第 3 章　格式化输入与输出

本章将学习如何使用标准库中的一些输入/输出函数，其中包括 printf()和 scanf()。虽然本书前面章节中使用过这些函数，但很多细节仍需加以解释，我们将对输入/输出的各种格式给予说明。

3.1　输出函数

printf()有两个很好的性质，这使得 printf()的使用非常灵活：

① 参数表的长度可以是任意的；

② 用简单的转换说明或格式控制显示。

printf()把它的字符流交付给标准输出流 stdout()（这个函数通常与屏幕相连）。printf()的参数有两部分：格式控制串和参数表。

我们看个例子：

```
printf("She buys %d %s for $%f", 1, "iPhone", 500.00);
```

格式控制串："She buys %d %s for $%f"

参数表：1, "iPhone", 500.00

其中，格式控制串中的格式控制符与参数表中的参数相匹配，如表 3-1 所示。

表 3-1　格式控制符与对应的参数

格式控制符	对应的参数
%d	1
%s	iPhone
%f	500.00

如果参数表中存在表达式，则对表达式求值并按照格式控制串中的格式控制符进行转换，然后把结果放进输出流。格式控制串包含两种类型的对象：普通字符和转换说明。输出时，普通字符将原样不动地复制到输出流中，而转换说明并不直接输出到输出流中，而是用于控制 printf()中参数的转换和打印。符号%用于引入转换说明或格式。单个转换说明用%开头并用一个转换字符结尾。表 3-2 是 printf()使用的转换字符。

表 3-2　printf()使用的转换字符

转换字符	对应的参数如何显示
c	作为字符
d, i	作为十进制整数
u	作为无符号十进制整数
o	作为无符号八进制整数
x, X	作为无符号十六进制整数
e, E	作为指数形式的浮点数，如 1.234e+3
f	作为带小数点的浮点数，如 3.1415
g, G	以 e（E）或 f 格式，都是较短的
s	作为串
p	相应的参数是指向 void 的指针，按十六进制数显示它的值
n	相应的参数是指向一个整数的指针，该整数是至今成功写到流或缓冲区中的字符个数，对参数不做转换
%	使用%%把%写入输出流，没有相应的参数被转换

在字符%和转换字符中间可能包含以下内容。

① 说明被转换参数的最小域宽的正整数，可选。C 语言把显示参数的位置称为域，把显示参数的格数称为域宽。如果被转换参数的字符数小于域宽，那么根据被转换参数是左调整或右调整，来决定在左边或右边填充空格。如果被转换参数的字符数大于域宽，那么就把域宽调整到所需的大小。如果定义域宽的整数从 0 开始，并且被转换参数在它的域中是右调整，那么就用 0 而不是空格进行填充。

② 精度，可选，用一个后跟非负整数的小数点描述它。对于 d、i、o、u、x 和 X 转换，描述了被显示的数字的最小个数。

③ e、E 和 f 描述了小数点右侧的数字个数。g 和 G 描述了有效数字的最大位数。S 描述了显示一个串的最大字符个数，可选项 h 或 l 分别是 short 或

long 修饰符。如果 h 后跟转换字符 d、i、o、u、x 或 X，那么转换描述适用于 short int 或 unsigned short int 型参数；如果 h 后跟转换字符 n，那么相应的参数是指向 short int 或 unsigned short int 型的指针。如果 l 后跟转换字符 d、i、o、u、x 或 X，那么转换描述适用于 long int 或 unsigned long int 型参数；如果 l 后跟转换字符 n，那么相应的参数是指向 long int 或 unsigned long int 型的指针。

④ L，可选，它是一个 long 修饰符。如果 L 后跟转换字符 e、E、g 或 G，那么转换说明适用于 long double 型参数。

⑤ 用于修改转换说明的零个或多个标志字符。

在转换说明中的标志字符包括下列内容：

- 减号，表示在它的域中被转换的参数为左对齐。若没有减号，则被转换的参数为右对齐。
- 加号，表示如果输出值是非负整数且有符号，就在其前面加一个“+”。它与 d、i、e、E、g 和 G 一起使用。所有的负数都用减号开头。
- 空格，表示如果输出值是非负的整数且有符号，就在其前面加一个空格。它与 d、i、e、E、g 和 G 一起使用。如果“+”和空格同时出现，就把空格忽略掉。
- #，表示依赖转换字符把结果转换成“可选择的形式”。与转换字符 0 一起使用，#自动在显示的八进制数前加 0。在 x 或 X 转换中，它自动在显示的十六进制数前加 0x 或 0X。在 g 或 G 转换中，它自动显示尾部的 0。在 e、E、f、g 或 G 转换中，它自动显示小数点，甚至精度为 0 也是如此。其他的转换没有定义这样的行为。
- 0，表示用 0 代替空格来填充域。与 d、i、o、u、x、X、e、E、f、g 和 G 一起使用，会导致用 0 作为前导。被显示的数之前的任何标记和任何 0x 或 0X 都优先于前导 0。

在转换说明中，宽度或精度可以用星号*表示，这时，其值通过转换下一个参数来计算。例如：

```
#include<stdio.h>
int main()
{
    int m,n;
    m=12;
    n=8;
    float x=3.1415;
    printf("x=%*.*f\n",m,n,x);      /*相当于 printf("x=%12.8f\n",,x);*/
    return 0;
}
```

在这里，宽度为 12，精度为 8，x= 3.14150000（左边填充两个空格）。

表 3-3 给出了格式化显示数字举例。我们用双引号把字符括起来以界定显示内容，实际上不显示双引号。

表 3-3　格式化显示数字举例

初始化和声明：int i=1234; double x=0.123456789;

格　式	对 应 参 数	在其域中的打印结果	说　明
%d	i	"1234"	默认域宽为 3
%6d	i	"001234"	填充 0
%7o	i	" 2322"	右调整，八进制数
%-9x	i	"4d2 "	左调整，十六进制数
%-#9x	i	"0x4d2 "	左调整，十六进制数
%10.5f	x	" 0.12346"	域宽 10，精度 5
%-12.5e	x	"1.23457e-001 "	左调整，e 格式

【例 3.1】　下面程序演示表 3-3 中的格式化显示数字举例。

```
#include<stdio.h>
int main()
{
    int i=1234;
    double x=0.123456789;
    printf("|%d|%6d|%7o|%-9x|%-#9x|\n",i,i,i,i,i);
    printf("|%10.5f|%-12.5e|\n",x,x);
    return 0;
}
```

程序的运行结果如下：

```
|1234|   1234|    2322|4d2      |0x4d2    |
|   0.12346|1.23457e-001|
```

程序第 6 行第一个格式控制符%d 表示按变量 i 的实际域宽显示；第二个格式控制符%6d 表示域宽为 6，而 i 的域宽为 4，所以显示时在 1234 的左边填充两个空格；第三个格式控制符%7o 表示域宽为 7，而 i 的八进制值只有 4 位，所以显示时在 i 的八进制值的左边填充三个空格；第四个格式控制符%−9x 表示左对齐显示 i 的十六进制值，域宽为 9，而 i 的十六进制值 4d2 只有 3 位，所以显示时在 4d2 的右边填充 7 个空格；第 5 个格式控制符%−#9x 表示在 i 的十六进制值前面自动显示 0x，即 0x4012。

程序第 7 行第一个格式控制符%10.5f 表示域宽为 10，精度为 5，即保留小数点后 5 位（四舍五入），所以显示时在 0.12346（0 和小数点各占 1 位）的左边填充三个空格；第二个格式控制符%−12.5e 表示域宽为 12，精度为 5，左对齐显示变量 x 的指数形式的浮点数，即有效数字为 7 位，指数为 5 位。

表 3-4 给出了格式化显示字符和串举例。我们用双引号把字符括起来以界定显示内容，实际上不显示双引号。

表 3-4　格式化显示字符和串举例

初始化和声明：char　ch='W',　s[]="Blue moon!"

格式控制符	对 应 参 数	在其域中的打印结果	说　　明
%c	ch	"W"	默认域宽为 1
%2c	ch	" W"	域宽为 2，右调整
%−3c	ch	"W　"	域宽为 3，左调整
%s	s	"Blue moon!"	默认域宽为 10
%3s	s	"Blue moon!"	需要更多的格
%.6s	s	"Blue m"	精度为 6
%−11.8s	s	"Blue moo　　"	精度 8，左调整

【例 3.2】　下面程序演示表 3-4 中的格式化显示字符和串举例。

```
#include<stdio.h>
int main()
{
    char ch='W';
    char s[]="Blue moon!";
    printf("|%c|%2c|%-3c|\n",ch,ch,ch);
    printf("|%s|%3s|%.6s|%-11.8s|\n",s,s,s,s);
    return 0;
}
```

程序运行结果如下：

```
|W| W|W    |
|Blue moon!|Blue moon!|Blue m|Blue moo      |
```

程序第 6 行显示字符变量 ch 的值'W'，第一个格式控制符%c 表示按字符实际域宽显示；第二个格式控制符%2c 设置域宽为 2，而字符'W'域宽为 1，所以显示时在'W'的左边填充一个空格；第三个格式控制符%-3c 设置域宽为 3，负号表示左对齐，所以显示时在'W'的右边填充两个空格。

程序第 7 行显示字符数组 s 的值（即字符串"Blue moon!"），第一个格式控制符%s 按字符串实际长度显示；第二个格式控制符%3s 设置的域宽小于字符串的实际长度，也按字符串的实际长度显示；第三个格式控制符%.6s 设置字符串的精度为 6，因而只能显示字符串的前 6 个字符（包括中间的一个空格）；第 4 个格式控制符%-11.8s 的负号表示左对齐，11 表示显示域宽为 11，尽管字符串的长度为 10，但由于规定精度为 8，所以只能显示其中的前 8 个字符（包括中间的一个空格），右边剩余 3 个字符用空格填充。

3.2 输入函数

scanf()有两个很好的性质，这使得 scanf()具有高度灵活性：① 参数表的长度可以是任意的；② 用简单的转换说明或格式控制输入。scanf()从标准输入流 stdin()（这个函数通常与键盘相连）中读字符。scanf()的参数有两部分：格式控制串和参数表。我们看个例子：

```
int    num;
char ch1, ch2, ch3, ch[25];
double x;
scanf("%c%c%c%d%s%lf", &ch1, &ch2, &ch3, &num, ch, &x);
```

格式控制串："%c%c%c%d%s%lf"

参数表：&ch1, &ch2, &ch3, &num, ch, &x

格式控制串后面的参数由用逗号分隔的指针或地址表达式组成。**注意**，本例中，如果用&ch，则会产生错误，因为表达式 ch 本身就是地址（ch 是字符数组，数组名为数组首地址）。

scanf()的格式控制串由 3 种字符组成：普通字符、空白字符和转换说明。普通字符是格式控制串中除空白字符和转换说明中的字符以外的字符。普通字符必须与输入流中的字符相匹配。例如：

```
float    price;
scanf("$%f", price);
```

字符$是一个普通字符，必须在输入流中匹配$。如果匹配成功，那么将跳过空白字符（如果有的话），并匹配能转换成浮点数的字符，把转换的值放在内存中 price 的地址处。我们再看一个例子，字符 h、a、i 都是普通字符：

```
scanf("hai");
```

首先匹配字符 h，其次匹配字符 a，最后匹配字符 i。如果在某处不能进行匹配，就在输入流中留下非法字符，scanf()返回。如果调用 scanf()成功，就可以对输入流中 h、a、i 后面的字符进行处理。

在格式控制串中但不在转换说明中的空白字符可以匹配输入流的空白字符，也可以不匹配。例如：

```
char c1,c2,c3;
scanf("   %c    %c    %c",&c1,&c2,&c3);
```

这条语句在各格式控制符前面均有一个空格，如果输入流含有字符 a、b、c，则无论它们有无前导空格，也无论是否用空白字符把它们隔开，都会把字符 a、b、c 分别读入变量 c1、c2、c3 中。空白指示使得输入流中的空白字符（如果有）被跳过。

注意：如果在格式控制符%c 的前面没有空白字符，则上例从键盘输入字符 a、b、c 时字符之间不能有空格，只能连续输入 abc，然后回车，才能使变量 c1、c2、c3 分别得到字符 a、b、c。

否则，如果输入 a　b　c，然后回车，那么变量 c1、c2、c3 只能分别得到字符 a、空格、字符 b。

在 scanf()的格式控制串中，转换说明用%开始，用一个转换字符结束。它决定了怎样匹配转换输入流中的字符。scanf()使用的转换字符如表 3-5 所示。

表 3-5　scanf()使用的转换字符

未加修饰的转换字符	在输入流中被匹配的字符	对应参数的类型
c	任何字符，包括空白	char
d	可选的有符号十进制整数	int
i	可选的有符号十进制整数、八进制整数或十六进制整数	int
u	可选的有符号十进制整数	unsigned
o	可选的有符号八进制整数，不需要前导 0	unsigned
x, X	可选的有符号十六进制整数，不允许前导 0x 或 0X	unsigned
e, E, f, g, G	可选的有符号浮点数	float
p	printf()中的%p 所产生的通常是无符号十六进制整数	void

scanf()还有三个特殊转换字符，如表 3-6 所示。

表 3-6　scanf()的特殊转换字符

未加修饰的转换字符	说　　明
n	在输入流中没有任何字符被匹配。相应的参数是指向 int 的指针，它存储的是已读入的字符数
%	转换字符%%使得输入流中的%被匹配，没有任何相应的参数
[…]	把在[ ]中的字符集称为扫描集，它决定了匹配什么和读入什么。相应的参数是指向字符数组的基地址。该数组足够大，能容纳被匹配的字符，并用自动加入的空字符\0 结尾

在字符%和转换字符中间可能包含以下内容。

① 可选的*，它表示赋值抑制，其后跟一个定义最大扫描宽度的可选整数，此外还跟有修饰符 h、l 或 L。

② 修饰符 h，它可在转换字符 d、i、o、u、x 或 X 的前面。它表示把被转换的值以 short int 或 unsigned short int 型存储。

③ 修饰符 l，它可在转换字符 d、i、o、u、x 或 X 的前面，或者在 e、E、f、g 或 G 的前面。在第一种情况下，它表示把被转换的值以 long int 或 unsigned long int 型存储，在第二种情况下，它表示把被转换的值以 double 型存储。

④ 修饰符 L，它可在 e、E、f、g 或 G 的前面，它表示把被转换的值以 long double 型存储。

按照格式控制串中的转换说明把输入流中的字符转换成值，并把值存储在由参数表中的相应指针表达式指向的地址处。除字符输入外，扫描域由连续的适合于指定转换的非空白字符组成。在遇到不适合的字符时，就结束扫描域。如果超过了扫描宽度或遇到了文件结束标记，也结束扫描域。

扫描宽度是被扫描的字符数，默认是输入流的长度。%s 表示跳过空白字符，然后读入非空白字符，直到遇见空白字符或文件结束标记为止。%5s 表示跳过空白字符，然后读入非空白字符，直到读入了 5 个字符，遇见空白字符或文件结束标记为止。在读入串时，假定在内存中已经分配足够的空间，它能容纳读入的串和自动加上的串结束标记\0。可以用%nc 读入 n 个字符，其中包括空白字符。在读入串时，假定在内存中已经分配足够的空间，它能容纳读入的串，字符\0 并没有自动加上。

%[string]表示读入具体的串，把[ ]中字符集称为扫描集。如果扫描集中的第一个字符不是抑制字符^，那么输入的串仅由扫描集中的字符组成。例如%[abc]仅输入含有字符 a、b 和 c 的串。如果在输入流中出现其他字符，包括空白字符，那么输入就停止。例如：

```
char   str[30];
scanf("20[ab   \t\n]", str);
```

它向字符数组 str 中读入一个最多 20 个字符的串，该串由 a、b、空格、制表位和换行符组成，并以\0 结尾。

如果扫描集中的第一个字符是抑制字符^，那么输入的串由除扫描集中的字符之外的所有字符组成，而不是由扫描集中的字符组成。例如%[^abc]输入一个由字符 a、b 或 c 终结的串，而不是由空白字符终结的串。例如：

```
char   line[100];
while(scanf("%[^\n]", line)==1)
      printf("%s\n", line);
```

上述代码的作用是去掉空行，并去掉任何行的前导空白。

调用 scanf()时，可能会发生输入失败或匹配失败的情况。如果在输入流中没有字符，就会发生输入失败的情况，返回的值是 EOF。在匹配失败时，非法的字符被留在输入流中，返回已成功转换的字符数。如果没有进行转换，返回的数就是 0。如果 scanf()调用成功，就返回成功转换的字符数，这个数也可能是 0。

因此，利用 scanf()的返回值可以判断是否成功读入了指定的数据项数给程序。当然，这得使用 if-else 语句编程实现，我们将在第 4 章中详细介绍。

【例 3.3】 编写一个程序，对用户录入的产品信息进行格式化。

程序运行后需得到如下运行结果：

```
Enter item number: 456
Enter unit price: 12.3
Enter purchase date (mm/dd/yyyy): 6/24/2019
Item                Unit                Purchase
                    Price               Date
456                 $   12.30           6/24/2019
```

其中，数字项和日期项采用左对齐方式，单位价格采用右对齐方式，美元的最大取值为 9999.99。

程序如下：

```
#include <stdio.h>
int main(void)
{
   int item_number, month, day, year;
   float unit_price;

   printf("Enter item number: ");
   scanf("%d", &item_number);
   printf("Enter unit price: ");
   scanf("%f", &unit_price);
   printf("Enter purchase date (mm/dd/yyyy): ");
   scanf("%d/%d/%d", &month, &day, &year);

   printf("\nItem\t\tUnit\t\tPurchase\n");
   printf("\t\tPrice\t\tDate\n");
   printf("%d\t\t$%7.2f\t%d/%d/%d\n", item_number, unit_price, month, day, year);
   return 0;
}
```

程序第 10 行按“月/日/年”的格式输入日期，所以从键盘输入的时候就要把'/'作为普通字符随日期一起输入，不能在月、日、年之间用空格分隔，而应用'/'分隔，否则变量 month、day、year 得不到正确的输入。

为了分隔不同的输出项，程序使用了转义字符制表位'\t'来对齐各输出列的数据。

【例 3.4】 编写程序，按如下数据输入格式从键盘输入一个整数乘法表达式：

```
整数 1 * 整数 2
```

然后计算并输出该表达式的计算结果，输出格式如下：

```
整数 1 * 整数 2=计算结果
```

程序如下：

```
#include<stdio.h>
int main()
{
    int i,j;
    char op;
    printf("Please input the expression i * j\n");
    scanf("%d%c%d",&i,&op,&j);
    printf("%d%c%d=%d\n",i,op,j,i%j);
    return 0;
}
```

运行程序，先输入 2，然后输入空格，接着输入*，随后输入空格，最后输入 3，运行结果为：

```
Please input the expression i * j
2 * 3
2 1=0
```

可以看到，出现了错误的结果，为什么呢？

原因是数据没有被正确地读入。下面我们先看一下输入过程：当输入 2 时，2 被 scanf()用 d 格式控制符赋值给变量 i。接着其后输入的空格被 scanf()用 c 格式控制符赋值给变量 op，因为在 C 语言中，空格也是一个字符。这样，变量 j 的值也是错的。

可以通过修改输入格式，来验证上面的分析结果是否正确。下面是程序两次测试的运行结果：

第 1 次测试先输入 2，接着输入空格，最后输入 3，运行结果如下：

```
Please input the expression i * j
2 3
2 3=6
```

第 2 次测试先输入 2，接着输入*，最后输入 3，运行结果如下：

```
Please input the expression i * j
2*3
2*3=6
```

从上面两次测试结果可以看出，在第 1 次测试中，输入的 2、空格、3 分别被赋值给整型变量 i、字符型变量 op、整型变量 j。在第 2 次测试中，输入的 2、*、3 分别被赋值给整型变量 i、字符型变量 op、整型变量 j。这说明，当用%c 读入字符时，空格字符和转义字符（包括\n）都将被作为有效字符读入，因此，使用%c 输入时要特别注意。

【例 3.5】 编写程序，从键盘依次先后输入 int 型、char 型和 float 型数据，要求每输入一个数据就显示这个数据的类型及其值。

程序如下：

```
#include<stdio.h>
int main()
{
    int i;
    char ch;
    float f;
```

```
    printf("Please input an integer:");
    scanf("%d",&i);
    printf("integer:%d\n",i);
    printf("Please input a character:");
    scanf("%c",&ch);
    printf("character:%c\n",ch);
    printf("Please input a float number:");
    scanf("%f",&f);
    printf("float:%f\n",f);
    return 0;
}
```

程序运行结果如下：

```
Please input an integer:5
integer:5
Please input a character:character:

Please input a float number:2.3
float:2.300000
```

这个程序与例 3.4 一样，问题也是出在%c 上面，在输入数据 5 后的回车符被作为一个有效字符赋值给字符变量 ch 了。

那么，如何解决数值与字符数据混合输入造成的问题呢？

我们知道，当调用 getchar()时，会将缓冲区中的回车符清空，从而避免回车符被作为随后的字符变量当作有效字符读入。上面的程序可用这种方法修改如下：

```
#include<stdio.h>
int main()
{
    int i;
    char ch;
    float f;
    printf("Please input an integer:");
    scanf("%d",&i);
    printf("integer:%d\n",i);
    getchar();    /*清空缓冲区，避免回车符作为后续字符变量的有效输入*/
    printf("Please input a character:");
    scanf("%c",&ch);
    printf("character:%c\n",ch);
    printf("Please input a float number:");
    scanf("%f",&f);
    printf("float:%f\n",f);
    return 0;
}
```

程序运行结果如下：

```
Please input an integer:5
integer:5
Please input a character:A
character:A
Please input a float number:2.3
float:2.300000
```

另一种可行的方法是，在%c 前面添加一个空格，这样程序将忽略前面数据输入时存入缓冲区中的回车符，避免其被后续的字符变量当作有效字符输入。与前一种方法相比，这种方法更简单，程序的可读性也更强。用这种方法修改的程序如下：

```
#include<stdio.h>
int main()
{
    int i;
```

```
    char ch;
    float f;
    printf("Please input an integer:");
    scanf("%d",&i);
    printf("integer:%d\n",i);
    printf("Please input a character:");
    scanf(" %c",&ch);   /*在%c 前面留了一个空格*/
    printf("character:%c\n",ch);
    printf("Please input a float number:");
    scanf("%f",&f);
    printf("float:%f\n",f);
    return 0;
}
```

程序的运行结果如下：

```
Please input an integer:5
integer:5
Please input a character:A
character:A
Please input a float number:2.3
float:2.300000
```

用这种方法在例 3.4 程序的%c 前加一个空格后，重新编译程序，那么无论用哪种方式输入乘法表达式，都能得到正确的结果。

习题 3

一、单项选择题

1．为了输出字符串，下列哪一条语句是正确的？（　　）。

A．printf("%f",a);　　B．printf("%d",a);　　C．printf("%c",a);　　D．printf("%s",a);

2．用 scanf("%d:%d",&a,&b);语句输入数据时，数据之间必须用（　　）隔开。

A．逗号　　B．分号　　C．冒号　　D．空格

3．若 a 为 int 型，且 a=125，执行 printf("%d,%o,%x\n",a,a+1,a+2);语句后的输出是（　　）。

A．25, 175, 7D　　B．125, 176, 7F　　C．125, 176, 7D　　D．125, 175, 2F

4．若 x, y 均定义为 int 型，z 定义为 double 型，以下不合法的 scanf 函数调用语句是（　　）。

A．scanf("%d %x, %le", &x, &y, &z);

B．scanf("%2d *%d, %lf", &x, &y, &z);

C．scanf("%x %*d %o", &x, &y);

D．scanf("%x %o%6.2f", &x, &y, &z)

5．只能向终端输出一个字符的函数是（　　）。

A．printf 函数　　B．putchar 函数　　C．getchar 函数　　D．scanf 函数

二、阅读程序，并写出运行结果

1．

```
main()
{
    int n;
      (n=6*4,n+6),n*2;
    printf("n=%d\n",n);
}
```

2．

```
main()
{
    int x=2,y,z;
    x*=3+1;
    printf("%d,",x++);
    x+=y=z=5;
    printf("%d,",x);
    x=y=z;
    printf("%d\n",x);
}
```

3.

```
main()
{
    int x, y, z;
    x=0;y=z=-1;
    x+=-z---y;{(-z--)-y}
    printf("x=%d\n",x);
}
```

4.

```
main()
{
    char c1='a'，c2='b'，c3='c';
    printf("a%cb%c\tc%c\n",c1,c2,c3);
}
```

5.

```
/*运行时从键盘输入 12345 和 abc*/
main()
{
    int a;
    char ch;
    scanf("%3d%3c",&a,&ch);
    printf("%d, %c" ,a, ch);
}
```

6.

```
main()
{
    unsigned x1;
    int b= -1;
    x1=b;
    printf("%u",x1);
}
```

7.

```
#include<stdio.h>
#include<math.h>
main()
{
    int a=1,b=4,c=2;
    float x=10.5, y=4.0, z;
    z=(a+b)/c+sqrt((double)y)*1.2/c+x;
    printf("%f\n", z);
}
```

8.

```
main()
{
    int a=2, c=5;
    printf("a=%%d, b=%%d\n", a, c);
}
```

9.

```
/*运行时从键盘输入 9876543210<CR>（<CR>表示回车）*/
main()
{
    int a;
    float b, c;
    scanf("%2d%3f%4f",&a,&b,&c);
    printf("\na=%d, b=%f, c=%f\n", a, b, c);
}
```

实验题

实验题目：国际标准图书编号。

实验目的：熟悉格式化 scanf()、printf()的使用，以及不同类型数据的输入/输出方法。

说明：图书用国际标准图书编号（ISBN）进行标识，如 0-393-30375-6。其中，第 1 位数字说明编写书籍所用的语言，第 1 组数字是出版社的编号，第 2 组数字则是由出版社指定的、用来识别图书的编号，最后 1 位数字是校验位，它用来验证前面数字的准确性。编写程序，分解用户输入的 ISBN，格式如下：

```
Enter ISBN: 0-393-30375-6
Language: 0
Publisher: 393
Book Number: 30375
Check digit: 6
```