第3章

运算符与表达式

3.1 概　　述

C语言中运算符和表达式数量之多，在高级语言中是少见的。正是丰富的运算符和表达式使C语言功能十分完善，这也是C语言的主要特点之一。C语言提供的运算符可以进行数据的运算处理。运算符按功能分为：算术运算符、赋值运算符、关系运算符、逻辑运算符和指针运算符等；按参与运算对象的个数分为：单目运算符、双目运算符和三目运算符。

使用运算符时，主要关注以下三个方面。

1. 运算符的目

运算符能连接运算对象的个数称为运算符的目，包括三种：

(1) 单目运算符：只能连接一个运算对象，如++、--和 & 等。

(2) 双目运算符：可以连接两个运算对象，如+、-、*、/、>和<等。C语言的运算符大多属于双目运算符。

(3) 三目运算符：可以连接三个运算对象。C语言中只有一个三目运算符，即条件运算符(?:)。

2. 运算符的优先级

当一个运算对象两侧的运算符优先级别不同时，应遵循优先级高的先处理的规则。

优先级是指在使用不同运算符进行计算时执行的先后次序。如在算术运算符中，乘除运算符的优先级高于加减运算符。一个表达式中若有多个运算符混合在一起，则计算的先后次序为先算括号，再根据相应运算符的优先级，高的优先。例如，x+y&&z，按照运算符的优先级要先计算 x+y，x+y 的计算结果再和 z 进行 && 操作。附录C中按优先级顺序列出了C语言的所有运算符。

3. 运算符的结合方向

结合方向又称为结合性，是指当一个运算对象连接两个同一优先级的运算符时，按运

算符的结合性所规定的结合方向处理。C 语言中各运算符的结合性分为两种，分别为左结合性(自左至右)和右结合性(自右至左)。如果先结合左边的运算符，称为“自左至右”的结合方向，如果先结合右边的运算符，则称为“自右至左”的结合方向。

结合方向是 C 语言独有的特点。C 语言中，赋值运算符、条件运算符以及所有的单目运算符都是“自右至左”的结合方向，其余都是“自左至右”的结合方向。例如，a＝15＋20；需要先计算＝右边的值 15＋20 为 35，再将计算结果 35 赋值给左边的变量 a。

表达式是用运算符将运算对象连接而成的符合 C 语言规则的算式。表达式包括算术表达式、关系表达式、逻辑表达式、赋值表达式、条件表达式和逗号表达式等。表达式中的运算对象可为常量、变量和函数等。表达式的运算主要按照运算符的优先级和结合性所规定的顺序进行，其次还要考虑参与运算的对象是否具有相同的数据类型以及是否需要类型转换。每个表达式代表着一个确定的值和确定的数据类型。作为特例，单个的常量、变量和函数也可以视为表达式，例如，printf("hello!");可以看做是一个表达式。

需要注意的是，C 语言表达式要求写在同一行上，并且在表达式求值时，同一优先级的运算符，运算次序由结合方向决定；不同优先级的运算符出现在同一表达式时，按运算符优先级的高低次序执行。

3.2　算术运算符与算术表达式

C 语言中，算术运算符是双目运算符。基本的算术运算符有：＋(加)、－(减)、＊(乘)、/(除)、％(取模，求余)。

其中，乘、除、取模运算符的优先级要高于加、减运算符，即先乘、除、取模，后加、减。算术运算符的优先级在所有运算符中是较高的，仅次于括号、单目运算符、类型转换运算符和求字节数运算符。算术运算符的结合方向为自左至右。

由算术运算符、括号以及操作对象组成的符合 C 语言语法规则的表达式称为算术表达式。例如，

```
int a=2;float b=4.5;
```

算术表达式 2＊a＋b 的值是多少？

计算过程是：首先计算整数 2 乘以整数 a，得到整数 4，然后将整数值 4 转换为单精度实数值 4.0，最后计算得到结果 8.5(实型)。又如，

```
int x=3,y=4,z=5;
```

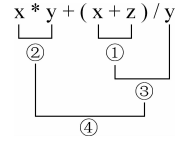

图 3-1　表达式计算顺序

算术表达式 x＊y＋(x＋z)/y 的计算过程是：遵循 C 语言对算术表达式计算顺序的“运算符优先级规则”。如图 3-1 所示，首先计算括号()内的值，得到 8；然后按自左向右的结合方向进行乘法和除法运算，分别得到 12 和 2；最后进行加法运算。因此表达式的结果为 14。

算术表达式的结果值可以是整数、单精度实数和双精度实数。

例 3-1：蛋白质相对分子质量计算问题。

蛋白质相对分子质量计算公式 Mpr＝$n*a-18(n-m)$，其中 Mpr 代表蛋白质的相对分子质量，n 代表构成蛋白质的氨基酸数，a 为氨基酸的平均相对分子质量，m 为构成蛋白质的肽链条数。

可以用 C 语言改写此公式如下：mpr＝n＊a－18＊(n－m)；

源程序代码：

```
#include<stdio.h>
void main()
{
    long int a,m,n,mpr;
    scanf("%ld%ld%ld",&a,&n,&m);
    mpr=n*a-18.0*(n-m);
    printf("mpr=%ld \n",mpr);
}
```

进行算术运算时要注意下面几个问题。

(1) 两个整数进行除法(/)运算，其结果仍为整数。例如，3/2 的结果为 1，舍去小数部分，相当于整除操作。如果整数与实数进行除法运算，则结果为实数。例如，3.0/2 的结果为 1.5。

(2) 求余运算(%)要求参与运算的两个操作数均为整型，不能为其他类型。例如，8%5，其值为 3。

(3) 除以 0(除数为 0)在计算机系统中是没有意义的，会导致溢出错误。

3.3 关系运算符与关系表达式

关系运算符用于对两个操作数进行比较，是双目运算符。C 语言提供的关系运算符包括：＞(大于)、＜(小于)、＞＝(大于等于)、＜＝(小于等于)、＝＝(等于)和！＝(不等于)。其优先级高于赋值运算符而低于算术运算符，其中＞、＜、＞＝、＜＝具有相同的优先级并且高于具有相同优先级的！＝、＝＝。

关系运算符的结合性均为左结合性。若有多个关系运算同时进行，则先按优先级次序运算，优先级相同时再自左向右计算。

关系表达式由关系运算符与两个表达式组成。一般形式为：

<表达式>关系运算符<表达式>

例如，a＞b、'a'＋'b'＜c、a＞(b＞c)、a＝＝(b＜c)等都是合法的关系表达式。

关系运算符中“关系”二字的含义是指一个数据与另一个数据之间的关系，这种关系只有成立与不成立两种可能情况，在 C 语言中用逻辑值来表示，逻辑上的真与假使用数字 1 与 0 来表示。关系成立时，表达式的结果为真(1)，否则表达式的结果为假(0)。

例如，int a=2；float b=3.4；表达式 a<b 的结果是什么？

由于关系运算符两侧的数据类型不统一，需要先将数据类型转换成同一个数据类型（实型），然后再进行比较，所以 a(2.0)<b(3.4)的结果值为真(1)。

使用关系表达式要注意以下几点。

(1) C 语言中，非 0 值即为真(1)，0 值即为假(0)，如 x=3，因为 x 等于非 0 值，则表达式 x 的逻辑值为真。

(2) 运算符>=、==、!=、<=是两个字符构成的一个运算符，如果在两个字符中间插入空格则会产生语法错误。例如，

```
a>=b;
```

是错误的。但是可以写成：

```
a>=b;                    /*在运算符的两侧增加空格可以提高可读性*/
```

(3) 由于计算机中存放的实数与实际中的实数存在着一定的舍入误差，因此对实数进行==(相等)或!=(不相等)的比较，容易产生错误结果，应该尽量避免。

(4) 不要将==写成=，例如，判断 x 和 y 是否相等时，如果将 x==y 写成 x=y，C 语言会将该表达式作为赋值表达式处理，将 y 的值赋给 x，并判断 x 的值是否为 0，如果 x 为 0 则表达式为假，否则表达式为真，造成逻辑含义上的错误。

3.4 逻辑运算符与逻辑表达式

“逻辑”是指连接“关系”的方式。C 语言提供的逻辑运算符可以将简单的条件组合成复杂的条件。逻辑运算符包括：&&(逻辑与)、||(逻辑或)和!(逻辑非)。其中，&& 和 || 是双目运算符，! 是单目运算符。

逻辑运算符中，! 运算级别最高、&& 和 || 运算同级。! 运算的优先级高于算术运算符，而 && 和 || 的运算优先级则低于关系运算符。! 运算符的结合性为右结合性；&& 和 || 运算符的结合性为左结合性。

逻辑表达式由逻辑运算符(&&、||、!)连接两个表达式而形成，其结果为逻辑真(1)或逻辑假(0)。一般形式为：

<表达式>逻辑运算符<表达式>

表达式也可以是逻辑表达式，例如：(a>b)&&(x>y)、(a>b)||x、!(a>b)、(a||b)&&(c||d)等。

例如，0≤x≤100 在 C 语言中的逻辑表达式形式，可以写成 x>=0 && x<=100，也可以写成 0<=x && x<=100，但不可以写成 0<=x<=100，该写法将被编译器解释为：首先计算关系表达式 0<=x 的值，再判断该值是否小于等于 100。这样，表达式 0<=x 的结果无论是 1(真)还是 0(假)，该值都小于 100，因此表达式 0<=x<=100 的值永远为真(1)。

参与逻辑运算的操作数(表达式)应该是 0 或非 0 的逻辑值，其运算规则见表 3-1(假

设两个操作数分别为 a 和 b)。

表 3-1　逻辑运算规则

a	b	a&&b	a‖b	！a	！b
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	1	0	0

C 语言规定逻辑表达式的执行过程如下：

对于任意的逻辑表达式 a&&b(与运算)，先计算 a 的值，若 a 的值为假(0)，则不再计算 b 的值，而直接得到 a&&b 的结果值为 0；若 a 的值为真(非 0)，则计算 b 的值，并依据 b 的值决定 a&&b 的结果值，若 b 的值也为真，则结果为 1，否则为 0。

对于任意的逻辑表达式 a‖b(或运算)，先计算 a 的值，若 a 的值为真，则不再计算 b 的值，而直接得到 a‖b 的结果值为 1；若 a 的值为假，则计算 b 的值，并且依据 b 的值决定 a‖b 的结果值，若 b 的值为真，则结果为 1，否则为 0。

对于任意的逻辑表达式!a(非运算)，先计算 a 的值，若 a 的值为假，则！a 的值为 1，若 a 的值为真，则！a 的值为 0。

例如，int a=3，b=4，c=5；则(a<b)&&(a>c)的结果为 0。因为 a<b 的结果为 1，而 a>c 的结果为 0，因此整个表达式为 0。

熟练掌握 C 语言的关系运算符和逻辑运算符，可以巧妙地利用逻辑表达式来构造一个复杂的程序运行条件。例如，判断闰年。闰年的条件是能被 4 整除，但不能被 100 整除的年份或者能被 400 整除的年份，设年份用变量 y 表示，则判断表达式可以描述为：

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

3.5　赋值运算符与赋值表达式

C 语言中赋值运算符为=，赋值表达式是由赋值运算符(=)连接表达式(右侧)和变量(左侧)形成的。一般形式为：

<变量名>=<表达式>

赋值运算用于将赋值运算符右侧表达式的结果值赋予左侧的变量，表达式可以是常量、变量、表达式或另外一个赋值表达式。在做赋值运算时，尽量做到赋值运算符两侧的数据类型一致，若不一致，则赋值时会自动将赋值运算符右侧表达式的值转换成与左侧变量相同的类型。

赋值运算符的优先级较低，在所有运算符中，它的优先级仅高于逗号运算符。赋值表达式按照自右向左的顺序结合。

例如：

```
a=1;          /* 表示把常量 1 赋给变量 a */
x=a*b;        /* 由于赋值运算符的优先级低于算术运算符,所以先计算 a*b,再做赋值运算,
                 将计算结果值赋给变量 x */
a=b=c=4;      /* c=4 表示一个赋值表达式,其值为 4;整个赋值表达式相当于是由 c=4、b=c 和
                 a=b 三个赋值表达式组合而成的。因此 b 的值为变量 c 的值(4);a 的值为变
                 量 b 的值(4);赋值表达式的值为变量 a 的值(4) */
```

表达式 a=(b=1)+(c=5)同样是合法的,其计算过程:首先计算 b=1,将 1 赋值给 b,而该表达式的结果值也为 1;其次计算 c=5,将 5 赋值给 c,表达式的结果值也为 5;然后进行加法运算;最后将 6 赋值给 a,因此 a 的值为 6。其中由于赋值运算的优先级低于算术运算,需要采用加()的方式改变运算次序。

在赋值表达式计算中,=不是数学中的等号,它表示一个动作:将其右侧的值赋予左侧的变量中(左侧只允许是变量,不能是表达式)。例如 i=i+1 表示将变量 i 中的值与 1 相加后重新赋给变量 i;而 x+5=y 则不是一个正确的赋值表达式。

例 3-2:从键盘输入两个整数分别存入变量 a 和 b,编写程序将变量 a 和 b 的值交换。

源程序代码:

```
#include<stdio.h>
void main()
{
    int a,b,temp;              /* 声明变量,其中 temp 为临时变量,用于单元存储中间结果 */
    scanf("%d%d",&a,&b);       /* 输入两个整数 */
    temp=a;                    /* 将其中的一个整数 a 存放在临时变量中,然后进行交换 */
    a=b;
    b=temp;
    printf("%d,%d\n",a,b);     /* 输出交换后的两个整数 */
}
```

C 语言允许在赋值运算符=之前加上其他运算符以构成复合的赋值运算符。凡是双目运算符,都可以和赋值运算符一起组合成复合的赋值运算符。在 C 语言中,可以使用的复合赋值运算符有:+=,-=,*=,/=,%=,<<=,>>=,&=,^=和|=。

复合赋值表达式是通过复合赋值运算符将一个变量和一个表达式连接起来的式子。一般形式为:

<变量名>复合赋值运算符<表达式>

复合赋值运算的作用等价于:

<变量名>=<变量名>运算符<表达式>

例如:

```
a+=5;          /* 等价于 a=a+5; */
a*=b+5;        /* 等价于 a=a*(b+5); */
a+=a-=a*a;     /* 等价于 a=a+(a=a-(a*a)),假设 a 为 5,先计算表达式 a*a 的结果(25),
                  再计算表达式 a=a-(a*a)的结果(a=5-25=-20);最后计算表达式 a=a+
                  (-20)的结果(-40) */
```

3.6 ++/--运算符与自增/自减表达式

自增(++)/自减(--)运算符分别用于使变量值自增1或自减1。++/--是C语言中较为独特的单目运算符,其操作对象可以是字符型、整型、指针型变量或数组元素,运算结果的数据类型与运算对象的数据类型一致。++/--运算符的优先级高于双目算术运算符,而低于括号()运算符,结合性为自右到左。一般形式为:

++变量名/--变量名
变量名++/变量名--

如果将++/--运算符放在某个变量之前,则称为前缀运算。前缀运算执行的是"先运算后使用"的处理过程,即将变量先加/减1,然后将结果值运用在出现该变量的表达式中。

如果将++/--运算符放在某个变量之后,则称为后缀运算。后缀运算执行的是"先使用后运算"的处理过程,即将变量当前的值参与到表达式的处理中,然后再对变量值加/减1。

例如:

```
int a=10,b=10;
x=++a;                    /*表示a的值先加1变成11后,再将新值11赋给x*/
x=b++;                    /*表示将b的当前值10赋给x后,b再加1变为11*/
```

当出现难以区分的若干个+或-所组成的运算符串时,C语言规定:从左到右取尽可能多的符号组成运算符。所以表达式x+++y应理解为(x++)+y的形式。

例如:

```
int x=5,y=5;
y=x+++y;              /*先进行x+y的操作,结果为10赋予y后,再对x自增变为6*/
```

例3-3:下面程序的输出结果是多少?

```
void main()
{
    int a=1,b=1,c=1;
    a=a+++b+++c++;
    printf("%d,%d,%d",a,++b,c++);
}
```

输出结果:

```
4,3,2
```

变量a、b、c以初始化的方式都被赋值为1。语句a=a+++b+++c++相当于a=(a++)+(b++)+(c++),表示表达式的加法操作完成后所有变量的值加1。因

此执行此语句后，得到 a=4、b=2、c=2；再执行库函数调用语句 printf，其中变量 b 是前缀运算，所以在输出前变量 b 的值增 1 变为 3，变量 c 是后缀运算，因此输出 2 后，c 再增 1 变为 3。

在使用++/--运算时应注意下面几个问题：

(1) ++/--运算只能作用于变量，不允许对常量、表达式或其他类型的变量进行此操作。例如 1++，--(x+y)均非法。

(2) 当++/--运算独立构成一条语句时，则前缀运算和后缀运算的效果相同。

例如：++x;等价于 x++；

--x;等价于 x--；

但当++/--运算出现在复杂表达式中，前缀运算和后缀运算的结果不同。

(3) 如果一个表达式对同一个变量进行多次++/--运算，例如 a+++++a+++++a，不仅表达式的可读性差，而且不同编译系统对表达式的处理也不尽相同，导致结果可能会不相同。因此，好的程序设计习惯是在一个表达式语句中尽量不要出现过多的++/--运算符，如果需要可以拆分成多条语句。

3.7 条件运算符与条件表达式

条件运算符是 C 语言中唯一的三目运算符，由？和：组合而成。它要求有三个运算对象，每个运算对象的类型可以是任意类型的表达式(包括任意类型的常量、变量或函数)。

条件运算符的优先级高于赋值运算符并低于逻辑关系运算符，其结合方向为自右向左。

由条件运算符连接三个运算对象构成的表达式称为条件表达式，其结果可以是任何类型。一般形式为：

<表达式 1>?<表达式 2>:<表达式 3>

条件表达式实现的功能是：根据条件(表达式 1)成立与否(真/假)，选择其中一个表达式(表达式 2 或表达式 3)的结果作为整个表达式的结果。通常<表达式 1>为关系或逻辑表达式，<表达式 2>和<表达式 3>一般为算术表达式、赋值表达式、函数表达式或条件表达式。

条件运算的计算过程为：计算<表达式 1>的值，如果为真(非 0)，则计算<表达式 2>的值，并将<表达式 2>的值作为整个条件表达式的结果值；否则计算<表达式 3>的值，并将<表达式 3>的值作为整个条件表达式的结果值。

例如：

```
int a=2;float b=5.2;
```

表达式!a?2*b:b 的结果为 5.2。因为!a 的结果值为 0(假)，因此表达式 b 的值即为整个条件表达式的结果值。

在条件表达式中，三个运算对象的类型可以互不相同，如：x?'a':0.5。

例 3-4：输入两个整数，找出最大的数并将其输出。

源程序代码：

```
#include<stdio.h>
void main()
{
    int a,b,max;
    scanf("%d%d",&a,&b);
    (a>b)?(max=a):(max=b);          /*利用条件表达式求最大值并存放在变量 max 中*/
    printf("最大的数是 %d \n",max);
}
```

3.8 逗号运算符与逗号表达式

C 语言中，逗号(,)也是一种运算符，称为逗号运算符。逗号运算符的运算对象可以是任何类型的表达式，逗号运算符的优先级在 C 语言所有运算符中最低，结合性是自左到右。

逗号表达式由逗号运算符(,)及两个以上的表达式连接而成。一般形式为：

<表达式 1>,<表达式 2>,…,<表达式 n>

其功能是依次计算＜表达式 1＞,＜表达式 2＞,…,＜表达式 n＞的值，最后将＜表达式 n＞的值作为整个表达式的结果值。

例如，

```
int a=2,c;float b=5.2;
c=a,2*a,2*b;
```

上述表达式最终的结果为 10.4(最后一个表达式的值)，因为逗号表达式的求值顺序是：先计算 c＝a，将 a 的值赋给 c(c＝2)，其次计算 2＊a 的值(为 4)，最后计算 2＊b 的值(为 10.4)。当整个表达式计算结束后，c 的值为 2，而整个表达式的值为 10.4。

在多数情况下，使用逗号表达式不是为了取得和使用这个逗号表达式的最终结果值，其目的是为了分别按顺序求得每个表达式的结果值，这在循环结构中经常使用。

例如，c＝(a＝10,b＝5,a＋b)；首先计算()内的逗号表达式的值，顺序求值的过程是：a＝10,b＝5,10＋5，并将 10＋5 的值赋予变量 c(c 的值为 15)。

逗号不仅出现在逗号表达式中，还用于函数参数的分隔等，例如 printf("%d,%d",a,b)。

3.9 sizeof 运算符

sizeof 是 C 语言的一种单目运算符，以字节为单位获取指定类型值所需要的存储空间大小。sizeof 的优先级高于双目运算符。一般形式：

sizeof(类型名)

或

sizeof(变量名)

sizeof()的结果值是无符号整数，表示存储属于类型名或变量名的值所需要的字节数。在不了解系统中各数据类型所占存储单元的字节数时，可用 sizeof 运算取得。例如，在 16 位机上，因为 int 类型占用 2 个字节的地址空间，所以 sizeof(int)的值通常为 2，而在 32 位机上 sizeof(int)的值为 4。

通常情况下，运算符 sizeof()也适用于常量、变量或表达式。例如，float x＝5.0;，由于 float 类型占用 4 个字节的存储空间，因此 sizeof(x)的值是 4。此外 sizeof()也可以与其他操作符一起组成复杂表达式，例如 i * sizeof(int)。

3.10 类型转换

C 语言中，不同类型的数据在进行混合运算时需要进行类型转换，即将不同类型的数据转换成同种类型的数据后再进行计算。转换的方式有两种：隐式转换和显式转换。

3.10.1 隐式转换

隐式转换由编译系统自动完成，可以将数据从一种数据类型自动转换为另外一种数据类型。

1. 算术运算中的自动数据转换

如果一个运算符有两个不同类型的运算分量，C 语言在计算该表达式时会自动转换为同一种数据类型后才能进行运算。自动转换要依据“类型提升”的原则：按数据类型提升(由低向高)的方向进行，以保证不降低精度，即先将较低类型的数据提升为较高的类型，从而使二者的数据类型一致(但数值不变)，然后再进行计算，其结果是较高类型的数据。数据类型的高低是根据其类型所占空间的大小来判定，占用空间越大，则类型越高。

混合运算时自动转换的具体规则如下：

(1) 单精度实型数据(float)自动转换成双精度实型数据(double)；

(2) 字符型数据(char)和短整型数据(short)自动转换成整型数据(int)；

(3) 整型数据(int)与无符号型数据(unsigned)运算时自动转换为无符号型数据，整型数据(int)或无符号型数据(unsigned)与长整型数据(long)运算时自动转换为长整型数据(long)；

(4) 整型数据(int)、无符号型数据(unsigned)、长整型数据(long)与实型数据(float/double)运算时都转换成实型数据(double)。

转换规则如图 3-2 所示。

例如，表达式 x＋'a'＋y，其中 x 是 int 型，y 是 double 型。首先将字符常量 a 转换为 int 型，然后按照整型数据运算规则从左向右进行算术运算(x＋'a')，遇到实型数据 y 时，将整型数据自动转换为 double 型，并按照实型数据的运算规则进行运算(与 y 进行加法计算)，最终表达式的结果为 double 型。

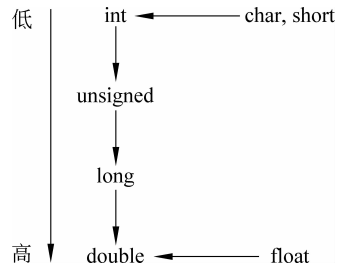

图 3-2　混合运算过程的转换规则

2. 赋值运算中的类型转换

在执行赋值运算时，如果赋值运算符两侧的数据类型不同，赋值运算符右侧表达式类型的数据将转换为赋值运算符左侧变量的类型。即计算出赋值运算符＝右侧表达式的值后，一律转换为＝左侧变量所属的类型，再赋值给左侧的变量。

转换规则：

(1) 将实型数据赋给整型变量时，舍弃小数部分。

例如：

```
int a;
a=15.5                          /*结果为 a=15(数据截取)*/
```

将整型数据赋给实型变量时，数值不变，将以实数的形式(在整数后添上小数点及若干个 0)存储到变量中。

例如：

```
float a;
a=10;                           /*结果为 a=10.0(数据填充)*/
```

(2) 将 double 型数据赋给 float 变量时，截取其前面 7 位有效数字存入 float 变量的存储单元中。相反，将 float 型数据赋给 double 变量时，数值不变，有效位扩展到 16 位。

(3) 将 char 型数据赋给整型变量时，由于字符型数据在运算时根据其 ASCII 码值自动转化为整型数据，因此只需将字符数据的 ASCII 码值存储到整型变量低 8 位中，高 8 位补 0。将 int 型数据赋给字符型变量时，只将其低 8 位存入字符型变量中。

(4) 将 unsigned int 型数据赋给 long int 型变量时，只需将高位补 0。将 unsigned int 型数据赋给字节数相同的整型变量时，将 unsigned 型变量在内存中的内容原样送入非 unsigned 型变量的内存中。将 int 型数据赋给 long 型变量时，只将 int 型数据放入 long 型变量的低字节中，高字节全部补充为 int 型数据的符号位。

不同类型数据之间的赋值运算，如果右侧的数据类型高于左侧时，将会丢失一部分数据，从而造成数据精度的降低；或者发生数据溢出，导致结果错误。

3.10.2　显式转换

一般情况下，数据类型的转换通常是由编译系统自动完成的，不需要人工干预，所以

被称为隐式类型转换。但如果程序要求一定要将某一类型的数据转换为另外一种类型，则可以利用强制类型转换运算符进行转换，这种强制转换过程称为显式转换。一般格式为：

(强制的类型名)<表达式>

显式转换用于强行将＜表达式＞的值转换成类型名所表示的数据类型。例如，(int) 4.2 的结果是 4；显式转换的目地是使表达式值的数据类型发生改变，从而使不同类型数据之间的运算能够进行下去。

如果表达式仅是单个常量或变量，则常量或变量不必用圆括号括起来；但是如果是含有运算符的表达式，则必须利用括号将其统一，否则容易发生歧义。

例如：

```
(int)a                    /* 表示将变量 a 的值强制转换为整型 */
(int)(a+b)                /* 表示将表达式 a+b 的计算结果强制转换为整型 */
(int)a+b                  /* 表示将变量 a 的值强制转换成整型后，再与 b 进行加运算 */
```

经显式转换后仅产生一个临时的、类型不同的数据继续参加运算，其常量、变量或表达式的原有类型以及原来数据值均不改变。

例如：

```
int x=5;float y;
y=(float)x/2;     /* 此时 x 的值被强制转换为实型(5.0)参与下一步运算，运算结果为 2.5。
                     但该操作并不改变 x 本身的数据类型，x 仍是整型 */
```

由于类型转换将占用系统时间，过多的转换将降低程序的运行效率。在设计程序时应尽量选择好数据类型，以减少不必要的类型转换。

*3.11 位运算符和位运算

C 语言提供的位运算是指进行二进制位的运算，这使得 C 语言也能像汇编语言一样编写系统程序。位运算是 C 语言的一种特殊运算功能，它以单独的二进制位为操作对象，这是它与其他运算符主要的不同之处。C 语言的位运算只有逻辑运算和移位运算两类。进行位运算的操作数只能是 int 型或 char 型数据。

3.11.1 位逻辑运算

C 语言所提供的位逻辑运算符包括 &(按位与运算符)、|(按位或运算符)、^(按位异或运算符)、～(按位取反运算符)。其中，&、|、^是双目运算符，优先级介于逻辑运算符与关系运算符之间，结合性是自左到右。～为单目运算符，优先级高于算术运算符，结合性为自右至左。

&、|、^运算的一般形式为：

<操作对象>位逻辑运算<操作对象>

～运算只有一个操作对象，形式为：

~ <操作对象>

1. &（按位与运算符）

按位与运算是将参与运算的两个数据，按对应的二进制数逐位进行逻辑与运算。只有当两个操作对象二进制数的相同位均为1时，结果数值的相应位才为1，否则为0。

例如，int型常量4和7进行按位与运算(4&7)，其运算过程为(仅取数据的两个字节分析)：

	4	(0000 0000 0000 0100)
&	7	(0000 0000 0000 0111)
=	4	(0000 0000 0000 0100)

而－4&7的运算过程为：

	－4	(1111 1111 1111 1100)
&	7	(0000 0000 0000 0111)
=	4	(0000 0000 0000 0100)

按位与运算通常用来对一个数据的某些位清0或保留某些位。例如，要保留变量x的低八位，而高八位清0，则可以执行运算：x&255(255的二进制数为0000 0000 1111 1111)。

2. |（按位或运算符）

按位或运算是将参与运算的两个数据，按对应的二进制数逐位进行逻辑或运算。只有当两个操作对象二进制数的相同位均为0时，结果数值的相应位才为0，否则为1。

例如：int型常量4和7进行按位或运算可表示为4|7，其运算过程如下：

	4	(0000 0000 0000 0100)
|	7	(0000 0000 0000 0111)
=	7	(0000 0000 0000 0111)

按位或运算一般用于将一个数据的某些位置1，而数据的其余位保持不变。例如，要将变量x的最低位置1，则可以执行运算：x|1(1的二进制数为0000 0000 0000 0001)。

3. ^（按位异或运算符）

按位异或运算是将参与运算的两个数据，按对应的二进制数逐位进行逻辑异或运算。只有当对应的两个二进制数位互斥的时候，对应位的结果才为1。

例如：int型常量4和7进行按位异或运算可表示为4^7，其运算过程如下：

	4	(0000 0000 0000 0100)
^	7	(0000 0000 0000 0111)
=	3	(0000 0000 0000 0011)

按位异或运算可以将一个数的某个或某些位翻转(即原来为1的位变为0，为0的变为1)，而其余位不变。例如，要将变量x的第四位取反，则可以执行运算：x&8(8的二进

制数为 0000 0000 0000 1000)。

4. ～(按位取反运算符)

按位取反运算是将参与运算的数据,按对应的二进制数逐位进行求反运算。即原来为 1 的位变成 0,原来为 0 的位变成 1。

例如:int 型常量 7 进行～运算可表示为～7,其运算过程如下:

～	7	(0000 0000 0000 0111)
=	−8	(1111 1111 1111 1000)

3.11.2 移位运算

移位运算包括左移运算和右移运算,可以实现二进制数值的移位(乘/除)处理。左移运算符<<和右移运算符>>是双目运算符,其优先级高于关系运算符而低于算术运算符,结合性自左到右。

1. 左移运算

左移运算的功能是将<<左侧操作对象的二进制数值逐位左移若干位,而左移的位数由<<右侧的数值指定。左侧被移出的位将被舍弃,而右侧空出的位置补 0。

例如:

```
int a,b;
a=5;
b=a<<2;
```

由于$(a)_{10}=(5)_{10}=(0000\ 0000\ 0000\ 0101)_2$,则 a 左移 2 位后得到 b 的结果为:$b=(0000\ 0000\ 0001\ 0100)_2=(20)_{10}$。

$b/a=20/5=4=2^2$,a 左移两位相当于扩大了 4 倍,由此可知,左移会引起数据的变化,左移一位相当于原数据乘以 2,左移 n 位则相当于将原数据乘以 2^n。由于位移操作的运算速度比乘法的运算速度高很多,因此在处理数据的乘法运算时,采用左移运算可以获得较快的速度。

2. 右移运算

右移运算的功能是将>>左侧操作对象的二进制值逐位右移若干位,右移的位数由>>右侧的数值指定。右侧被移出的位将被舍弃。

如上例:$(a)_{10}=(5)_{10}=(0000\ 0000\ 0000\ 0101)_2$;

若 b=a>>2;则 $b=(0000\ 0000\ 0000\ 0001)_2=(1)_{10}$。

右移同样会引起数据的变化,右移一位相当于将原数据除以 2。右移 n 位则相当于将原数据除以 2^n。

右移运算时,如果当前的数据为无符号数,则高位补零。如果当前的数据为有符号数,符号位为 0(正数),则左边补 0;符号位为 1(负数),则取决于所使用的系统:补 0 的称为“逻辑右移”,补 1 的称为“算术右移”。

3.11.3 复合位运算及补位原则

C 语言提供 5 个复合按位运算符，分别为 &=、!=、>>=、<<=和^=。

例如：a&=0x11 等价于 a=a&0x11，其他运算符依此类推。按位取反不存在复合运算。

不同长度的数据之间进行位运算，将按右端对齐的原则进行处理，即按长度最大的数据进行处理，将数据长度小的数据左端补 0 或 1。例如，char a 与 int b 进行按位运算时，需将字符 a 先转化为 int 型数据，在左端补 0，再进行位运算。

补位原则是：

(1) 对于有符号数据：如果为正数，则左端补 0；如果为负数，则左端补 1。

(2) 对于无符号数据：左端补 0。

3.12 表达式运算

表达式是由常量、变量、函数等通过运算符连接起来而形成的一个有意义的算式。特别地，一个常量、一个变量、一个函数都可以作为一个表达式。

表达式的计算过程是数据加工的过程，根据表达式中各个运算符的优先级和结合性进行运算。可以利用小括号()改变表达式的计算次序。

C 语言中，为了和英文字母 x 进行区分，用 * 表示乘号，除号用/表示。所有表达式都必须写在一行上，例如 a/b，不允许写成普通的数学算式$\frac{a}{b}$(分上、下行)；幂运算可以采用连乘的方式实现(例如 a^3 写成 a * a * a)，也可以调用数学库函数实现(例如 a^3 写成 pow(a, 3))；此外，数学库函数还提供常用的数学运算，如求平方根、求 sin 值等。例如，计算并显示 97 的平方根，可书写为：

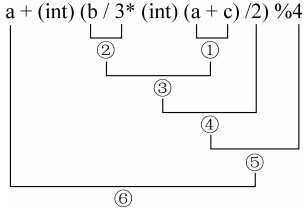

图 3-3 表达式计算过程

```
printf("%f",sqrt(97));
```

使用数学库函数，需要在源文件开始处添加预编译指令 #include<math.h>，否则会导致错误结果。

例 3-5：已知有：int b=7；float a=2.5，c=4.7；计算表达式 a+(int)(b/3 * (int)(a+c)/2)%4 的结果。

计算过程如图 3-3 所示。

3.13 其他语言中的运算符与表达式

运算符和表达式是程序设计语言的基础，C ++、Java、C # 中都包含了+、-、*、/、%、++和--等算术运算符；>、>=、==、!=、<=和<等关系运算符；&&、||和!等逻辑运算符及赋值运算符、逗号运算符等，这些运算符在意义上与 C 语言一致。

此外，根据语言的需要，C++、Java、C#等又扩充了自己的运算符，如C++增加了类定义运算符(class)、作用域运算符(::)、动态分配内存运算符(new)、删除动态分配内存运算符(delete)；Java增加了instanceof、interface等运算符；C#增加了is、sealed、abstract等运算符。

运算符的本质是针对特定对象的一种操作。例如加法运算符(+)，C语言将其规定为针对两个数值型数据的操作，操作的结果是这两个数据的和；而C++语言，加法运算符除了针对数值操作外，又被扩充为针对两个字符串的操作，运算结果是将这两个字符串连接到一起，甚至可以对加法运算符进行无限扩充，例如将其扩充为对两个向量进行操作，结果是两个向量的和。

3.14 案　　例

3.14.1 固体密度测量问题

问题：假设固体为不规则物体，通过在不同介质中的重量变化计算密度。

1. 问题陈述

对于规则物体而言，固体密度的体积容易测得，而对于不规则物体则比较难测量，一般常采用阿基米得定理法。

阿基米得定理指出：浸在液体中的物体受到一个向上的浮力，其大小等于物体所排开液体的重量。利用电子天平分别称得固体在空气中和在液体中的重量W_a和W_{fl}，如果忽略空气的浮力，已知液体的密度为ρ_{fl}，基于阿基米得定理测量固体密度的基本公式为：

$$\rho = \frac{W_a \cdot \rho_{fl}}{W_a - W_{fl}}$$

假设有一个不规则固体，经实验测量其在空气中的重量为10牛顿，其在水中的重量为7.5牛顿。水的密度为1000千克/立方米。根据上述公式编程计算该固体的密度。

2. 输入输出描述

输入数据是固体在空气中和水中的重量、水的密度，输出数据是该固体的密度。

3. 流程图和源代码

根据问题要求，利用密度计算公式计算固体密度的流程图如图3-4所示，其对应的源程序代码如下：

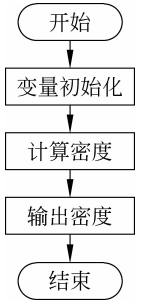

图3-4　程序流程图

源程序代码：

```
/* Filename:      ex1_3.c
 * Author:        《程序设计基础(c)》课程组
 * Discription:   根据阿基米得公式计算不规则固体的密度 */
#include<stdio.h>
```

```
void main()
{
    const int waterDensity=1000;                          /*水的密度*/
    float airWeight=10.0f;                                /*固体在空气中的重量*/
    float liquidWeight=7.5f;                              /*固体在液体中的重量*/
    float density;                                        /*固体自身密度*/
    density=airWeight*waterDensity/(airWeight-liquidWeight);   /*计算密度*/
    printf("The density of this material is %f\n",density);
}
```

4. 对程序进行测试验证(略)

3.14.2 超市收银系统

问题：根据消费的优惠折扣计算实际消费金额。

1. 问题陈述

超市消费往往伴随着各种各样的优惠促销活动，假设今日为某超市的周年庆促销活动，所有消费一律享受 9 折优惠，试编写程序，在商品的单价和数量基础上计算消费金额及折后的实际消费额。

2. 输入输出描述

输入数据是商品的单价及数量。输出数据为折扣优惠前后的消费金额。

3. 流程图和源代码

根据问题要求，计算实际消费的流程图如图 3-5 所示，其对应的源程序代码如下：

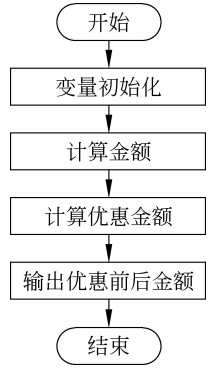

图 3-5 程序流程图

源程序代码：

```
/*Filename:      ex2_3.c
*Author:        《程序设计基础(c)》课程组
*Discription:   已知商品单价及数量,输出金额及折后金额*/
#include<stdio.h>                          /*引用头文件*/
void main()
{
    int n=10;                              /*商品数量*/
    float unitprice=20,sumprice,discounted;
    sumprice=unitprice*n;                  /*计算金额*/
    discounted=sumprice*0.9;               /*计算折后的金额*/
    printf("The total amount is %f\n",sumprice);
```

```
    printf("The discount is %f\n",discounted);
    printf("Your comsumption saves %f for you",sumprice-discounted);
}
```

4. 对程序进行测试验证(略)

练　习　题

1. ++arg 和 arg++有什么区别?

2. 写出求两个数中最小值的表达式。

3. 对于任意一个无符号整数,如何判断其第 1 位和第 4 位的二进制数位是否为 1?

4. 设 a=1,分析下列表达式的值。

(1) !a‖a　　(2) a&&!a　　(3) !a‖(a&&1)　　(4) a&&(!a‖1)

5. 假设有 int m,n=15,x=21,y=13,z=4;下面的计算结果分别是多少?

(1) m=(n<x)?n:x　　(2) m=(m<y)?m:y

(3) m=(m<z)?m:z

6. 设 x、y、z 均为 int 类型变量,请用 C 语言的表达式描述以下内容。

(1) x 或 y 中有一个小于 z;　　(2) x、y 和 z 中有两个为负数;

(3) x 是偶数

7. 假设程序中包含下列声明:

```
short int a=2;
int b=-10;
long int c=50;
float x=3.1
double y=3.99
char ch='\10';
```

请给出下列表达式的值和类型:

(1) ch*3　(2) x/ch　(3) x-y　(4) a+c　(5) y/a　(6) (int)y

(7) ch='a'

8. 假设有:

```
int a=2,b=5;
float x=7.5,y=2.4,z;
```

分析 z=(float)(a+b)/2+(int)x%(int)y+(int)x/a;的计算过程。

9. 设有 int n=6;计算表达式 n%=n+=n-=n*n 的结果。

10. 设有 int a=4,b=8,c;计算表达式 c=(b==a)&&(a+b!=20)的结果。

第4章

输 入 输 出

4.1 概　　述

在程序的执行过程中，经常需要由用户输入数据，程序依据这些数据进行计算处理，并将处理结果返回给用户，实现人与计算机的交互功能。因此，程序设计中输入输出不可或缺。

从计算机输入设备（如键盘、鼠标等）将数据送入计算机内部的操作称为输入，而将数据从计算机内部送给计算机外部设备（如显示器、打印机等）的操作称为输出。例如，用户通过键盘输入两个数，程序处理后将其中较大的数据输出到显示器上。由于输入和输出涉及应用程序与外部设备的交互，实现较为复杂，程序设计语言中通常会提供相应的函数完成输入输出工作。

C 语言的标准输入输出函数库中提供了多种输入输出函数。其中，最常用的输入输出函数包括 scanf/printf（格式输入/格式输出）、getchar/putchar（字符输入/字符输出）、gets/puts（字符串输入/字符串输出）。调用标准输入输出库函数时，需要在源文件开始处使用预编译指令：

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

或

```
#include"stdio.h"
```

4.2 printf 函数

printf 函数称为格式化输出函数，其功能是按用户指定的格式将指定的数据项输出到标准输出设备（显示器）上。

4.2.1 printf 函数的调用格式

printf 函数的调用格式：

```
printf("格式控制字符串",输出项列表);
```

其中：

(1) 由双引号""括起来的格式控制字符串用以指定数据的输出格式，由格式控制字符(包括转换说明符、标志、域宽、精度)和普通字符组成。其中，转换说明符和百分号(%)一起使用，用以说明输出数据的数据类型，普通字符原样输出，标志、宽度和精度为可选项。

(2) 输出项列表指出输出数据，当有多个输出项时，各输出项之间用逗号(,)分隔。输出项可以是常量、变量和表达式。

例如：

```
printf("%d,%f\n",a,x+1);
```

输出项必须与格式控制字符在类型和数量上完全对应。

当 printf 函数没有输出项，格式控制字符串中只有普通字符时，函数完成的功能是将双引号中的字符串输出。

例 4-1：输出字符串 hello C program!。

```
#include<stdio.h>
void main()
{
   printf("hello C program!");
}
```

在显示器上输出结果为：

```
hello C program!
```

printf 函数不会自动换行，如果想将 hello C program! 分行输出，则需要引入转义字符\n，或者多次调用 printf 函数分段输出。例如：

```
void main()
{
  printf("hello\n C program!");
}
```

或：

```
void main()
{
  printf("hello\n");
  printf("C program!");
}
```

其中的转义字符\n 作为输出控制字符，它的作用是使计算机执行 printf 函数时，换新一行输出。所以显示器上出现了两行字符。如果在字符串中忘记了\n，即使是多次调用 printf 函数，输出结果也将没有换行。例如：

```
#include<stdio.h>
void main()
{
    printf("hello");
    printf("C program!");
}
```