第3章 chapter 3

数据的输入与输出

在用计算机解决实际问题时，一般都伴随有一些初始数据的输入以及结果的输出，因此，一个程序一般都要包含数据的输入与输出过程。数据的输入与输出是程序与用户之间的一个界面。

数据的输入与输出应包括以下几项：

(1) 用于输入或输出的设备；

(2) 输入或输出数据的格式；

(3) 输入或输出的具体内容。

在C语言中，提供了用于输入与输出的函数，在这些函数中，键盘是标准输入设备，显示器是标准输出设备。因此，在没有指定输入或输出设备时，其输入设备是键盘，输出设备是显示器。

另外要注意，如果在程序中要使用C语言所提供的输入函数或输出函数，则在使用前(即在程序的开头)应该使用包含命令：

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

将C语言中输入与输出的库函数包含进来。

本章具体介绍C语言中的输入与输出函数。

3.1 格式输出函数

3.1.1 基本的格式输出语句

在C语言中，格式输出语句的一般形式如下：

```
printf("格式控制",输出表);
```

其中printf()是C编译系统提供的格式输出函数。格式控制部分要用一对双撇号括起来，它用于说明输出项目所采用的格式。输出表中的各项目指出了所要输出的内容。

在格式控制中，用于说明输出数据格式的格式说明符总是以%开头，后面紧跟的是具体的格式。用于输出的常用格式说明符有以下几种。

1. 整型格式说明符

整型格式说明符用于说明整型数据的输出格式。在C语言中，整型常量可以用十进制形式、十六进制形式以及八进制形式三种形式表示，因此，对于整型数据的输出也具有这三种格式。

(1) 十进制形式

以十进制形式输出整型数据。其格式说明符为：

%d 或 %md　　用于基本整型

%ld 或 %mld　　用于长整型

%hd 或 %mhd　　用于短整型

%u 或 %mu　　用于无符号基本整型

%lu 或 %mlu　　用于无符号长整型

%I64d　　用于64位超长整型

(2) 八进制形式

以八进制形式输出整型数据。其格式说明符为：

%o 或 %mo　　用于基本整型

%lo 或 %mlo　　用于长整型

(3) 十六进制形式

以十六进制形式输出整型数据。其格式说明符为：

%x 或 %mx　　用于基本整型

%lx 或 %mlx　　用于长整型

在以上各种整型格式说明符中，m表示输出的整型数据所占总宽度(即列数)。当实际数据的位数不到m位时，数据前面将用空格补满。如果在格式说明符中没有用m来说明数据所占的宽度，则以输出数据的实际位数为准。如果在格式说明符中说明了宽度m，但实际输出的数据位数大于m，则也以输出数据的实际位数为准进行输出。

2. 实型格式说明符

实型格式说明符用于说明实型数据的输出格式。

如果以十进制数形式输出实型数据，则其格式说明符为：

%f 或 %m.nf

如果以指数形式输出实型数据，则其格式说明符为：

%e 或 %m.ne

在输出实型数据时，格式说明符中的m表示整个数据所占的宽度，n表示小数点后面所占的位数。如果在小数点后取n位后，所规定的数据宽度m不够输出数据前面的整数部分(包括小数点)，则按实际的位数进行输出。

需要指出的是，在C语言中，用于输出单精度实型数据与双精度实型数据格式说明符是一样的。但对于双精度实型数据也可以用格式说明符%lf与%le输出。

3. 字符型格式说明符

字符型格式说明符用于说明字符型数据的输出格式。其格式说明符为：

%c 或 %mc

其中 m 表示输出的宽度，即在这种情况下，在输出字符的前面将要补 m－1 个空格。

下面对各种基本类型数据的格式输出作几点说明：

(1) 输出表中可以有多个输出项目，但各输出项目之间要用“,”分隔。各输出项目可以是常量、变量以及表达式。

(2) 格式输出函数中的“格式控制”是一个字符串，其中每一个%后面的字符是格式说明符，用于说明相应输出数据的输出格式，而每一个格式说明符的结束符分别为 d(整型)、f(实型)、c(字符型)、s(字符串，将在 9.3.3 节中介绍)。而格式控制中除格式说明符外的其他字符将按原样输出。

例 3.1 设有以下程序：

```
#include<stdio.h>
main()
{  int a, b;
   float  x, y, s;
   a=34; b=-56;
   x=2.5f; y=4.5f; s=x*x+y*y;
   printf("a=%d,b=%d\n", a, b);
   printf("x=%6.2f,y=%6.2f,s=%6.2f\n", x, y, s);
}
```

这个程序经编译连接后，运行输出的结果为(□表示空格)：

```
a=34,b=-56
x=□□2.50,y=□□4.50,s=□26.50
```

在上述程序中的第一个格式输出语句：

```
printf("a=%d,b=%d\n", a, b);
```

中，其输出顺序为：输出字符串“a＝”，以%d 的格式输出变量 a 的值，输出字符串“,b＝”，以%d 的格式输出变量 b 的值，最后输出一个换行(即下一次的输出将另起一行)。

(3) 格式输出函数的执行过程如下：

首先，在计算机内存中开辟一个输出缓冲区，用于存放输出项目表中各项目数据。

然后，依次计算项目表中各项目(常量或变量或表达式)的值，并按各项目数据类型应占的字节数依次将它们存入输出缓冲区中。

最后，根据“格式控制”字符串中的各格式说明符依次从输出缓冲区中取出若干字节的数据(如果是非格式说明符，则将按原字符输出)，转换成对应的十进制数据进行输出。其中从输出缓冲区中取多少个字节的数据是按照对应格式说明符说明的数据类型。例

如，格式说明符%hd为短整型，应依次取2个字节；格式说明符%ld为长整型，应依次取4个字节；格式说明符%f用于输出时为双精度型(以后会知道，在C语言中，实型数据均转换成双精度计算)，应依次取8个字节等。

下面举例说明格式输出语句的执行过程。

例3.2 设有如下C程序：

```
#include<stdio.h>
main()
{  int xx, yy, zz;
   xx=1; yy=-65535; zz=1;
   printf("xx=%ld, yy=%ld, zz=%ld\n", xx, yy, zz);
   printf("xx=%hd, yy=%hd, zz=%hd\n", xx, yy, zz);
   printf("xx=%d, yy=%d, zz=%d\n", xx, yy, zz);
}
```

该程序运行的结果如下：

```
xx=1, yy=-65535, zz=1
xx=1, yy=1, zz=1
xx=1, yy=-65535, zz=1
```

在这个程序中，为整型变量xx赋的值为1，在计算机内存中占4个字节，其十六进制补码(正数的补码为原码本身)表示为0x00000001；为整型变量yy赋的值为－65535，在计算机内存中占4个字节，其十六进制补码(负数的补码为反码加1)表示为0xffff0001；为整型变量zz赋的值为1，在计算机内存中占4个字节，其十六进制补码(正数的补码为原码本身)表示为0x00000001。在计算机中，存储的基本单位是字节，一般来说，如果一个数据占多个字节时，则数据的低字节部分存放在存储空间的后面，而数据的高字节部分存放在存储空间的前面。因此，整型变量xx，yy，zz经赋值后，这三个整型数在计算机中的存放顺序如下(每个数据占8位十六进制位，每两个十六进制位为一个字节，并且，低字节在后，高字节在前)：

xx：01 00 00 00　　对应十六进制数 0x00 00 00 01

yy：01 00 ff ff　　对应十六进制数 0xff ff 00 01

zz：01 00 00 00　　对应十六进制数 0x00 00 00 01

现在考虑第一个格式输出语句：

```
printf("xx=%ld, yy=%ld, zz=%ld\n", xx, yy, zz);
```

的执行情况。首先，在这个输出语句的输出项目表中有三项xx，yy，zz，每个项目均为整型数据。因此，将输出项目表中的这三个整型输出项目依次存入输出缓冲区后，缓冲区的存储情况如下：

01000000	0100ffff	01000000
xx	yy	zz

然后，根据这个输出语句中的“格式控制”，依次有三个长整型格式说明符%ld，它们与输

出缓冲区中数据的对应情况如下(每个长整型格式说明符%ld对应4个字节的数据):

01000000	0100ffff	01000000
%ld	%ld	%ld

因此,依次从输出缓冲区中取出三个长整型数据,分别转换成十进制形式输出,其中格式说明符以外的字符在相应的位置上输出,其输出结果为:

```
xx=1, yy=-65535, zz=1
```

与实际情况相符合。

现在考虑第二个格式输出语句:

```
printf("xx=%hd, yy=%hd, zz=%hd\n", xx, yy, zz);
```

的执行情况。首先,在这个输出语句的输出项目表中有三项xx, yy, zz,每个项目均为基本整型数据,即现在输出项目表中的这三个基本整型输出项目依次存入输出缓冲区后,缓冲区的存储情况如下(与前面的相同):

01000000	0100ffff	01000000
xx	yy	zz

然后,根据这个输出语句中的"格式控制",依次有三个短整型格式说明符%hd,它们与输出缓冲区中数据的对应情况如下(每个短整型格式说明符%hd对应2个字节的数据):

01000000	0100	ffff	01000000
%hd	%hd		%hd

即,依次从输出缓冲区中取出三个基本整型数据(因为短整型数据处理时一定转换成基本整型),并分别将每个数据的前两个字节转换成十进制形式输出,其中格式说明符以外的字符在相应的位置上输出,其输出结果为:

```
xx=1, yy=1, zz=1
```

显然,在这种情况下,输出结果与实际情况不符合。

现在考虑第三个格式输出语句:

```
printf("xx=%d, yy=%d, zz=%d\n", xx, yy, zz);
```

的执行情况。由于在32位编译系统中,int等价于long,%d与%ld也完全相同,因此其输出结果为:

```
xx=1, yy=-65535, zz=1
```

输出结果与实际情况符合。

由这个例子可以看出,在格式输出语句中,格式控制中的各格式说明符与输出表中的各输出项目在个数、次序、类型等方面必须一一对应,否则会造成错误的输出结果。

下面再举一个例子来说明这个问题。

例3.3 设有如下C程序:

```
#include<stdio.h>
main()
```

```
{  double  x=34.567;
   printf("x=%f\n", x);
   printf("x=%d\n", x);
   printf("x=%d\n", (int)x);
}
```

这个程序的实际运行结果为：

```
x=34.567000
x=1958505087
x=34
```

显然，这个程序中的第二个格式输出语句输出的结果是错误的，这是因为在第二个格式输出语句中，格式说明符%d是基本整型格式说明符，而输出项目是双精度型的数据，它们是不匹配的。

(4) 在“格式控制”的格式说明符中，如果带有宽度说明，则在左边没有数字的位置上用空格填满(即输出的数字是右对齐)。但如果在宽度说明前加一个负号(－)，则输出为左对齐，即在右边补空格。例如，如果将例3.1中的程序改成如下：

```
#include<stdio.h>
main()
{  int a, b;
   float  x, y, s;
   a=34; b=-56;
   x=2.5f; y=4.5f; s=x*x+y*y;
   printf("a=%d, b=%d\n", a, b);
   printf("x=%-6.2f, y=%-6.2f, s=%-6.2f\n", x, y, s);
}
```

则这个程序经编译连接后，运行输出的结果为：

```
a=34, b=-56
x=2.50□□, y=4.50□□, s=26.50□
```

即在输出的宽度范围内，空格补在数据的后面。

3.1.2 printf函数中常用的格式说明

格式控制中，每个格式说明都必须用“%”开头，以一个格式字符作为结束，在此之间可以根据需要插入“宽度说明”、左对齐符号“－”、前导零符号“0”等。

1. 格式字符

%后允许使用的格式字符和它们的功能如表3.1所示。在某些系统中，可能不允许使用大写字母的格式字符。因此为了使程序具有通用性，在写程序时，尽量不用大写字母的格式字符。

表 3.1 格式字符和它们的功能

格式字符	说 明
c	输出一个字符
d 或 i	输出带符号的十进制整型数，%ld 为长整型(16 位编译器上必须使用)，%hd 为短整型，%I64d 为 64 位长整数(VC++ 4.0 以上版本输出_int64 类型的整数)
o	以八进制格式输出整型数，%o 不带先导 0。例如十进制数 15 用%o 输出为 17；%#o 加先导 0，例如十进制数 15 用%#o 输出为 017
x 或 X	以十六进制格式输出整型数，但不带先导 0x 或 0X。例如十进制数 2622 用%x 数据格式输出为 a3e，用%X 数据格式输出为 A3E。%#x 或%#X 输出带先导 0x 或 0X 的十六进制数。例如十进制数 2622 用%#x 数据格式输出为 0xa3e，而用%#X 数据格式输出为 0XA3E
u	以无符号十进制形式输出整型数
f	以带小数点的数学形式输出浮点数(单精度和双精度数)
e 或 E	以指数形式输出浮点数(单精度和双精度数)，格式是：[－]m.ddddddde±xxx 或[－]m.ddddddE±xxx。小数位数(d 的个数)由输出精度决定，隐含的精度是 6。若指定的精度为 0，则包括小数点在内的小数部分都不输出。xxx 为指数，保持 3 位，不足补 0。若指数为 0，输出指数是 000
g 或 G	由系统决定采用%f 格式还是采用%e(或%E)格式输出，以使输出宽度最小
s	输出一个字符串，直到遇到"\0"。若字符串长度超过指定的精度则自动突破，不会截断字符串
p	输出变量的内存地址
%	也就是%%形式，输出一个%

2. 长度修饰符

在%和格式字符之间，可以加入长度修饰符，以保证数据输出格式的正确和对齐。对于长整型数(long)应该加 l，如%ld。对于短整型数(short)可以加 h，如%hd。

3. 输出数据所占的宽度说明

当使用%d、%c、%f、%e、%s、…的格式说明时，输出数据所占的宽度(域宽)由系统决定，通常按照数据本身的实际宽度输出，前后不加空格，并采用右对齐的形式。但可以用以下 3 种方法人为控制输出数据所占的宽度(域宽)，按照使用者的意愿进行输出。

(1) 在%和格式字符之间插入一个整数常数来指定输出的宽度 n(例如%4d，n 代表整数 4)。如果指定的宽度 n 不够，输出时将会自动突破，保证数据完整输出。如果指定的宽度 n 超过输出数据的实际宽度，输出时将会右对齐，左边补以空格，达到指定的宽度。

(2) 对于 float 和 double 类型的实数，可以用“n1.n2”的形式来指定输出宽度(n1 和

n2 分别代表一个整常数),其中 n1 指定输出数据的宽度(包括小数点),n2 指定小数点后小数位的位数,n2 也称为精度(例如%12.4f,n1 代表整数 12,n2 代表整数 4)。

对于 f、e 或 E,当输出数据的小数位多于 n2 位时,截去右边多余的小数,并对截去部分的第一位小数做四舍五入处理;当输出数据的小数位少于 n2 时,在小数的最右边补 0,使得输出数据的小数部分宽度为 n2。若给出的总宽度 n1 小于 n2 加上整数位数和小数点(e 或 E 格式还要加上指数的 5 位),则自动突破 n1 的限制;反之,数字右对齐,左边补空格。

也可以用".n2"格式(例如%.6f),不指定总宽度,仅指定小数部分的输出位数,由系统自动突破,按照实际宽度输出。如果指定"n1.0"或".0"格式(例如%12.0f 或%.0f),则不输出小数点和小数部分。

对于 g 或 G,宽度用来指定输出的有效数字位数。若宽度超过数字的有效数字位数,则左边自动补 0;若宽度不足,则自动突破。不指定宽度,将自动按照 6 位有效数字输出,截去右边多余的小数,并对截去部分的第一位小数做四舍五入处理。

(3) 对于整型数,若输出格式是"0n1"或".n2"格式(例如%05d 或%.5d),如果指定的宽度超过输出数据的实际宽度,输出时将会右对齐,左边补以 0。

对于 float 和 double 类型的实数,若用"0n1.n2"格式输出(例如%012.4f),若给出的总宽度 n1 大于 n2 加上整数位数和小数点(e 或 E 格式还要加上指数的 5 位),则数字右对齐,左边补 0。

对于字符串,格式"n1"指定字符串的输出宽度,若 n1 小于字符串的实际长度,则自动突破,输出整个字符串;若 n1 大于字符串的实际长度,则右对齐,左边补空格。若用".n2"格式指定字符串的输出宽度,则若 n2 小于字符串的实际长度,将只输出字符串的前 n2 个字符。

注意:输出数据的实际精度并不完全取决于格式控制中的域宽和小数的域宽,而是取决于数据在计算机内的存储精度。通常系统只能保证 float 类型有 7 位有效数字,double 类型有 15 位有效数字。若指定的域宽和小数的域宽超过相应类型数据的有效数字,则输出的多余数字是没有意义的,系统只是用来填充域宽而已。

4. 输出数据左对齐

由于输出数据都隐含右对齐,如果想左对齐,就可以在格式控制中的"%"和宽度之间加一个"-"号来实现。

5. 使输出数据总带+号或-号

通常输出数据,如果是负数,则前面有符号位"-",但正数的"+"都省略了。如果要每一个数前面都带正负号,则可以在%和格式字符间加一个"+"号来实现。

若 k 为 int 型,值为 1234,f 为 float 型,值为 123.456。表 3.2 列举了各种输出宽度和不指定宽度情况下的输出结果(表中数字数据中的符号□代表一个空格)。

表 3.2 各种输出宽度情况下的输出结果

输 出 语 句	输 出 结 果
printf("%d\n", k);	1234
printf("%6d\n", k);	□□1234
printf("%2d\n", k);	1234
printf("%f\n", f);	123.456
printf("%12f\n", f);	□□123.456000
printf("%12.6f\n", f);	□□123.456000
printf("%2.6f\n", f);	123.456000
printf("%.6f\n", f);	123.456000
printf("%12.2f\n", f);	□□□□□□123.46
printf("%12.0f\n", f);	□□□□□□□□□123
printf("%.0f\n", f);	123
printf("%e\n", f);	1.234560e+002
printf("%13e\n", f);	1.234560e+002
printf("%13.8e\n", f);	1.23456000e+002
printf("%3.8e\n", f);	1.23456000e+002
printf("%.8e\n", f);	1.23456000e+002
printf("%13.2e\n", f);	□□□□1.23e+002
printf("%13.0e\n", f);	□□□□□□□1e+002
printf("%.0e\n", f);	1e+002
printf("%g\n", f);	123.456
printf("%5g\n", f);	123.456
printf("%10g\n", f);	□□123.456
printf("%g\n", 123.456789);	123.457
printf("%06d\n", k);	001234
printf("%.6d\n", k);	001234
printf("%012.6f\n", f);	00123.456000
printf("%013.2e\n", f);	00001.23e+002
printf("%s\n", "abcdefg");	abcdefg
printf("%10s\n", "abcdefg");	□□□abcdefg
printf("%5s\n", "abcdefg");	abcdefg
printf("%.5s\n", "abcdefg");	abcde

续表

输出语句	输出结果
printf("%-6d\n", k);	1234□□
printf("%-12.2f\n", f);	123.46□□□□□□
printf("%-13.2e\n", f);	1.23e+002□□□□
printf("%+-6d%+-12.2f\n", k, -f);	+1234 -123.46□□□□□□
printf("%4.1f%%\n", 12.5);	12.5%

3.1.3 使用 printf 函数时的注意事项

(1) printf 的输出格式为自由格式,是否在两个数之间留逗号、空格或回车,完全取决于你的格式控制,如果不注意,很容易使数字连在一起,使得输出结果没有意义。例如:printf("%d%d%f\n", k, k, f);语句的输出结果是:12341234123.456,无法分辨其中的数字含义。而如果改为 printf("%d %d %f\n", k, k, f);输出结果是:1234 1234 123.456,看起来就一目了然。

(2) 格式控制中必须含有与输出项一一相对应的输出格式说明,类型必须匹配。若格式说明与输出项的类型不一一对应匹配,则不能正确输出。而且编译时不会报错。若格式说明个数少于输出项个数,则多余的输出项不予输出;若格式转换说明个数多于输出项个数,则将输出一些毫无意义的数字乱码。

(3) 在格式控制中,除了前面要求的输出格式,还可以包含任意的合法字符(包括汉字和转义符),还可利用'\n'(回车)、'\r'(回行但不回车)、'\t'(制表)、'\a'(响铃)等控制格式。这些字符输出时将"原样照印"。

(4) 如果要输出%符号,可以在格式控制中用%%表示,将输出一个%符号。

(5) printf 函数有返回值,返回值是本次调用输出字符的个数,包括回车等控制符。

(6) 尽量不要在输出语句中改变输出变量的值,因为可能会造成输出结果的不确定性。例如:int k=8; printf("%d,%d\n", k, ++k); 输出结果不是你预想的 8,9,而是 9,9。这是因为调用函数 printf 时,其参数是从右至左进行处理的,将先进行++k 运算。

(7) 输出数据时的域宽可以改变。若变量 m、n、i 和 f 都已正确定义并赋值,则语句 printf("%*d", m, i); 将按照 m 指定的域宽输出 i 的值,并不输出 m 的值。而语句 printf("%*.*f", m, n, f); 按照 m 和 n 指定的域宽输出浮点型变量 f 的值,并不输出 m、n 的值。

3.2 格式输入函数

3.2.1 基本的格式输入语句

在 C 语言中,格式输入的一般形式如下: