Chapter 3

第3章 常用运算符、表达式和库函数

本章学习目标

- 掌握C语言提供的各种运算符的功能、优先级和结合性。
- 掌握表达式的书写规则，表达式的应用及表达式值的计算方法。
- 掌握常用库函数的功能和使用。

C语言提供的运算符非常丰富，按其功能分类为：算术运算符、关系运算符、逻辑运算符、赋值运算符、条件运算符、位运算符、逗号运算符等，程序中对数据的加工是通过运算符完成的。由运算符将对应的运算对象联系起来的式子，就是表达式，根据运算符的不同，有各种类型的表达式。根据运算符的优先级和结合性可以计算出各个表达式的值。另外，为了用户使用方便，C语言处理系统提供了许多已编写好的函数，这些函数被称为库函数。用户在使用这些函数时，用#include预处理命令将对应的头文件嵌入程序中即可。通过本章的学习，读者将对C语言提供的各种运算符的功能、优先级和结合性，表达式的书写规则、表达式的应用及表达式值的计算方法，以及常用库函数的功能和使用等有一个全面的认识。

3.1 常用运算符和表达式

3.1.1 算术运算符和表达式

C语言中的算术运算符有以下8种，按优先级从高到低排列依次为

－（取负）、＋＋（自增）、－－（自减）

＊（乘）、/（除）、%（求余）

＋（加）和－（减）

其中，－（取负）、＋＋（自增）和－－（自减）为单目运算符，优先级相同，结合性为从右到左；其余为双目运算符，结合性均为从左到右。

1. 双目运算符

需要有两个运算对象的运算符是双目运算符，C 语言中的双目算术运算符为：

＋（加）　　－（减）　　＊（乘）　　/（除）　　％（取余）

（1）＋、－、＊、/与数学中的运算相似，先乘除后加减，即按运算符优先级进行计算。结合性均为从左到右。注意/（除）运算符的运算对象若都是整型数据时，结果也是整数，自动舍去小数部分。例如：1.0/2 为 0.5，1/2 为 0，9/5 为 1，－9/5 为－1。

运用运算符“/”的整除特性，我们可以对整型变量作特殊的有效处理。例如，若整型变量 k 是一个三位数，求 k 的百位上的数字，可以写作“k/100”，因整数除法的特性，结果只取整数部分，如 k＝123，则“k/100”为 1，即求出 k 百位上的数字。

（2）字符型的数据以该字符的 ASCII 码值参加运算。例如：'a'＋2 为 99，'5'－'0' 为 5 等。

（3）取余运算符“％”是求整数除法的余数，余数符号与被除数的符号相同。“％”不能用于实型数据的运算。例如：2％3 为 2，4％2 为 0，'a'％6 为 1，－9％5 为－4，9％－5 为 4 等。再如：

假设有“int　k＝12345;”，求 k 的十位上的数字，可以写作“k％100/10”，因为“k％100”得 45，“45/10”得 4，即为变量 k 十位上的数字。同样，要得到 k 个位上的数字可写为“k％10”。

2. 单目运算符

C 语言中的单目算术运算符包括自增、自减运算；另外，“＋”“－”运算当只有一个运算对象时也是单目运算符，分别表示正、负，其优先级高于双目算术运算符，如表达式“5＊－3”的值为－15。

自增、自减运算的作用是使变量的值增 1 或减 1，自增（减）的运算对象只能是变量，不能是常量和表达式。

自增、自减运算符只有一个运算对象，是单目运算符，优先级别比＊、/、％高，结合性为从右到左。自增、自减运算在使用时有以下两种格式。

前缀格式：运算符 变量

后缀格式：变量 运算符

两种格式的区别在于：前缀格式中，先使变量加（减）1，再使用变量的值；后缀格式则是，先使用变量的原值，再使变量加（减）1。

对于变量 i，“i＋＋”和“＋＋i”都表示“i＝i＋1”，前缀格式和后缀格式在使用上没有什么区别；另外，“i－－”和“－－i”也是一样的。

但是，当自增、自减运算作为表达式的一部分时，采用不同格式对表达式来说结果是不一样的。

【例 3.1】 注意下列程序运行后变量 x 与 y 的区别。

```
#include <stdio.h>
void main()
```

```
{
    int a=2,b=2,x,y;
    x=--a+2;
    y=b--+2;
    printf("x=%d  y=%d  a=%d  b=%d\n",x,y,a,b);
}
```

程序运行：

```
x=3  y=4  a=1  b=1
```

程序说明：变量 a 的自减运算是前缀格式，先使变量 a 减 1，此时 a 的当前值为 1，再执行“x＝a＋2”，结果 a 为 1、x 为 3；变量 b 的自减运算是后缀格式，b 的原值为 2，先执行“y＝b＋2；”，再使变量 b 减 1，结果 b 为 1、y 为 4。

【例 3.2】 分析下面程序的运行结果。

```
#include <stdio.h>
void main()
{
    int x=3,y;
    y=--x+--x+x++;
    printf("x=%d   y=%d\n",x,y);
}
```

程序运行：

```
x=2   y=3
```

程序说明：语句“y＝(－－x)＋(－－x)＋(x＋＋)；”的执行步骤是：先执行前缀格式的自增、自减运算，即执行“－－x”和“－－x”，x 值为 1；y 被赋值 1＋1＋1，即为 3；再执行后缀格式的自增、自减运算“x＋＋”，使 x 从 1 改变为 2。

建议读者在使用自增、自减运算时要慎重，尤其不要用自增、自减运算构造复杂的表达式，以免降低程序的易读性。在不同的编译环境下，这种表达式会引起歧义，产生不同的结果。

3. 算术运算中的类型转换

1）自动转换

C 语言中，允许不同类型的数据进行混合运算，例如整型、实型、字符型数据等都可以进行混合运算。C 语言系统在这类表达式的计算过程中，会自动进行操作数类型的转换。转换规则如下：

规则 1：凡 char 型、short 型数据一律自动转换成 int 型；float 型数据一律自动转换成 double 型。转换后如果两个操作数类型相同，就进行算术运算，计算结果的类型与转换后的类型相同。

规则 2：相同类型(除 char、short、float 型外)的操作数作算术运算的结果为同一类型。

例如：两个整型操作数做除法运算，其结果一定是整型，即只取结果的整数部分。所以 5/2 结果为 2，－5/2 结果为－2。

规则 3：不同类型的操作数如果在经过规则 1 的转换后仍然是不同类型，则其中级别低的类型自动转换成级别高的类型后再进行运算，计算结果的类型与转换后的类型相同。各类型的级别高低(从低到高)如下：

char＜short＜int＜unsigned＜long＜unsigned long＜float＜double

如表达式：

```
2.0+5/2*3
```

根据运算符的优先级和结合性计算表达式的值。原表达式相当于 2.0＋[(5/2)＊3]，首先求 5/2 的值，两操作数均为 int 型，按规则 2，不需进行类型转换，得结果 2，类型为 int；中间结果 2 再与 3 相乘，同样不需要进行类型转换，得结果 6；最后进行 2.0＋6 运算，按规则 1，操作数 2.0 类型转换为 double 后，两操作数类型仍然不同，按规则 3，将 6 转换成 double 型后再进行运算。计算结果为 8.0，类型是 double 型。

2) 强制类型转换

除了自动实现数据类型转换外，还可以在程序中进行强制类型转换，将一个表达式转换成所需的类型。表示形式为

```
(类型标识符)表达式
```

例如：int i＝5，j＝2；则 i/j 只能做整除运算，得到整数部分 2，如要保留小数部分，需做实数除法，可以写作“(double)i/j”，运算步骤是先将 i 值强制转换为 double 类型，再相除，其结果为 2.5。

值得注意的是：对表达式中的变量而言，无论是自动类型转换还是强制类型转换，仅仅是为了本次运算的需要，取得一个中间值，而不改变定义语句中对变量类型的定义。例如，计算“(double)i/j”后，变量 i 还是 int 类型，并没有变成 double 类型。

3.1.2 关系运算符和表达式

关系运算是双目运算，用于对两个运算对象的大小进行比较。用关系运算符将一些运算对象连接起来构成的式子，称为关系表达式。关系运算的结果是“成立”或“不成立”，也就是逻辑意义上的“真”或“假”。在 C 语言中没有设置表示逻辑值的数据类型，但规定用数值 1 代表“真”，用数值 0 代表“假”；在判断参加运算的对象的真、假时，将非零的数值认作“真”，0 认作“假”。

关系表达式和随后即将介绍的逻辑表达式常常在程序中用于对某些条件作出判断，根据条件成立与否，决定程序的流程。

1. 关系运算符

C 语言提供的关系运算符共 6 个，分别为

＞(大于)　　＞＝(大于或等于)　　＜(小于)　　＜＝(小于或等于)
＝＝(等于)　　!＝(不等于)

其中：

(1) 在关系运算符中，“＞、＞＝、＜、＜＝”运算的优先级相同，“＝＝、!＝”运算的优先级相同，且前者高于后者。

(2) 关系运算符的优先级比算术运算符的优先级低、比赋值运算符的优先级高。

2. 关系表达式举例

【例 3.3】 分析下面程序的运行结果。

```
#include <stdio.h>
void main()
{
    int a,b,c;
    scanf("%d%d%d",&a,&b,&c);
    a=b!=c;                                    //第 6 行
    printf("a=%d,b=%d,c=%d\n",a,b,c);
    a==(b=c++ * 3);                            //第 8 行
    printf("a=%d,b=%d,c=%d\n",a,b,c);
    a=b>c>2;                                   //第 10 行
    printf("a=%d,b=%d,c=%d\n",a,b,c);
}
```

程序运行：

```
2 3 4↙
a=1,b=3,c=4
a=1,b=12,c=5
a=0,b=12,c=5
```

程序说明：根据运算符的优先级，程序中第 6 行语句运行后，变量 a 的值为“b!＝c”的结果(即 1)；第 8 行用括号改变了运算的优先顺序，因没有对变量 a 赋值，所以 a 不变而变量 b、c 的值由赋值运算和自增运算而改变；第 10 行语句在运算时根据运算符的结合性，先计算 b＞c，结果为 1，再与 2 进行“＞”比较，所以赋给 a 的值一定是 0。

【例 3.4】 若有 int x＝2，y＝3，z＝5；，计算下列关系表达式的值。

(1) x％2＝＝0

表达式值是 1。“x％2”值为 0，再计算“0＝＝0”结果为 1。

(2) z＝x－1＞＝y－2＜y－x

表达式值是 0。根据关系运算符优先级，先进行算术运算，得“z＝1＞＝1＜1”，关系运算“＞＝”与“＜”优先级相同，根据结合性，从左到右运算，计算“1＞＝1”其值是 1，得“z＝1＜1”，再计算“1＜1”为 0，故 z 及表达式的计算结果为 0。

(3) 8<5<3

表达式值是1。根据运算符的结合性从左到右计算,先计算"8<5",值为0;再计算"0<3",值为1,表达式的值为1。

3.1.3 逻辑运算符和表达式

逻辑运算用于判断运算对象的逻辑关系,通常表示一些比较复杂的条件。逻辑运算的对象除了关系运算结果(逻辑量)外,可以是任何类型的数据(包括整型、实型、字符型等),以运算对象的值是零还是非零判断它们是"真"还是"假"。

1. 逻辑运算符

C语言提供的逻辑运算符共3个,分别为

!(逻辑非)　　&&(逻辑与)　　||(逻辑或)

1) 逻辑非

一般形式:

```
!表达式
```

功能:是单目运算符,其结果为运算对象逻辑值的取"反"。若表达式值为0,"!表达式"值为1;否则,"!表达式"值为0。

例如:"!x"作用是判别x是否为0,x为0时,值为1,否则值为0,与"x==0"等价。

2) 逻辑与

一般形式:

```
表达式 && 表达式
```

功能:若参加运算的两个表达式值均为非0,则结果为1;否则结果为0。

例如:判断"c是一个大写字母"的逻辑表达式如下:

```
c>='A' && c<='Z'
```

其中"&&"用于判断两个条件"c>='A'"和"c<='Z'"(ASCII码值比较)是否"同时成立"。

3) 逻辑或

一般形式:

```
表达式 ‖ 表达式
```

功能:若参加运算的两个表达式值均为0,结果为0;否则结果为1。

例如:判断"c是一个字母"的逻辑表达式如下:

```
c>='A' && c<='Z' || c>='a' && c<='z'
```

其中"||"用以判断两个条件是否"有一个成立"。

2. 逻辑运算符优先级

(1) 逻辑运算符的运算优先级从高到低依次为:!、&&、||。

例如：判别“a 小于 4 且|b|＞3”的逻辑表达式如下：

```
a<4&&(b>3||b<-3)
```

其中的括号用来改变表达式中各运算符的运算顺序。本例写成“a＜4&&b＞3||b＜－3”是错误的，因为逻辑与优先于逻辑或，只要 b 小于－3，则不管 a 是否小于 4，表达式值为 1。

(2) 与其他运算符比较：

① !(逻辑非)是单目运算符，与＋＋、－－优先级相同，结合性为从右到左。

② 其他逻辑运算符比关系运算符的优先级低，比赋值运算符的优先级高。

3. 逻辑表达式

逻辑表达式在程序中一般用在控制语句(if、for、while、do-while 等)中，对某些条件作出判断，根据条件成立(真)还是不成立(假)，决定程序的流程。参加逻辑运算的对象将非 0 值视为“真”、将 0 视为“假”。

【例 3.5】 设“int a＝2，b＝3；char c＝'A'；float x＝3.6，y＝－4.4”，写出下列表达式的值。

(1) a＞b||!(c－'A')&&x＜y

代入各变量值：0||!(0)&&0 即 0||1&&0，亦即 0||0，结果为 0。

(2) a－b&&!c－5||y＊2＜x

按照各运算符的优先级加括号，改写作“((a－b)&&((!c)－5))||((y＊2)＜x)”，代入各变量值后为“(－1&&(0－5))||(－8.8＜3.6)”，又可以写作“(1&&1)||1”，计算结果为 1。

【例 3.6】 根据下列条件，写出 C 的逻辑表达式。

(1) 条件“长度分别为 a、b、c 的三条线段能够组成三角形”。

逻辑表达式：

```
a+b>c && a+c>b && b+c>a
```

(2) 条件“|x|是一个两位数”。

逻辑表达式：

```
x>=10 && x<=99 || x>=-99 && x<=-10
```

(3) 条件“y 年是闰年”。

逻辑表达式：

```
y%4==0 && y%100!=0 || y%400==0
```

(4) 条件“x、y 落在圆心在(0,0)半径为 1 的圆外、中心点在(0,0)边长为 2 的矩形内”。

逻辑表达式：

```
x*x+y*y>1 && x>=-2 && x<=2 && y>=-2 && y<=2
```

以下用两个程序例子试图说明 C 语言在计算逻辑表达式值时的特点，请读者注意。在由 && 和||运算符组成的逻辑表达式中，为提高程序的执行效率，C 语言规定，只对能够确

定整个表达式值的最少数目的子表达式进行计算，即当计算出某个子表达式值后就可以确定整个逻辑表达式的值时，后面的子表达式就不再计算了。也就是说，在计算“A && B”类型的表达式时，若 A 为 0，则不再计算 B；同样，在计算“A||B” 类型的表达式时，若 A 为 1，则不再计算 B。

【例 3.7】 分析下面程序的运行结果。

```
#include"stdio.h"
void main()
{
   int x,y,z,m;
   x=y=z=0;
   m=++x&&++y|| ++z;
   printf("m=%d x=%d y=%d z=%d",  m,x,y,z);
}
```

程序运行：

```
m=1  x=1  y=1  z=0
```

程序说明：由于“++x&&++y”为 1，表达式“++x&&++y||++z”值已完全确定，所以表达式中的“++z”被忽略。

【例 3.8】 分析下面程序的运行结果。

```
#include"stdio.h"
void main()
{
   int x,y,z,m;
   x=y=z=-1;
   m=++x&&++y|| ++z;
   printf("m=%d x=%d y=%d z=%d",  m,x,y,z);
}
```

程序运行：

```
m=0  x=0  y=-1  z=0
```

程序说明：“++x&&++y||++z”；由于“++x”为 0，表达式中的“++y”被忽略，因还不能得出整个表达式的结果，再计算“0 || ++z”。

3.1.4 赋值运算符和表达式

1. 简单赋值运算

1）赋值运算

在 C 语言中，将赋值作为一种运算。赋值运算符为赋值符号“=”，它的作用是执行一次

赋值运算，将右边表达式的值赋给左边的变量。赋值表达式的形式为

变量名=表达式

赋值表达式功能是先计算表达式的值，再将计算结果送到变量。表达式又可以是赋值表达式。赋值表达式本身是一个运算表达式，它也有值，其值就是给左边变量赋的值。

赋值运算符的优先级低于算术运算符、关系运算符和逻辑运算符，结合性方向为从右到左。

2）赋值语句

（1）语句形式

变量名=表达式；

赋值表达式后加分号即为赋值语句，赋值语句执行赋值操作。

例如：赋值语句“a＝1＋2＊3.14159；”的执行步骤是：计算“＝”右边的表达式的值，赋给左边变量。

例如：赋值语句“a＝b＝c＝1；”的执行步骤是：根据运算符的右结合性，①先执行赋值表达式“c＝1”，且表达式值亦为1；②再执行赋值表达式“b＝1”，且表达式值亦为1；③最后a与b、c一样被赋值1。

（2）赋值时数据类型的转换

在赋值语句中，右边表达式和左边变量的类型不同时，系统会自动完成类型转换，将表达式的值转换为与左边变量相同类型的数据，再赋值，具体规则见表3.1。

例如：int x＝3；，则表达式：x＝x＋1.999的值是4。

例如：char c；c＝1345；putchar(c)；执行该程序段后输出结果为A。因将1345赋值给c，取其低字节内容，为65。

例如：int a＝－1；unsigned b＝65534；执行语句b＝a；后，b的值为65535，带符号整数赋值给无符号整型变量时，将符号位看作数值位；语句改为a＝b；a的值为－2，无符号整数赋值给带符号整型变量时，最高位按符号位处理。

表3.1　不同类型数据的赋值转换规则

变量类型	表达式类型	转换规则	举　例
char	int	取表达式值的低8位内容	表达式值65或577，变量为'A'
	float、double	取表达式整数部分的低8位	表达式值65.789，变量为'A'
int	char	将对应ASCII码值赋给变量	表达式值'A'，变量为65
	float、double	舍弃小数部分	表达式值1.999，变量为1
float	char、int、double	浮点形式，有效位数为6	执行float t＝3.14159265，t为3.14159
double	char、int、float	浮点形式，有效位数为16	

说明：带符号整数赋值给无符号整型变量时，符号位将作为数值位对待，如“unsigned int a; int b=-1; a=b;”则 a 的值为 65535。反之，无符号整数赋值给带符号整型变量时，最高位按符号位理解。

2. 复合赋值运算

C 语言的双目运算符与赋值运算符的结合，构成复合赋值运算符，亦称自反算术赋值运算符。由双目算术运算符构成的复合赋值运算有：

+=、-= 、*= 、/=、%=

这类运算符实际上是把“算术运算”与“赋值运算”两个动作结合在一起，作为一个复合运算符使用。运算符的优先级与结合性与“=”相同。

复合赋值运算构成的表达式在计算时，先把左边变量的当前值与右边整个表达式的值进行相应的算术运算，然后把运算的结果赋给左边的变量，整个复合赋值表达式的值也是左边变量的值。

复合赋值运算可以简化程序，但降低了程序的可读性，且易导致错误。

例如：表达式“x*=y+5”的等效形式是“x=x*(y+5)”，不能理解为“x=x*y+5”；表达式“x/=2*y-10”的等效形式是“x=x/(2*y-10)”，不能理解为“x=x/2*y-10”。即右边的表达式是个整体要先计算，再将与左边变量进行相应的算术运算后的结果赋值给左边的变量。

建议读者在编写程序时，使用的复合赋值表达式尽可能简单易于理解，如：i+=2；少用复杂的复合赋值运算，如：x/=2*y-10 等。

3.1.5 逗号运算符和表达式

1. 逗号运算符

逗号运算符“,”的功能是将两个或两个以上的表达式连接起来，从左到右求解各个表达式，最后一个表达式的值为整个逗号表达式的值。

逗号运算符的优先级是所有运算符中最低的，其结合性为从左到右。

2. 逗号表达式

逗号表达式的一般形式：

```
表达式 1, 表达式 2, …, 表达式 n
```

逗号表达式的计算过程是：先计算表达式 1，再计算表达式 2，……最后计算表达式 n，逗号表达式的值为表达式 n 的值。

例如：表达式“a=2+4,3*5”的执行步骤是：计算 2+4 为 6，因“=”的优先级比“,”高，所以先将 6 赋值给 a，再计算 3*5 的值 15，最后整个表达式的值为 15。

再如：若有定义 int a=2,c;,语句“c=(b=a++,a+2);”的执行步骤是：将 a 的原值 2 赋值到变量 b，然后 a 自增为 3；再计算 a+2 值为 5，逗号表达式的值取后一个表达式的值即 5，将 5 赋值到变量 c。

3.2　常用库函数

在用C语言编制程序时，用户可以根据需要将某些独立功能写成自定义函数。为了用户使用方便，C语言处理系统提供了许多已编写好的函数，这些函数被称为库函数。用户在使用这些函数时，用#include预处理命令将对应的头文件包含到程序中即可。

在本节中，将介绍常用的输入输出函数、数学函数和字符函数等，其他函数的使用方法将在以后各相关章节中陆续介绍。

3.2.1　输入输出函数

C语言没有提供输入输出语句，数据的输入输出操作是通过调用库函数实现的。编程者只要调用合适的系统库函数，就可以完成各种数据的输入输出工作。下面介绍C语言中常用的输入输出库函数，使用输入输出库函数时，应在源文件中包含头文件stdio.h。

1. 字符输出函数 putchar

函数原型：

```
int putchar(char x)
```

功能：向标准输出设备(一般是屏幕)输出一个字符x。

说明：输出的x可以是字符常量(包括转义字符)、变量或表达式，还可以是整型数据。

例如：

```
#include <stdio.h>
void main()
{
   putchar('A');              //输出'A'
   putchar('\101');           //输出ASCII码值为101(八进制)对应的字母'A'
   putchar(65);               //输出ASCII码值为65(十进制)对应的字母'A'
   putchar('\n');             //输出一个回车符
}
```

2. 字符输入函数 getchar

函数原型：

```
int getchar()
```

功能：从标准输入设备(一般是键盘)读取一个字符。

例如：

```
#include <stdio.h>
void main()
```

```
{
    char c1,c2,c3;
    c1=getchar();
    c2=getchar();
    c3=getchar();
    putchar(c1);
    putchar(c2);
    putchar(c3);
}
```

从键盘输入 a　b　c↙,则 c1 得到'a'、c2 得到空格、c3 得到'b',字符'c'没有使用。

3. 格式输出函数 printf

函数原型:

```
int printf(格式控制字符串,表达式列表)
```

功能:按照格式控制字符串所给定的输出格式,把各表达式的值在显示器上输出。

例如:printf("a=%d,x=%f\n",a,x)的输出结果是,先输出 a=,接着按格式符%d 指定的格式输出变量 a 的值,再输出逗号,接着输出 x=,然后按格式符%f 指定的格式输出变量 x 的值,最后输出一个换行符。

格式控制字符串由格式说明符和其他字符组成,printf 函数从格式控制字符串的首字符开始输出,到格式控制字符串尾部结束输出,基本规则为

(1) 遇格式说明符(以"%"开头),则以此格式输出列表中对应表达式的值。

(2) 遇非格式说明符则原样输出(若要输出一个"%"字符,在字符串中用"%%"表示)。

格式说明符的作用是将对应的数据转换为指定的格式输出,其作用见表 3.2。

表 3.2　printf 的格式说明符及其作用

格式说明符	作　　用
%d 或%i	以带符号的十进制形式输出整数(正数不显示符号)
%o	以八进制无符号形式输出整数(不显示前导 0)
%x 或%X	以十六进制无符号形式输出整数(不显示前导 0x),格式说明字符用 x 时,以小写形式输出十六进制数码 a～f;用 X 时,输出对应的大写字母
%u	以十进制无符号形式输出整数
%c	以字符形式输出一个字符
%s	输出字符串中的字符,直到遇到'\0'
%f	以小数形式输出单、双精度型数,默认输出 6 位小数
%e 或%E	以标准指数形式输出单、双精度型数,小数部分默认输出 6 位小数,格式说明字符用 e 时指数以"e"表示,用 E 时指数以"E"表示
%g 或%G	由系统自动选定%f 或%e 格式,以使输出宽度最小,不输出无意义的 0,用 G 时若以指数形式输出则选择大写字母"E"

在格式说明符中，%与格式说明字符之间可以加可选的附加说明符，见表 3.3。

表 3.3　printf 的附加说明符及其意义

附加说明符	意　　义
字母 l 或 L	输出长整型数据时，加在 d、i、o、x、X、u 前
m(正整数)	数据输出的域宽(列数)，当数据实际输出列数超过 m 时，则按实际宽度输出(m 不起作用)，如数据实际输出列数少于 m 时，则在数据前补空格到 m 列
.n(正整数)	对于实数，表示输出 n 位小数；对于字符串，表示从左开始截取的字符个数
－	输出的数据在域内左对齐，右边补空格
＋	输出的数字前带有正负号
0	输出的数据在域内若右对齐时，在左边补 0
#	用在格式字符 o、x、X 前，使输出八进制或十六进制数时，输出前导的 0 和 0x

1）整型数据的输出

(1) 格式符 %d、%i 输出基本整型数据。

例如：

```
int a=10,b=20;
printf("a=%d,b=%i\n",a,b);
```

输出结果为

```
a=10,b=20
```

在格式控制字符串中，如有除格式说明符以外的其他字符，则会原样输出这些字符，它们改进了输出效果，提高了输出结果的易理解性。

例如：

```
int a=10,b=20;
printf("两数之和:%d   两数之差:%d\n",a+b,a-b);
```

输出结果为

```
两数之和:30   两数之差:-10
```

(2) 格式符%u、%o、%x 分别以十进制、八进制、十六进制输出无符号整数。

例如：

```
Short int a=1,b=-1;
printf("%d,%d,%u,%u,%o,%o,%x,%x\n",a,b,a,b,a,b,a,b);
```

输出结果为

```
1,-1,1,65535,1,177777,1,ffff
```

变量a、b在内存单元中以二进制补码形式存放：

a：00000000 00000001

b：11111111 11111111

其中最高位是符号位。以%u、%o、%x格式输出整数时，如果是负数，将补码表示的符号位也看成是数值输出。

(3) 指定宽度输出。

格式符%md、%mo、%mx、%mu均可指定输出数据的宽度(列数)，m(正整数)小于实际宽度时不起作用，m大于实际宽度时，结果右对齐输出，左边补空格。m前加负号，则在域宽内左对齐输出，右边补空格。

例如：

```
int a=12345,b=-1;
printf("a=%4d,b=%4d\n",a,b);
```

输出结果为

```
a=12345,b=  -1
```

例如：

```
int a=12,b=3;
printf("a=%-4d,b=%-4d左对齐\n",a,b);
```

输出结果为

```
a=12  ,b=3   左对齐
```

(4) 输出长整型数据%ld、%lu、%lo、%lx。

例如：

```
long int a=12345678,b=-1;
printf("%5ld,%5ld\n",a,b);
```

输出结果为

```
12345678,    -1
```

2) 字符、字符串的输出

(1) 格式符%c输出一个字符。

例如：

```
char c='$';
printf("%c,%4c\n",c,c);
```

输出结果为

```
$,  $
```

字符型数据在内存中是以其 ASCII 码值存放的，所以字符数据可以用整数形式输出它的 ASCII 码值，整型数据只要它的值在 ASCII 码值范围之内，也可以用字符形式输出。

例如：

```
char c='A'; int a=66;
printf("%c,%d,%c,%d\n",c,c,a,a);
```

输出结果为

```
A,65,B,66
```

(2) 格式符%s 输出字符串。

例如：

```
printf("%s %s\n", "Windows", "XP");
```

输出结果为

```
Windows  XP
```

(3) 指定宽度输出。

%mc、%ms 可以按指定宽度输出字符、字符串。m 小于实际宽度时不起作用，m 大于实际宽度时左边补空格，m 前加负号右边补空格。%m.ns 指定只输出字符串的前 n(正整数)个字符。

例如：

```
printf("开始%10s %-5s 结束\n","Windows","XP");
printf("开始%10.3s %.3s %-5.3s 结束\n","Windows","Windows","XP");
```

输出结果为

```
开始    Windows XP   结束
开始    Win Win XP   结束
```

3) 实型数据的输出

(1) 用格式符%f 输出实型数据。

例如：

```
float x=123.45; double y=1234.567898765;
printf("x=%f  y=%f\n",x,y);
```

输出结果为

```
x=123.450000  y=1234.567899
```

float、double类型数据都可以用相同的格式符(即double类型不必用%lf格式输出)。格式符%f使得输出结果保留6位小数(不足6位后面补0,超过6位在第七位四舍五入)。

例如:

```
float x=9876.54321; double y=9876.54321;
printf("x=%f   y=%f\n",x,y);
```

输出结果为

```
x=9876.542969  y=9876.543210
```

因float型数据有效位数为6位,double类型有效位数为16位,变量x只有前6位数字有意义,而变量y的所有数字均有效。

(2) 用格式符 %m.nf、%.nf输出实型数据。

例如:

```
float x=123.456,y=65.4321; double z=-123.4567898765;
printf("x=%4.2f  y=%.2f  z=%18.8f\n",x,y,z);
```

输出结果为

```
x=123.46  y=65.43  z=-123.45678977
```

m、n为正整数,m为输出数据所占列数(小数点及符号位各占一列),n为输出小数点后面的位数。

m大于实际宽度时,左边补空格至m位,m前有负号则右边补空格至m位,m小于实际数据输出宽度时自动失效。格式符 %.nf表示不规定域宽,只设定输出小数位数。

(3) 用格式符%e、%m.ne、%.ne以指数形式输出实型数据。

例如:

```
float x=123.456,y=65.4321; double z=-123.4567898765;
printf("x=%e  y=%.2e  y=%18.8e\n",x,y,z);
```

输出结果为

```
x=1.234560e+002  y=6.54e+001  z=  -1.23456790e+002
```

格式符%e的输出形式为:

×.××××××e±×××

格式符%m.ne的输出形式为:

×.×…×e±×××

小数点后 n 位，总宽度 m 位。总宽度与实际宽度不符时，所做处理与%m.nf 类似。

(4) 用格式符%g、%m.ng、%.ng 输出实型数据。

由系统自动选定%f 或%e 格式，以使输出宽度最小，不输出无意义的 0。

例如：

```
float x=12345.6; double y=-123456789.8765;
printf("x=%g  y=%18.8g\n",x,y);
```

输出结果为

```
x=12345.6  y=   -1.2345679e+008
```

4. 格式输入函数 scanf

函数原型：

```
int scanf(格式控制字符串,地址列表)
```

scanf 语句的功能是：按照格式控制字符串所给定的输入格式，把输入数据按地址列表存入指定的存储单元。

“格式控制字符串”的含义与 printf 函数相似；“地址列表”是由若干个地址组成的列表，可以是变量的地址、数组的首地址、数组元素的地址。

1) 格式控制字符串

格式控制字符串由格式说明符和其他字符组成，scanf 函数从格式控制字符串的首字符开始输入，到格式控制字符串尾部结束输入，与 printf 函数用法大体相似，基本规则如下。

(1) 遇非格式说明符则必须原样输入与之抵消。

例如：

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

必须输入 10，20，两数用逗号隔开。

(2) 遇格式说明符则以此格式输入数据存放到地址列表中对应的变量内存单元中。

scanf 函数可以使用的格式说明符和作用见表 3.4，在%与格式字符之间也可以有附加说明符，见表 3.5。

表 3.4　scanf 的格式说明符及其作用

格式说明符	作　　用
%d	输入一个十进制整数
%I 或%i	输入一个整数，可以是十进制数、带前导 0 或 0x 的八进制或十六进制数
%o	以八进制形式输入一个整数(可带前导 0，也可不带前导 0)

续表

格式说明符	作　　用
%x	以十六进制形式输入一个整数(可带前导 0x 或 0X,也可不带)
%u	输入一个无符号的十进制整数
%c	输入一个字符
%s	输入一个字符串,将输入的整个字符串存入到一个字符数组中。输入时遇空格或回车键结束,并自动在最后加存一个'\0',作为字符串的结束标识
%f 或%e	两个格式相同,用来输入实数,可以以小数形式或指数形式输入

表 3.5　scanf 的附加说明符及其意义

附加说明符	意　　义
字母 l 或 L	加在 d、I(或 i)、o、x、u 前,表示输入长整型数据;加在 f、e 前,表示输入双精度型数据
字母 h 或 H	加在 d、I(或 i)、o、x、u 前,表示输入短整型数据
m(正整数)	指定输入数据所占的域宽(列数)
*	表示对应的输入项在读入后不赋给相应的变量,不需要为其指定地址参数

(3) 格式说明符:

输入单个字符用格式符%c,输入字符串,用格式符%s。

输入各类整型数据,用格式符%d、%i、%o、%x、%u。

输入不同字长的整型数据时,格式符中应当加入长度修饰符,如%hd、%ld 分别用于 short、long 类型数据的输入。

输入 float 型数据,用格式符%f;输入 double 类型数据,用格式符%lf。

(4) 指定输入数据的宽度:

用格式符%mc 读入 m 个字符,将首字符赋值给相应的字符变量。

以输入 int 类型数据为例,格式符%md、%mo、%mx 均可指定输入数据的宽度为 m 位,符号占一位,其他整型数据的指定宽度输入格式与之类似。

以输入 float 型数据为例,格式符%mf 可指定输入数据的宽度,符号和小数点各占一位,不可以指定小数位数,double 类型数据的指定宽度输入格式符为%mlf。

例如:

```
int a;float x;
scanf("%3d%f ",&a,&x);
```

输入数据:−1234 56.78,则变量 a 和 x 的值分别为−12、34.0。

例如:

```
int a;float x;
```

```
scanf("%d%4f ",&a,&x);
```

输入数据：－1234 56.78，则变量 a 和 x 的值分别为－1234、56.7。

2）地址列表

在 scanf 中，必须列出变量的地址列表，表示将输入的数据送到相应的地址所代表的内存存储单元中。变量的内存地址，通过取址运算符“&”得到。如 &x 即得到 x 在内存中的地址。

3）输入缓冲区

scanf 函数执行时，并不是输完一个数据项就被送入对应的变量中，而是将输入的数据项先存放在内存输入缓冲区中，只有在按回车键后，才按 scanf 中的格式控制字符串所给定的输入格式从缓冲区依次读数据，存入地址列表所对应的变量。

若缓冲区中的数据个数比 scanf 函数所需数据个数少，等待用户继续输入数据；若缓冲区中的数据个数比 scanf 函数所需数据个数多，多出数据留在缓冲区内，可被下一个输入语句使用；程序在首次执行输入数据库函数前，缓冲区为空。

【例 3.9】 已知等差数列的第一项为 a，公差为 d，求前 n 项之和，a、d、n 由键盘输入。

```
#include <stdio.h>
void main()
{
    int a,d,n,sum;
    printf("input a n");
    scanf("%d%d",&a,&n);
    printf("input d");
    scanf("%d",&d);
    sum=a*n+n*(n-1)*d/2;
    printf("sum=%d\n",sum);
}
```

程序运行：

```
input a n 1 100↙
input d 2↙
sum=10000
```

程序说明：输入 1 和 100 两个数据，第一条 scanf 语句接收并分别存放到变量 a 和 n 中。再输入 2，第二条 scanf 语句接收该数据，并存放到 d 中。按等差数列前 n 项和的计算公式计算出 1＋3＋5＋…前 100 项之和并输出。若输入 3 个数据：1　100　2↙，第一次执行 scanf 语句后，缓冲区内还有一个数据“2”，执行下一条 scanf 语句时直接从缓冲区内读入 2 赋值给变量 d。

4）输入项之间的分隔符

scanf 函数从缓冲区中读取数据时，数据与数据之间必须有分隔符，以便确定哪些信息

作为一个数据项。C 语言确定一个数据项的结束,有下列几种方法:

(1) 数值类型数据可用空格、换行符('\n')、制表键 Tab('\t')作分隔符。

(2) 与对应的格式说明项不匹配或已到规定宽度。

【例 3.10】 用 scanf 函数输入数据。

```
#include <stdio.h>
void  main()
{
    int i1,i2;
    float f1,f2; char c1;
    scanf("%d%d%5f%c%5f",&i1,&i2,&f1,&c1,&f2);
    printf("i1=%d,i2=%d,f1=%.3f,f2=%.3f,c1=%c\n",i1,i2,f1,f2,c1);
}
```

程序运行:

```
123  456  34.5656.789↙
i1=123,i2=456,f1=34.560,f2=6.789,c1=5
```

程序运行:

```
123  456  34.#56.789↙
i1=123,i2=456,f1=34.000,f2=56.780,c1=#
```

程序说明:不指定输入宽度的各数值数据之间可以用空格、回车符、Tab 键作为间隔符;指定宽度输入的各种类型数据之间可以不需要间隔符;无论指定宽度与不指定宽度的输入,遇到与规定格式不符的字符时结束一个数据项的输入。

(3) 在格式控制字符串中可以包含其他的普通字符,执行 scanf 函数时,这些普通字符在屏幕上不显示,而是要求在数据输入时,作为数据分隔符,照原样输入这些普通字符。

如:

```
scanf("%d,%f,%c",&i1,&f1,&c1);
```

则每一数据项必须用逗号分隔。

输入 123,34.56,r↙则各变量的赋值如下:

```
i1=123    f1=34.56    c1='r'
```

输入数据中的逗号是不可缺少的,如输入项用空格分隔,则各变量无法正确接收数据。

若输入语句为"scanf("i1=%d i2=%d,f1=%f",&i1,&i2,&f1);",则必须用"i1="“i2=""",f1="作为分隔符,例如输入 i1=12 i2=345,f1=67.89↙。

5) 抑制字符"*"

格式符符号"%"后加入"*",表示读入的数据不赋值给任何变量,不需要为此格式符指