第 3 章 格式化输入与输出

计算机程序离不开输入与输出。简单来说,输入与输出是指计算机程序与环境或用户之间进行的数据或信息交换。编写的程序代码经编译、链接成为可执行文件(通常称为计算机程序)之后,在计算机操作系统环境下运行时,往往需要用户输入相关数据或信息,可执行文件经过计算处理以后,通过输出对环境产生某种影响作为计算机程序的运行结果。

标准 C 定义了一系列输入与输出库函数,在编程时可直接调用这些函数来输入和输出数据或格式化信息。本章对字符输入与输出函数,以及针对各种类型数据的格式化输入与输出函数,从函数原型、调用方式和使用举例等多个侧面进行说明。其他标准输入与输出函数将在第 10 章文件中加以介绍。输入输出函数在使用前,需包含 `stdio.h` 头文件:

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

3.1 字符输入与输出

标准库实现了简单的文本输入与输出模式。文本流由一系列行组成,每一行的结尾是一个换行符。若系统未遵循此模式,则标准库将通过一些措施使得该系统在输入端和输出端都适应此模式。例如,标准库可以在输入端将回车符和换页符都转换成换行符,而在输出端进行方向转换。最简单的输入输出机制是字符的输入与输出。

3.1.1 字符输入函数 `getchar`

字符输入函数 `getchar` 的功能是从标准输入(一般为键盘)一次读取一个字符。其函数原型为:

```
int getchar(void);
```

说明:函数返回值的类型为 `int`,参数表中的 `void` 表示函数不需要参数。

`getchar` 函数在每次调用时,从输入流中读取一个字符,并将所读取字符的 ASCII 码(`int` 类型)作为函数的返回值。若遇文件结尾,则返回 `EOF`。

符号常量 `EOF` 用作文件结束标志(End Of File),在头文件 `<stdio.h>` 中定义,其值一般为 `-1`,但是在程序中应该使用 `EOF` 来测试文件是否结束,这样才能保证程序同 `EOF` 的特定值无关。

在 Microsoft 公司的操作系统(如 DOS 和 Windows 等)中,按 `Ctrl+Z` 键(先按 `Ctrl` 键,再按 `Z` 键,然后松开这两个键,后面组合键的按法相同),或者在类 UNIX 系统(如 Linux 和 MacOS 等)中按 `Ctrl+D` 键,可以输入一个文件结束标志 `EOF`。

如果需要使用输入的字符数据,则应该把输入的字符赋值给一个字符变量。同时也可以对输入的字符不予处理。

【例 3.1】 getchar 函数执行流程分析。

```
#include <stdio.h>
int main(void)
{
    char  ch1, ch2, ch3;
    ch1 = getchar();      /* 输入的字符 (getchar 函数的返回值) 赋值给字符变量 ch1 */
    getchar();           /* 输入的字符不赋值给任何变量 */
    ch2 = getchar();     /* 输入的字符赋值给字符变量 ch2 */
    ch3 = getchar();     /* 输入的字符赋值给字符变量 ch3 */
    printf("\n%c%c%c", ch1, ch2, ch3);
    printf("%d %d %d", ch1, ch2, ch3);
    return 0;
}
```

程序执行时,如果输入(↵表示回车键):

```
a↵
b↵
```

那么将会输出:

```
ab↵
97 98 10
```

分析: 程序开始运行后,执行语句 `ch1=getchar();` 时,由于输入流中没有字符, `getchar` 函数进入等待输入状态。此时,输入 `a`,并按下回车键,则字符 `'a'` 和 `'\n'` 被送入输入流。由于按下回车键会激活处于等待状态的 `getchar` 函数, `getchar` 函数从输入流中读取第一个字符 `'a'`,变量 `ch1` 被赋值为字符 `'a'`,输入流中还剩下换行字符 `'\n'`。

程序接着执行语句 `getchar();`,由于输入流中有一个字符 `'\n'`, `getchar` 函数直接从输入流读取字符 `'\n'`,读取字符 `'\n'` 后不做任何处理,同时输入流中字符被取空。

程序继续执行语句 `ch2=getchar();`,由于输入流中没有字符, `getchar` 函数再次进入等待输入状态。此时,输入 `b`,并按下回车键,则字符 `'b'` 和 `'\n'` 被送入输入流。同样地,回车键激活处于等待输入状态的 `getchar` 函数, `getchar` 函数从输入流中读取第一个字符 `'b'`,变量 `ch2` 被赋值为字符 `'b'`,输入流中还剩一个字符 `'\n'`。

程序继续执行语句 `ch3=getchar();`,由于输入流中有一个字符 `'\n'`, `getchar` 函数直接从输入流读取字符 `'\n'`,并将其赋值给字符变量 `ch3`。

程序继续执行语句 `printf("\n%c%c%c", ch1, ch2, ch3);`,输出格式字符串中的普通字符 `'\n'` 并按转换说明以字符形式输出 `ch1('a')`、`ch2('b')` 和 `ch3('\n')`。

程序执行语句 `(printf("%d %d %d", ch1, ch2, ch3);)`,按转换说明以十进制整数形式输出 `ch1('a')`、`ch2('b')` 和 `ch3('\n')` 的值,即输出这些字符的 ASCII 码 97、98 和 10。

3.1.2 字符输出函数 putchar

字符输出函数 `putchar` 的功能是从标准输出(一般为显示器)一次输出一个字符。其函

数原型为

```
int putchar(int ch);
```

说明：函数返回值的类型为 int，形式参数 int ch 表明 putchar 函数需要一个 int 类型的参数，该参数为所要输出字符的 ASCII 码值。如果函数正确执行，则返回所输出字符的字符码，否则返回 EOF。

putchar 函数在每次调用时，将实际参数 ch 的字符输出到标准输出设备上。实际参数 ch 可为 char、short 与 int 类型的表达式，其值是要输出字符的字符码。

【例 3.2】 使用 putchar 函数输出字符串"HUST"。

```
#include <stdio.h>
int main(void)
{
    char c1 = 'H', c2 = 'U';
    short c3 = 83;
    int c4 = 84;
    putchar(c1); putchar(c2); putchar(c3); putchar(c4);
    return 0;
}
```

【例 3.3】 设变量说明为 char c = 'a', 则下列函数调用表达式都可输出字符'a'的图形符号。

```
putchar('a')  putchar(c)  putchar(97)  putchar('\141')
```

【例 3.4】 在键盘上循环输入任意一个字符，直到文件结束为止，若输入的字符是小写字母，则输出该小写字母对应的大写字母，否则直接输出该字符。

```
#include <stdio.h>
int main(void)
{
    char c;
    while ((c=getchar())!=EOF)
        putchar((c>='a'&& c<='z') ? c-'a'+'A': c);
    return 0;
}
```

分析：putchar 函数的参数是一个条件表达式：((c>='a'&& c<='z') ? c-'a'+'A' : c)。putchar 函数在执行前首先要对该条件表达式求值。如果变量 c 的值大于或等于字符'a'并且小于或等于字符'z'，那么变量 c 就是小写字母，表达式 c-'a'+'A'的值是 c 对应的大写字母，作为 putchar 函数的参数将被输出；否则，字符 c 作为 putchar 函数的参数被输出。

对于语句 while ((c=getchar())!=EOF)，如果从键盘上输入一段文字后，再按 Ctrl+Z 和回车键，在 Windows 系统上是这样处理的：由于回车键的作用，Ctrl+Z 以及其前面的字符一并被送到输入缓冲区，其后字符被忽略不被送到缓冲区。如果 Ctrl+Z 前面有其他

字符, Ctrl+Z 在缓冲区的字符码是 26, 否则是 -1。然后从缓冲中读取相应的数据, 如果都读取完了, 则输入缓冲区重新变为空, getchar 函数等待新的输入。

可见, Windows 下的 Ctrl+Z 只有在行首输入才表示结束输入流, 否则表示结束本行。请读者自行上机验证, 比较在行首和不在行首输入 Ctrl+Z 两种方式的输出有何不同。

3.2 格式化输入与输出

getchar 和 putchar 函数只能输入和输出单个字符, 输入和输出过程中不进行数据格式上的任何转换。scanf 和 printf 函数能够按照用户指定的格式输入和输出若干个数据, 称为格式输入和格式输出。

3.2.1 格式输出函数 printf

printf 的功能是按照用户指定的格式向标准输出(一般默认为显示器)输出若干个数据。其函数原型为

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

说明: 函数 printf 在输出格式 format 的控制下, 将其参数进行转换与格式化, 并在标准输出设备上显示出来。其返回值的类型为 int, 返回值为实际输出的字符个数。

第一个形式参数 format 是一个字符串, 称为格式字符串, 用来指定输出数据的个数和输出格式; “...”表示其余参数的数目可变, 可以是 0 个到多个。其余参数是要被输出的数据, 参数的个数和数据类型应与格式字符串中转换说明的个数和转换字符一致。

格式字符串包含两种字符: 普通字符和转换说明字符。普通字符照原样输出, 转换说明字符并不直接输出, 它用来控制其余参数的转换和输出。每个转换说明都由一个百分号字符(%)开始, 并以一个转换字符结束。

在字符 % 和转换字符之间可以加域宽说明, 用来指出输出时的对齐方式、输出的数据域宽度、小数部分的位数等格式要求。域宽可以依次是下列常用字符中一个或多个的组合: -m.nX, 其中:

- (1) 负号(-), 用于指定被转换的参数按照左对齐的形式输出。
- (2) 正整数(m), 用于指定输出数据的最小宽度(列数)。转换后的参数将输出不小于 m 的宽度, 如果数据的实际宽度小于 m, 则左边(左对齐时为右边)用空格填充以保证最小宽度。
- (3) 小数点(.), 用于将数据宽度和精度分开。
- (4) 正整数(n), 用于指定精度, 即指定字符串中要输出的最大字符数、浮点数小数点后的位数、整数最少输出的数字数目。
- (5) 字母(X), 可以是 h、l 或 L, 字母 h 表示将整数作为 short 类型输出, 字母 l 表示将整数作为 long 类型输出, 字母 L 表示输出参数是 long double 类型。

表 3-1 列出了 printf 函数的常用转换字符。其中最常用的转换字符是: d(输出十进制整数)、u(输出十进制无符号整数)、c(输出单个字符)、s(输出字符串)、f(输出小数形式的浮点数)。

表 3-1 printf 函数的常用转换字符

转换字符	参数类型	输出格式
c	char	输出单个字符
s	char *	顺序输出字符串(必须以'\0'结束或在域宽说明中给出长度限制)
d 或 i	int	以十进制形式输出带符号整数(正数不输出符号)
o	int	以八进制形式输出无符号整数(不输出前导 0)
x 或 X	int	以十六进制形式输出无符号整数(不输出前导 0x 或 0X)
u	int	以十进制形式输出无符号整数
f	double	输出小数形式的浮点数,小数的位数默认为 6 位
e 或 E	double	以标准指数形式输出浮点数,尾数部分的位数默认为 6 位
g 或 G	double	在不输出无效 0 的前提下,按输出域宽度较小的原则从 %f 和 %e 两种格式中自动选择
p	void *	指针值(输出格式与具体实现有关)
%	不转换参数	输出一个 % 字符

【例 3.5】 整数数据的输出。

```
#include <stdio.h>
int main()
{
    int x=32768,y=-1;
    printf("x=%d, y=%i\n", x, y);
    printf("x=%d, x=0%o, x=0x%x\n", x, x, x);
    printf("y=%d, y=0%o, y=0x%x\n", y, y, y);
    printf("x=%2d, x=%8d, x=%08d, x=%-8d, x=%hd\n", x, x, x, x, x);
    return 0;
}
```

程序输出结果:

```
x=32768, y=-1
x=32768, x=0100000, x=0x8000
y=-1, y=03777777777, y=0xffffffff
x=32768, x=   32768, x=00032768, x=32768  , x=-32768
```

分析: x 和 y 按照八进制和十六进制数输出时,可以在 % 前面增加 C 语言的八进制和十六进制数前导说明符,该说明符作为普通字符输出。

在第四个 printf 函数中,第 1 个格式说明为 %2d,其格式说明的最小宽度小于数据的实际位数,按照实际位数输出,输出为 32768。第 2 个格式说明为 %8d,由于输出的最小宽度为 8,默认右对齐方式,而数据 x 只有 5 位,所以在输出数据 x 左边填补了 3 个空格。第 3 个格式说明为 %08d,0 表示多出的空位填充 0,对应输出为 00032768,左边填补了 3 个 0。第 4 个格式说明为 %-8d,负号表示左对齐,空格填充在右边,对应输出为 32768。

请读者思考,变量 y 按照无符号八进制和十六进制形式输出时,其值为什么发生了变化? 变量 x 按照格式说明 `%hd` 输出时,其值为什么也发生了变化?

【例 3.6】 浮点型数据的输出。

```
#include <stdio.h>
int main()
{
    float x=3.2768;
    double y=314.15926;
    printf("x=%f, y=%f\n", x, y);
    printf("x=%5.2f, y=%6.3f\n", x, y);
    printf("x=%-10.5f, y=%4f\n", x, y);
    printf("x=%12.8f, y=%012.4f\n", x, y);
    printf("x=%e, x=%12.2E, x=%g\n", x, x, x, x);
    return 0;
}
```

程序输出结果:

```
x=3.276800, y=314.159260
x= 3.28, y=314.159
x=3.27680 , y=314.159260
x= 3.27679992, y=0000314.1593
x=3.276800e+000, x= 3.28E+000, x=3.2768
```

分析:

(1) 输出结果第 1 行分别输出 x 、 y 的值,不加域宽限制时,默认输出 6 位小数。

(2) 输出结果第 2 行,按照 `%5.2f` 格式输出 x 的值,5 表示输出数据的最小域宽,即占 5 个字符宽度,2 表示输出数据的精度,即 2 位小数。由于 x 有 4 位小数,按四舍五入保留 2 位,即 x 的值为 3.28,其域宽(含小数点)只有 4 位,所以输出 x 的值时左边添加了一个空格。类似地,按照 `%6.3f` 格式输出 y 的值为 314.159,由于最小域宽 6 比 y 的实际域宽(7)小,所以按照实际宽度输出。

(3) 输出结果第 3 行,按照 `%-10.5f` 格式输出 x ,为“3.27680 ”,注意右边有 3 个空格,负号(-)表示按左对齐方式输出,10 表示输出 x 值时占 10 个字符宽度, x 的域宽只有 7 位,不足的 3 位补空格,数字 5 表示输出 x 值有 5 位小数,不足 5 位时最后补 0。按照 `%4f` 输出 y 时,4 表示输出的域宽,由于 y 的域宽大于 4,所以按照 y 的实际域宽输出,数据精度默认为 6 位,不足的话就补 0。

(4) 输出结果第 4 行,`%12.8f` 表示按照输出宽度 12、8 位小数、右对齐格式输出 x 的值,注意小数部分是有误差的。`%012.4f` 中的 0 表示左边不足的位填充 0。

(5) 输出结果第 5 行,按照 `%e` 格式输出 x 的指数形式 3.276800e+000,没有加域宽限制,默认输出 6 位小数,指数部分的位数固定是 4 位(含符号位)。按照 `%12.2E` 格式输出 x 时,小数部分占 2 位,数据域宽只有 9 位,不足的 3 位在数据左边补空格。注意,转换字符是大写 E,输出时也采用大写 E。按照 `%g` 格式输出 x 时,选择小数形式或指数形式中输出宽度较短的那种形式输出数据,所以输出 $x=3.2768$,此时不输出无意义的 0。

【例 3.7】 字符和字符串的输出。

```
#include <stdio.h>
int main()
{
    char ch='A',str[]="Hello";
    printf("ch=%c,ch=%3c,ch=%-3c,ch=%03c,\n",ch,ch,ch,ch);
    printf("%s$\n",str);          /* $作为普通字符输出,用以标识字符串输出的结束 */
    printf("str[]={%-8s$,str[]={%8s$,str[]={%4.2s$\n",str,str,str);
    return 0;
}
```

程序输出结果：

```
ch=A, ch= A, ch=A , ch=00A,
Hello$
str[]={Hello $, str[]={ Hello$, str[]={ He$
```

分析：输出结果第 1 行，分别按照 %c、%3c、%-3c、%03c 格式输出字符变量 ch。3c 表示输出变量 ch 的域宽为 3，右对齐，左边补 2 个空格；-3c 表示域宽为 3，左对齐，右边补 2 个空格；03c 表示域宽为 3，右对齐，用 0 填充。

输出结果第 3 行，%-8s 表示按照域宽 8、左对齐格式输出字符串 str，而串 str 只有 5 个字符，不足的 3 位在右边补空格。类似地，%8s 按照域宽 8、右对齐格式输出字符串 str，在左边补 3 个空格。%4.2s 中的 4 表示输出域宽为 4，2 表示最多输出串 str 的前面 2 个字符，默认右对齐，左侧补 2 个空格。

【例 3.8】 h、l、L 的用法。

```
short a;
long b;
double x;
long double y;
printf("a=%hd,b=%ld,x=%lf,y=%Lf",a,b,x,y);
```

输出短整数用 %hd，输出长整数用 %ld，输出 double 类型的浮点数用 %f 和 %lf 都可以，输出 long double 类型的浮点数用 %Lf。

在使用 printf 函数格式化输出时，要注意以下三点。

(1) 转换说明与输出项的数据类型不一致时，系统不会提示错误，但不会得到正确的输出结果。例如：

```
int x=123;
float y=123.456;
printf("x=%f, x=%c\n",x,x);
printf("y=%d\n",y);
```

输出结果：

```
x=0.000000, x=.
```

```
y=536870912
```

请读者编程验证以上结果。

(2) 一般要求输出项列表中的每个输出项都对应一个格式说明。如果格式说明的个数多于待输出数据项个数,则多出来的格式说明将对应输出 0 或者输出一个不确定的值;如果格式说明的个数少于待输出数据项个数,则多出的数据项不被输出。例如:

```
int i=-6; double x=5.7, y=123.4567;
printf("%d,%g\n", i, y, x);          /* 输出 -6, 123.457 */
```

说明: 数据项 x 因为无对应的转换说明而未被输出。

```
printf("i=%d\n");
```

说明: 输出 i=4109,4109 是一个不确定的值,原因在于转换说明 %d 没有对应的输出数据项。

(3) 如果需要输出字符 %, 则可以在转换说明中用两个 % 表示。例如:

```
printf("%f%%", 1.0/4);              /* 输出 0.250000% */
```

3.2.2 格式输入函数 scanf

格式输入函数 scanf 执行的是格式输出函数 printf 的逆操作,其功能是按照用户指定的格式,从标准输入设备读取字符流,存放到输入参数指定的内存单元。其函数原型为

```
int scanf(const char * format, ...);
```

说明: 函数 scanf 从标准输入设备中读取字符序列,按照 format 的格式说明对字符序列进行解释,并把结果保存到其余的参数中,这些参数都必须是指针,用于指定经格式转换后的对应的输入数据存放的位置。

函数 scanf 的返回值类型为 int,值是成功输入的数据项个数。如果出错或遇到文件尾,则返回值为 EOF。

参数表中,format 是格式说明字符串,用来指定输入数据的数目、类型和格式,“...”表示其他参数数目是可变的,其他参数必须是用来存放输入数据的内存地址。

在调用函数 scanf 时,一般至少需要读入一个数据。也就是说,除格式说明字符串外,实际参数至少应有一个输入参数。输入参数 1 至输入参数 n 可以是基本类型变量的地址(即指针)或指针类型变量。用于输入字符串数据的参数应该是字符类型的指针,可以是字符数组名或指向字符数组首元素的指针变量。此外,输入参数在类型、数目和次序上应与格式说明字符串中的转换说明一致。

函数 scanf 的格式说明字符串与 printf 相似,用于控制输入的转换。格式说明字符串可能包含下列三个部分。

- (1) 空白字符: 包括如空格符、制表符,它们被忽略,对数据的输入来说没有影响。
- (2) 普通字符(不包括 %): 在输入流中相应位置必须有相同的字符与之匹配。
- (3) 转换说明: 以 % 开头,以转换字符为结尾的转换说明,形式是:

% [选项]转换字符

说明：[选项]由程序设计人员根据实际输入的需要来选择，可供选择的字符有以下三种。

(1) 赋值禁止字符 *：用于表示跳过它对应的某个输入域，称为虚读。

(2) 正整数 m：用于指定输入数据的最大域宽。系统自动地截取数据，即从自然输入域中取前 m 个字符作为实际输入域。自然输入域不足 m 个字符，则用完自然输入域为止。

(3) h、l 或 L 字符：用于和整数或浮点数转换字符一起输出各种类型的整型数或浮点数。字符 h 修饰 d、i、o、u 或 x，用于输入短整数；字符 l 修饰 d、i、o、u 或 x，用于输入长整数；字符 l 修饰转换字符 e、f 或 g，用于输入双精度浮点数；字符 L 修饰转换字符 e、f 或 g，用于输入长双精度浮点数。

常用的 scanf 函数转换字符如表 3-2 所示。

表 3-2 常用的 scanf 函数转换字符

转换说明字符	参数类型	输入数据
d	int *	十进制整数
i	int *	整数。可以是八进制(有前导 0)或十六进制(有前缀 0x 或 0X)
o	int *	八进制整数(有无前导 0 均可)
u	unsigned int *	无符号十进制整数
x	int *	十六进制整数(有无前缀 0x 或 0X 均可)
c	char *	字符。输入字符数由域宽给定，未指定域宽时输入一个字符，参数可为字符或 int 变量的地址，存放时不在尾部添加 '\0'。不跳过输入流中的空白字符。若需读入一非空白字符，可使用 %1s
s	char *	无空白字符的字符串(不加引号)。跳过输入流中的空白字符，存放在参数指定的内存地址时在尾部添加 '\0'
e, f, g	float *	浮点数。可以无符号，可以无小数点，也可以无指数部分
%	无参数	%。不进行任何赋值操作

在实际使用中，scanf 函数的格式字符串一般只包含转换说明，对于除空格和制表符外的其他普通字符，在输入流中相应位置必须输入相同的字符与之匹配，否则当 scanf 函数碰到某些输入无法与格式控制说明匹配的情况时，该函数将终止执行。

由此可见，在 scanf 函数的格式说明字符串中加入了除空格和制表符以外的普通字符，不仅给数据输入带来麻烦，而且容易出错。但是，如果在格式字符串中每个转换说明之间适当添加空格或制表符(按 Tab 键输入)，则可以使转换说明看上去清晰明了，同时对数据的输入没有影响。

【例 3.9】 数值型数据的输入。

```
#include <stdio.h>
int main()
{
    int x1, x2, x3, x4;
    float y1, y2;
    scanf("%d%d", &x1, &x2);
```

```
scanf("%3d% * 2d%3d% * d", &x3, &x4);
scanf("%f%4f", &y1, &y2);
printf("x1=%d, x2=%d, x3=%d, x4=%d\n", x1, x2, x3, x4);
printf("y1=%f, y2=%f\n", y1, y2);
return 0;
}
```

用□表示一个空格,↵表示回车符,其输入数据格式如下:

```
12□345↵
18140669932↵
3.1415926□6.18931↵
```

程序输出结果:

```
x1=12, x2=345, x3=181, x4=669
y1=3.141593, y2=6.180000
```

分析:

(1) 第1行输入数据12和345,两个数据用一个空格隔开,整数12存入变量x1中,整数345存入变量x2中。

(2) 第2行输入数据18140669932,根据函数scanf的格式字符串%3d% * 2d%3d% * d,第1个格式说明符%3d,系统自动读取前面3个数字,即181,将其存入变量x3中;第2个格式说明符% * 2d,即跳过2个数字,即40(虚读);第3个格式说明符%3d,系统自动读取下面的3个数字,即669,将其存入变量x4中;第4个格式说明符% * d用来跳过最后剩余的数据。

(3) 第3行输入两个实数,并用一个空格隔开。第1个格式说明符%f,默认为6位小数(多于6位则四舍五入);第2个格式说明符%4f,限制域宽为4个字符,读取前面4个字符,存入变量y2中,默认为6位小数,所以后面补0,即y2=6.180000。

【例 3.10】 字符型数据的输入。

```
#include <stdio.h>
int main()
{
    char ch1, ch2, ch3, ch4, str[100];
    scanf("%c %c", &ch1, &ch2);
    getchar();
    scanf("%c%c", &ch3, &ch4);
    scanf("%s", str);
    printf("ch1=%c, ch2=%c, ch3=%c, ch4=%c\n", ch1, ch2, ch3, ch4);
    printf("str[]={%s\n", str);
    return 0;
}
```

用□表示一个空格,↵表示回车符,其输入数据格式如下:

```
A□B↵
```

```
CD ✓  
Hello ✓
```

程序输出结果：

```
ch1=A, ch2=B, ch3=C, ch4=D  
str[]=Hello
```

分析：

(1) 输入数据第 1 行,用格式说明符 `%c%c` 控制输入格式,输入的字符 A 和 B 用一个空格隔开,用回车符结束,此时输入流中缓存了字符 A、一个空格字符、字符 B 和一个换行符,执行完第一个函数 `scanf` 之后,字符 A 存入字符变量 `ch1` 的地址空间,空格字符与格式说明符中空格字符相匹配,字符 B 存入字符变量 `ch2` 的地址空间,还有一个换行符留在输入流中,然后通过下一语句 `getchar` 函数读取该换行符,不予赋值。

(2) 输入数据第 2 行,用格式说明符 `%c%c` 控制输入格式,输入的字符 C 和 D 是连续的。

(3) 输入数据第 3 行,用格式说明符 `%s` 控制输入格式,输入字符串 Hello。

假如输入数据第 1 行输入的字符 A 和 B 不用空格隔开,输入数据第 2 行输入不变,请读者写出输出结果,并解释其原因。

假如输入数据第 1 行输入不变,输入数据第 2 行输入的字符 C 和 C 之间添加一个空格,请读者写出其输出结果,并解释其原因。

假如将程序中的函数 `getchar` 删除,输入数据第 1 行和第 2 行的输入不变,请读者写出输出结果,并解释其原因。

【例 3.11】 整数和字符串混合输入。

```
int day, year;  
char month[10];  
scanf("%d%s%d", &day, month, &year);
```

执行时输入：

```
11 Jun 2020 ✓
```

或者输入：

```
11Jun 2020 ✓
```

两种输入形式下,变量的赋值结果相同:整数 25 被赋给变量 `day`,字符串 "Jun" 被赋给数组 `month`,整数 2020 被赋给变量 `year`。但是,如果输入:

```
11 Jun2020 ✓
```

由于 Jun 与 2020 之间没有分隔,上面的输入只能构成两个输入域,而格式字符串中三个转换说明需要三个输入域,因此函数 `scanf` 等待输入第三个数据。

例 3.11 表明,在整数后面接着输入字符串时,整数和字符串之间可以有也可以没有空白字符;而在字符串后面接着输入整数时,字符串和整数之间必须用空白字符分隔。

【例 3.12】 短整型数、长整型数、双精度浮点数、长双精度浮点数输入。

```
#include <stdio.h>
int main()
{
    short i;
    long j;
    double x;
    long double y;
    scanf("%hd %ld %lf %Lf", &i, &j, &x, &y);
    printf("i=%hd, j=%ld, x=%lf, y=%Lf\n", i, j, x, y);
    return 0;
}
```

输入数据：

```
32767 2147483647 18140669939.456 1867296993931.9312567
```

程序输出结果：

```
i=32767, j=2147483647, x=18140669939.456001, y=1867296993931.931200
```

注意：输入 double 类型数据时转换说明是 %lf，不能用 %f 或 %Lf。

此外，请读者输入的 i 值大于 32767，j 值大于 2147483647，写出其输出的结果，并分析产生该结果的原因。

在使用函数 scanf 时，需要注意以下几点。

(1) 函数 scanf 中的参数地址列表必须是变量的地址，即地址运算符 &+变量名，或者是字符数组名或指向字符数组首元素的指针变量。如果只写变量名，不加地址符 &，程序在编译时不会提示错误，但在运行程序时会出错，无法正确运行。例如：

```
int day, year;
scanf("%d%d", day, year);
```

当输入 21 2020 时，程序异常终止运行。

(2) 转换说明字符与输入的数据不一致时，可能导致不可预期的后果。

如果转换字符与输入参数的类型不匹配，则导致读入的数据值不正确或程序非正常终止。如果转换说明的个数比输入数据的个数少，则无对应转换说明的变量不被赋值；如果转换说明的个数比输入参数的个数多，则可能死机。例如：

```
int i, j;
double x;
scanf("%d%d", &i, &j);
```

执行时若输入

```
12a ✓
```

则整数 12 被赋予 i，由于第 2 个输入域是字符 a，与第 2 个转换说明 %d 不匹配，因而不能被转换，j 未被赋值；scanf 函数执行时返回值为 1，表明从输入流中读取并转换的数据个数是

1. 又如:

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

因为上面 scanf 函数的格式字符串中有两个转换说明,但后面只有一个输入参数,则该语句执行时可能造成死机。再如:

```
scanf("%f", &x);
```

上面语句中的转换说明 %f 与参数 &x 的类型不匹配,假如输入数据 3.14159,而实际赋值给变量 x 的却是一个错误的值(不是 3.14159)。所以,输入 double 类型的浮点数要用转换说明 %lf。

(3) 当函数 scanf 的格式字符串中包含有非空白字符的普通字符时,在输入流中相应位置必须有相同的字符与之一一对应。例如:

```
scanf("%d,%d,%d", &i, &j, &k);
```

上面语句中的格式字符串里有两个普通字符“,”,在输入数据时,必须要在输入的三个整数间分别加一个字符“,”以与格式字符串中的字符“,”相对应或相匹配,当然也可以另加空白字符,但不能加其他非空白字符的普通字符;否则,程序运行到这个地方会发生错误。

例如,以下输入可以使程序正常执行:

```
10, 20, 30 ✓
```

或

```
10 ✓  
, 20, 30 ✓
```

但是,输入以下格式的数据会使程序运行出错:

```
10 20, 30 ✓
```

或

```
10,, 20, 30 ✓
```

读者可编写程序验证,以加深对函数 scanf 的理解。

(4) 输入实数时不能规定小数点的位数。例如:

```
float x, y;  
scanf("%3f %3.2f", &x, &y);  
printf("x=%f ,y=%f\n", x, y);
```

输入数据格式:

```
931 456
```

输出结果:

```
x=931.000000,y=0.000000
```

显然,由于在函数 scanf 中的转换说明 %3.2f 中含有指定的小数位数说明,所以变量 y

无法得到正确的结果。

本章小结

在编写程序时,函数 `getchar`、`putchar`、`scanf` 和 `printf` 使用频度非常高,因此需要理解和掌握其用法,做到灵活应用。格式化输入输出中常用转换说明符 `%c`、`%d`、`%s`、`%f` 等的用法需要记住。至于函数 `printf` 转换说明的域宽说明字符和函数 `scanf` 转换说明选项的用法,不必死记硬背,了解即可,在使用时这些细节可以查阅相关书籍和手册。

习题 3

3.1 简述函数原型的概念和用途。写出 `putchar`、`getchar`、`printf` 和 `scanf` 函数的函数原型的入口参数和出口参数的含义。

3.2 写出下面程序的运行结果。

```
#include<stdio.h>
int main(void)
{
    unsigned int x1 = 65535;
    int x2 = -3;
    float y1 = 123.4567, y2 =123.4500;
    printf("x1=%d,%o,%x,%u\n", x1, x1, x1, x1);
    printf("x2=%d,%o,%x,%u\n", x2, x2, x2, x2);
    printf("y1=%10f,%10.3f,% .3f,%-10.3f\n", y1, y1, y1, y1);
    printf("y2=%f,%e,%g\n", y2, y2, y2);
    printf("x1 (%#4d) =%4d", x1);
    return 0;
}
```

3.3 阅读下列程序,为保证程序正确执行,将应填入的内容写在程序后面对应的编号后。

```
#include<stdio.h>
int main(void)
{
    unsigned char uc;
    short h;
    unsigned long ul;
    float f;
    printf("input a character:");
    scanf("    (1)    ", &uc);
    printf("    (2)    ", uc >= 'a' && uc <= 'z'? uc - 'a' + 'A': '? ');
    printf("    (3)    ", f = uc);
    printf("    (4)    ", 3.14 * uc * uc);
    printf("    (5)    ", (ul = uc, ul + 1));
    printf("    (6)    ", h = uc << 8 | uc + 1);
}
```

```

printf("    (7)    ", uc << 8 | uc - 1);
printf("    (8)    ", uc %2L);
printf("\ninput a float:");
scanf("    (9)    ", &f);
printf("    (10)   ", (long double)f);
return 0;
}
(1) _____ (2) _____
(3) _____ (4) _____
(5) _____ (6) _____
(7) _____ (8) _____
(9) _____ (10) _____

```

3.4 现有以下格式输入输出语句,为了得到正确的输出结果,请写出正确的输入内容,同时写出输出结果。

```

int a,b;
float x,y;
scanf("a=%d,b=%d",&a,&b);
scanf("%f,%f",&x,&y);
printf("a=%d,b=%d",a,b);
printf("%f%%, %f",x,y);

```

3.5 根据变量说明指出下面的语句哪些是正确的,哪些是错误的。

```

char c = 'A';
int i1 = 1;
const int i2 = -1;
long i3 = 3;
unsigned i4 = 0;
float x1 = 1;
double x2 = 3;
long double x3 = 1000;

printf("%d", (17 / 15) %i4 + c); /* (1) */
putchar(c - 'A' + 'a' + 1); /* (2) */
printf("% * c", i2 & 0xf, ' * '); /* (3) */
printf("x2=%f, x3=%lf\n", x2, x3); /* (4) */
printf("%lf\n", getchar() != EOF ? x1 * 3.14 : x3); /* (5) */
scanf("%u%f", &i4, &x2); /* (6) */
printf("%x,%lo\n", c << ++i4 | 1 / 2, i3 + '0'); /* (7) */
putchar(i2 = getchar()); /* (8) */
printf("%ld,%f\n", c + i1 * i4, (x2, x1, i2, c)); /* (9) */
scanf("% * c%lf", &x2); /* (10) */

```

3.6 如果调用函数 scanf("%d%d",&a,&b)来输入两个数,试比较和分析以下几种输入得到的结果,并通过编程验证比较分析的结果是否正确。

- (1) 在同一行中输入 39 和 8,并以空格分隔。

(2) 在不同的两行中输入 39 和 8。

(3) 分别在(1)和(2)的输入数字的前面和后面加入大量的空格或水平制表符或空行。

(4) 把 8 换成字符 E,重复(1)至(3)的输入操作。

3.7 编程利用函数 printf 实现以下几种输出。

(1) 仅用一条 printf 语句,打印 1+2 和 2+4 的值,用两个空行隔开。

(2) 试着把 %d 中的两个字符(%和 d)输出到屏幕。

(3) 试着把 \n 中的两个字符(\和 n)输出到屏幕。

3.8 从终端输入一个字符,如果该字符是十六进制数字,则输出它对应的整数,否则输出它的字符码。

3.9 输入无符号短整数 k,输出将 k 的高 4 位和低 4 位交换后的结果。

3.10 输入某轿车行驶的公里数和消耗的汽油(升),然后计算和显示该轿车每升汽油行驶的公里数以及该轿车的百公里油耗值。均为 1 位小数。

3.11 输入三角形三边长度值(均为正整数),判断它是否能成为直角三角形的三个边长。如果可以构成直角三角形,则输出“Yes”;如果不能构成直角三角形,则输出“No”。如果根本无法构成三角形,则输出“not a triangle”。

3.12 输入无符号短整数 x、m、n, $0 \leq m \leq 15$, $1 \leq n \leq m+1$,取出 x 从第 m 位开始向右的 n 位(m 对二进制位从右向左编号为 0~15),并使其向左端(第 15 位)靠齐,输出处理后的结果。