第3章 顺序结构程序设计

前两章介绍了简单的C程序，其中在第2章中介绍了程序常用的一些基本要素（常量、变量、数据类型等），在本章中继续介绍运算符和表达式，以及C语言输入输出语句的使用。

变量用来存放数据，运算符则用来处理数据。用运算符将变量和常量连接起来的符合C语言语法规则的式子称为表达式。每个表达式都有值。

根据运算符所带的操作数的数量进行划分，C语言的运算符有三种类别：

（1）单目运算符：只带一个操作数的运算符，如++、--运算符。

（2）双目运算符：带两个操作数的运算符，如+、-运算符。

（3）三目运算符：带三个操作数的运算符，如?:运算符。

C语言中运算符和表达式之多，在高级语言中是少见的。正是具有丰富的运算符和表达式，使得C语言功能十分完善。这也是C语言的主要特点之一。所以要学好和用好C语言，务必要熟练掌握其运算符的功能、特点及应用。

具体学习过程中应掌握如下几个方面。

- 运算符的功能：该运算符主要用于什么运算。
- 与操作数关系：要求操作数的个数及操作数的类型。
- 运算符的优先级：表达式中包含多个不同运算符时运算符运算的先后次序。
- 运算符的结合性：同级别运算符的运算顺序（指左结合性还是右结合性）。
- 运算结果的类型：表达式运算后最终所得到的值的类型。

3.1 赋值运算符与赋值表达式

3.1.1 赋值运算符

赋值符号“=”就是赋值运算符，其作用是将一个值（常量、变量、表达式）赋给一个变量，实际上是将特定的值写到变量所对应的内存单元中。赋值运算符是双目运算符，因为“=”两边都要有操作数。“=”左边是待赋值的变量，“=”右边是要赋给的值。

在定义变量的同时给变量赋初值称为变量的初始化。在变量定义中赋初值的一般形式为：

类型说明符 变量1 =值1,变量2 =值2,…

例如：

```
int a =5;
int b , c =6;
float x =3.6, y =4f, z =0.67;
char ch1 ='A', ch2 ='m';
```

应注意,在定义中不允许连续赋值,如 int a = b = c =6 是不合法的。

3.1.2 赋值表达式

由赋值运算符或复合赋值运算符(后面即将介绍)将一个变量和一个表达式连接起来的表达式,称为赋值表达式。

赋值表达式的一般格式为：

变量 (复合)赋值运算符 表达式

被赋值表达式的值,就是该表达式的值。例如,a = 5,变量 a 的值 5 就是该赋值表达式的值。

3.1.3 赋值语句

按照 C 语言规定,任何表达式在其末尾上加上分号“;”就构成语句。赋值表达式在其后加上分号“;”就构成了赋值语句。

其一般形式为：

变量 =表达式 ;

例如,

```
x =8;a =b =c =5;
```

注意：变量可以连续赋值但不可以连续初始化。

【例 3-1】 变量赋值。

```
1    #include <stdio.h>
2    int main()
3    {
4        int a =3, b, c =7;
5        b =a +c;
6        printf("a =%d, b =%d, c =%d\n", a, b, c);
7        return 1;
8    }
```

【运行结果】

```
a = 3, b = 10, c = 7
```

3.1.4 左值和右值

因为不是所有对象都可以更改值的，C 语言使用术语“可修改的左值”来表示那些值是可以被更改的，所以，赋值运算符的左边应该是一个可以修改的左值。

术语“右值”指的是能赋给可修改的左值的量。例如，下面的语句：

```
cow =1235;
```

这里 cow 是一个可修改的左值，1235 是一个右值。

3.1.5 不同数据类型间的赋值规则

C 语言变量的数据类型是可以相互转换的。转换的方法有两种，一种是自动转换，另一种是强制转换。

1. 自动转换

以赋值符号“=”左边变量的数据类型为准，对其右边的数据进行处理。

(1) 短长度的数据类型转换为长长度的数据类型，包括以下几种形式。

① 无符号短长度的数据类型转换为无符号或有符号长长度的数据类型。直接将无符号短长度的数据类型的数据作为长长度的数据类型数据的低位部分，长长度的数据类型数据的高位部分补“0”，不损失精度，如图 3.1 所示。

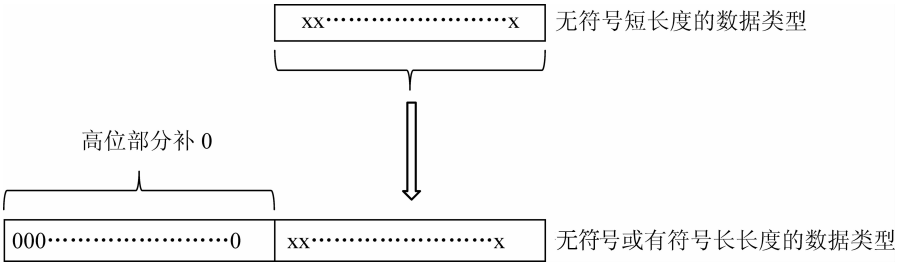

图 3.1 无符号短长度的数据类型转换为无符号或有符号长长度的数据类型

例如：

```
unsigned char ch =0xfc;
unsigned short a =0xff00;
int b;
unsigned long u;
```

```
b =ch;          /* b 的值将是 0x000000fx */
b =a;           /* b 的值将是 0x0000ff00 */
```

② 有符号短长度的数据类型转换为无符号或有符号长长度的数据类型。直接将有符号短长度的数据类型的数据作为长长度的数据类型数据的低位部分，然后将低位部分的最高位(即符号位)向长长度的数据类型数据的高位扩展，不损失精度，如图 3.2 所示。

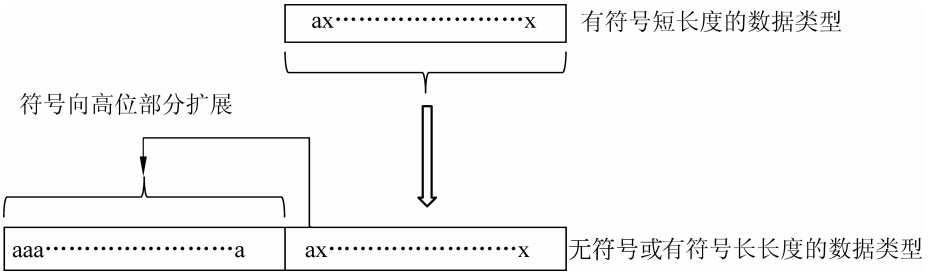

图 3.2　有符号短长度的数据类型转换为无符号或有符号长长度的数据类型

例如：

```
char ch =2;
short a =-2;
int b;
unsigned long u;
b =ch;          /* b 的值将是 2 */
u =a;           /* u 的值将是 0xfffffffe,前 4 个 f 是符号扩展的结果 */
```

(2) 长长度的数据类型转换为短长度的数据类型(隐式强制转换)。直接截取长长度的数据类型数据的低位部分(长度为短长度数据类型的长度)作为短长度数据类型的数据，损失精度，如图 3.3 所示。

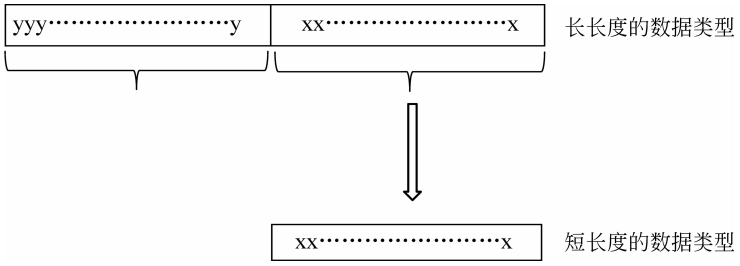

图 3.3　长长度的数据类型转换为短长度的数据类型

例如：

```
int a =-32768;
int b =0xffffaa00;
char ch;
short int c;
```

```
ch =a;          /* ch的值将是 0          */
c =b;           /* c的值将是 0xaa00   */
```

2. 显式强制转换

显式强制类型转换是通过类型运算来实现的。其一般形式为：

(类型说明符)　(表达式)

其功能是把表达式的运算结果强制转换成类型说明符所表示的类型。例如：

- (float) a：把 a 转换为单精度类型。
- (int) (x+y)：把 x+y 的结果转换为整型。

在使用显式强制转换时应注意以下问题：

- 类型说明符和表达式都必须加括号(单个变量可以不加括号)。如把(int) (x+y)写成(int) x+y 则成了把 x 转换成 int 类型之后再与 y 相加了。
- 无论是强制转换或是自动转换，都只是为了本次运算的需要而对变量的数据长度进行临时转换，不是改变数据声明时对该变量所定义的类型。

【例 3-2】 强制类型转换应用。

```
1    #include <stdio.h>
2    int main()
3    {
4        float f =5.76;
5        printf("(int)f =%d, f =%f\n", (int)f, f);
6        return 1;
7    }
```

【运行结果】

```
(int)f = 5, f = 5.760000
```

本例表明，f 虽然强制转换为 int 类型，但只在运算中起作用，是临时的，而 f 本身的类型并不改变。因此，(int)f 的值为 5(删去了小数)，而 f 的值仍为 5.76。

3.2　算术运算符与算术表达式

3.2.1　算术运算符

C 语言提供的算术运算符包括 5 种：加(+)、减(-)、乘(*)、除(/)和取余(%)。它们均是双目运算符。+、-、*、/ 运算既可以用于整型数据的算术运算，又可以用于实型数据的算术运算，而%只能用于整数的运算。

注意：

(1) C 语言规定：两个整数相除，其商为整数，小数部分被舍弃。例如，7/2 的值是 3，不是 3.5。要得到 3.5，则应写成 7.0/2 或 7/2.0。

(2) %不能用于浮点型数据，否则会出错。例如，5.4%2 是非法的，因为%只能用于整型数据的运算。

3.2.2 算术表达式

用算术运算符将操作对象连接起来的表达式称为算术表达式。例如，3＋6＊9，(x ＋ y)＊8/2 等都是算术表达式。

【例 3-3】 算术运算符“%”和“/”的使用。

```
1    #include <stdio.h>
2    int main()
3    {
4        printf("%d, %d, %d, %d\n", 20/7, 20%7, -20/7, 20/(-7));
5        printf("%f, %f,%f\n", 20.0/7, -20.0/7, 20.0/(-7));
6        return 1;
7    }
```

【运行结果】

```
2, 6, -2, -2
2.857143, -2.857143,-2.857143
```

本例中，20/7、－20/7 的结果均为整型，小数部分全部舍去了。而 20.0/7 和－20.0/7 由于有实数参与运算，因此结果也为实型。

3.2.3 运算符的优先级和结合性

当一个表达式中存在有多个算术运算符时，计算顺序取决于运算符的优先级和结合性。

1. 运算符的优先级

在 C 语言中，运算符的优先级共分为 15 级，1 级最高，15 级最低。在表达式中，优先级较高的先于优先级较低的进行运算。而在一个操作数两侧的运算符优先级相同时，则按运算符的结合性所规定的结合方向处理。

例如，图 3.4 详细地表示了表达式 10 ＋ 5＊4－7/3 的求解过程，图中的①～④表示求解过程的先后。

算术运算符、赋值运算符和类型强制转换运算符的优先级的关系如下：

类型强制转换运算符 >算术运算符 >赋值运算符

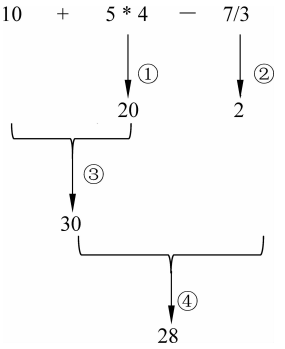

图 3.4　表达式 10＋5＊4－7/3 的求解过程

因此，执行下面的语句后，a 的值是 13。

```
int a;
a = (int)2.5 * 4+5;
```

2. 运算符的结合性

C 语言中各运算符的结合性分为两种，即左结合性（自左至右）和右结合性（自右至左）。例如，算术运算符的结合性是自左至右，即先左后右。如有表达式 x－y＋z，则 y 应先与“－”结合，执行 x－y 运算，然后再执行＋z 的运算。这种自左至右的结合方向就称为“左结合性”。而自右至左的结合方向称为“右结合性”。最典型的右结合性运算符是赋值运算符。如 x ＝ y ＝ z，由于“＝”的右结合性，应先执行 y ＝ z 再执行 x ＝ (y ＝ z)运算。C 语言运算符中有不少为右结合性，应注意区别，以避免理解错误。

3.2.4　自增自减运算符

在 C 语言中，减号（－）既是一个运算符，又是一个负号运算符。负号运算符是单目运算符。例如，a＝2，那么－a 的值就是－2。负号运算符的优先级比较高，与强制类型转换符是同一级别。

C 语言还提供了另外两个用于算术运算的单目运算符：自增（＋＋）和自减（－－）。

自增运算（＋＋）使单个变量的值增 1，自减运算（－－）则使单个变量的值减 1。

自增、自减运算都有以下两种用法。

- 前置运算：运算符放在变量的前面，如＋＋i、－－i，表示先使变量 i 的值增（或减）1，然后再以变化后的值参与其他运算，即先增减后运算。
- 后置运算：运算符放在变量的后面，即 i＋＋、i－－，表示变量 i 先参与其他运算，然后再使其值增（或减）1，即先运算后增减。

【例 3-4】　自增自减运算符的使用。

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

```
2    int main()
3    {
4        int a =2, b =4;
5        int c, d;
6        c =a++;              /* 等价于 c =a ;和 a =a +1;两条语句 */
7        d =--b;              /* 等价于 b =b -1 ;和 d =b;两条语句 */
8        printf(" a =%d, b =%d\n", a, b);
9        printf(" c =%d, d =%d\n", c, d);
10       return 1;
11   }
```

【运行结果】

```
a = 3, b = 3
c = 2, d = 3
```

注意：

- ++和--运算符只能用于变量，不能用于常量和表达式，因为++和--蕴含着赋值操作。如5++、--(a+b)都是非法的表达式。
- 负号运算符、++、--和强制类型转换运算符的优先级相同，当这些运算符连用时，按照从右向左的顺序计算，即具有右结合性。
- 两个+和-符号之间不能有空格。
- 在表达式中，连续使用同一变量进行自增或自减运算时，很容易出错，所以最好避免这种用法。如++i++是非法的。
- 自增、自减运算，常用于循环语句中，使循环控制变量加(或减)1，以及指针变量中，使指针指向下(或上)一个地址。

假设有 int p,i = 2,j = 3。现分几种情况来讨论++和--的使用方法。

- 情况一：p = -i++，相当于 p = -(i++)，即先把-i 的值赋值给 p，然后 i 增1。执行后 p 的值为-2，i 的值为 3。
- 情况二：p = i+++j，相当于 p = (i++) +j，即先将 i+j 的值赋值给 p，然后 i 增 1。执行后 p 的值为 5，i 的值为 3，j 的值不变。
- 情况三：p = i+--j，相当于 p = i+(--j)，即先将 j 减 1，然后把 i+j 的值赋值给 p。执行后 p 的值为 4，i 的值不变，j 的值为 2。
- 情况四：p = i+++--j，相当于 p = (i++) +(--j)，即先将 j 减 1，然后把 i+j 的值赋值给 p，再把 i 的值增 1。执行后 p 的值为 4，i 的值为 3，j 的值为 2。
- 情况五：p = i+++i++，相当于 p = (i++) +(i++)，即先将 i+i 的值赋值给 p，再把 i 的值增 1 两次。执行后 p 的值为 4，i 的值为 4。
- 情况六：p = ++i+(++i)，相当于 p = (++i) +(++i)，即先将 i 的值增 1 两次，再把 i+i 的值赋值给 p。执行后 p 的值为 8，i 的值为 4。注意，该式不能写成：p = ++i+++i。

3.2.5 算术运算中数据类型转换规则

在C语言中,整型、实型和字符型数据间可以混合运算。如果一个运算符两侧的操作数的数据类型不同,则系统按“先转换后运算”的原则,首先将数据自动转换成同一类型,然后在同一类型数据间进行运算。转换规则如图3.5所示。

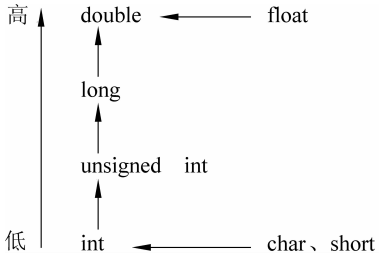

图3.5 数据类型转换规则

图3.5中横向向左的箭头,表示必需的转换。char和short类型必须转换成int类型,float类型必须转换成double类型。

图3.5中纵向向上的箭头,表示不同类型的转换方向。例如,int类型与double类型数据进行混合运算,则先将int类型数据转换成double类型数据,然后在两个同类型的数据间进行运算,结果为double类型。

注意:图3.5中箭头方向只表示数据类型的由低到高转换,不要理解为:int类型先转换成unsigned类型,再转换成long类型,最后转换成double类型。

下面以一个实例来看看算术表达式中不同数据类型间的转换。

假设变量的定义为:

```
char  ch;
int  i;
float  f;
double  d;
```

现计算ch/i + f * d-(f+i)的值,其类型转换如图3.6所示。

【例3-5】 不同数据类型数据的算术运算。

```
1    #include <stdio.h>
2    int main()
3    {
4        float a, b, c;
5        a =7 / 2;            /* 计算7/2得int型值3,因此a的值为3.0 */
6        b =7 / 2 * 10;       /* 计算7/2得int型值3,再与10相乘,因此b的值为30.0 */
7        c =1.0 * 7 / 2;      /* 先计算1.0*7得double型的结果7.0,再计算7.0/2,
                                 因此c的值是3.5 */
8        printf(" a =%f, b =%f, c =%f\n", a, b, c);
```

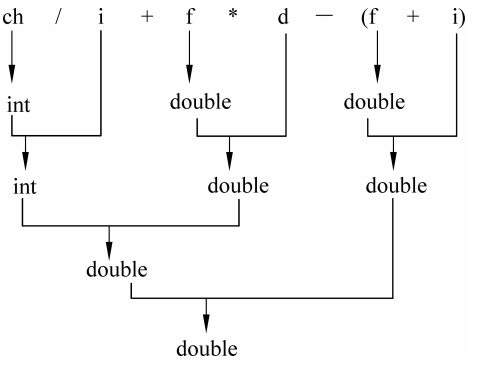

图 3.6　计算表达式 ch/i ＋ f ＊ d－(f＋i)的数据类型转换示意图

```
9       return 1;
10   }
```

【运行结果】

```
a = 3.000000, b = 30.000000, c =3.500000
```

3.2.6　sizeof 运算符、复合赋值运算符

1. sizeof 运算符

C 语言中提供了一个能获取变量和数据类型所占内存大小(字节数)的运算符——sizeof。其使用格式是：

```
sizeof  表达式
sizeof  (数据类型名或表达式)
```

比如，sizeof(int)的值在 VC++ 2010 中是 4，sizeof(long)的值是 4，sizeof(10L)的值也是 4。

sizeof 运算符的优先级比较高，与＋＋、－－是同一个优先级。

2. 取模运算符%

取模运算符用于整数运算。该运算符计算出它左边的整数去除以它右边的整数得到的余数。例如，13 % 5 的结果为 3。

3. 复合赋值运算符

C 语言除了提供赋值运算符“＝”以外，还提供了各种复合赋值运算符。将算术运算符与赋值运算符组合在一起就构成了复合赋值运算符。复合赋值运算符既包括了算术运算，又包含了赋值操作。复合赋值运算符具体有＋＝、－＝、＊＝、/＝、%＝等几种。

其含义为：

```
exp1 op=exp2
```

等价于

```
exp1 =exp1 op exp2
```

例如：

```
a +=3
```

等价于

```
a =a +3
```

又如：

```
x *=y +8
```

等价于

```
x =x * (y +8)
```

复合赋值运算符与赋值运算符是同一个优先级，具有右结合性。

3.3 数据的格式化输出

在C语言中，用于输出数据的主要函数是printf函数。在前面的章节中，已简单介绍了利用printf函数输出单个变量的值，在本节中将重点介绍printf函数的用法。实际上，printf函数的功能绝非只是输出变量的值，它还可以输出表达式的值，并且可以同时输出多个表达式和变量的值。printf函数称为格式化输出函数，其函数名最末一个字母f即为“格式(format)”之意。也就是说，它可以按照某种输出格式在屏幕上输出相应的数据信息。

它的函数原型在头文件stdio.h中，所以使用printf函数时，在程序最开始处应有如下的语句：

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

printf函数的一般形式为：

```
printf("格式控制字符串",表达式1, 表达式2,…,表达式n);
```

其功能就是按照“格式控制字符串”的要求，将表达式1,表达式2,…,表达式n的值显示在计算机屏幕上。

格式控制字符串用于指定输出格式。它包含如下所示的两类字符。

- 常规字符：包括可显示字符和用转义字符表示的字符。

- 格式控制符：以%开头的一个或多个字符，以说明输出数据的类型、形式、长度、小数位数等。比如前面介绍的%d、%f 等。其中，%后面的 d 和 f 称为格式转换字符。

例如，在格式控制字符串“the value is %d\n”中，除了%d 是格式控制符（表示以十进制形式输出整型数据）外，其他的字符均是常规字符。

使用 printf 函数要注意以下几点：

（1）格式控制字符串可以不包含任何格式控制符。在这种情况下，使用 printf 函数时，其参数一般只有一个字符串即可。例如：

```
printf ("how are you?\n "); /* 只有一个字符串参数,输出"how are you?" */
```

（2）当格式控制字符串中既包含有常规字符，又包含有格式控制符时，则表达式的个数应与格式控制符的个数一致。此时常规字符原样输出，而在格式控制符的位置上输出对应的表达式的值，其对应的顺序是：从左到右的格式控制符对应从左到右的表达式。这种对应关系如图 3.7 所示。

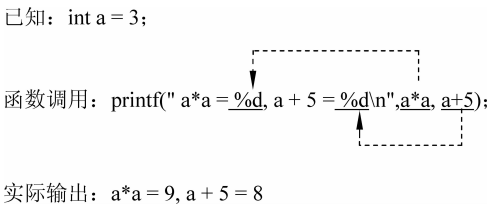

图 3.7　格式控制符与 printf 的后续参数的对应关系

（3）如果格式控制符字符串中格式控制符的个数多于表达式的个数，则余下的格式控制符的值将是不确定的，所以这一点编程时要注意。当然，如果格式控制符字符串中格式控制符的个数少于表达式的个数，则不影响输出。例如：

```
printf("5+3 =%d, 5-3 =%d, 5 * 3 =%d", 5+3, 5-3);
```

输出结果是：

```
5+3 =8,5-3 =2, 5 * 3 =-28710
```

这里输出 5 * 3 = −28710，就是因为第三个格式控制符没有对应的表达式所造成的，所以其输出的值是随机值。

（4）不同类型的表达式要使用不同的格式转换符，比如输出 int 型表达式要使用%d，输出字符要使用%c，而输出实型表达式则要使用%f。同一表达式如果按照不同的格式转换符来输出，其结果可能是不一样的。例如：

```
char ch ='A';
printf("ch =%c", ch);       /* 输出结果: ch =A (以字符形式输出) */
printf("ch =%d", ch);       /* 输出结果: ch =65(以字符的 ASCII 码形式输出) */
```

“格式控制符字符串”中的格式转换符，必须与所对应的表达式的数据类型一致，否则

会引起输出错误。

各种数据类型对应的格式转换符如表 3.1 所示。

表 3.1　printf 函数中的格式转换符及其含义

格式转换符	含　义	对应的表达式数据类型

%d 或 %i	以十进制形式输出一个整型数据。例如：
	`int a =20;`
	`printf("%d", a);　　/* 输出 20 */`
		有符号整型

%x、%X	以十六进制形式输出一个无符号整型数据。例如：
	`int a =164;`
	`printf("%x", a);　　/* 输出 a4 */`
	`printf("%X", a);　　/* 输出 A4 */`
		无符号整型

%o（字母 o）	以八进制形式输出一个无符号整型数据。例如：
	`int a =164;`
	`printf("%o", a);　　/* 输出 244 */`
		无符号整型

%u	以十进制形式输出一个无符号整型数据。例如：
	`int a =-1;`
	`printf("%u", a); /* VC++ 2010 下输出 4294967295 */`
		无符号整型

%c	输出一个字符型数据。例如：
	`char ch ='A';`
	`printf("%c", ch);　　/* 输出 A */`
		字符型

%s	输出一个字符串。例如： `printf("%s", "china");　　/* 输出 china */`	字符型

%f	以十进制小数形式输出一个浮点型数据。例如：
	`float f =-12.3;`
	`printf("%f", f);　　/* 输出-12.300000 */`
		浮点型

%e、%E	以指数形式输出一个浮点型数据。例如：
	`float f =1234.8998;`
	`printf("%e", f); /* VC++ 2010 下输出 1.234900e+003 */`
	`printf("%E", f); /* VC++ 2010 下输出 1.234900E+003 */`
		浮点型

%g、%G	按照%f 或%e 中输出宽度比较短的一种格式输出	浮点型

%p	以主机的格式显示指针，即变量的地址。例如：
	`int a =3;`
	`printf("%p", &a);　　/* VC++ 2010 下输出 0018FF44 */`
		指针类型

另外，可以在格式转换字符和%之间插入一些辅助的格式控制字符。因此，格式控制字符的一般形式为：

```
%[ flag ][ width ][ .precision ][ size ] Type
```

其中，[ ]表示其中的内容是可以缺省的，没有用[ ]括起来的是不能缺省的。width 和

precision 都必须是无符号的整数，注意，precision 前面的句点(.)不能省略。Type 就是表 3.1 中列出的格式转换符。

3.3.1 整数的输出

1. 有符号整数的输出

输出有符号整数的格式控制符的一般形式为：

```
%[ -][ +][ 0 ][ width ][ .precision ][ l ][ h ] d
```

其中各控制符的说明如下：

- [ ]：表示可选项，可以缺省。
- －：表示输出的数据左对齐，默认是右对齐。
- ＋：输出正数时，在数据的前面加上“＋”号。
- 数字 0：右对齐时，如果实际宽度小于 width，则在左边的空位补 0。
- width：无符号整数，表示输出整数的最小域宽(即占屏幕的多少格)。若实际宽度超过了 width，则按照实际宽度输出。
- .precision：无符号整数，表示至少要输出 precision 位。若整数的位数大于 precision，则按照实际位数输出，否则在左边的空位上补 0。
- 字母 l：如果在 d 的前面有字母 l(long)，表示要输出长整型数据。
- 字母 h：如果在 d 的前面有字母 h(short)，表示输出短整型数据。

例 3-6 说明了常用的使用方法。

【例 3-6】 有符号整数的常用方法。

```
1    #include <stdio.h>
2    int main()
3    {
4        int a =123;
5
6        printf("    12345678901234567890\n");
7        printf("a =%d----------(a=%%d)\n", a);
8        return 1;
9    }
```

【运行结果】

```
   12345678901234567890
a = 123---------(a=%d)
```

例 3-7 说明了上述格式控制符的作用。

【例 3-7】 有符号整数的格式化输出。

```
1    #include <stdio.h>
2    int main()
```

```
3    {
4        int a =123;
5        long L =65537;
6
7        printf("    12345678901234567890\n");
8         printf("a =%d---------(a=%%d)\n", a);
9        printf("a =%6d-------(a=%%6d)\n", a);
10       printf("a =%+6d-------(a=%%+6d)\n", a);
11       printf("a =%-6d-------(a=%%-6d)\n", a);
12       printf("a =%-06d-------(a=%%-06d)\n", a);
13       printf("a =%+06d-------(a=%%+06d)\n", a);
14       printf("a =%+6.6d-------(a=%%+6.6d)\n", a);
15       printf("a =%6.6d-------(a=%%6.6d)\n", a);
16       printf("a =%-6.5d-------(a=%%-6.56d)\n", a);
17       printf("a =%6.4d-------(a=%%6.4d)\n", a);
18       printf("L =%ld----------(L=%%ld)\n", L);
19       printf("L =%hd----------(L=%%hd)\n", L);
20       return 1;
21   }
```

【运行结果】

```
   12345678901234567890
a =123---------(a=%d)
a =   123-------(a=%6d)
a =  +123-------(a=%+6d)
a =123   -------(a=%-6d)
a =123   -------(a=%-06d)
a =+00123-------(a=%+06d)
a =+000123-------(a=%+6.6d)
a =000123-------(a=%6.6d)
a =00123 -------(a=%-6.56d)
a =  0123-------(a=%6.4d)
L =65537----------(L=%ld)
L =1----------(L=%hd)
```

【程序分析】

- 第 9 行要求输出 123 时占 6 格宽，右对齐。默认情况下，左边多出来的空位用空格填充，即在左边填补 3 个空格。
- 第 10 行同第 9 行一样，因为格式控制符中有“＋”号，且 a 为正数，输出时要输出“＋”号，“＋”占一个空位，所以在左边还多出来 2 个空位，用空格填补。
- 第 11 行要求输出 123 时占据 6 格宽，左对齐，右边多出来的 3 个空位用空格填补。
- 第 12 行等同于 11 行。格式控制符中的“0”，其作用是在左边填补 0，但因为是左对齐，所以不可能再在左边填补 0。
- 第 13 行要求输出 123 时占 6 格宽，右对齐。因为要输出“＋”号，所以左边还需填补 2 个 0（格式控制符中“0”的作用）。
- 第 14 行要求输出 123 时占 6 格宽，右对齐，并且至少输出 6 为数字，因此在左边

要填补 3 个 0(“+”号除外)。

- 第 15 行要求输出 123 时占 6 格宽,右对齐,并且至少输出 6 为数字,因此在左边要填补 3 个 0。
- 第 16 行要求输出 123 时占 6 格宽,左对齐,并且至少输出 5 为数字,因此在左边要填补 2 个 0,右边留有一个空格。
- 第 17 行要求输出 123 时占 6 格宽,右对齐,并且至少输出 4 为数字,因此在左边要填补 2 个空格和 1 个 0。
- 第 19 行是输出一个有符号的短整型数,因为 L 是一个长整型数 65537,其值为十进制的 0X0001**0001**,所以要将其转换成短整型,即取低 16 位 0X0001,将其输出,故输出为 1。

2. 无符号整数的输出

输出无符号整数的格式控制符的一般形式为:

```
%[ -][ #][ 0 ][ width ][ .precision ][ l ][ h ]u | o| x | X
```

其中各控制符的说明如下:

- [ ]:表示可选项,可以缺省。
- | :表示互斥关系。
- #:表示当以八进制形式输出数据(%o)时,在数字前输出 0;当以十六进制形式输出数据(%x 或%X)时,在数字前输出 0x 或 0X。
- .precision:其含义与前面介绍的相同。注意:在 VC++ 2010 下,不包含 0x 或 0X 所占的位数。
- 其他字段的含义与前面介绍相同。

例 3-8 说明了上述格式控制符的作用。

【例 3-8】 无符号整数的格式化输出。

```
1    #include <stdio.h>
2
3    int main()
4    {
5        int a =-1;
6        unsigned u =32767;
7        unsigned long L =-32768;
8
9        printf("a =%d, a =%u ---(a=%%d, a=%%u)\n", a, a);
10       printf("a =%hx, a =%X ---(a=%%hx, a=%%X)\n", a, a);
11       printf("u =%o, a =%X ------(a=%%o, a=%%X)\n", u, u);
12       printf("u =%#010X ----------(a=%%#010X)\n", u);
13       printf("u =%#10.10X ---------(a=%%#10.10X)\n", u);
14       printf("L =%lX -------------(L=%%lX)\n", L);
15       printf("L =%-#14.10X -------(L=%%-#14.10X)\n", L);
```

```
16      return 1;
17  }
```

【运行结果】

```
a = -1, a = 4294967295 ---(a=%d, a=%u)
a = ffff, a = FFFFFFFF ---(a=%hx, a=%X)
u = 77777, a = 7FFF ------(a=%o, a=%X)
u = 0X00007FFF ----------(a=%#010X)
u = 0X0000007FFF ---------(a=%#10.10X)
L = FFFF8000 -------------(L=%lX)
L = 0X00FFFF8000   -------(L=%-#14.10X)
```

【程序分析】

- 第 9 行是将 a 的值以十进制整数有符号(%d)和无符号(%u)的形式输出。在 VC++ 2010 下,因为一个整数变量占 4 字节的存储单元,所以变量 a 在内存中的实际存放值为 0XFFFFFFFF(无符号数为 4 294 967 295)。
- 第 10 行是将 a 的值以十六进制无符号短整型数(%hx)和整型(%X)的形式输出。在 VC++ 2010 下短整型和整型所占内存单元的大小是不同的,短整型是占 2 字节,所以在输出时,存在数据类型的转换问题,即将整型转换为短整型,则取变量 a 的值(其值为 0XFFFFFFFF)低 16 位(0XFFFF)输出;整型占 4 字节,所以将 a 的值(0XFFFFFFFF)以无符号十六进制形式输出。
- 第 11 行是将 u 的值分别以无符号的八进制(%o)和十六进制(%X)形式输出。因为 u 的值是 32 767,其八进制为 77 777,十六进制为 7FFF。
- 第 12 行是将 u 的值分别以无符号的十六进制形式(%#010X)输出,右对齐,输出占 10 位(包括 0X),因为格式控制符中有"0",所以在输出的 7FFF 前面补 4 个 0。
- 第 13 行要求输出 u 的值(十六进制形式)必须是 10 个数码,在 VC++ 2010 下,10 位不包含 0X 两位,所以在 7FFF 前补 6 个 0。
- 第 14 行要求以长整型十六进制无符号的形式(%lX)输出 L 的值。L 在内存中的表示为 0XFFFF8000(十进制－32 768 的补码)。
- 第 15 行要求以整型十六进制无符号(%－#14.10X)的形式输出 L 的值,其中输出占 14 位,必须输出数据 10 位,左对齐,并显示 0X。L 的值在内存中是 0XFFFF8000,在 VC++ 2010 下,整型占 4 字节,将 L 中的值全部显示(FFFF8000),不够 10 位,在其左边填补 2 个 0(0X 不包括在 10 个必显示的字符之列),但总的输出要占 14 位,所以还必须在输出的右边填补 2 个空格。

3.3.2 实数的输出

输出实数的格式控制符的一般形式为:

```
%[ -][ +][ #][ 0][ width][ .precision][ l]f|e|E|g|G
```

其中各控制符的说明如下:

- [ ]:表示可选项,可以缺省。

- |：表示互斥关系。
- #：表示必须输出小数点。
- .precision：规定输出实数时，小数部分的位数。
- l：输出 double 型数据。
- 其他字段的含义与前面介绍相同。

例 3-9 说明了上述控制符的作用。

【例 3-9】 实数的格式化输出。

```
1    #include <stdio.h>
2
3    int main()
4    {
5        double f =2.5e5;
6
7        printf("    12345678901234567890\n");
8        printf("f =%15f---------(f=%%15f)\n", f);
9        printf("f =%015f-------(f=%%015f)\n", f);
10       printf("f =%-15.0f-------(f=%%-15.0f)\n", f);
11       printf("f =%#15.0f-------(f=%%#15.0f)\n", f);
12       printf("f =%+15.4f-------(f=%%+15.4f)\n", f);
13       printf("f =%15.4E-------(f=%%15.4E)\n", f);
14       return 1;
15   }
```

【运行结果】

```
   12345678901234567890
f =  250000.000000---------(f=%15f)
f =00250000.000000-------(f=%015f)
f =250000          -------(f=%-15.0f)
f =         250000.-------(f=%#15.0f)
f =  +250000.0000-------(f=%+15.4f)
f =    2.5000E+005-------(f=%15.4E)
```

【程序分析】

- 第 8 行利用%15f 使变量 f 的输出占 15 位，由于没有规定输出小数的位数，默认输出 6 位小数，并且是右对齐，因此输出数据的左边要填补两个空格。
- 第 9 行同第 8 行一样，输出也是占 15 位，右对齐，但因有%0，所以输出数据的左边要填补两个 0，而不是空格。
- 第 10 行利用%－15.0f 使变量 f 的输出占 15 位，但要左对齐，并且不输出小数部分。
- 第 11 行同样要求不输出小数部分，但＃要求输出小数点，右对齐。
- 第 12 行格式控制符中有“＋”，因此输出了加号。输出总共占 15 位，其中小数部分占 4 位，右对齐。
- 第 13 行要求以指数形式输出，并且规定整个输出占 15 位，其中小数部分占 4 位。

3.3.3 字符和字符串的输出

输出字符和字符串的格式控制符的一般形式为：

输出字符：%[-][0][width]c
输出字符串：%[-][0][width][.precision]s

其中各控制符的说明如下：

- [ ]：表示可选项，可以缺省。
- .precision：表示输出字符串的前 precision 个字符。
- 其他字段的含义与前面介绍相同。

例 3-10 说明了上述格式控制符的作用。

【例 3-10】 字符和字符串的格式化输出。

```
1    #include <stdio.h>
2
3    int main()
4    {
5        char ch = 'A';
6
7        printf("    12345678901234567890\n");
8        printf("ch =%c----------(ch=%%c)\n", ch);
9        printf("ch =%4c---------(ch=%%4c)\n", ch);
10       printf("ch =%-4c--------(ch=%%-4c)\n", ch);
11       printf("ch =%04c--------(ch=%%04c)\n", ch);
12       printf("st =%s---------(st=%%s)\n", "CHINA");
13       printf("st =%6s---------(st=%%6s)\n", "CHINA");
14       printf("st =%06.3s---------(st=%%06.3s)\n", "CHINA");
15       return 1;
16   }
```

【运行结果】

```
    12345678901234567890
ch =A----------(ch=%c)
ch =   A---------(ch=%4c)
ch =A   ---------(ch=%-4c)
ch =000A---------(ch=%04c)
st =CHINA---------(st=%s)
st = CHINA---------(st=%6s)
st =000CHI---------(st=%06.3s)
```

【程序分析】

- 第 9 行利用%4c 输出变量 ch(字符'A')，占 4 位，右对齐，所以左边填补 3 个空格。
- 第 10 行利用%－4c 输出 ch(字符'A')，占 4 位，左对齐，所以右边填补 3 个空格。
- 第 11 行利用%04c 输出变量 ch(字符'A')，占 4 位，右对齐，因为格式控制符中有

0，所以左边填补 3 个 0。

- 第 13 行利用%6s 输出字符串“CHINA”，占 6 位，右对齐，所以左边填补 1 个空格。
- 第 14 行利用%06.3s 输出字符串“CHINA”中的前 3 个字符，占 6 位，右对齐，因为格式控制符中有 0，所以左边填补 3 个 0。

3.3.4 格式化输出总结

在前面通过整型数(有符号、无符号)、实型数、字符和字符串的格式化输出的详细讨论，读者可能感到 printf 函数使用起来还是相当复杂的，要真正熟练掌握和正确应用似乎难度很大，其实，printf 函数使用时的复杂之处就在“格式控制符”上，其中“格式转换字符”(如 d、u、o、x、c、s、f、e 等)相对来讲比较容易把握，最为烦琐的就是在%和“格式转换字符”之间插入的一些“辅助格式控制符”了，为了让读者深刻理解和记忆 printf 函数的格式化输出的功能，将前面介绍过的“辅助格式控制符”以表 3.2 的形式汇集在一起，希望读者能够对照所介绍的相关内容，正确领会和把握它们的应用。

表 3.2 printf 函数中的辅助格式控制字符(修饰符)及其含义

修饰符	功　能	例　子
width	输出数据域宽，当数据长度＜width 时，填补空格；当数据长度＞width 时，按实际数位输出	%4d 表示输出至少占 4 列
.precision	对于整数，表示至少要输出 precision 位，当数据长度小于 precision，左边填补 0	%6.4d 表示至少要输出 4 位数
	对于实数，指定小数点后的位数(四舍五入)	%6.2 表示输出 2 位小数
	对于字符串，表示只输出字符串的前 precision 个字符	%.3s 表示输出字符串前 3 个字符
－	输出数据在域内左对齐(默认是右对齐)	%－8d 表示输出数据左对齐
＋	输出有符号数时，在其前面显示正负号	%＋d 表示输出整数的正负号
0	输出数值时指定左边不使用的空格自动补 0	%08X 表示输出十六进制无符号整数，不足 8 位时左补 0
#	对于无符号数，在八进制和十六进制数前显示 0、0x 或 0X	%#X 表示输出的十六进制前显示前导 0X
	对于实数，必须输出小数点	%#100f 表示输出的浮点数必须输出小数点.
h	在 d、o、x、u 前，指定输出为短整型数	%hd 表示输出短整型数
l	在 d、o、x、u 前，指定输出为长整型数	%ld 表示输出长整型数
	在 e、f、g 前，指定输出精度为 double 型	%lf 表示输出 double 型数

此外，使用 printf 函数还要注意以下几点：

(1) 格式控制字符后面表达式的个数一般要与格式控制字符的个数相等。