scanf("%p", &p);
  printf("Value at location %p is %c\n", p, *p);
```

```
  return 0;
}
```

8.6.7 %n限定符

使用%n限定符时，scanf( )将遇到%n以前读过的输入流中的字符数赋给对应变元所指的变量。

8.6.8 使用一个扫描集

scanf( )函数支持一种通用格式的限定符，称为扫描集。扫描集定义一组字符。当用scanf( )处理扫描集时，它将读入一串属于扫描集定义的字符，并将它们赋给与此扫描集中变元对应的字符数组。定义扫描集时，将需要扫描的字符放到方括号中，且开头的方括号必须以百分号作为前缀。例如，下面的扫描集告诉scanf( )只读字符X，Y，Z：

```
%[XYZ]
```

使用扫描集时，scanf( )一边读字符一边将它们放入相应的字符数组，直到遇到第一个不在扫描集中的字符为止。scanf( )完成执行后，数组将包含一个以空格结束的字符串，该字符串由已读出的字符组成。为了弄清楚这是如何工作的，请看下面的程序：

```
#include <stdio.h>

int main(void)
{
  int i;
  char str[80], str2[80];

  scanf("%d%[abcdefg]%s", &i, str, str2);
  printf("%d %s %s", i, str, str2);

  return 0;
}
```

键入123abcdtye并以回车键结束，结果将显示123 abcd tye。由于t不在扫描集中，因此scanf( )遇到t后将终止读入字符到str。余下的字符放入str2中。

可在扫描集中的一个字符前加前缀^来定义求反集，^指示scanf( )接受不被扫描集定义的任何字符。

在大多数实现中，也可用破折号定义接受字符的范围。下例告诉scanf( )接受字符A到Z。

```
%[A-Z]
```

记住，扫描集是区分大小写的。如果想扫描大写和小写字母，必须分别指定它们。

8.6.9 丢弃不想要的空白符

控制字符串中的空白符使scanf( )跳过流中的一个或多个空白符。空白符包括空格、纵向制表符、换页符、制表符或换行符。实质上，控制字符串中的空白符将导致scanf( )读入但不存储从空白符到第一个非空白符的任何数目（包括0）的空白符。

8.6.10 控制字符串中的非空白符

控制字符串中的非空白符导致scanf( )读入并丢弃输入流中的一个匹配字符。例如，“%d，%d”使得scanf( )首先读一个整数，然后读并丢弃一个逗号，最后再读入另一个整数。如果没有找到特定的字符，则scanf( )终止执行。如果希望读并丢弃一个百分符号，那么在控制字符串中书写%%即可。

8.6.11 必须向scanf( )传递地址

所有通过scanf( )接受数据值的变量都必须用它们的地址传递。这意味着所有变元必须是指向用做变元的变量的指针。记住，这是产生按引用调用的方法，它允许函数改变变元的内容。例如，使用下面的scanf( )调用可以读入一个整数并存入变量count中：

```
scanf("%d", &count);
```

字符串被读入字符数组，其中没有任何下标的数组名是数组中的第一个元素的地址。因此，为了把一字符串读入数组str中，需使用

```
scanf("%s", str);
```

在这种情况下，str已经是指针，而不必在前面再添加运算符&。

8.6.12 格式修饰符

像printf( )一样，scanf( )允许修饰它的一些格式限定符。

格式限定符可以包含最大域宽修饰符，它是放在百分号与格式限定符之间的一个整数，它限制了读入字符的最大长度。例如，向str中读入不超过20个字符的串，应使用

```
scanf("%20s", str);
```

如果输入流超过20个字符，则超长部分将丢失。例如，执行上例时若输入：

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

则仅前20个字符或A-T被放入str中，其余字符UVWXYZ丢失（由于最大域宽限定符的缘故）。若这样使用：

```
scanf("%s", str);
```

则UVWXYZ也被置于str中了。在输入流中若遇到空白符，即使没有读到最大长度，输入也将终止。在这种情况下，scanf( )跳到下一个数据。

为了读长整型数据，可以把l放在格式限定符前面。为了读短整型数据，可以把h放在格式限定符前面。这些修饰符也可以与d, i, o, u, x和n格式代码一起使用。

默认时，%f、%e和%g使scanf( )读入浮点数。如果将l置于上述三者之一的前面，则读入双精度数。L告诉scanf( )接受数据的变量是长整型双精度变量。

8.6.13 压缩输入

如果在域的格式码前加上*，则用户就可以告诉scanf( )读这个域，但并不把它赋予任何变量。例如：

```
scanf("%d%*c%d", &x, &y);
```

当键入10，10时，逗号将被读过但不赋予任何变量。赋值压缩在只需要处理输入流的一部分时是十分重要的。