第 5 章 输入与输出

在程序的运行过程中，往往需要由用户输入一些数据，这些数据经机器处理后要输出反馈给用户。通过数据的输入与输出来实现人与计算机之间的交互，所以在程序设计中，输入与输出语句是一类必不可少的重要语句。

常用的输入与输出函数有 getchar()、putchar()、scanf()、printf()。ANSI 标准精确地定义了这些库函数，所以，在任何可以使用 C/C++ 语言的系统中都有这些函数的兼容形式。如果程序的系统交互部分仅仅使用了标准库提供的功能，则可以不加修改地从一个系统移植到另一个系统中。

5.1 getchar()函数

getchar()函数的原型如下：

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

此函数的功能是从 stdio 流中读字符。C 语言中，在没有输入时，getchar()函数将返回一个特殊值，这个特殊值与任何实际字符都不同，这个值称为 EOF(end of file，文件结束)，它的值通常是－1。

getchar()函数只能接收单个字符，如果输入的是数字也按字符处理。输入多于一个字符时，只接收第一个字符。

【例 5.1】 统计输入中的行数及字符数。

输入样例：

```
The
C
Programming
Language
```

输出样例：

```
4 27
```

【分析】

标准库保证输入文本流以行序列的形式出现，每一行均以换行符'\n'结束。因此，统计行数等价于统计换行符的个数。而字符计数，就是读一个字符，计数一次。

```
#include<bits/stdc++.h>
using namespace std;
```

```
int main()
{
    //定义 3 个整型变量,并且给变量 nLine 和 nChar 赋初值 0
    int c, nLine =0, nChar =0;

    //每次读一个字符,直到文件结束
    while((c =getchar()) !=EOF)
    {
        nChar ++;              //字符计数
        if(c =='\n')           //如果读入的字符是换行符
            nLine ++;          //nLine 的值增加 1。此语句与 nLine=nLine+1 语句等价
    }
    printf("%d %d\n", nLine, nChar);

    return 0;
}
```

在程序中测试符号常量 EOF,而不是测试－1,这可以使程序具有可移植性。EOF 定义在头文件＜stdio.h＞中,ANSI 标准强调,EOF 是负的整数值,但没有必要一定是－1。因此,在不同的系统中,EOF 可能具有不同的值。在 Microsoft 公司 Windows 这样的系统中,EOF 是通过键入“Ctrl＋Z”的方式来输入的。

在声明变量 c 的时候,必须让它大到足以存放 getchar()函数返回的任何值。这里不把 c 声明成 char 类型,是因为它必须足够大,能存储任何可能的字符,所以,将 c 声明成 int 类型。

5.2　putchar()函数

putchar()函数的原型如下:

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

该函数的功能是将指定的表达式的值所对应的字符输出到标准输出终端上。表达式可以是字符型或整型,它每次只能输出一个字符。如果没有发生错误,则函数 putchar()将返回输出的字符;如果发生了错误,则返回 EOF。

【例 5.2】　把从键盘输入的字符逐个输出到显示器。

输入样例:

```
The C Programming Language
```

输出样例:

```
The C Programming Language
```

【分析】根据题意,此过程由下列两个步骤构成:

```
Step1 读入一个字符。
Step2 如果该字符不是文件结束符,则输出这个字符,重复 Step1;否则结束。
```

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
    int ch;                    //定义整型变量

    ch =getchar();             //读入一个字符
    while(ch !=EOF)
    {
        putchar(ch);           //输出一个字符
        ch =getchar();         //读入一个字符
    }

    return 0;
}
```

5.3 scanf()函数

输入函数 scanf()对应输出函数 printf(),它在后者相反的方向上提供同样的转换功能。

由于 C 语言的规则以及输出方向的不同转换,scanf()和 printf()有着很多不对称的地方,最重要的不对称性在于,printf()需要从其调用函数处获得多个值,而 scanf()则要将多个值返回给它的调用函数。

具有变长参数表的 scanf()函数的原型如下:

```
int scanf(char *format, …);
```

scanf()函数从标准输入中读取字符序列,按照 format 中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。

除格式参数 format 外,其他所有参数都必须是指针,用于指定经格式转换后的相应输入保存位置。

当 scanf()函数扫描完其格式串或者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,同时,成功匹配并赋值的输入项的个数将作为函数值返回,所以该函数的返回值可以用来确定已匹配的输入项的个数。

如果到达文件的结尾,该函数将返回 EOF。注意,返回 EOF 与 0 是不同的,0 表示下一个输入字符与格式串中的第一个格式说明不匹配,下一次调用 scanf()函数将从上一次转换的最后一个字符的下一个字符开始继续搜索。

格式串通常都包含转换说明,用于控制输入的转换。格式串可能包含以下部分:

- 空格或制表符。在处理过程中将被忽略。
- 普通字符(不包括%)。用于匹配输入流中的下一个非空白符字符。
- 转换说明。依次由一个%,一个可选的赋值禁止字符 *,一个可选的数值(指定最大字段宽度),一个可选的 h、l 或 L 字符(指定目标对象的宽度)以及一个转换字符组成。

scanf()函数基本的转换说明如表5-1所示。

表5-1　scanf()函数基本的转换说明

字　符	输入数据;参数类型
d	十进制整数;int * 类型
i	整数;int * 类型,可以是八进制(以0开头)或十六进制(以0x或0X开头)
o	八进制整数(可以以0开头,也可以不以0开头);int * 类型
u	无符号十进制整数;unsigned int * 类型
x	十六进制整数(可以以0x或0X开头,也可以不以0x或0X开头);int * 类型
c	字符;char * 类型,将接下来的多个输入字符(默认为一个字符)存放到指定位置。该转换规范通常不跳过空白符。如果需要读入下一个非空白符,可以使用%ls
s	字符串(不加引号);char * 类型,指向一个足以存放该字符串(还包括尾部的字符'\0')的字符数组。字符串的末尾将被添加一个结束符'\0'
e、f、g	浮点数,它可以包括正负号(可选)、小数点(可选)及指数部分(可选);float * 类型
%	字符%;不进行任何赋值操作

说明:

(1) 转换说明d、i、o、u及x的前面可以加上字符h或l。前缀h表明参数表的相应参数是一个指向short类型而非int类型的指针,前缀l表明参数表的相应参数是一个指向long类型的指针。

(2) 转换说明e、f和g的前面也可以加上前缀l,它表明参数表的相应参数是一个指向doulbe类型而非float类型的指针。

(3) scanf()函数忽略格式串中的空格和制表符。此外,在读取输入值时,它将跳过空白符(空格、制表符、换行符等)。

(4) 如果要读取格式不固定的输入,最好每次读入一行,然后再用scanf()函数将合适的格式分离出来读入。

(5) scanf()函数可以和其他输入函数混合使用。无论调用哪个输入函数,下一个输入函数的调用将从scanf()没有读取的第一个字符处开始读取数据。

5.4　printf()函数

printf()函数的原型如下:

```
int printf(char * format, arg1, arg2, …);
```

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打印出来。它的返回值为打印的字符数。

格式字符串包含两种类型的对象:普通字符和转换说明。

(1) 在输出时,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。

(2) 每个转换说明都以一个百分号字符(即%)开始,并以一个转换字符结束。

(3) 在字符%和转换字符中间可能依次包含下列组成部分:

- 负号:用于指定被转换的参数按照左对齐的形式输出。
- 数:用于指定最小字段宽度。转换后的参数将打印不小于最小字段宽度的字段。如果有必要,字段左边(如果使用左对齐的方式,则为右边)多余的字符位置用空格填充以保证最小字段宽度。
- 小数点:用于将字段宽度和精度分开。
- 小数点后的数:用于指定精度,即指定字符串中要打印的最多字符数、浮点数小数点后的位数。
- 字母 h 或 l:字母 h 表示将整数作为 short 类型打印,字母 l 表示将整数作为 long 类型打印。

printf()函数基本的转换说明如表 5-2 所示。

表 5-2 printf()函数基本的转换说明

字符	参数类型;输出形式
d、i	int 类型;十进制数
o	int 类型;无符号八进制数(没有前导 0)
x、X	int 类型;无符号十六进制数(没有前导 0x 或 0X)
u	int 类型;无符号十进制数
c	int 类型;单个字符
s	char * 类型;顺序打印字符串中的字符,直到遇到 '\0' 或已打印了由精度指定的字符数为止
f	double 类型;十进制小数[-]m.dddddd,其中 d 的个数由精度指定(默认值为 6)
e、E	double 类型,以指数形式输出单、双精度实数;十进制小数[-]m.dddddd e±xx 或[-]m.dddddd E±xx ,其中 d 的个数由精度指定(默认值为 6)
g、G	double 类型,以%f 或%e 中较短的输出宽度输出单、双精度实数
p	void * 类型;指针(取决于具体实现)
%	不转换参数;打印一个百分号%

说明:

(1) 如果%后面的字符不是一个转换说明,则该行为是未定义的。

(2) 在转换说明中,宽度或精度可以用星号“*”表示,这时宽度或精度的值通过转换下一参数(必须为 int 类型)来计算。

(3) printf()函数使用第一个参数判断后面参数的个数及类型。如果参数的个数不够或者类型错误,则将得到错误的结果。请注意下面两个函数调用之间的区别:

```
printf(s);                    /* 如果字符串 s 含有字符%,输出将出错 */
printf("%s", s);              /* 正确 */
```

【例 5.3】 写出下列程序的运行结果。

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
    char *s="hello, world";

    printf("%s\n", s);
    printf("%10s\n", s);
    printf("%.10s\n", s);
    printf("%-10s\n", s);
    printf("%.15s\n", s);
    printf("%-15s\n", s);
    printf("%15.10s\n", s);
    printf("%-15.10s\n", s);

    return 0;
}
```

运行结果：

```
hello, world
hello, world
hello, wor
hello, world
hello, world
hello, world
     hello, wor
hello, wor
```

5.5 C++格式化控制台输出

C++提供了附加函数来格式化一个要显示的值。这些函数称为流操作，包含在iomanip头文件中。下面介绍几种有用的流操作。

1. setprecision操作

使用setprecision(n)操作给一个浮点数指定总的显示位数，其中n是所需数字位数(小数点后位数的总和)。

setprecision操作的作用是直到精度改变之前，一直保持效果。所以

```
cout<<setprecision(3)<<12.34567<<" ";
cout<<9.34567<<" "<<121.3457<<" "<<0.2367<<endl;
```

显示为

```
12.3 9.35 121 0.237
```

如果精度的宽度不足够一个整数，setprecision操作将会被忽略。例如：

```
cout<<setprecision(3)<<23456<<endl;
```

显示为

```
23456
```

2. 修改操作

有时候,计算机会自动用科学记数法显示一个很长的浮点数。在 Windows 系统上,语句

```
cout <<232123434.357 <<endl;
```

显示为

```
2.32123e+008
```

可以使用 fixed 操作来强制数字显示为非科学记数法的形式,显示小数后的位数。例如:

```
cout <<fixed <<232123434.357 <<endl;
```

显示为

```
232123434.357000
```

默认情况下,能修复小数点后 6 位。可以用 fixed 操作和 setprecision 操作一起来改变原来的设置。当在 fixed 操作之后使用 setprecision 操作时,setprecision 操作用来指定小数点后的位数。例如:

```
cout <<fixed <<setprecision(2) <<232123434.357 <<endl;
```

显示为

```
232123434.36
```

3. showpoint 操作

默认情况下,没有小数部分的浮点数是不显示小数点的。但可以使用 fixed 操作来强制浮点数显示小数点和指定小数点后位数。除此之外,还可以使用 showpoint 和 setprecision 操作一起来解决这个问题。

```
cout <<setprecision(6) <<1.23 <<endl;
cout <<showpoint <<1.23 <<endl;
cout <<showpoint <<123.0 <<endl;
```

显示为

```
1.23
1.23000
123.000
```

setprecision(6)函数设置精度值为 6。所以,第一个数 1.23 被显示为 1.23。第二个数是 1.23,被显示为 1.23000,因为 showpoint 操作强制浮点数显示小数点,并在需要的位置上补充 0;第三个数是 123.0,显示为 123.000,与第二个数同理,有小数点和补充的 0。

4. setw(width)、left、right 操作

使用 setw(width)函数指定输出的最小列数。setw 操作默认使用右对齐。可以使用 left 操作来设置输出为左对齐，或者 right 操作设置输出为右对齐。例如：

```
cout <<right;              //或者 cout <<left
cout <<setw(8) <<1.23 <<endl;
cout <<setw(8) <<351.34 <<endl;
```

显示为

```
    1.23
  351.34
```

【例 5.4】 已知华氏温度(F)与摄氏温度(C)的转换公式为 C＝(5/9) * (F－32)，将华氏温度从 0～100℉每隔 20℉分别转换成相应的摄氏温度并输出。

输入样例：

本题无输入。

输出样例：

```
  0  -17.8
 20   -6.7
 40    4.4
 60   15.6
 80   26.7
100   37.8
```

【分析】

变量 F 用于存放华氏温度，变量 C 用于存放摄氏温度，然后

```
F =0 时,C =(5 / 9) * (0 -32);
F =20 时,C=(5 / 9) * (20 -32);
⋮
F =100 时,C =(5 / 9) * (100 -32);
```

这些语句非常类似，只是 F 的值从 0 变化到 100。于是可以写成

```
循环执行,F 的值从 0 变化到 100
{
  C =(5.0 / 9) * (F -32);
  输出华氏温度相应的摄氏温度
}
```

注意：C ＝ (5.0/9) * (F－32)在语句中是 5.0 / 9，因为在 C/C++ 语言中，整数相除的结果为整数，也就是说 5 / 9 ＝ 0。

```
#include<bits/stdc++.h>
using namespace std;

int main()
```

```
{
    double C;                                   //定义一个双精度浮点型变量
    for(int F =0; F <=100; F +=20)
    {
        C =(5.0 / 9) * (F -32);                 //计算,实现华氏温度转换为摄氏温度
        printf("%3d %6.1f\n", F, C);            //输出
    }

    return 0;
}
```

在 printf()函数中,%3d 表示输出整型数据,占 3 列,右对齐,如果需要左对齐则用%-3d;%6.1f 表示输出浮点型数据,共占 6 列,小数点占 1 列,小数点后面有一位。

程序中的 printf("%3d %6.1f\n", F, C); 语句可以改成用 cout 输出:

```
cout <<setw(3) <<F <<" ";
cout <<setw(6) <<fixed <<setprecision(1) <<C <<endl;
```

5.6 应 用 实 例

本节给出了一些求和实例,主要是帮助大家解决多组数据的输入和输出问题的。

【例 5.5】 计算 a+b。输入数据首先包括一个整数 t,表示输入数据的组数。然后是 t 行,每行有两个整数 a、b,这两个整数用空格隔开。对于每组输入数据 a 和 b,输出一行,即 a+b 的值。

输入样例:

```
2
1 5
10 20
```

输出样例:

```
6
30
```

【分析】根据题意可知,输入的组数由第 1 行的 t 所决定。我们可以采用 for 循环或 while 循环来解决这个问题。

【第 1 种方法】用 for 循环语句

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
    int t, a, b;
    scanf("%d", &t);
    for(int i =1; i <=t; i ++)
    {
```

```
        scanf("%d%d", &a, &b);
        printf("%d\n", a +b);
    }

    return 0;
}
```

【第2种方法】用 while 循环语句

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
    int t, a, b;
    scanf("%d", &t);
    while(t --)
    {
        scanf("%d%d", &a, &b);
        printf("%d\n", a +b);
    }

    return 0;
}
```

【例 5.6】 计算 a+b。输入数据有多组，每组占一行，它有两个整数 a、b，这两个整数用空格隔开，如果 a=0 并且 b=0，则表示输入结束，该行不做处理。对于每组输入数据 a 和 b，输出一行，即 a+b 的值。

输入样例：

```
1 5
10 20
0 0
```

输出样例：

```
6
30
```

【分析】根据题意可知，输入数据有多组，但是并没有说到底有多少组，那输入数据什么时候结束呢？直到 a=0 并且 b=0 时输入就结束。

【第1种方法】

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
    int a, b;
    while(true)
    {
```