第3章 数据的运算与输入输出

数据和操作是构成程序的两个要素。第2章介绍了如何描述数据，本章主要介绍C语言程序中对数据的基本操作，即对数据的运算及输入输出。

3.1 运算符和表达式概述

运算(即操作)是对数据的加工，被运算的对象——数据称为运算量或操作数。一个表达式包含一个或多个操作，操作的对象称为操作数，而操作本身是通过运算符(也称操作符)体现的。例如a、a－b、k＝1等都是表达式。一个表达式完成一个或多个运算，最终得到一个结果。其表现形式多种多样，最简单的表达式是只含一个常量或一个变量，即只含一个操作数而不含运算符。

C语言提供有多种运算符，可以构成多种表达式，主要有算术表达式、赋值表达式、关系表达式、逻辑表达式、条件表达式和逗号表达式。C语言运算符按其功能分为算术运算符、赋值运算符、关系运算符、逻辑运算符、逗号运算符、位运算符等。运算符按其参加运算的操作数的个数分为一目运算符、二目运算符和三目运算符。

本章主要学习下面7种运算符及其构成的表达式。

- 算术运算符：加(＋)、减(－)、乘(＊)、除(/)、求余(或称模运算，％)、自增(＋＋)、自减(－－)共7种。
- 赋值运算符：分为简单赋值(＝)和复合赋值运算两大类。复合赋值运算又分为复合算术赋值(＋＝，－＝，＊＝，/＝，％＝)和复合位运算赋值(&＝，｜＝，^＝，＞＞＝，＜＜＝)两类共10种。
- 关系运算符：大于(＞)、小于(＜)、等于(＝＝)、大于或等于(＞＝)、小于或等于(＜＝)和不等于(！＝)6种。
- 逻辑运算符：与(&&)、或(‖)、非(！)。
- 条件运算符：这是一个三目运算符，用于条件求值(？：)。
- 逗号运算符：用于把若干表达式组合成一个表达式(，)。
- 位操作运算符：参与运算的量，按二进制位进行运算。包括位与(&)、位或(｜)、位非(～)、位异或(^)、左移(＜＜)、右移(＞＞)6种。

3.2 算术运算符和算术表达式

算术运算符和算术表达式跟以前接触过的算术运算类似，本节主要介绍算术运算符和

算术表达式。

3.2.1　算术运算符

1. 基本算术运算符

C 语言算术运算符分为一目运算符和二目运算符。一目运算符是对单个变量进行操作的运算符，常见的一目运算符有正号运算符(＋)、负号运算符(－)、自增运算符(＋＋)和自减运算符(－－)。二目运算符就是在运算中有两个变量或常量的运算符，常见的二目运算符有加法(＋)、减法(－)、乘法(＊)、除法(/)和求余(％)，这 5 种二目算符也是 C 语言基本算术算符。

说明：

(1) 对于加法(＋)、减法(－)和乘法(＊)运算的操作，可以是整型或实型的常量、变量，其运算规则与一般的数学运算规则相同。

(2) 对于除法(/)运算，其操作数可以是整型或实型的常量、变量。当 x/y 中如果两个操作数中有一个是实型，运算结果为实型；若两个操作数都是整型，运算结果为整型(舍去小数部分)。如：5/2.0＝2.5000000，5/2＝2。

(3) 对于求余％运算，用于计算两个数相除后得到的余数，当被除数可以被除数整除时，余数为零。适用于求余运算的两个操作数必须为整型，不能对实型的常量和变量进行求余运算。如：5％2＝1，6％2＝0，7％4＝3。

【例 3-1】 将华氏温度换算成摄氏温度。

思路分析：华氏温度(F)与摄氏温度(C)的换算公式为 C＝5/9×(F－32)。如果按照整型数计算，公式中 5/9 计算结果为零，所以该公式中常量和变量应当定义为实型变量，将 5/9 写成 5.0/9.0 进行计算以保证数值换算的准确性。

```
1 #include<stdio.h>
2 int main()
3 {
4     float F=48.0, C;
5     C=5.0/9.0*(F-32.0);
6     printf("与之相匹配的摄氏温度为: %.2f\n",C);
7     return 0;
8 }
```

程序运行结果：

```
与之相匹配的摄氏温度为: 8.89
```

注意：本例中，由于需要确保运算结果的准确性，所以将变量 F 和 C 定义为浮点数 float。通过 printf 函数中的格式控制字符将输出结果保留小数点后两位(四舍五入)(关于 printf 函数和格式控制字符在本章后面章节会进行介绍)。

2. 自增、自减运算符

自增“＋＋”、自减“－－”运算符的作用是使变量的值增 1 或减 1，自增或自减运算符都

可以放在变量的前面或后面,其一般用法如下:

```
++k或--k          /*在使用k之前,先将k的值加(减)1*/
k++或k--          /*在使用k之后,再将k的值加(减)1*/
```

其中k是一个整型变量。把运算符放在操作数之前,称为前置运算符,前置运算符是对变量的值先增加1或减去1,然后参与其他运算,即先改变变量的值后使用;把运算符放在操作数之后,称为后置运算符,后置运算符则是让变量先参与其他运算,然后对变量值增加1或减去1,即先使用后改变。

说明:

(1) 自增运算符(++)、自减运算符(--)只能用于变量,不能用于常量或表达式,如3++或(a+b)++都是不合法的。因为3是常量,常量的值不能改变。(a+b)++也是不合法的。即使假设a+b的自增实现,那么得到的值将没有地方存放。

(2) ++和--运算符的结合方向是"自右向左"。如-k++,k的左面是负号运算符,右面是自增运算符,由于负号运算符和自增运算符同优先级,而结合方向为"自右至左"(右结合性),即它相当于-(k++)。如果k的原值等于3,执行printf("%d", -k++)时,先取出k的值使用,输出-k的值-3,然后使k自增为4。注意k++是先用k的原值进行运算后,再对k加1,不要认为先加完1后再加负号,输出-4,这是不对的。

【例3-2】 自增、自减运算符前置、后置形式的差异程序示例。

```
 1 #include <stdio.h>
 2 int main()
 3 {
 4     int k,x,y;
 5     k=10;
 6     x=k++;y=++k;
 7     printf("k=%d,x=%d,y=%d\n",k,x,y);
 8     k=10;
 9     x=--k;y=k--;
10     printf("k=%d,x=%d,y=%d\n",k,x,y);
11     return 0;
12 }
```

程序运行结果:

```
k=12,x=10,y=12
k=8,x=9,y=9
```

注意: k++是"先使用,后增值"。所谓"先使用"是指在表达式中按k的原值进行运算得到表达式的结果,然后才使k加1。语句"x=k++;"等价于"x=k;k=k+1;"两个语句。而++k是"先增值,后使用"。所谓"先增值"是指先使k加1,在表达式中按k改变后的值进行运算得到表达式的结果。语句"y=++k;"等价于"k=k+1;y=k;"两个语句。同理可分析语句"x=--k"和语句"x=k--;"。

3. 算术运算符的优先级及结合方向

(1) 二目运算符:乘(*)、除(/)、求余(%)的优先级相同,高于加(+)、减(-),结合方

向为“自左至右”，即先左后右。

(2) 一目运算符：负(－)、自增(＋＋)、自减(－－)的优先级相同，高于二目运算符加(＋)、减(－)、乘(＊)、除(/)、求余(％)，结合方向为“自右至左”，即先右后左。

3.2.2　算术表达式

用算术运算符和括号将运算对象(常量、变量和函数等)连接起来的、符合 C 语言语法规则的式子，称为算术表达式，例如，3＋a＊b/2－1＋c。

如何求解这个算术表达式的值？在 C 语言中规定，对表达式求值时，按运算符的优先级别，先进行一目运算符计算，再进行二目运算符计算；从运算符优先级上看，从高到低进行运算，先乘除、后加减。因此，在上面的表达式中，应先计算 a＊b/2，然后再进行加法运算和减法运算。由于表达式 a＊b/2 的运算符号优先级别相同，因此按运算符的结合方向进行运算。算术运算符的结合方向全部都是“自左至右”的，即先左后右，这样 a＊b/2 就应先算 a＊b 再和 2 相除。

根据算术运算符的优先顺序和结合方向，上式运算过程为：先计算 a＊b 的值，再计算 a＊b/2 的值，然后从左到右依次计算。

【例 3-3】 程序示例。

```
1 #include <stdio.h>
2 int main()
3 {
4     int a=3,b=8,c=2,d;
5     d=3+a*b/2-1+c;
6     printf("d=%d\n",d);
7     return 0;
8 }
```

程序运行结果：

```
d=16
```

3.3　赋值运算符和赋值表达式

本节主要介绍赋值运算符和赋值表达式。C 语言采用赋值运算的方式改变变量的值，或者说为变量赋值。

3.3.1　赋值运算符

1. 赋值运算符

赋值号(＝)就是赋值运算符。赋值运算符的基本形式是将右边表达式的值直接赋给左边的变量，赋值号左边是单个的变量，右边是表达式，如：a＝3，c＝a＋b 和 r＝x％y。复合赋值运算符的形式有＋＝，－＝，＊＝，/＝和％＝五类。这些符合赋值形式代表先将两个变量进行相应的计算，然后再将计算结果赋给赋值运算符左边的变量，如 a＋＝3，表示 a＝a＋

3,b * =4,表示 b=b * 4。

2. 赋值运算符的优先级及结合方向

赋值运算符(包括下面将要讲的复合赋值运算符)的优先级低于算术运算符的优先级,结合方向是“从右至左”进行运算。

3.3.2 赋值表达式

由赋值运算符将一个变量和一个表达式连接起来的式子称为赋值表达式。它的一般形式为:

<变量><赋值运算符><表达式>

例如:

```
a=5
```

是一个赋值表达式。对赋值表达式的求解过程是:先计算赋值运算符右侧的“表达式”的值,再将该值赋给左侧的变量,赋值表达式的值就是被赋值的变量的值。

在赋值表达式的一般形式中,表达式仍可以是一个赋值表达式,也就是说,赋值表达式是可以嵌套的。例如:

```
x=(y=8)
```

括号内的表达式也是一个赋值表达式。其运算过程为:先把常量 8 赋给变量 y,赋值表达式 y=8 的值为 8,再将这个表达式的值赋给变量 x,因此运算结果 x 和 y 的值都是 8,整个赋值表达式的值也是 8。

C 语言规定,赋值运算的结合方向是“自右至左”,即整个计算过程从右至左进行。因此表达式 x=(y=8)中的括号可以省略,写成 x=y=8。在计算的过程中“从右至左”先将 8 赋值给 y,然后 y 的值再赋给 x。

下面再看几个赋值表达式的例子:

```
a=b=c=5;          /* 整个表达式的值为 5,a,b,c 的值也为 5 */
a=5+(c=6);        /* 整个表达式的值为 11,a 的值也为 11,c 的值为 6 */
x=(y=4)/(z=3);    /* 整个表达式的值为整数 1,x 的值为 1,y 值为 4,z 的值为 3 */
a=5+c=6;          /* 这个表达式不合法,因为赋值表达式左值应该为单个变量 */
```

在 C 语言中有一个术语“左值”,它标识程序中占用存储单元的实体,其原意是可以出现在赋值号的左侧,如变量名就是一个“左值”,它的值是可以改变的。例如,上面例子中的最后一个赋值表达式,表达式“5+c”不是“左值”,所以不能出现在赋值号的左侧。另外,a=3+5,a 是变量名,它的值是可以改变的,它是“左值”,赋值运算符右侧的表达式“3+5”不能作为“左值”,写成下面这样是错误的:3+5=a。并不是所有的数据对象都能作为“左值”的,变量名、数组元素名、指针指向的单元可以作为“左值”,能出现在赋值运算符的左侧。

3.3.3 复合的赋值运算符

赋值运算符有基本赋值运算符和复合赋值运算符两种形式。在 C 语言中,可以在赋值

运算符之前加上其他运算符，构成复合的赋值运算符，包括复合算术赋值运算符(+=，-=，*=，/=，%=)和复合位运算赋值运算符(&=，|=，^=，>>=，<<=)。本节主要介绍复合算术赋值运算符。

例如：

```
x+=3;          等价于  x=x+3;
y*=y+z;        等价于  y=y*(y+z);
x%=3;          等价于  x=x%3;
```

如果赋值运算符的右边是包含若干项的表达式，则相当于它有括号。如：

```
y*=y+z
```

正确的含义是 y=y*(y+z)(不要错写成 y=y*y+z)。

C 语言规定，复合的赋值运算符的结合方向也是“自右至左”，即从右至左进行运算。例如：

```
a+=a-=a*a;
```

这是一个赋值表达式。如果 a 的初值为 12，此赋值表达式的求解步骤如下：

(1) 先从右面进行“a-=a*a”的运算，它相当于 a=a-a*a=12-144=-132；

(2) 然后进行“a+=-132”的运算，相当于 a=a+(-132)=-132-132=-264。

3.4　关系运算符和关系表达式

本节主要介绍关系运算符和关系表达式。

3.4.1　关系运算符

1. 关系运算符的种类

关系运算符用来对两个操作数进行大小或相等比较的运算，其运算结果是“真”或“假”。由于 C 语言中没有逻辑类型数据，所以通常以非零表示“真”，零表示“假”。C 语言提供了 6 种关系运算符。

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

2. 关系运算符的优先级及结合方向

(1) 前 4 种关系运算符(<、<=、>、>=)的优先级别相同，后两种(==和!=)的优先级别相同。前 4 种高于后两种。

(2) 关系运算符的优先级低于算术运算符。

(3) 关系运算符的优先级高于赋值运算符。

(4) 结合方向为自左至右,即同级关系运算自左至右算。

以上的关系如图 3-1 所示。

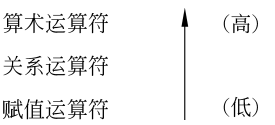

图 3-1 运算符的优先级 1

3.4.2 关系表达式

用关系运算符将两个表达式(可以是算术表达式、关系表达式、逻辑表达式、赋值表达式)连接起来的式子,称为关系表达式。

例如:

```
a>b,a+b>b+c,(a=3)>(b=5),'a'<'b',(a>b)>(b>c)
```

关系表达式的值是逻辑值“真”或“假”。C 语言中没有专门的逻辑数据类型,在求解关系表达式的值时,以 1 代表“真”,以 0 代表“假”。当关系表达式成立时,表达式的值为 1,否则,表达式的值为 0,如关系表达式“5==3”的值为 0,“5>=0”的值为 1。

例如,若有变量定义:int a=3,b=2,c=1,d,f;,则下列关系表达式:

`a>b`	其值为“真”,表达式的值为 1。
`(a>b)==c`	其值为“真”(因为 a>b 的值为 1,等于 c 的值),表达式的值为 1。
`b+c<a`	其值为“假”(因为 b+c 为 3,不小于 a),表达式的值为 0。
`d=a>b`	d 的值为 1,因为关系运算优先级高于赋值运算。从右往左是先进行关系运算 a>b,为真,表达式值为 1,再进行赋值运算,将 1 赋给 d。
`f=a>b>c`	f 的值为 0,因为关系运算优先级高于赋值运算,先进行关系运算 a>b>c。因为“>”运算符是自左至右的结合方向,先执行“a>b”得值为 1,再执行关系运算“1>c”,表达式 a>b>c 的值为 0。最后,通过赋值运算将 0 赋给 f。

3.5 逻辑运算符和逻辑表达式

本节主要介绍逻辑运算符和逻辑表达式。

3.5.1 逻辑运算符

1. 逻辑运算符

C 语言提供三种逻辑运算符:

- &&(逻辑与);
- ‖(逻辑或);
- !(逻辑非)。

其中,&& 和 ‖ 是二目运算符,它要求有两个操作数,而“!”是一目运算符,只要求有一个操作数。例如:

a&&b 的运算结果为:当 a 和 b 均为“真”时,其结果为“真”,否则为“假”。

a‖b 的运算结果为:当 a 和 b 之一为“真”时,其结果为“真”,只有二者均为“假”时,其结果才为“假”。

!a 的运算结果为：当 a 为假时，其结果为“真”，否则为“假”。

逻辑运算的值可由表 3-1 表示的逻辑运算的“真值表”来判断。它表示 a 和 b 的值为不同组合时，各种逻辑运算的计算结果。

表 3-1　逻辑运算的真值表

a	b	!a	!b	a&&b	a‖b
真	真	假	假	真	真
真	假	假	真	假	真
假	真	真	假	假	真
假	假	真	真	假	假

2. 逻辑运算符的优先级及结合方向

在一个逻辑表达式中如果包含多个逻辑运算符，如：

```
!a&&b||x>y&&c
```

按以下的优先级排序：

(1) !(非)→&&(与)→‖(或)，即“!”为三者中最高的。

(2) 逻辑运算符中的 && 和 ‖ 的优先级低于关系运算符，由于“!”是一目运算符，所以其优先级高于基本算术运算符，见图 3-2。

(3) &&、‖ 的结合方向为自左至右，即同级逻辑运算自左至右算，!(非)的结合方向为自右至左。例如，a=1，b=!!a，则从右往左执行两次“非”运算后，b 的值为 1。

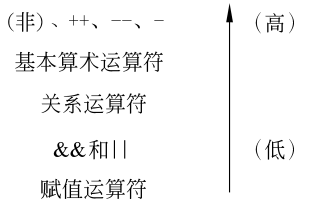

图 3-2　运算符的优先级 2

3.5.2　逻辑表达式

逻辑表达式是通过逻辑运算符将关系表达式或逻辑量连接起来所构成的表达式。逻辑表达式的值应该是“真”或“假”。C 语言中没有逻辑型数据“真”和“假”，以数值 1 代表逻辑“真”，以 0 代表逻辑“假”，来表示逻辑运算的结果。但在判断一个变量值或常量的逻辑值时，以 0 代表逻辑“假”，以非 0 代表“真”，即将一个非 0 的数值视为“真”。例如：若有 a=4，则 a 在逻辑上为真，逻辑值为 1；非 a，即!a 在逻辑上为假，逻辑值为 0。若 a=4，b=5，由于 a 和 b 数值均非 0，则两个变量的逻辑值均为 1，表达式 a&&b 的逻辑值为“真”，值为 1。逻辑运算符两侧的运算对象可以是任何类型的数据，即是字符型、实型或指针型等，系统最终以非 0 和 0 来判定它们属于“真”或“假”。

例如：

```
'c'&&'d'
```

其值为 1。由于字符 c 和字符 d 对应的 ASCII 值分别为 99 和 100，进行逻辑“与”运算后，表达式的逻辑值为“真”(查询字符的 ASCII 码值可以看书后附录 A)。

注意：*在逻辑表达式的求解中，并不是所有的逻辑运算符都被执行，只是在必须执行下*

一个逻辑运算符才能求出表达式的解时,才执行该逻辑运算符。这种现象我们称为逻辑运算中的“短路”现象。

1.“与”运算中的“短路”现象

对于逻辑表达式:

```
(表达式1)&&(表达式2)
```

根据自左至右的运算规则,首先计算表达式1的值,若该值为“真”,必须接着计算表达式2的值,才能最后确定整个表达式的值;若表达式1的值为“假”,则已经能够确定整个表达式的值必定为“假”,不再计算表达式2的值,这就是“与”运算中的“短路”现象,即无须全部完成“与”运算中所有表达式的计算就可以获得逻辑表达式的值。

例如,a的值为3,b的值为4,在进行以下逻辑运算后a的值为4,b的值不变仍为4。

```
(a++==4)&&(b++==5)
```

计算过程是:首先,判断a的值是否等于4,由于自增运算符后置,所以表达式a++==4等价于判断a是否为4。因为a的初始值为3,判断a==4结果为“假”,即表达式“a++==4”的值为0。根据“与”运算的规则,整个逻辑表达式的值必定为0,所以系统不再对逻辑运算符&&右边的表达式,即b++==5,进行计算。因此,变量a加1后值为4,右边的表达式因“短路”未进行运算,所以b没有自增,值仍为4。

2.“或”运算中的“短路”现象

对于逻辑表达式:

```
(表达式1)||(表达式2)
```

根据自左至右的运算规则,首先计算表达式1的值,若该值为“假”,必须计算出表达式2的值,才能最后确定整个表达式的值;若表达式1的值为“真”,根据“或”运算规则,可以确定整个逻辑表达式的值也为1,不再计算表达式2的值,这就是“或”运算中的“短路”现象。

例如,a的值为4,b的值为4,在进行以下逻辑运算后a的值为5,b的值不变仍为4。

```
(a++==4)||(b++==5)
```

计算过程是:首先判断a的值是否等于4,将判断结果作为表达式“a++==4”的值,然后a再自加1。由于a值为4,判断结果为“真”,即表达式“a++==4”的值为1。根据“或”运算规则,整个逻辑表达式的值必定为1,系统不再计算表达式(b++==5)。因此,变量a自加1后值为5,b因“短路”未进行自加运算,所以值仍为4。

3.6 条件运算符和条件表达式

条件运算的主要作用是依据指定的条件,在两个表达式中选择一个表达式进行计算。条件表达式由条件运算符连接构成。条件运算符是C语言提供的一个唯一的三目运算符,由两个运算符号“?”和“:”组成。条件表达式的一般形式为:

表达式 1?表达式 2：表达式 3

条件运算符的求解顺序：当表达式 1 的值为非零时，求解表达式 2 的值，此时表达式 2 的值就是整个条件表达式的值；当表达式 1 的值为零时，求解表达式 3 的值，此时整个表达式的值是表达式 3 的值。

例如：

```
c=(c>='A'&&c<='Z')?(c+32):c;
```

如果 c 是大写字母，则将 c 的 ASCII 值增加 32，转换成相应的小写字母赋给 c；否则，将保持 c 的值不变。

说明：

(1) 条件表达式可以嵌套，即一个条件表达式又可以与另一个条件表达式组成一个新的表达式。如：条件表达式 a＞b?a:c＞d?c:d。在对这种嵌套表示的条件表达式进行计算时，如果没有括号将嵌套的部分括起来，条件运算符的结合方向是自右至左，相当于计算 a＞b?a:(c＞d?c:d)。运算顺序为：若 a＞b 则整个表达式的值为 a，否则，再计算表达式 c＞d?c:d 的值，整个表达式的值为表达式 c＞d?c:d 的值。

(2) 条件运算符的优先级仅高于赋值运算符和逗号运算符，比其他的运算符都低。例如：

```
y=x>=0? x:-x;
```

先进行条件运算，求出 x 的绝对值，再进行赋值运算，将 x 的绝对值赋给 y。

【例 3-4】 求变量 a、b 的绝对值的较大者。

```
#include<stdio.h>
int main()
{
    int na,nb,a,b,max;
    a=-8;
    b=6;
    max=(na=a>=0?a:-a)>(nb=b>=0?b:-b)?na:nb;
    printf("%d\n",max);
    return 0;
}
```

程序运行结果：

```
8
```

【例 3-5】 有一个五位整数 a，若 a 的百位数大于 5，使变量 k 的值为 1，否则使变量 k 的值为 0。

思路分析：要得到整数 a 的百位数字，可以先求得 a/100 的商，再对所得的商进行求余操作，即 a/100%10，最后通过条件表达式判断百位数是否大于 5。

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

```
3 {
4     int a,k;
5     a=12300;
6     k=a/100%10>5?1:0;
7     printf("k=%d\n",k);
8     return 0;
9 }
```

程序运行结果:

```
k=0
```

3.7 逗号运算符和逗号表达式

C 语言中逗号也是一种运算符,用逗号将若干表达式连接起来就构成逗号表达式。逗号表达式的一般形式为:

表达式 1,表达式 2,表达式 3,…,表达式 *n*

逗号表达式的求解过程是:先计算表达式 1 的值,再计算表达式 2 的值……一直计算到表达式 n 的值。整个逗号表达式的值是最后一个表达式 *n* 的值。

例如:

```
i=4,j=6,k=8;      /*整个逗号表达式的值是 8*/
x=8*2,x*4;        /*先计算 x=8*2,x 被赋值为 16,整个表达式的值是 x*4 为 64*/
```

说明:

(1) 逗号表达式可以嵌套,即一个逗号表达式又可以与另一个表达式组成一个新的表达式。如:(x=8*2, x*4), x*2,先计算表达式 1,即被嵌套的逗号表达式 x=8*2, x*4,通过计算被嵌套逗号表达式中的第一个表达式 x=8*2,x 被赋值为 16,被嵌套的逗号表达式值为 x*4 为 64,然后计算表达式 2,x*2 为 32。根据逗号表达式的计算规则,整个逗号表达式的值是 32,x 的值是 16。

(2) 程序中并不是所有的逗号都要看成逗号运算符。如:printf("%d,%d,%d",x,y,z);中的逗号是作为分隔符用的,而不是逗号运算符。

(3) 逗号运算符的另一个用法是为了在一行代码语句中书写更多的内容。例如:

```
int i=4,j=6,k=8;
x=y=0,z=1;
```

(4) 逗号运算符的优先级在所有运算符中级别最低。

(5) 在计算过程中,注意需要求解的是整个逗号表达式的值,还是某一个变量的值。例如:

```
int i, a;
i=(a=2*3, a*4), a+6;
```

逗号表达式从左到右进行计算,表达式 i=(a=2*3, a*4)中嵌套了逗号表达式 a=

2＊3，a＊4，计算得到变量 a 的值为 6，被嵌套的逗号表达式的值为 a＊4＝24，通过 i＝(a＝2＊3，a＊4)的赋值，变量 i 的值为 24，而整个逗号表达式的值为表达式 a＋6 的值，即 6＋6＝12。

3.8　位运算符和位运算表达式

字节是计算机内存的最小可存取单位。尽管如此，有时需要处理的数据对象可能更小，例如，需要处理二进制位。在一个二进制位中，可以存储二进制值 0 或 1，在一个字节中，有 8 个二进制位。提供对位的操作是 C 语言突出的特点之一。由位运算符构成的表达式就是位运算表达式。表 3-2 所列出的是 C 语言所支持的位运算符。

表 3-2　位运算符

运算符	含　义	运算符	含　义
&	按位与	～	取反
|	按位或	＜＜	左移
^	按位异或	＞＞	右移

3.8.1　按位取反运算符(～)

按位“取反”的一般形式为：

```
~a
```

其中，a 为整型或字符型数据，可以转换成二进制数。

运算规则：～是一目运算符，用来对一个二进制数按位取反。即将 0 变 1，1 变 0。

例如：将八进制数 25 取反：～025

运算过程：

$$\begin{array}{r} (\sim)00010101 \\ \hline 11101010 \end{array}$$

即～025＝11101010。

3.8.2　按位“与”、按位“或”、按位“异或”运算

本节主要介绍按位“与”、按位“或”、按位“异或”运算符的构成及其运算规则。

1. 按位“与”运算符(&)

按位与的一般形式为：

```
a&b
```

其中 a、b 均为整型或字符型数据。

运算规则：参加运算的两个数据，每一位二进制数(包括符号位)进行“与”运算。如果两个对应位的二进位都为 1，则该位的结果值为 1，否则为 0。即：0&0＝0，0&1＝0，1&0＝

0,1&1=1。

例如,将十进制数 3 与十进制数 5 进行按位与运算:3&5。

运算过程:

```
    00000011
(&) 00000101
------------
    00000001
```

即 3&5=1。

2. 按位“或”运算符(|)

按位“或”的一般形式为:

```
a|b
```

其中,a、b 均为整型或字符型数据,可以转换成二进制数。

运算规则:参加运算的两个数据,每一位二进制数(包括符号位)进行“或”运算。如果对应的二进制位只要有一个为 1,该位的结果值就为 1。即:0|0=0,0|1=1,1|0=1,1|1=1。

例如:将八进制数 60 与八进制 17 进行按位或运算:060|017。

运算过程:

```
    00110000
(|) 00001111
------------
    00111111
```

即 060|017=00111111。

3. 按位“异或”运算符(^)

按位“异或”的一般形式为:

```
a^b
```

其中 a、b 均为整型或字符型数据,可以转换成二进制数。

运算规则:每一位二进制数(包括符号位)均参与运算。若对应的二进制位同为 1 或同为 0,则结果为 0。对应的二进制位值不同时,结果为 1。即:0^0=0,0^1=1,1^0=1,1^1=0。

例如:将八进制数 71 与八进制 52 进行按位异或运算:071^052。

运算过程:

```
    00111001
(^)00101010
------------
    00010011
```

即 071^052=023。

3.8.3 移位运算

下面主要介绍移位运算符的构成及其运算规则。

1. 左移运算符(<<)

左移运算的一般形式为：

```
a<<b
```

其中，a、b 均为整型或字符型数据，a 是被左移的对象，b 为左移的位数。

运算规则：用来将数的各二进制位全部左移若干位。

例如：将 a 左移 2 位

```
a=a<<2
```

将 a 的二进制位全部左移 2 位，右补 0。若 a＝15，即二进制数 00001111，左移两位得 00111100，即结果为十进制数 60。左移一个二进制位，相当于乘以 2 操作。左移 n 个二进制位，相当于乘以 2^n 操作。

左移运算有溢出问题。整数的最高位是符号位，当左移一位，若符号不变，则相当于乘以 2 操作，但当符号位变化时，就会溢出。例如：

```
char a=127, x;
x=a<<2;
```

运算后 x 的值为 1111 1100，表示的十进制数是－4，此时即发生了溢出。溢出后，变量所表示的数必须根据它在内存中的存储形式进行判断。

2. 右移运算符(>>)

右移运算的一般形式为：

```
a>>b
```

其中，a、b 均为整型或字符型数据，a 是被右移的对象，b 为右移的位数。

运算规则：用来将数的各二进制位全部右移若干位。

例如：将 a 右移 2 位

```
a=a>>2
```

将 a 的二进制位全部右移 2 位，移出的低 n 位舍弃。若 a 是有符号的整型数，高位补符号位，即若符号位为 0，则补 0，若符号位为 1，则补 1。若 a 是无符号的整型数，则高位补 0。

例如：

```
char a=-4, b=4, x, y;
x=a >>2;
y=b >>2;
a: 1111 1100 -->x: 1111 1111
b: 0000 0100 -->y: 0000 0001
```

x 的值是－1，y 的值是 1。右移 1 位相当于该数除以 2。

3.8.4　位运算符的优先级及结合方向

位运算符的优先级及结合方向为：

(1) 位运算符(~,<<,>>,&,^,|)的优先级由高到低依次为:~→<< ,>>→&→^→|,~运算符的结合方向自右至左,其他的结合方向自左至右。

(2) 按位取反运算符(~)是一目运算符,优先级与!(非)、++、--、-(负)相同。左移运算符(<<)、右移运算符(>>)的优先级相同,优先级低于算术运算符,高于关系运算符。按位"与"、按位"或"、按"位异"运算符的优先级低于关系运算符,高于逻辑运算符。

注意:

(1) 当不同长度的数值进行位运算时,右端对齐,左端补 0。

(2) 如果参加位运算的是负数,则以补码形式按位进行位运算。

3.9 数据类型的转换

C 语言允许整型、实型和字符型数据进行混合运算。不同类型的数据进行混合运算时要考虑以下问题:

- 运算符的优先级。
- 运算符的结合方向。
- 数据类型转换。

在计算一个合法的 C 语言表达式时,先要将表达式中不同类型的数据转换为同一类型,然后再进行运算。C 语言数据类型转换可归纳为两种方式:自动转换和强制类型转换。

3.9.1 自动转换规则

当要对不同类型的数据进行操作或混合运算时,应首先将其转换成相同的数据类型,然后进行操作或求值。通常情况下,转换过程是由编译程序自动进行的,类型之间的转换遵守如下自动转换的规则。

(1) 不同数据类型的数据在赋值时的类型转换规则是"就左不就右",即将赋值运算符右边表达式的数据转换成左边变量的数据类型,然后进行赋值。例如:

```
int a:
float b;
b=2/3;          /*2/3为0,将0转化为0.0赋给b*/
a=5.0/2.0;      /*5.0/2.0的值为2.5,将2.5转化为2赋给a*/
```

实型数据赋给整型变量时,舍弃小数部分,将整数部分赋给整型变量,不进行四舍五入。例如:int i=2.78,则 i 得到的值为 2。整型数据赋给实型变量时,数值不变,有效数字位数增加。例如:float f=26,则 f 的值为 26.000000。

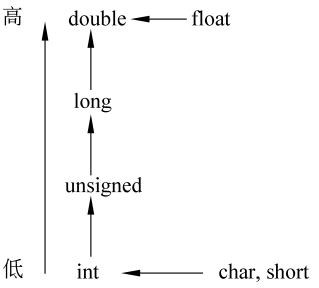

图 3-3 运算符的转换次序

(2) 同一表达式中各数据的类型不同,编译程序会自动将不同数据类型转变成同一类型后再进行运算。转换规则如图 3-3 所示。

图 3-3 中横向向左的箭头表示必定的转换,如 char 数据必定先转换为整数,short 型必定先转换为 int 型,float 型数据在运算时一律先转换成 double 型,以提高运算精度(即使

是两个 float 型数据相加，也都先转换成 double 型，然后再进行相加）。

纵向的箭头表示当运算对象为不同类型时转换的方向。例如 int 型与 double 型数据进行运算时，先将 int 型的数据直接转换成 double 型，然后在两个同类型（double 型）数据间进行运算，结果为 double 型。注意，竖向向上的箭头只表示数据类型级别的高低，由低向高转换，但不代表要进行逐步转换，不要理解为 int 型先转换成 unsigned 型，再转成 long 型，再转成 double 型。例如：已定义 i 为整型变量，f 为 float 型变量，d 为 double 型变量，e 为 long 型变量，则下面表达式：

```
10+'a'+i*f-d/e
```

计算机自动处理后的运算次序为：

（1）进行 10＋'a'的运算，先将 char 型'a'转换成 int 型 97，运算结果为 int 型 107；

（2）进行 i * f 的运算，先将 int 型 i 与 float 型 f 都转成 double 型，运算结果为 double 型；

（3）整型 107 与 doube 型 i * f 的积相加，先将 int 型数 107 转换成双精度数（小数点后加若干个 0，即 107.000…00），结果为 double 型；

（4）将 long 型变量 e 化成 double 型，d/e 结果为 double 型；

（5）将 10＋'a'＋i * f 的结果与＋d/e 的商相减，结果为 double 型。

上述的类型转换是由编译系统自动进行的。比较容易记忆的方式是：自动转换总是由少字节类型转换成多字节类型。

如果希望将多字节类型转换为少字节类型就不能进行自动转换，而必须通过强制转换来完成，需要注意的是，这样转换可能会造成数据丢失。

3.9.2　强制转换

在表达式的运算中，若要违反自动类型转换规则，则可以采用强制类型转换。强制类型转换算符的一般形式为：

(类型)(表达式)

通过一元运算符()将“表达式”的值强制转换成为“类型”所说明的数据类型。例如(int)6.000 就是把浮点常量强制转换成整型常量，结果为 6。而(int)6.832 同样把浮点常量强制转换成整型常量，结果也是 6。但是需要注意的是，前面的例子中强制转换方式并没有对数据的值产生影响；而在后一个例子中小数点后面的数据“.832”由于从浮点数转换到整型的时候丢失了。

注意：需要进行类型转换的表达式应该用括号括起来，否则将有不同的含义。例如：

```
(int)(x+y)          /*将 x +y 的值转换成为整数*/
(int)x+y            /*只将 x 转换成整数,然后与 y 相加*/
```

有时运算表达式必须借助强制类型转换运算否则不能实现目的，如%要求两侧均为整型量，若 x 为 float 型，则“x%3”不合法，必须用“(int)x%3”。强制类型转换运算的优先级高于%运算符的优先级，因此先进行(int)x 的运算，得到一个整型的中间变量，然后再对 3 求余。

3.10 数据的输入输出

把数据从计算机内部送到计算机的外设上的操作称为“输出”。例如,将计算机运算的结果显示在屏幕上或送到打印机上或者是送到磁盘上保存下来。相反,从计算机外部设备将数据送入计算机内部的操作称为“输入”。

输入输出是对数据的一种重要操作。没有输出的程序是没有用的,没有输入的程序是缺乏灵活性的,因为程序在多次运行时,用到的数据可能是不同的。在程序运行时,由用户临时输入所需的数据,可以提高程序的通用性和灵活性。

C 语言本身不像其他高级语言一样有输入和输出语句,其输入和输出是由标准的输入输出函数完成的。在使用系统库函数时,要用预编译命令 # include 将有关的“头文件”包含到用户源程序中。在头文件中包含了调用函数时所需的有关信息。在使用标准输入输出库函数时,要用到 stdio.h 文件中提供的信息。因此要使用标准输入输出函数时,一般应在程序开头有以下预编译命令: # include "stdio.h" ,以便把 I/O 函数要使用的信息包含到用户的源程序中。本节介绍标准 I/O 函数库中常用的 getchar 函数、putchar 函数、printf 函数、scanf 函数。

3.10.1 字符数据的输入输出

下面介绍单个字符数据的输入输出函数。

1. 字符输出函数 putchar

putchar 函数的作用是向标准输出设备(通常是显示器或打印机)输出一个字符。其一般形式为:

```
putchar(c);
```

输出字符变量 c 的值,c 可以是字符型变量或整型变量。在使用该函数时,应在程序前使用预编译命令:

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

【例 3-6】 输出单个字符。

思路分析:举例说明 putchar 的两种用法。一种是通过使用字符变量引用,逐个输出字符;另一种是直接输出字符常量。在输出字符常量的时候,可以使用字符、转义字符和 ASCII 码值三种方法表示字符常量。

```
1 #include <stdio.h>
2 int main()
3 {
4     char a,b,c;
5     a='B'; b='O'; c='Y';
6     putchar(a); putchar(b); putchar(c );
7     putchar('\n');
```

```
 8     /* 下面使用字符、转义字符和 ASCII 码值三种方法表示字符常量 */
 9     putchar('A'); putchar('\101'); putchar(65);
10     putchar('\n');
11     return 0;
12 }
```

程序运行结果：

```
BOY
AAA
```

2. 字符数据输入函数 getchar

此函数的作用是从标准输入设备(通常是键盘)输入一个字符，其一般形式为：

getchar()

该函数没有参数，函数的作用就是从输入设备得到的字符。getchar 函数只能接收一个字符，如果想输入多个字符就要多次调用 getchar 函数。在使用该函数时，也应在程序前使用预编译命令：

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

【例 3-7】 字符输入输出函数应用示例。

```
1 #include <stdio.h>
2 int main()
3 {
4     char c;
5     c=getchar();
6     putchar(c);
7     return 0;
8 }
```

程序运行结果：

```
a<CR>(输入 a，按 Enter 键)
a (输出字符变量 c 的值'a')
```

注意：getchar 函数一次只能输入一个字符。

说明：上面带下画线的部分是要输入的数据，＜CR＞表示回车符，其他部分为程序的输出内容，本书其他章节程序运行结果格式与本例相同。

3.10.2　格式输出函数 printf

printf 函数是 C 语言中常用的输出函数。下面介绍格式输出函数 printf 的几种使用方法及格式符的构成。

1. printf 函数最简单的用法

最简单的 printf 函数调用的一般形式为：

printf(字符串常量);

例如:

```
printf("Hello world,I want to learn C program language.");
```

输出的结果是原样字符串:

```
Hello world, I want to learn C program language.
```

在字符串中也可以包含有转义字符,如'\n'、'\t'、'\r'等,这样可以对输出的字符串格式进行调整。

【例 3-8】 printf 函数字符串中包含转义字符。

```
1 #include <stdio.h>
2 int main()
3 {
4     printf("The score of my classes are:\n");
5     printf("No.\tName\tScore\n");
6     printf("1\tLiPing\t495\n");
7     printf("2\tLiuHua\t465\n");
8     return 0;
9 }
```

程序运行结果:

```
The score of my classes are:
No.   Name      Score
1     LiPing    495
2     LiuHua    465
```

在上面的输出结果中,printf 函数中包含'\t'(Tab 制表符)和'\n'(换行)两种转义字符。所以,每一个字符串输出后都有换行。在第二、三、四个 printf 函数的输出结果中,有 Tab 制表符。

2. printf 函数的格式化输出数据

(1) 格式化输出函数 printf。

格式化输出函数 printf 调用的一般形式为:

printf(格式控制,输出表列)

看如下语句:

```
printf("a=%d,b=%d",a,b);
```

其中,在圆括号中用双引号括起来的字符串"a=%d,b=%d"称为“格式控制字符串”;a、b 是输出表列的两个输出项。函数的功能是按照“格式控制字符串”中指定的格式,将输出项表列中诸项一一输出到标准输出文件中(通常指显示器)。

“格式控制字符串”包括以下两种信息:

- 为各输出项提供格式声明，由“%”和格式字符组成，如%d、%f 等。格式转换声明的作用是将要输出的数据转换为指定的格式。它总是由“%”符号开始，紧跟其后的是格式字符。当输出项为 int 类型时，系统规定用 d 作为格式字符，其形式为%d，如上面的例子；当输出项为 float 或 double 类型时，用 f 或 e 作为格式字符，其形式为%f 或%e，具体参照表 3-3。
- 需要原样输出的字符。如以上输出语句中的“a=”“，b=”都是要原样输出的字符。假若 a，b 的值分别是 4 和 4，则以上输出语句的输出结果为：a=4，b=4。

“输出表列”是需要输出的一些数据。输出表列中的各输出项要用逗号隔开，输出项可以是常量、变量或表达式。

格式声明的个数要与输出项的个数相同，使用的格式字符也要与输出项一一对应且类型匹配。如：

```
float a=4.1415;
printf("i=%d,a=%f,a*10=%e\n",2518,a,a*10);
```

运行后的输出结果为：

```
i=2518,a=4.141500,a*10=4.141500e+01
```

在以上的 printf 格式控制字符串中，“i=”“，a=”“，a * 10=”都是原样输出，而在%d、%f、%e 的位置上则输出了常量、变量及表达式的值。

(2) printf 函数中常用的格式说明。

每个格式说明都必须用%开头，以一个格式字符作为结束，在此之间可以根据需要插入“宽度说明”、左对齐符号“-”、前导零符号“0”等。

格式描述字符。允许使用的格式字符和它们的功能如表 3-3 所示。

表 3-3　printf 函数中的格式字符及其说明

格式字符	说　明
c	输出一个字符
d 或 i	输出带符号的十进制整型数
o	以八进制无符号形式输出整型数(不带前导 0)
x 或 X	以十六进制无符号形式输出整型数(无前导 0x 或 0X)。若是 x 则输出 abcdef；若是 X 则输出 ABCDEF
u	按无符号的十进制形式输出整型数
f	以带小数点的形式输出单精度和双精度数
e 或 E	以[-]m.dddddde±xx 或[-]m.ddddddE±xx 的指数形式输出。d 的个数由精度指定，隐含的精度为 6；若指定精度为 0，则小数部分不输出
g 或 G	由系统决定采用%f 格式还是采用%e 格式，以使输出的宽度最小
s	输出字符串，直到遇到空字符'\0'，或者输出由精度指定的字符

在格式说明中，在%和上述格式字符之间可以插入表 3-4 所示的几种附加符号(又称修饰符)。