第 3 章

数据的输入和输出

数据的输入/输出(简称 I/O)是程序与用户进行交互的手段,通过输入将用户的数据告知程序,通过输出将程序的运行结果反馈给用户。C 语言本身并不提供 I/O 函数,需要在 #include 预处理命令中包含 stdio.h 头文件。

为了得到准确的输入数据和满足用户的输出需求,可以用格式化输入函数 scanf()和格式化输出函数 printf()来指定具体的格式。此外,还可以使用单字符输入函数 getchar()、单字符输出函数 putchar()、字符串输入函数 gets()和字符串输出函数 puts()等实现数据的输入和输出。

本章将对 scanf()、printf()、getchar()和 putchar()函数进行介绍。关于 gets()和 puts()的使用将在第 6 章中介绍。

3.1 预备知识

3.1.1 缓冲区

计算机在内存空间中预留了一定大小的存储空间用来缓冲输入或输出的数据(图 3-1),这部分预留的空间称为缓冲区。根据其对应的是输入设备还是输出设备,分为输入缓冲区和输出缓冲区。

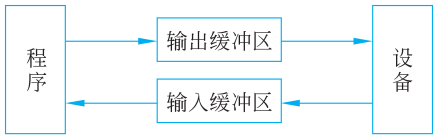


图 3-1 缓冲区示意图

缓冲区对于输入和输出有着很重要的意义,它允许用户输错数据后可以及时地修改,也可以避免可能的数据丢失。

3.1.2 缓冲区的类型

缓冲区实际是一段内存,它的大小由 stdio.h 头文件中的符号常量 BUFSIZ 表示,可通过 BUFSIZ 查看:

```
printf("%d",BUFSIZ);     //缓冲区一般是 512 字节
```

根据系统何时处理缓冲区中的数据,可以把缓冲区分为如下 3 种类型。

1. 全缓冲区

所谓全缓冲区，指只有当缓冲区空间被放满数据时，程序才会开始真正的 I/O 操作。假设缓冲区的大小是 512 字节，哪怕此时缓冲区中已有 511 字节，程序也不会执行 I/O 操作。全缓冲区的典型代表是对磁盘文件的读写。

2. 行缓冲区

所谓行缓冲区，指当用户按下回车后（回车符也会放在缓冲区中），程序才会开始真正的 I/O 操作。行缓冲区的典型代表是 stdin（标准输入）和 stdout（标准输出）。

3. 无缓冲区

所谓无缓冲区，就是没有缓冲的意思，用户输入一个字符，程序就会执行一次操作。无缓冲区的典型代表是 stderr（标准出错情况），这使得出错信息可以尽快地显示出来。

3.1.3　读取缓冲区的数据

当调用标准输入函数时，系统会将用户输入的数据先保存到输入缓冲区，直到缓冲区已满或者用户按下回车后，系统才真正开始从输入缓冲区中读取数据，未读取的部分仍保存在输入缓冲区。此外，如果调用标准输入函数时输入缓冲区中仍有数据，系统会继续从输入缓冲区中读取数据，直到输入缓冲区已空后，才会让用户输入新的数据。可以想象系统对用户说“你先输入，等你输入完毕我再读取”，以及“我先读取你已经输入好的，等我读取完了你再接着输入”的方式体会系统从输入缓冲区中读取数据的过程。

如果在输入过程中，发现之前输入的数据有错误，只要没有按下回车，都可以退回至错误处进行修改，这也是系统设计输入缓冲区的目的之一。

3.1.4　缓冲区的刷新

以下情况会引发缓冲区的刷新：

- 缓冲区满时。
- 执行特定函数刷新缓冲区（如 fflush 语句）。
- 遇到回车时。
- 关闭文件时。

问：如何清空缓冲区中的内容？

清空输入缓冲区是指清除掉输入缓冲区中的所有内容；清空输出缓冲区是把输出缓冲

区中的所有数据立刻输出到显示器屏幕上。其实，如果程序是单进程的，是否清空输出缓冲区对于用户来说一般是感觉不明显的，因为要输出的内容总归会显示到屏幕上，是分批次显示还是一次性显示对于用户来说差别不大。不过，当程序是多进程的，且多个进程都在进行输出时，可能会出现输出错误的问题，此时就可以通过清空输出缓冲区避免输出错误的发生。

C 标准规定 fflush()函数可以用来清空标准输出，语句是 fflush(stdout)，但对于标准输入，它是没有定义的。不过有些编译器也定义了 fflush(stdin)，可以清空输入缓冲区中的内容。

例 3-1 输入缓冲区的作用示例。

这里设计了 3 个示例来展示输入缓冲区的作用。例 3-1-1 没有清空输入缓冲区，例 3-1-2 和例 3-1-3 有清空输入缓冲区的动作。

如表 3-1 所示，例 3-1-2 与例 3-1-1 的区别在于多了一条 fflush(stdin)语句，它在第一条 scanf 语句后清空了输入缓冲区，所以系统在执行第二条 scanf 语句时会要求用户输入新的数据。例 3-1-3 则验证了用户输入的回车符也会保存在输入缓冲区。

表 3-1 输入缓冲区的作用示例

例 3-1-1 代码

```
int main()
{
    int a,b;
    scanf("%d",&a);
    scanf("%d",&b);
    printf("%d,%d\n",a,b);
    return 0;
}
```

例 3-1-2 代码

```
int main()
{
    int a,b;
    scanf("%d",&a);
    //清空缓冲区
    fflush(stdin);
    scanf("%d",&b);
    printf("%d,%d\n",a,b);
    return 0;
}
```

例 3-1-3 代码

```
int main()
{
    int a,b;
    char c;
    scanf("%d",&a);
    //清空缓冲区
    fflush(stdin);
    scanf("%d",&b);
    printf("%d,%d\n",a,b);
    scanf("%c",&c);
    printf("%d",c);
    return 0;
}
```

3 个程序的运行效果分别如图 3-2(a)、图 3-2(b)和图 3-2(c)所示。

```
111 222 333 444
111,222
Process returned 0 (0x0)
```

(a) 例3-1-1的运行效果

```
111 222 333 444
555
111,555
Process returned 0 (0x0)
```

(b) 例3-1-2的运行效果

```
111 222 333 444
555
111,555
10
Process returned 0 (0x0)
```

(c) 例3-1-3的运行效果

图 3-2 例 3-1 的运行效果示例

上面 3 个示例中的第一条 scanf 语句都会让用户进行输入，用户输入“111 222 333 444”并回车后，系统才从输入缓冲区中读走 111 并把它赋给变量 a，此时的输入缓冲区中还剩下“ 222 333 444\n”(即当前缓冲区中的第一个字符是空格，最后一个字符是回车符)。

在例 3-1-1 中，当执行第二条 scanf 语句时，由于输入缓冲区中仍有数据，所以继续从输入缓冲区中读取 222 并把它赋给变量 b，此时的输入缓冲区中还剩下“ 333 444\n”（即当前缓冲区中的第一个字符是空格，最后一个字符是回车符）。

程序结束后，缓冲区被清空。

问题来了

在输入缓冲区中，222 的前面还有一个空格，系统是以何种方式读走这个空格呢？

C 语言规定，隐含使用空格、Tab 符和回车符作为非字符型数据（如整型、浮点型、字符串）之间的分隔符，系统在从输入缓冲区中读取非字符型数据时，会“智能”地读走这些分隔符（包括连续出现的空格、Tab 符和回车符）。因此，在输入时，输入“111　　222”的效果与“111 222”一样。

在例 3-1-2 中，在第一条 scanf 语句之后是 fflush(stdin)语句，其功能是清空输入缓冲区，所以在执行第二条 scanf 语句时，由于输入缓冲区已被清空，于是要求用户继续输入数据。用户输入“555”并回车后，系统才从输入缓冲区中读取 555 并把它赋给变量 b，此时的输入缓冲区中还剩下“\n”。

在例 3-1-3 中，为了验证例 3-1-2 中输入缓冲区还剩下“\n”，定义了一个字符型变量 c，让它从输入缓冲区读走一个字符并显示该字符的 ASCII 码，结果是 10，而回车符的 ASCII 码正好是 10（表 2-5）。

3.2　格式化输入函数 scanf()

3.2.1　scanf()的使用形式

scanf()函数通过标准输入设备（键盘）来获取数据，并将其存放到指定变量所占的内存空间，也就是给变量赋值。它的使用形式如下：

```
int scanf("格式描述",变量地址列表);
```

其中，

- scanf()的返回值类型是 int，代表已读取到的数据个数。
- 格式描述是用户在输入数据时应遵循的格式。
- 变量地址列表指的是要被赋值的变量地址列表（通常用“& 变量名”表示变量地址，“&”在此处是取址符）。

3.2.2　scanf()的格式描述

格式描述是用双引号括起来的字符串，主要包括格式控制符和分隔符两部分，有时也会有一些说明文字或提示信息。格式描述的作用就是提醒用户按照给定的格式“照猫画虎”地

输入数据，而系统在读取数据时，也是按照格式描述一一从输入缓冲区中读取内容的。

例如，格式描述"%d,%f"就是要求用户输入一个整数和一个实数，两个数之间用逗号分隔。再如，对于格式描述"x:%d,y:%d"，如果用户想让系统正确读取到 5 和 96 并赋值，应输入"x:5,y:96"。注意：格式描述中的"x"和"y"不是变量名，它们只是说明信息，在输入时要求用户照样输入。

可以发现，用户在输入时，除了**用实际的数据代替格式控制符**之外，其余部分必须按照格式描述中的内容照样输入。例如，对于 scanf("请输入 x 和 y：%d%d",&x,&y)，若想让 x 的值为 10，y 的值为 20，用户要在键盘上输入"请输入 x 和 y：10 20"后回车，即用 10 代替第一个格式控制符%d，用 20 代替第二个格式控制符%d，其余内容原样输入。显然，为简化输入内容，scanf()中的格式描述应越简单越好，建议不要有除了格式控制符和分隔符之外的其他字符，如果一定要添加说明文字或提示信息，可以用 printf 语句配合 scanf()使用，如：

```
printf("请输入 x 和 y:");    //用 printf 语句进行提示
scanf("%d%d",&x,&y);         //用户只需要输入数据，这里隐含分隔符为空格、Tab 符和回车符
```

【小思考】

如果 scanf()中的格式描述是"请输入 x：%d，请输入 y：%d"，用户应如何把 7 和 18 正确赋值给对应的变量？答案见脚注①。

1. 格式控制符

如图 3-3 所示，格式控制符由%开始并以一个格式字符结束，还可以根据需要在它们之间添加赋值抑制符"*"、域宽"m"或类型长度修正"l/h"等格式修饰符。

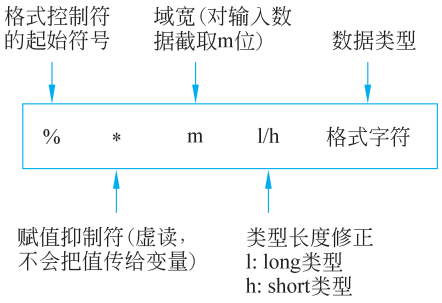


图 3-3　scanf()中的格式控制符的组成

表 3-2 列举了可用于 scanf()的格式字符。

① 用户应输入"请输入 x：7，请输入 y：18"，除了具体数值，其余部分和格式描述中的一模一样。

表 3-2　可用于 scanf()的格式字符

格式字符	说　　明
d	以带符号的十进制整数的形式读取数据，不能组成合法十进制数的字符为非法字符
u	以十进制无符号整数的形式读取数据，不能组成合法十进制数的字符为非法字符
o	以八进制无符号整数的形式读取数据，不能组成合法八进制数的字符为非法字符
X 或 x	以十六进制无符号整数的形式读取数据，不能组成合法十六进制数的字符为非法字符
c	读取一个字符，所有字符都合法
s	读取一个以非空格字符开始，到空格符结束的一个字符串
f、e、g	以小数形式或指数形式读取浮点数，指数形式要满足 e/E 前面有数，后面是整数。不能组成合法浮点数的字符为非法字符

表 3-3 列举了可用于 scanf()的格式修饰符。

表 3-3　可用于 scanf()的格式修饰符

格式修饰符	说　　明
*	赋值抑制符，指定输入项读入数据后不赋值给变量，可理解为虚读一个数据
m	域宽，指定输入数据的最大宽度，遇空格或非法字符后结束
l	加在 d、o 和 x 前面，指定输入类型为 long 型；加在 e、f、g 前面，指定输入类型为 double 型
h	加在 d、o 和 x 前面，指定输入类型为 short 型

格式控制符指定系统要按照何种数据类型以及何种修饰方式来读取数据，对于同样的用户输入，不同的格式控制符会导致不同的读取结果。例如，如果输入缓冲区的内容是“178as\n”，%o 的格式控制符会读取 17 作为输入数据（因为 8 不是八进制的有效数字），%d 的格式控制符会读取 178 作为输入数据（因为 a 不是十进制的有效数字），而%x 的格式控制符会读取 178a 作为输入数据（因为 s 不是十六进制的有效数字）。

例 3-2　scanf()中不同格式控制符的作用示例。

这里设计了两个示例展示不同格式控制符的作用。

如表 3-4 所示，例 3-2-1 测试了不同格式字符对数据读取的影响，例 3-2-2 测试了不同格式修饰符对数据读取的影响。

表 3-4　不同格式控制符对数据读取的影响

例 3-2-1 代码

```
int main()
{
    int a;
    float b;
    char c;
    scanf("%o,%f,%c",&a,&b,&c);
    printf("a=%d,b=%f,c=%c\n",a,b,c);
    return 0;
}
```

例 3-2-2 代码

```
int main()
{
    int a,b;
    double c;
    scanf("%3d%*2d%d%lf",&a,&b,&c);
    printf("a=%d,b=%d,c=%lf", a,b,c);
    return 0;
}
```

两个程序的运行效果分别如图 3-4(a)和图 3-4(b)所示。

```
10,54.6,*
a=8,b=54.599998,c=*
```

(a) 例3-2-1的运行效果

```
1234567890
999
a=123,b=67890,c=999.000000
```

(b) 例3-2-2的运行效果

图 3-4　例 3-2 的运行效果示例

在例 3-2-1 中，用户输入“10,54.6,*”并回车后，系统按照格式描述，先在输入缓冲区中按%o 读走 10(不能读作“一十”)并赋值给变量 a(逗号不是八进制数的有效数字，所以停止读取数据)，此时的输入缓冲区中还剩下“,54.6,*\n”。然后系统按照格式描述，在读走分隔符','后按%f 读走 54.6 并赋值给变量 b，此时输入缓冲区中还剩下“,*\n”。类似地，系统按照格式描述，在读走分隔符','后按%c 读走字符'*'并赋值给变量 c。printf 语句在输出变量 a 的值时是按照%d 即十进制数输出的，所以输出的数值是 8。变量 b 的值有误差，这是因为浮点数在保存时就有误差的缘故。

在例 3-2-2 中，用户输入“1234567890”并回车后，系统按照格式描述，在输入缓冲区中按%3d 读取 3 位数字的宽度即 123 并赋值给变量 a，此时的输入缓冲区中还剩下“4567890\n”。接着，系统按照格式描述，按%*2d 读取 2 位数字的宽度即 45，但并不进行赋值，也就是虚读。然后，系统按照格式描述，按%d 读取 67890 并赋值给变量 b，此时输入缓冲区中还剩下“\n”。最后系统按%lf 读取数据，但此时的输入缓冲区中已无数据(回车符隐含是非字符型数据的分隔符，系统会自动读走)，所以要求用户继续输入。用户输入“999”并回车后，系统读走 999 并赋值给变量 c。

2. 分隔符

分隔符的作用是分隔数据。当 scanf()从输入缓冲区中读取数据时，遇到分隔符或非法字符时会停止数据的读取，并将已读取的数据赋值给对应变量。

scanf()隐含**空格、Tab 符('\t')和换行符**作为非字符型数据之间的分隔符，也可指定某个符号(如逗号、冒号)作为分隔符。但对于字符型数据，空格、Tab 符和换行符将不再作为分隔符，而是作为有效字符被读取。

如果要求用户使用空格或回车符等隐含分隔符作为非字符型数据的分隔，在格式描述中可以省略隐含分隔符。例如格式描述"%d%d%d"，用户在输入数据时，既可以用空格进行分隔，也可以用 Tab 符或换行符进行分隔。

注意

当用 scanf 语句读取非字符型数据时，它前面的空格、Tab 符和回车符都是隐含分隔符，系统会自动读走它们。但在读取字符型数据时，所有字符都是有效字符。

例 3-3　分隔符和非法字符对读取数据的影响示例。

如表 3-5 所示，例 3-3-1 测试了分隔符对读取数据的影响，例 3-3-2 测试了非法字符对读取数据的影响。

表 3-5　分隔符和非法字符对读取数据的影响

例 3-3-1 代码	例 3-3-2 代码

int main()
{
　　int a,b;
　　float c;
　　char d;
　　scanf("%d%d%f%c",&a,&b,&c,&d);
　　printf("a=%d,b=%d,c=%f,d=%d",a,b,
c,d);
　　return 0;
}
	int main()
	{
	　　int a=0,c=0;
	　　float b=0;
	　　int n;　　//n 是 scanf()读取的数据个数
	　　n=scanf("%o%f%d",&a,&b,&c);
	　　printf("a=%d,b=%f,c=%d\n",a,b,c);
	　　printf("读取了%d 个数据",n);
	　　return 0;
	}

两个程序的运行效果分别如图 3-5(a)和图 3-5(b)所示。

```
18      981
270 *
a=18,b=981,c=270.000000,d=32
```

(a) 例3-3-1的运行效果

```
17890.2e3.56
a=15,b=890200.000000,c=0
读取了2个数据
```

(b) 例3-3-2的运行效果

图 3-5　例 3-3 的运行效果示例

在例 3-3-1 中，用户输入“18”后按下 Tab 键再输入“981”并回车后，系统按照格式描述，用%d 读取 18 并赋值给变量 a(Tab 符对于%d 来说是分隔符，会停止读取数据)，输入缓冲区中还剩下“\t981\n”，接着按%d 读取数据时会自动读走隐含分隔符'\t'，再读走 981 并赋值给变量 b(回车符对于%d 来说是分隔符，会停止读取数据)。接着按%f 读取数据时会自动读走隐含分隔符'\n'，然后输入缓冲区中已没有数据，所以用户再次输入“270 *”并回车后，系统读走 270 并赋值给变量 c(对于%f 来说空格是分隔符)，此时的输入缓冲区中还剩下“ *\n”(第一个字符是空格)。最后按%c 读取一个空格并赋值给变量 d(对于%c 来说空格是有效字符)。printf 语句中输出的是变量 d 的 ASCII 码，结果是 32，而这正是空格的 ASCII 码(参见表 2-5)。

在例 3-3-2 中，所有变量在定义时都被赋初值为 0。用户输入“17890.2e3.56”并回车后，系统按照格式描述，用%o 读取 17 并赋值给变量 a(在八进制数中，8 是非法数字，停止读取数据)，输入缓冲区中还剩下“890.2e3.56\n”。接着按%f 读走 890.2e3 并赋值给变量 b(实数的指数形式中，e 后面必须是整数，所以 e 后面出现的小数点是非法字符)，输入缓冲区中还剩下“.56\n”。接着按%d 读取数据但不成功，因为'.'对于%d 来说是非法字符。至此，scanf 总共读取了 2 个数据(变量 a 和变量 b 的值)，未能给变量 c 赋值(c 仍是原值 0)，所以 scanf()的返回值是 2。

问题来了

已知有语句 int a; char b;，程序运行时，用户会用隐含分隔符输入 78 和'*'，如“78*”或“78↙*”(用↙表示回车)，应该怎么写 scanf 语句才能把 78 赋值给变量 a，把'*'赋值给变量 b?

这里提供两种方法：

(1) scanf("%d%c%c",&a,&b,&b)。

(2) scanf("%d%*c%c",&a,&b)。

第一种方法：读两次%c，第一次的%c 会读走分隔符并赋值给变量 b，第二次的%c 会读走'*'并赋值给变量 b，相当于覆盖了前一次读取的分隔符。

第二种方法：用赋值抑制符虚读走分隔符。

3.2.3 scanf()的变量地址列表

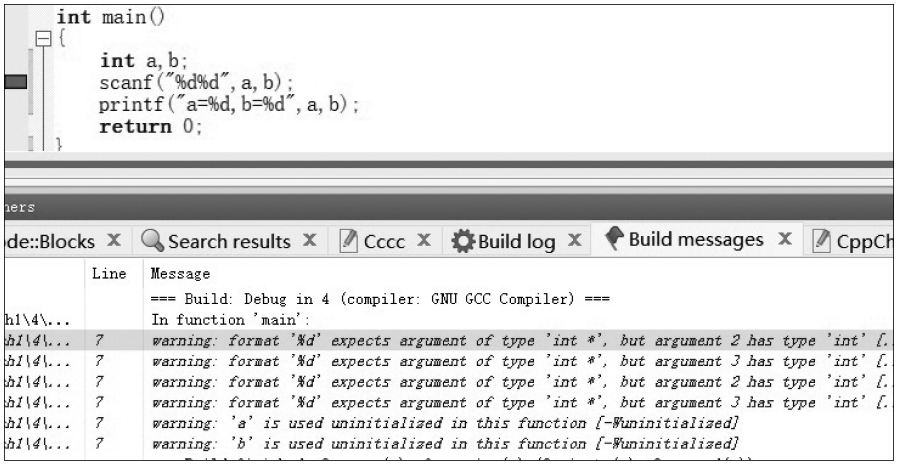
想象一下自己是一名快递员，若要把包裹送到小 a 的家，是不是要知道小 a 家的地址？那小 a 家的地址是多少呢？就是 &a。所以，scanf()中使用的是变量地址就理所当然了。

变量地址列表中的变量类型和个数与格式描述中的格式控制符类型和个数是一一对应的，即用%d、%o、%u、%x 格式读取的数据要放在整型变量所占的空间，用%f 格式读取的数据要放在 float 型变量所占的空间，用%lf 格式读取的数据要放在 double 型变量所占的空间等。

问题来了

已知 int a, b;语句，输入数据时如果写成 scanf("%d%d",a,b);语句，系统会提示出错吗？

对于内存来说，地址就是一个个的编码，如果把输入语句写成 scanf("%d%d",a,b)，编译器会把 a 值和 b 值当作地址，不会报错但会给出警告(图 3-6(a))。但是在运行时，由于用 a 值和 b 值所表示的地址编码很可能并不在系统为 C 程序分配的可用内存空间范围内(表 2-1)，所以当用户输入数据后，系统会因无法把数据存储到指定的地址而非正常退出(图 3-6(b))。



(a) 警告信息为：%d对应的应该是整型变量的地址，但代码中给的是整型变量的值

(b) 输入数据后非正常退出（正常退出应返回0）

图 3-6 scanf()中的变量地址列表中不能是变量值

【小建议】

检查程序错误的一个小技巧

运行程序后，如果程序非正常退出（如图 3-6(b)的箭头所示，返回值不为 0），首先就要检查 scanf 语句，看是不是忘了在变量名前面加上取址符 &。

3.3　格式化输出函数 printf()

3.3.1　printf()的使用形式

printf()函数通过标准输出设备（显示器屏幕，简称屏幕）输出数据，其作用是按照用户指定的格式，将信息显示到屏幕上。printf()的使用形式有两种：

```
(1)printf("字符串");
(2)printf("格式描述",输出参数列表);
```

第一种形式是输出双引号里面的字符串，通常用于提示信息的显示。

第二种形式与 scanf()差不多，只是在格式描述上有些许差异，输出参数列表是以逗号分隔的若干输出项，可以是常数、变量或表达式。

3.3.2　printf()的格式描述

格式描述是一个用双引号括起来的字符串，主要包含提示信息和格式控制符。格式描述的输出规则如下：

（1）从左到右依次输出。

（2）遇到格式控制符（如%d 或%f），将输出参数列表中与之对应的输出项按格式控制符的格式输出。

（3）遇到转义字符，输出它所代表的转义含义（如'\n'表示换行）。

（4）遇到其他字符，原样输出。

1. 格式控制符

如图 3-7 所示，格式控制符由%开始并以一个格式字符结束，还可以根据需要在它们之间添加在有符号的正数前面显示正号的“+”、左对齐输出的“-”、空位补 0 的“0”、域宽“m”、保留小数点位数“n”、八进制或十六进制数显示前导的“#”或类型长度修正“l/h”等格式修饰符。

表 3-6 列举了可用于 printf()的格式字符。

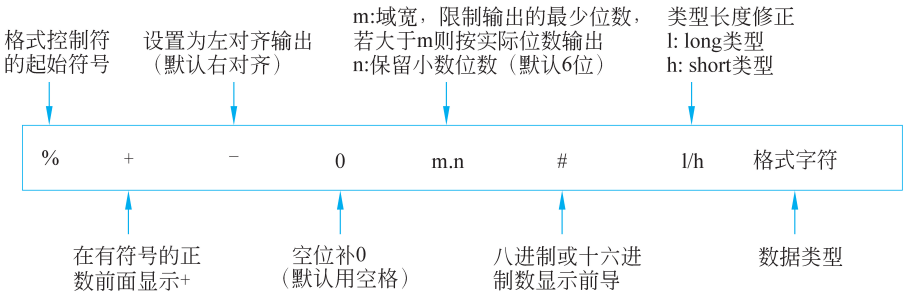


图 3-7　printf()中的格式控制符的组成

表 3-6　可用于 printf()的格式字符

格 式 字 符	说　　明
d	输出带符号的十进制整数(正数不输出符号)
u	输出十进制无符号整数
o	输出八进制无符号整数(不输出前导 0)
X 或 x	输出十六进制无符号整数(不输出前导 0X 或 0x) 用 X 时,十六进制数中的 A～F 以大写形式输出,用 x 时,则以小写形式输出
c	输出一个字符
s	输出一个字符串
f	以小数形式输出浮点数,隐含保留小数点后 6 位
E 或 e	以规范化指数形式输出浮点数,指数部分有正负号,隐含小数部分 6 位
G 或 g	选用%f 或%e 格式输出宽度较短的一种形式,不输出无意义的 0

表 3-7 列举了可用于 printf()的格式修饰符。

表 3-7　可用于 printf()的格式修饰符

格式修饰符	说　　明
+	指定在有符号数的正数前面显示正号(+),不对无符号数添加
#	在八进制和十六进制数前显示前导 0 或前导 0x,需配合对应的格式字符
m	域宽,限制输出的最少位数,若输出的实际位数超过 m,按实际输出,否则在空位补空格。对于浮点数的小数形式,输出位数包括整数部分、小数点和小数部分。对于浮点数的指数形式,还要包括 e 及后面的正负号和指数
n	按照四舍五入的方法保留小数点后 n 位小数,如果不写,默认保留 6 位小数
-	指定输出数据左对齐,默认是右对齐
0	只能用于右对齐时的空位补 0,即只能在数据前面补 0
l	加在 d、o 和 x 前面,指定输出类型为 long 型;加在 e、f、g 前面,指定输出类型为 double 型
h	加在 d、o 和 x 前面,指定输出类型为 short 型

注意

不可以在 scanf()的格式描述中指定浮点数的输入精度，如 scanf("%4.2f",&a)是错误的，这一点与 printf()完全不同。可以指定输出数据的精度，但不能指定输入数据的精度。

例 3-4　printf()中不同格式控制符的作用示例。

```
int main()
{
    int a=12,b=28902,c=672;
    double d=123.456789126;
    float f=54.678;
    printf("不同进制的输出:\n");              //输出提示信息
    printf("%+d,%+#o,%+#x\n",a,a,a);      //同一个整数用不同的进制输出并显示前导
    printf("右对齐方式下的域宽测试\n");
    printf("a=%4d,b=%4d,c=%04d\n",a,b,c);      //右对齐时在前面补位,空位可以补 0
    printf("左对齐方式下的域宽测试\n");
    printf("a=%-4d,b=%-4d,c=%-04d\n",a,b,c); //左对齐时只能在后面补空格
    printf("测试浮点数的域宽与精度\n");
    printf("f=%9.2f,d=%.7lf\n",f,d);       //域宽包括浮点数的整数、小数点与小数部分
    printf("格式字符与变量类型不匹配时\n");
    printf("a=%f,f=%d\n",a,f);             //不匹配时出现数据错误
    printf("格式控制符超过参数个数时\n");
    printf("a=%d,b=%d,c=%d\n",a,b);
    printf("格式控制符少于参数个数时\n");
    printf("a=%d,b=%d\n",a,b,c);
    printf("输出%%的方法\n");                 //用%%可以输出%
    printf("%d%%\n",a);
    printf("参数可以是表达式,会先计算表达式的值再输出\n");
    printf("%d",a*2);
    return 0;
}
```

程序的运行效果分别如图 3-8 所示。

```
不同进制的输出:
+12,014,0xc
右对齐方式下的域宽测试
a=  12,b=28902,c=0672
左对齐方式下的域宽测试
a=12  ,b=28902,c=672
测试浮点数的域宽与精度
f=    54.68,d=123.4567891
格式字符与变量类型不匹配时
a=0.000000,f=-1073741824
格式控制符超过参数个数时
a=12,b=28902,c=0
格式控制符少于参数个数时
a=12,b=28902
输出%的方法
12%
参数可以是表达式，会先计算表达式的值再输出
24
```

图 3-8　例 3-4 的运行效果示例

由此可以总结使用 printf()时的注意事项如下。

(1) 在不同进制的输出中,“+”只针对有符号数有效。

(2) 右对齐方式下,若输出数值不够域宽,会在前面的空位补空格或 0。输出数值达到域宽要求时,按实际输出。

(3) 左对齐时,若输出数值不够域宽,会在后面的空位补空格。“0”补位符不起作用。

(4) 当输出数据为浮点数时,输出位数是包括整数部分、小数点和小数部分的,如果输出位数不够域宽,会相应地补位,具体补位方法与(2)和(3)项相同。

(5) 格式控制符一般要与参数的数据类型一一对应,否则会出现数据错误。但对于 char 型变量,%d 是输出它的 ASCII 码值,%c 是输出它的字符形式。

(6) 格式控制符的个数超过输出参数个数时,对于多出的格式控制符,系统会输出一些无意义的数据。

(7) 格式控制符的个数少于输出参数个数时,不输出多余的参数。

3.4 单字符 I/O 函数

3.4.1 单字符输入函数 getchar()

getchar()的功能是从输入缓冲区中读取一个字符,使用形式是:

```
a=getchar();          //把输入的字符赋值给字符变量 a
```

a=getchar()与 scanf("%c",&a)的作用完全相同,读者可任意选用。

3.4.2 单字符输出函数 putchar()

putchar()的功能是输出一个字符到屏幕上,使用形式是:

```
putchar(a);          //把字符变量 a 的值输出到屏幕上
```

putchar(a)与 printf("%c",a)的作用完全相同,读者可任意选用。

3.5 编程实战

实战 3-1 用户输入一个 4 位的整数,请输出各个位上数字的和。

输入示例:

```
1562
```

输出示例:

```
14
```

【问题分析】

(1) 程序需要几个数据？它们应该是什么类型？

(2) 怎样拆分各个位上的数字及求和？

【程序设计】

(1) 首先看怎样拆分各个位上的数字。

第一种方案，把用户输入的数据当成一个整体看待，先用%d 读取数据，然后想办法拆分成个位、十位、百位和千位上的数字。如此，至少需要 5 个整型变量，分别存储输入的数据和 4 个位上的数字。

至于个位，以 1562 为例，1562%10 的结果就是个位。

至于十位，用 1562/10%10 即可得到。因为 1562/10 是 156(整数相除的结果仍是整数)。

百位和千位的求法与十位相似，只需要 1562/100%10 和 1562/1000 即可。

读者们不妨找一下求某位上数字的规律，当学会循环后，就可以用这个规律得到每一位上的数字。

第二种方案，把用户输入的数据当成 4 个字符读取，将它们分别减去'0'进行数值化，就可得到个位、十位、百位和千位上的数字(两个字符相减，得到的是它们的 ASCII 码的差值，如'5'－'0'＝5)。如此，需要 4 个字符变量。

(2) 把所有位上的数字相加，就可得到它们的和。

(3) 这个和是否需要存放到内存中？如果需要保存，是定义一个新的变量，还是借用某个已有的变量？

【程序实现】

表 3-8 分别是两种方案的代码，它们都是直接输出各个位上数字的和，程序运行结果如图 3-9 所示。

表 3-8　实战 3-1 的两种实现方案

方　案　一

```
int main()
{
    int a,ones,tens,hundreds,thousands;
    scanf("%d",&a);
    ones=a%10;
    tens=a/10%10;
    hundreds=a/100%10;
    thousands=a/1000;
    printf("%d", ones+tens+hundreds+
thousands);
    return 0;
}
```

方　案　二

```
int main()
{
    char ones,tens,hundreds,thousands;
    scanf("%c%c%c%c",&ones,&tens,&hundreds,
&thousands);
     printf ( "%d", ones + tens + hundreds +
thousands-4 * '0');
    return 0;
}
```

```
3471
15
Process returned 0 (0x0)
```

图 3-9　实战 3-1 的运行结果示例

就本例而言，方案二用字符的形式分别读取各位数字会更方便，但当改为“用户输入不超过 4 位的整数”时，方案二就需要用循环来配合读取字符，而方案一不需任何改动就可使用。

实战 3-2 妈妈问大宝写了多久的作业，大宝会按“h:m”的方式回答，如“1:16”就表示他学习了 1 小时 16 分钟。请编写程序，输出大宝一共学习了多少分钟。

输入示例：

```
1:16
```

输出示例：

```
大宝一共学习了 76 分钟
```

【问题分析】

(1) 程序需要几个数据？它们应该是什么类型？

(2) 用户输入数据用的是什么分隔符？

(3) 怎样把小时换算成分钟？

【程序设计】

(1) 需要两个整型变量 hour 和 min，分别表示小时和分钟。

(2) 用户用冒号分隔数据，所以 scanf()的格式描述里也要用冒号作为分隔符。

(3) 1 小时＝60 分钟。

【程序实现】

```
int main()
{
    int hour,min;
    scanf("%d:%d",&hour,&min);
    printf("大宝一共学习了%d分钟",hour*60+min);
    return 0;
}
```

程序运行结果如图 3-10 所示。

```
2:15
大宝一共学习了135分钟
Process returned 0 (0x0)
```

图 3-10 实战 3-2 的运行结果示例

习题

一、单项选择题

1. 已知有 int a,b; scanf("x＝%d,y＝%d",&a,&b);语句，用户若希望把 10 赋给变

量 a，把 20 赋给变量 b，下面选项正确的是(　　)。

A. x=10,y=20　　B. 10,20　　C. a=10,b=20　　D. 10 20

2. 已知有 int a=10;语句，则 printf("%d,%o,%x",a,a,a);语句的输出结果是(　　)。

A. 10,012,0xa　　B. 10,10,10　　C. 10,12, 'a'　　D. 10,12,a

3. 已知有 int a=28;语句，则 printf("|%-04d|",a);语句的输出结果是(　　)。

A. |2800|　　B. |28 |　　C. |0028|　　D. | 28|

4. 已知有 int a=2890;语句，则 printf("%3d",a);语句的输出结果是(　　)。

A. 289　　B. 290　　C. 890　　D. 2890

5. 已知有 int a; char b;语句，运行程序时用户会输入“89 t”，若要把 89 赋给 a，把't'赋给 b，下面输入语句错误的是(　　)。

A. scanf("%d%*c%c",&a,&b);　　B. scanf("%d%c%c",&a,&b,&b);

C. scanf("%d%c",&a,&b);　　D. scanf("%d %c",&a,&b);

6. 已知有 int a; float b;语句，下面选项能正确输入数据的是(　　)。

A. scanf("%3d%6f",&a,&b);　　B. scanf("%d%d",&a,&b);

C. scanf("%d%6.2f",&a,&b);　　D. scanf("%d%lf",&a,&b);

7. 已知有 int a, b; scanf("%d: %d",&a,&b);语句，下面选项能正确输入数据的是(　　)。

A. 14 789　　B. 16,65　　C. 167: 2　　D. %17: %78

8. 用户输入 goodbye，下面程序的运行结果是(　　)。

```
int main()
{   char a,b;
    a=getchar();
    b=getchar();
    putchar(a+2);
    putchar(b+4);
    return 0;
}
```

A. go　　B. is　　C. oy　　D. a+2b+4

9. 已知有 int a=10,b=20;语句，则 printf("a+b=%d",a+b);语句的输出结果是(　　)。

A. 30　　B. a+b=30　　C. a+b=%30　　D. 10+20=30

10. 已知有 int a=0; float b=0; scanf("%d%f",&a,&b);语句，运行时用户输入“12e6”，下面选项描述正确的是(　　)。

A. a 的值是 1，b 的值是 2e6

B. a 的值是 12，b 的值是 e6

C. a 的值是 12，b 的值是 0

D. a 的值是 12e6，光标闪动等待用户继续输入

11. 已知有 float a=123.45678;语句，则 printf("%-8.2f",a);语句的输出结果是(　　)。这里用□代替空格。

A. 123.46□□　　B. □□123.46　　C. -123.46□□　　D. □□-123.46

12. 已知有 int a; float b;语句,关于 scanf()和 printf(),下面选项正确的是(　　)。

A. scanf("%3d",&a)和 printff("%3d",a)中的%3d 的作用相同

B. printf("%.0f",b)可实现以四舍五入的方式输出数据的整数部分

C. 可以用 scanf("%4.2f",&b)的方式指定输入数据的精度

D. scanf("%－05d",a),当 a 的位数不够 5 位时,会在后面补 0,以实现左对齐

13. 下面选项能正确输出%的是(　　)。

A. printf("%");　B. printf('%');　C. printf("\%");　D. printf("%%");

14. 已知有 int a,b,c;scanf("%d%d%d",&a,&b,&c);语句,用户的输入不能正确地把 10 赋给 a,把 20 赋给 b,把 30 赋给 c 的选项是(　　)。用↙代表回车,□代表空格。

A. 10□20↙
30↙
30↙　B. 10↙
20↙　C. 10↙
20□30↙　D. 10,20,30

15. 已知有 char a='d';语句,则 putchar(a);语句与 putchar('a');语句的输出结果是(　　)。

A. a'a'　B. aa　C. da　D. dd

二、编程题

1. 小明开了一家奶茶店,每月房租是 r 元,材料成本是 m 元,月底共进账 t 元(r、m 和 t 都不超过 10 万元)。他现在请你帮忙计算房租和材料成本加起来占进账的百分比(保留小数点后一位)。

输入示例:

```
2000 680 5800
```

输出示例:

```
房租和材料成本占 46.2%
```

2. 用户按 yyyy/mm/dd 的方式输入日期(保证输入数据正确),请转换成 yyyy 年 mm 月 dd 日的方式输出。

输入示例:

```
2009/5/12
```

输出示例:

```
2009 年 05 月 12 日
```

3. 假设 1 美元可以兑换 r 元人民币,用户输入汇率 r 和要兑换的美元数 a(r 和 a 都不超过 50000),输出可兑换的人民币(保留小数点后两位)。

输入示例:

```
6.7891,200
```

输出示例：

```
1357.82
```

4. 用户输入 4 个大小写字母，请将其中的大写转换为小写，小写则转换为大写后输出。
输入示例：

```
abkD
```

输出示例：

```
ABKd
```