第3章

数据的输入与输出

在编程解决实际问题时，一般都会有一些数据的输入以及结果的输出，即使没有输入也一定会有输出。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。

C 语言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，在没有特别指定输入或输出设备时，默认其输入设备是键盘，输出设备是显示器。

另外要注意，如果在程序中要使用 C 语言所提供的输入或输出函数，在程序的开头应该使用包含命令

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

将 C 语言中标准输入输出库函数包含进来。

本章将具体介绍一些 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为：

```
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引号括起来，用于说明输出项目所用的格式。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、十六进制以及八进制 3 种形式表示，因此，对于整型数据的输出也具有这 3 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 或%lld　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 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，不受宽度 m 的限制。

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，其格式说明符为%f 或%m.nf；如果以指数形式输出实型数据，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，不受宽度 m 的限制。

需要指出的是，在 printf 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。

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，会在输出字符的前面补 m－1 个空格。

下面对各种基本类型数据的格式输出做几点说明。

（1）输出表中可以有多个输出项目，但各输出项目之间要用逗号（,）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);
}
```

程序经编译、连接后，运行的结果为(␣表示一个空格)：

```
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，再输出字符串"，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：

① 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。

② 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自右至左依次将它们存入输出缓冲区中。

③ 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)，转换成对应的指定进制数据进行输出。其中从输出缓冲区中取多少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，应取 2 字节；格式说明符%ld 为长整型，应取 4 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)，应取 8 字节；以此类推。

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);
}
```

程序的运行的结果为：

```
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 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－65 535，在计算机内存中占 4 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 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 3 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 1 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 0x 00 00 00 01

yy：01 00 ff ff　　对应十六进制数 0x ff ff 00 01

zz：01 00 00 00　　对应十六进制数 0x 00 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 3 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 3 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 3 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 字节的数据）：

01000000	0100ffff	01000000
%ld	%ld	%ld

因此，依次从输出缓冲区中取出 3 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 3 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 3 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 3 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 字节的数据）：

0100 0000	0100 ffff	0100 0000
%hd	%hd	%hd

即，依次从输出缓冲区中取出 3 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);
}
```

程序实际运行的结果为:

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

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中的程序改为:

```
#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及其功能如表 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 或%lld 为 64 位超长整数(VC6 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.ddddddE±xxx 或[-]m.dddddddE±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	由系统根据输出数据决定采用%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"。对%ms 若字符串长度超过指定的输出宽度 m 则自动突破，不会截断字符串；对%.ms 若字符串长度超过指定的输出宽度 m，则只输出字符串前 m 个字符
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。对于超长整型数(long long)应该加 ll，如%lld。

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)，按照用户的意愿进行输出。

(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(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,填满指定的宽度。

(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(包括小数点和符号位),n2 指定小数点后小数位的位数,n2 也称为精度(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、小数点(e 或 E 格式还要加上指数的 5 位)和符号位,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(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。如果采用“n1.0”或“.0”格式(如%12.0f 或%.0f),则不输出数据的小数点和小数部分。

对于 g 或 G,可以用%m.ng 或%m.nG 的形式来输出(m 和 n 分别是一个常整数),其中 m 指定输出数据的宽度(包括小数点),n2 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,则左边自动补空格;若宽度不足,则自动突破。若用%g,%G,%mg,%mG 的形式不指定输出的有效数字位数,将自动按照最多 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。若指定或默认的输出有效数字位数超过实际数字的有效位数,则把实际数字原样输出。

(3) 对于整型数,若输出格式是“0n1”或“.n2”格式(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(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。通常计算机系统只能保证 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,输出的多余数字是没有意义的,只是系统用来填充域宽而已。

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,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,如果负数,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,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(表中输出结果中的符号␣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);	123.456000
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", 123.4);	123.4
printf("%g\n", 123.4567);	123.457
printf("%5g\n", 123.4567);	123.457
printf("%10g\n", 123.4567);	␣␣␣123.457
printf("%g\n", 1234567.89);	1.23457e+006
printf("%5G\n", 1234567.89);	1.23457e+006
printf("%10.8g\n", 1234567.89);	␣1234567.9
printf("%10.7G\n", 1234567.89);	␣␣␣1234568
printf("%10.5G\n", 1234567.89);	1.2346E+006
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
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、Tab 或回车,完全取决于格式控制,如果不注意这个问题,很容易造成数字连在一起,使得输出的结果没有意义。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,则不能输出正确结果。而且编译时还不会报错。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或乱码。

(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,%d\n", k, ++k, ++k);
```

用微软 VS 系列编译器,输出结果不是 8,9,10,而是 10,10,10。这是因为在调用 printf()函数之前,先执行了两次++k。但如果换一个非微软编译器执行结果可能就不一样,这种现象属于 C 语言国际标准中的未定义行为(undefined behavior),国际标准中并没有规定一定要怎么做,由各家的 C 编译器自行确定。

(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,但不输出 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,但不输出 m、n 的值,这被称作变场宽输出。

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为：

```
scanf("格式控制",内存地址表);
```

其中scanf()是C编译系统提供的格式输入函数。格式控制部分要用一对双引号括起来，用于说明输入数据时应使用的格式。内存地址表中的各项目给出各输入数据所存放的内存地址。

与格式输出一样，在格式控制中，用于说明输入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。用于数据输入的常用格式说明符有以下几种。

1. 整型格式说明符

整型格式说明符有以下几种。

1）十进制形式

十进制形式的格式说明符用于输入十进制形式的整型数据，其格式说明符为：

- %d或%md　用于一般整型。
- %ld或%mld　用于长整型。
- %lld或%I64d或%mlld　用于超长整型。
- %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。

2）八进制形式

八进制形式的格式说明符用于输入八进制形式的整型数据，其格式说明符为：

- %o或%mo　用于一般整型。
- %lo或%mlo　用于长整型。

3）十六进制形式

十六进制形式的格式说明符用于输入十六进制形式的整型数据，其格式说明符为：

- %x或%mx　用于一般整型。
- %lx或%mlx　用于长整型。

由此可以看出，用于输入与输出整型数据的格式说明符是完全一样的，m表示输入数据时的宽度（即列数）。

与输出情形一样，对于八进制形式与十六进制形式的输入格式，主要用于输入无符号整型的数据。

2. 实型格式说明符

用于输入的实型格式说明符与用于输出的情形稍有不同。并且，单精度实型与双精度实型的输入格式说明符是不同的。

用于输入的单精度实型格式说明符为%f或%e，用于输入的双精度实型格式说明符为%lf或%le。

由此可以看出，与输出不同，在用于输入时，无论是单精度实型还是双精度实型，都不能用m.n来指定输入的宽度和小数点后的位数。

3. 字符型格式说明符

用于输入的字符型格式说明符为%c或%mc。

下面是用到格式输入的一个程序：

```
#include <stdio.h>
main()
{ int  a;
  float  b;
  char  c;
  scanf("%d%f%c", &a, &b, &c);
}
```

下面对格式输入做几点说明。

(1) 在格式输入中，内存地址表中的各项目必须是变量地址，而不能是变量名，且彼此间用“,”分隔。为此，C语言专门提供了一个取地址运算符 &。例如，&a，&b，&c 分别表示变量 a，b，c 在内存中的首地址。

(2) 当用于输入整型数据的格式说明符中没有宽度说明时，则在具体输入数据时分以下两种情况。

① 如果各格式说明符之间没有其他字符，则在输入数据时，两个数据之间可以用“空格”“Tab”或“回车”来分隔。

② 如果各格式说明符之间包含其他字符，则在输入数据时，应输入与这些字符相同的字符作为间隔。例如，设有如下说明

```
int  a, b;
float  c, d;
```

现要利用格式输入函数输入 a=12 ，b=78 ，c=12.5 ，d=7.6。采用不同的格式说明，其输入数据的形式也是不同的。

如果输入语句为：

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

即格式说明符中没有宽度说明，各格式说明符之间也没有其他字符，则输入数据的形式应为：

```
12  78  12.5  7.6↵
```

即在输入的两个数据之间用空格来分隔，当然也可用“Tab”或“回车”来分隔。

如果输入语句为：

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

即格式说明符中没有宽度说明，但各格式说明符之间有其他字符，即逗号，则输入数据的形式应为：

```
12,78,12.5,7.6↵
```

即在输入的两个数据之间同时要输入逗号。

如果输入语句为：

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

即格式说明符中没有宽度说明，但各格式说明符之间有其他字符，则输入数据的形式应为：

```
a = 12,b = 78,c = 12.5,d = 7.6 ↵
```

即在输入的两个数据之间同时要输入这些非格式说明符的字符。

(3) 当整型或字符型格式说明符中有宽度说明时,按宽度说明截取数据。

例 3.4　设有以下程序:

```
#include <stdio.h>
main()
{ int  a, d;
  char  b, c;
  printf("input a, b, c, d: ");
  scanf("%3d%3c%2c%2d", &a, &b, &c, &d);
  printf("a = %d, b = %c, c = %c, d = %d\n", a, b, c, d);
}
```

若从键盘输入如下(其中"input a, b, c, d:"为输出的字符串):

```
input a, b, c, d: 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6
                    3d   3c  2c  2d
```

则它们与各格式说明符之间的对应关系如上,最后赋给各变量的值应为:

```
a = 123, b = 4, c = 7, d = 90
```

但此程序运行时会出现致命错误。因为一个字符型变量只能存放一个字符。在截取的字符中取第一个字符赋给字符型变量,但其后的字符会赋值到 b、c 内存单元以外的地方,产生内存越界的致命错误。

(4) 在用于输入的实型格式说明符中不能用 m.n 来指定输入的宽度和小数点后的位数(这是与输出的不同之处)。例如,下列用法是错误的:

```
scanf("%7.2f", &a);
```

(5) 为了便于程序执行过程中从键盘输入数据,在一个 C 程序开始执行时,系统就在计算机内存中开辟了一个输入缓冲区,用于暂存从键盘输入的数据。开始时该输入缓冲区是空的。

当执行到一个输入函数时,就检查输入缓冲区中是否有数据。

如果输入缓冲区中已经有数据(上一个输入函数剩下的),则依次按照格式控制中的格式说明符从输入缓冲区中取出数据转换成计算机中的对应表示形式(二进制),最后存放到内存地址表中指出的对应地址中。

如果输入缓冲区中没有数据(即输入缓冲区位空),则等待用户从键盘输入数据并依次存放到输入缓冲区中。当输入一个<回车>或<换行>符后,将依次按照格式控制中还未用过的格式说明符从输入缓冲区中取出数据转换成计算机中对应的表示形式(二进制),最后存放到内存地址表中指出的对应地址中。

无论上述哪一种情况,在从输入缓冲区中取数据时,如果遇到<回车>或<换行>符,则将输入缓冲区清空。此时如果格式控制中的格式说明符还未用完,则继续等待用户从键盘输入数据并依次存放到输入缓冲区中,直到输入一个<回车>或<换行>符后,再依次按照格式控制中还未用过的格式说明符从输入缓冲区中取出数据转换成计算机中对应的表示形式(二进制),最后存放到内存地址表中指出的对应地址中。这个过程直到格式控制中的格式说明符用完为止。此时如果输入缓冲区中的数据还未取完,则将留给下一个输入函数使用。

从以上输入函数的执行过程可以看出，从键盘输入数据是以<回车>或<换行>符作为结束的。当一行输入的数据不够时，可以在下一行继续输入；当一行上的数据用不完时，可以留给下一个输入函数使用。

需要注意的是，由于<回车>或<换行>符是作为键盘输入数据的结束符，因此，在输入函数的格式控制中，最后不能加换行符'\n'。

(6) 与格式输出一样，格式输入格式控制中的各格式说明符与内存地址表中的变量地址在个数、次序、类型方面必须一一对应。

下面举一个例子来说明这个问题。

例 3.5 设有 C 程序如下：

```
#include <stdio.h>
main()
{ double  x;
  printf("input x: ");
  scanf("%f", &x);
  printf("x = %f\n", x);
}
```

程序的运行结果为(其中有下画线的部分为键盘输入)：

```
input x: 123.456
x =-92559604945281701000000000000000000000000000000000000000000000000000000.000000
```

显然，输出语句输出的 x 值是错误的。这是因为，x 定义为双精度型的实型变量(占 8 字节)，但它使用的是单精度实型的输入格式说明符。当输入一个实型数 123.456 后，将按照单精度输入格式说明符将它转换成计算机中的表示形式(只占 4 字节)，最后存放到为双精度实型变量 x 所分配的存储空间的低 4 字节中，而为双精度实型变量 x 所分配的存储空间的高 4 字节中的各位均是未初始化的随机数，输出结果将是一个毫无意义的数。

3.2.2 scanf 函数中常用的格式说明

每个格式说明都必须用%开头，以一个格式字符作为结束。

通常允许用于输入的格式字符和相应功能如表 3.3 所示。

表 3.3 用于输入的格式字符和相应功能

格式字符	说　明
c	输入一个字符
d	输入带符号的十进制整型数
i	输入整型数，整型数可以是带先导 0 的八进制数，也可以是带先导 0x(或 0X)的十六进制数
o	以八进制格式输入整型数，可以带先导 0，也可以不带
x	以十六进制格式输入整型数，可以带先导 0x 或 0X，也可以不带
u	以无符号十进制形式输入整型数
f(lf)	以带小数点的数学形式或指数形式输入浮点数(单精度用 f，双精度数用 lf)
e(le)	以带小数点的数学形式或指数形式输入浮点数(单精度用 f，双精度数用 lf)
s	输入一个字符串，直到遇到"\0"。若字符串长度超过指定的场宽则自动突破，不会截断字符串

说明：

(1) 在格式串中，必须含有与输入项一一相对应的格式转换说明符。若格式说明与输入项的类型不一一对应匹配，则不能正确输入，而且编译时不会报错。若格式说明个数少于输入项个数，scanf 函数结束输入，则多余的输入项将无法得到正确的输入值；若格式转换说明个数多于输入项个数，scanf 函数也结束输入，多余的数据作废，不会作为下一个输入语句的数据。

(2) 在 32 位编译器上，输入 short 型整数，格式控制必须用%hd。要输入 double 型数据，格式控制必须用%lf(或%le)。否则，数据不能正确输入。

(3) 在 scanf 函数的格式字符前可以加入一个正整数指定输入数据所占的宽度，但不可以对实数指定小数位的宽度。

(4) 由于输入是一个字符流，scanf 从这个流中按照格式控制指定的格式解析出相应数据送到指定地址的变量中。因此当输入的数据少于输入项时，运行程序等待输入，直到满足要求为止。当输入的数据多于输入项时，多余的数据在输入流中没有作废，而是等待下一个输入操作语句继续从此输入流读取数据。

(5) scanf 函数有返回值，其值就是本次 scanf 调用正确输入的数据项的个数。

3.2.3　通过 scanf 函数从键盘输入数据

当用 scanf 函数从键盘输入数据时，每行数据在未按下回车键(Enter 键)前，可以任意修改。但按下回车键后，这一行数据就送入了输入缓冲区，不能再回去修改。

1. 输入数值数据

在输入整数或实数这类数值型数据时，输入的数据之间必须用空格、回车符、制表符(Tab 键)等间隔符隔开，间隔符个数不限。即使在格式说明中人为指定了输入宽度，也可以用此方式输入。例如，若 k 为 int 类型变量，a 为 float 类型变量，y 为 double 类型变量，有以下输入语句：

```
scanf("%d%f%le", &k, &a, &y);
```

若要给 k 赋值 10，a 赋值 12.3，y 赋值 1234567.89，输入格式可以是(输入的第一个数据之前可有任意空格)：

```
10  12.3  1234567.89<CR>
```

此处<CR>表示回车键。也可以是：

```
10<CR>
12.3<CR>
1234567.89<CR>
```

只要能把 3 个数据正确输入，可以按任何形式添加间隔符。

2. 指定输入数据所占的宽度

可以在格式字符前加入一个正整数指定输入数据所占的宽度。例如上面代码改为：

```
scanf("%3d%5f%5le", &k, &a, &y);
```

若从键盘上从第 1 列开始输入：

```
123456.789.123
```

用 printf("%d　%f　%f\n", k, a, y);打印的结果是：

```
123  456.700000  89.120000
```

可以看到，由于格式控制是%3d，因此把输入数字串的前 3 位 123 赋值给了 k；由于对应于变量 a 的格式控制是%5f，因此把输入数字串中随后的 5 位数(包括小数点)456.7 赋值给了 a；由于格式控制是%5e，因此把数字串中随后的 5 位(包括小数点)89.12 赋值给了 y。

由以上示例可知，数字之间不需要间隔符，若插入了间隔符，系统也将按指定的宽度来读取数据，从而会引起输入混乱。除非数字是已经“粘联”在一起，否则不提倡指定输入数据所占的宽度。

3. 跳过某个输入数据

可以在%和格式字符之间加入“*”号，作用是跳过对应的输入数据。例如：

```
int   x, y, z;
scanf("%d%*d%d%d", &x, &y, &z);
printf("%d  %d  %d\n", x, y, z);
```

若是输入：

```
12  34  56  78
```

则输出是：

```
12  56  78
```

系统将 12 赋给 x，跳过 34，把 56 赋给 y，把 78 赋给 z。

4. 在格式控制字符串中插入其他字符

scanf 函数中的格式控制字符串是为了输入数据用的，无论其中有什么字符，也不会输出到屏幕上，因此若想在屏幕上输出提示信息，应该首先使用 printf 函数输出。例如：

```
int  x, y, z;
scanf("Please input x,y,z: %d%d%d", &x, &y, &z);
```

屏幕上不会输出 Please input x,y,z :，而是要求输入数据时按照一一对应的位置原样输入这些字符。必须从第一列起以下面的形式进行输入：

```
Please input x,y,z: 12 34 56
```

包括 Please input x,y,z、字符的大小写、字符间的间格等必须与 scanf 中的完全一致，这些字符又被称为通配符。

但如果使用以下的形式：

```
int  x, y, z;
printf("Please input x,y,z: ");
scanf("%d%d%d", &x, &y, &z);
```

运行时，由于 printf 语句的输出，屏幕上将出现提示：Please input x,y,z:，只需按常规输入下面的数即可：

```
12  34  56
```

如果在以上 scanf 中，在每个格式说明之间加一个逗号作为通配符：

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

则输入数据时，必须在前两个数据后面紧跟一个逗号，以便与格式控制中的逗号一一匹配，否则就不能正确读入数据。例如，输入：

```
12,34,56
```

能正确读入。输入：

```
12,    34,    56
```

也能正确读入。因为空格是间隔符，将全部被忽略掉。但输入：

```
12    ,34    ,56
```

将不能正确读入，因为逗号没有紧跟在输入数据后面。

需要提醒的是，为了减少不必要的麻烦，尽量不要在输入格式中使用通配符。

3.3　字符输出函数

3.1 节和 3.2 节介绍了几种基本类型数据的输入与输出，其中包括了字符型数据的输入与输出。为了使用方便，C 语言还提供了专门用于字符输入与输出的函数。本节先介绍字符输出函数，3.4 节再介绍字符输入函数。

字符输出函数的形式为：

```
putchar(c)
```

这个函数的功能是，在显示屏的当前光标位置处输出项目 c 所表示的一个字符。其中 c 可以是字符型常量、字符型变量、整型变量或整型表达式。

字符输出函数的执行过程与格式输出函数的执行过程完全相同。

例 3.6　设有如下 C 程序：

```
#include <stdio.h>
main()
{  int x=68;
   char y='B';
   putchar('A');
   putchar(y);
   putchar(67);
   putchar(x);
   putchar(34+25);
}
```

程序的运行结果为：

```
ABCD;
```

在上述程序中，第 1 个字符输出函数输出字符型常量'A'；第 2 个字符输出函数输出字符型变量 y 的值，为字符型数据'B'；第 3 个字符输出函数输出以整型常量 67 作为 ASCII 码的

字符,即字符型数据'C';第 4 个字符输出函数输出以整型变量 x 中的值为 ASCII 码的字符,由于整型变量 x 的值为 68,即字符型数据'D'的 ASCII 码;第 5 个字符输出函数输出以整型表达式 34+25 的值为 ASCII 码的字符,由于该表达式的值为 59,即字符型数据';'的 ASCII 码。

字符输出函数 putchar()也可以输出转义字符。

例 3.7 设有 C 程序如下:

```
#include <stdio.h>
main()
{ int x = 68;
  char y = 'B';
  putchar('A'); putchar('\n');
  putchar(y); putchar('\n');
  putchar(67); putchar('\n');
  putchar(x); putchar('\n');
  putchar(34 + 25); putchar('\n');
}
```

在这个程序中,在输出每一个字符后,紧接着输出一个换行,最后运行结果为:

```
A
B
C
D
;
```

3.4 字符输入函数

在 C 语言中,字符输入函数的形式为:

```
getchar()
```

这个函数的功能是接收从键盘输入的一个字符。

字符输入函数的执行过程与格式输入函数完全相同。例如,下面的程序执行过程中,将等待从键盘输入一个字符赋给字符型变量 x:

```
#include <stdio.h>
main()
{ char  x;
  x = getchar();
}
```

需要说明的是,在执行字符输入函数时,由键盘输入的字符(依次存放在输入缓冲区中)同时也在屏幕上显示,并且以<回车>结束,但一个字符输入函数只顺序接收一个字符,输入缓冲区中剩下的字符数据(包括回车符)将留给下面的字符输入函数或格式输入函数使用。

在 C 语言中,还有字符输入函数_getch()或_getche()。这两个函数的功能是在按下相应键的同时接收从键盘输入的一个字符。

特别需要指出的是,getch()和_getche()不是 stdio.h 中的函数,而是 conio.h 中的函数。使用字符输入函数_getch()时,由键盘输入的字符不在屏幕上显示;但使用字符输入函数

_getche( )时，由键盘输入的字符同时也在屏幕上显示。并且这两个函数都不等待<回车>符，在按下相应键的同时，此函数就读取(接收)了相应字符。

例如，下面的程序执行过程中，将等待从键盘输入一个字符赋给字符型变量 x，只要按下键盘上任何一个键，_getch()将会立刻把这个字符读入赋值给 x。

```
#include <conio.h>              /* 注意: 这里引用的不是 stdio.h */
main()
{    char  x;
     x = _getch();
}
```

还有一个比较有趣的函数可以将刚读入的字符放回输入流，使得输入流看起来未被读过。函数形式为：

```
_ungetch(c);
```

这个函数的功能是把刚从键盘接收(输入)的一个字符 c 回写到输入流。

例如，有如下程序：

```
#include <stdio.h>
#include <conio.h>
main()
{    char  x, y;
     x = _getche();             /* 读入一个字符 */
     _ungetch(x);               /* 将读入的字符放回输入流中 */
     y = _getche();             /* 读入一个字符 */
     putchar(y);                /* 输出字符 */
}
```

程序运行时，只需输入一个字符就会出现结果，而且是输入的相应字符。

练习 3

1. 在下列 C 程序的前两行中填入应包含的文件名：

```
#include <        >
#include <        >
main()
{ double a, b, z;
  printf("input a and b :");
  scanf("%lf, %lf", &a, &b);
  z = exp(a * a + b * b);
  printf("z = %f\n", z);
}
```

2. 设有下列定义和输入语句：

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

如果要求变量 x，y，c，d 的值分别为 20，30，X，Y，则正确的数据输入格式是什么？

3. 设有下列 C 程序：

```
#include <stdio.h>
main()
{ int  x=4617;
  printf("x=%8d\n", x);
  printf("x=%-8d\n", x);
}
```

这个程序运行的结果是什么？

4. 设有输入语句如下：

```
scanf("x=%d, y=%d, z=%d", &x, &y, &z);
```

为使变量 x 的值为 12，变量 y 的值为 34，变量 z 的值为 62，则从键盘输入数据的正确格式是什么？

5. 编写一个 C 程序，从键盘输入直角三角形的斜边 c 与一条直角边 a 的长，计算并输出另一条直角边 b 的长。

6. 编写一个 C 程序，从键盘输入一个数字字符('0'～'9')，然后将它转换为相应的整数后再输出。如输入数字字符'5'，然后将它转换为十进制整数 5 后输出。

7. 编写一个 C 程序，将从键盘输入的小写字母转换为大写字母后输出。

8. 分析下列 C 程序的输出结果：

```
#include <stdio.h>
main()
{ double x;
  int  a=1, b=1, c=1, d=1;
  x=97.6875;
  printf("x=%f\n", x);
  printf("x=%d\na=%d\nb=%d\nc=%d\nd=%d\n", x, a, b, c, d);
}
```