第3章

基本运算与顺序结构

3.1 概　　述

借鉴数学中的运算符和表达式概念,C 语言引入运算符,并为之定义运算规则。例如使用+、-、*、/运算符实现加、减、乘、除四则运算。数学中不同类别的运算有先后顺序,C 语言同样定义了运算符的优先级,用于确定表达式中运算符的运算顺序。例如“先乘除、后加减”。同时,C 语言还规定了运算符与运算对象的结合性,例如-1/3 代表-1 除以 3。解决问题的数学计算公式则通过 C 语言的表达式实现计算,表达式由若干常量、变量、函数和运算符构成。C 语言的基本运算包括算术运算、字符运算、位运算、赋值运算、逗号运算、复数运算、关系运算、逻辑运算、条件运算、数组运算、字符串运算、指针运算以及结构体运算等。由于计算机使用整数存储字符数据,因此字符数据也支持算术运算。本章重点介绍赋值运算、算术运算、字符运算、位运算、逗号运算、类型转换运算以及标准输入输出库函数和数学库函数的使用,其他运算参见后续相关章节。

计算机程序运行时,通过计算机输入设备(如键盘、鼠标等)获得输入信息,通过计算机输出设备(如显示器、打印机等)实现运算结果的输出。由于输入输出设备种类繁多,为保证 C 程序的可移植性,并没有定义专门的 C 语言指令实现输入和输出,而是通过调用输入输出函数(在不同设备上定义具体的函数)实现。对于文本型界面的输入输出,C 语言通过标准输入输出库函数实现键盘输入信息和屏幕输出信息。对于图形化的输入输出(例如 Windows 桌面程序和三维游戏),则通过调用 GDI、OpenGL 或 DirectX 等图形化函数库实现。

基于图灵机模型和冯·诺依曼计算原理,装载到内存的程序按照顺序自动执行。C 语言定义相应函数实现特定功能,其函数功能由构成函数的语句完成,语句执行计算、调用、控制等操作。

3.2 运算符与表达式

C 语言提供了丰富的运算符参与数据的运算处理。运算符与运算对象共同构成了表达式,运算对象又称为操作数。使用运算符时应注意以下几点。

1. 运算符的目

运算符能够连接运算对象的个数称为运算符的目，从这个角度可将运算符分为3类：

（1）单目运算符，只能连接一个运算对象，如＋＋、－－、&、[]、!等。

（2）双目运算符，可以连接两个运算对象，如 ＋、－、*、/、＞、＜、&& 等。

（3）三目运算符，可以连接3个运算对象。C语言只有条件运算符（?：）一个三目运算符。

2. 运算符的优先级

优先级是指使用不同运算符进行计算时执行的先后次序。当一个运算对象两侧的运算符优先级不同时，应遵循优先处理优先级高的运算符的原则。例如，在算术运算中，乘除运算符的优先级高于加减运算符的优先级。C语言所有运算符的优先级顺序参见附录B。

3. 运算符的结合性

结合性是指运算符与运算对象的组合规定，又称为运算符的结合方向。当一个运算对象两侧连接同一优先级的两个运算符时，需要根据运算符的结合性进行处理。C语言中运算符的结合性分为左结合性和右结合性两种。如果先结合左边的运算符，称为自左至右结合方向（左结合性），例如算术运算符、关系运算符、逻辑运算符等；如果先结合右边的运算符，称为自右至左结合方向（右结合性），例如赋值运算符、条件运算符以及所有单目运算符。

表达式是一个可以计算的算式，其计算过程按照运算符的优先级高低和结合性的方向顺序进行，同时还要考虑运算对象是否具有相同的数据类型以及是否需要类型转换。每个表达式代表一个确定的值和确定的数据类型。根据运算符，C语言的基本表达式有以下几种：

（1）单目运算表达式。由自增（＋＋）/自减（－－）、内存大小运算sizeof、取地址运算（&）、取值运算（*）、正数运算（＋）、负数运算（－）、位反运算（～）等运算符构成的计算表达式。

（2）算术运算表达式。由加（＋）、减（－）、乘（*）、除（/）和取余（%）等运算符构成的算术表达式。

（3）关系运算表达式。由小于（＜）、大于（＞）、小于或等于（＜＝）、大于或等于（＞＝）、等于（＝＝）、不等于（!＝）等运算符构成的关系表达式。

（4）逻辑运算表达式。由逻辑与（&&）、或（‖）、非（!，也是单目运算符）等运算符构成的逻辑关系表达式。

（5）位移运算表达式。由左移（＜＜）、右移（＞＞）等运算符构成的表达式。

（6）位逻辑运算表达式。由位与（&）、位或（|）、位异或（^）、位取反（～）（也是单目运算符）等运算符构成的表达式。

（7）条件表达式。由运算符（?：）构成的表达式。

（8）赋值表达式。由运算符（＝）构成的表达式。

（9）逗号表达式。由运算符（，）构成的表达式。

此外，C语言的表达式还包括由数组下标运算符[]构成的表达式、指针运算表达式以

及动态数组和动态结构运算表达式等。

当一个表达式中包含多个运算符时，计算时优先计算高级别的运算符。例如，计算表达式 x+y&&z，由于算术运算符优先级高于逻辑运算符优先级，因此先计算 x+y，再将 x+y 的计算结果与 z 进行 && 运算。如果希望 x 与 y&&z 的结果相加，则需要写成 x+(y&&z)，通过小括号提升 y&&z 运算的优先级。基本表达式一般完成简单的运算，复杂的运算可以调用函数库实现，常用的函数库包括数学库(math)、复数库(complex)、标准库(stdlib)、字符串库(string)、字符库(ctype)、宽字节字符库(wchar)等。

特殊地，一个常量、一个变量、一个函数都可以看作一个独立的表达式。例如，printf("hello!")是一个函数调用表达式。

3.3 赋值运算

C 语言中赋值运算符为“=”，赋值表达式由赋值运算符(=)连接表达式(右侧)和变量(左侧)构成。

语法格式：

```
<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。赋值运算符的优先级较低，结合方向为自右向左。例如：

(1) 表达式 a=1 表示将常量 1 赋给变量 a。

(2) 表达式 i=i+1 表示将变量 i 中的值加 1 后重新赋给变量 i。

(3) 表达式 x=a*b 表示先计算 a*b 的值，再进行赋值运算，将 a*b 结果值赋给变量 x。

(4) 表达式 a=b=c=4 相当于由 c=4、b=c 和 a=b 这 3 个赋值表达式组合而成。变量 c 的值为 4(c=4)，b 的值为变量 c 的值(4)，a 的值为变量 b 的值(4)，整个表达式的值为变量 a 的值(4)。

(5) 表达式 a=(b=1)+(c=5)，由于赋值运算优先级低于算术运算优先级，利用加括号的方式改变了运算次序。首先计算 b=1，将 1 赋值给 b，结果值为 1；其次计算 c=5，将 5 赋值给 c，结果值为 5；最后进行加法运算并将运算结果值赋给 a，a 的值为 6。

进行赋值运算时，应尽量保证赋值运算符两侧的数据类型一致，若不一致，赋值时会自动将右侧表达式的值转换为与左侧变量相同的类型。

赋值表达式中的“=”不是数学中的等号，它表示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)。例如表达式 x+5=y 是一个错误赋值表达式。

例 3-1 编写程序，将键盘输入的两个整数进行交换。

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

```
    int a,b,temp;                    /* 声明变量,a、b 存放从键盘输入的整数;temp 为临时
                                        变量,存储中间结果 */
    scanf("%d%d",&a,&b);             /* 输入两个整数 */
    temp=a;                          /* 交换,将整数 a 存放在临时变量中 */
    a=b;
    b=temp;
    printf("%d,%d\n",a,b);           /* 输出交换后的两个整数 */
    return 0;
}
```

3.4 算术运算

所有的整型数据、实型数据、字符数据都支持算术运算，进行算术运算的常用运算符包括括号、加法（＋）、减法（－）、乘法（＊）、除法（/）、取余（％）、前置自增（＋＋）/自减（－－）、后置自增（＋＋）/自减（－－），以及正运算（＋）和负运算（－）。此外，位运算中的左移（<<）与右移（>>）可以实现对 2 的乘法和除法操作，复合赋值运算同样支持算术运算。这些运算符优先级从高到低的顺序如下：

（1）括号，结合方向是自左至右，用于提高表达式的优先级。

（2）前置自增（＋＋）/自减（－－）、后置自增（＋＋）/自减（－－）、正运算（＋）、负运算（－），结合方向为由右向左。

（3）乘法（＊）、除法（/）和取余（％），结合方向为自左至右。

（4）加法（＋）、减法（－），结合方向为自左至右。

（5）左移（<<）、右移（>>），结合方向为自左至右。

（6）复合赋值运算。

3.4.1 基本算术运算

基本算术运算主要是数学意义上的计算，由算术运算符＋、－、＊、/、％和运算对象组成，运算对象可以是常量、变量或表达式。

语法格式：

<运算对象>算术运算符<运算对象>

可以利用括号提高表达式的优先级，例如，int x＝3，y＝4，z＝5;，算术表达式 x＊y＋（x＋z）/y 的计算过程如图 3-1 所示，遵循 C 语言对算术表达式计算顺序的运算符优先级规则，首先计算括号内的值，结果为 8；然后按自左向右的结合方向进行乘法和除法运算，结果分别是 12 和 2；最后进行加法运算。最终表达式计算结果为 14。

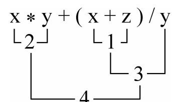

图 3-1 表达式计算顺序

例 3-2 编写程序，计算蛋白质相对分子质量。

蛋白质相对分子质量的计算公式 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。

```
#include <stdio.h>
int main()
{
    long a,m,n,mpr;
    scanf("%ld%ld%ld",&a,&n,&m);
    mpr=n*a-18*(n-m);
    printf("mpr=%ld \n ",mpr);
    return 0;
}
```

C 语言还提供了语法意义上的正运算(＋)和负运算(－)。

语法格式：

+表达式
-表达式

正运算(＋)不会改变表达式的值，运算符仅仅用于标明某整数或实数为正数，通常省略＋。例如＋100 和 100 等同。负运算(－)计算表达式的相反值。例如，int x＝200，y＝－300;，则表达式－(x＋y)的值为 100。

3.4.2 自增或自减运算

前缀自增(＋＋)/自减(－－)、后缀自增(＋＋)/自减(－－)运算可以实现整型、实型、字符型变量以及指针类型变量的自增 1 或自减 1。

语法格式：

++变量名　　(前缀)
--变量名　　(前缀)
变量名++　　(后缀)
变量名--　　(后缀)

自增/自减运算符放置于变量前(左侧)称为前缀运算。前缀运算执行先运算后使用规则，即将变量先加/减 1，然后将其结果值作为表达式值。自增/自减运算符放置于变量后(右侧)称为后缀运算。后缀运算执行先使用后运算规则，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;        /* 变量 a 的当前值 10 先加 1 变成 11 后,将新值 11 赋给 x(x 值为 11) */
x=b++;        /* 变量 b 的当前值 10 赋给 x(x 值为 10)后,变量 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，应按照运算规则从左至右取

尽可能多的符号组成运算符。例如：

```
int x=5, y=5;
y=x+++y;     /*表达式 x+++y 应理解为(x++)+y,先进行 x+y 操作,将结果 10 赋予 y,x 自
                增变为 6*/
```

例 3-3 分析程序的输出结果。

```
#include <stdio.h>
int main()
{
    int a=1,b=1,c=1;
    a=a+++b+++c++;       /*等价于 a=a+b+c;a++;b++;c++*/
    printf("%d,%d,%d",a,++b,c++);
    return 0;
}
```

变量 a、b、c 以初始化的方式被赋值为 1。语句 a＝a＋＋＋b＋＋＋c＋＋；相当于 a＝(a＋＋)＋(b＋＋)＋(c＋＋)；语句，先执行加法后再执行赋值操作，a＝a＋b＋c 完成后所有变量的值加 1。执行此语句后 a＝4、b＝2、c＝2。当执行调用 printf 输出函数语句时，变量 b 为前缀运算，其值自增 1 变为 3 后输出，变量 c 为后缀运算，输出其值 2 后自增 1 变为 3。

使用自增/自减运算符时应注意几个问题：

(1) 自增/自减运算只能作用于变量，不允许对常量和表达式进行自增/自减运算。例如，1＋＋，－－(x＋y)都是非法表达式。

(2) 当自增/自减运算独立构成一条语句时，前缀运算和后缀运算的效果相同。例如，＋＋x；等价于 x＋＋；。

(3) 如果一个表达式中对同一个变量多次进行自增/自减运算，例如语句 a＝＋＋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方式不尽相同，可能导致运算结果不一致。因此，建议自增/自减运算尽可能简单，仅仅用于单个变量的自增或自减表达式中。可将 a＝a＋＋＋b＋＋＋c＋＋；改写为 a＝a＋b＋c； a＋＋； b＋＋； c＋＋；。

3.4.3 整数运算

整数域内的所有运算的结果依然为整数。例如，3/2 的结果为 1。取余运算是两个整数相除的余数，例如，12％5 的结果为 2。取余运算仅适用于整数运算。

C 语言定义的整数类型仅仅涵盖整数域的某一子集，由于不同的数据类型规定了不同的机内表示长度，决定了对应数据量的变化范围，当某一整数超出此范围时，计算机将其截取为一个表示范围内允许的数，这种情况称为溢出处理。假设 int 型长度为 4B，表示 int 型数据使用范围应限制为－2 147 483 648～2 147 483 647，可以正确表示－15、0、2 147 483 647 等数据，当表示整数 2 147 483 650 时发生溢出错误。

例 3-4 整型数据的溢出分析。

```
int main()
{
    short int x, y;
    x=32767;
    y=x+1;
    printf("%d,%d\n",x,y);
    return 0;
}
```

程序中的 x、y 都是 short int 型变量,计算结果存储在变量 y 中,如图 3-2 所示,按照计算机对整数的编码,最高位为 1 时表示负数,说明此数为−32 768,造成溢出错误。

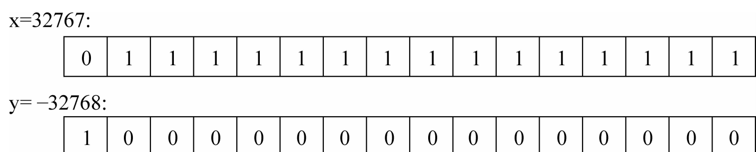

图 3-2 计算溢出

本例解决溢出的方法是将变量 y 声明为整型(int)。但 int 型同样存在溢出错误,当需要处理的数据达到 10 亿数量级时就应该十分小心。

C 语言在头文件 limits.h 中定义了整数的相关边界常量,还在 stdint.h 中补充定义了整型数据的边界常量,使用时通过预编译指令#include 包含相应的头文件。

```
#include <stdint.h>  或  #include "stdint.h"
#include <limits.h >  或  #include "limits.h"
```

常用的整数边界常量如表 3-1 所示。

表 3-1 整数边界常量

类型	最大边界常量	最大值	最小边界常量	最小值
signed char	SCHAR_MAX	+127	SCHAR_MIN	−127
unsigned char	UCHAR_MAX	255		0
char	CHAR_MAX	+127	CHAR_MIN	−127
short int	SHRT_MAX	$2^{15}-1$	SHRT_MIN	$-(2^{15}-1)$
unsigned short int	USHRT_MAX	$2^{16}-1$		0
int	INT_MAX	$2^{15}-1$	INT_MIN	$-(2^{15}-1)$
unsigned int	UINT_MAX	$2^{16}-1$		0
long int	LONG_MAX	$2^{31}-1$	LONG_MIN	$-(2^{31}-1)$
unsigned long int	ULONG_MAX	$2^{32}-1$		0
long long int	LLONG_MAX	$2^{63}-1$	LLONG_MIN	$-(2^{63}-1)$
unsigned long long int	ULLONG_MAX	$2^{64}-1$		0
INTn*	INTn_MAX	$2^{n-1}-1$	INTn_MIN	$-(2^{n}-1)$
UINTn	UINTn_MAX	2^{n-1}		0

续表

类型	最大边界常量	最大值	最小边界常量	最小值
INT_LEASTn	INT_LEASTn_MAX	$2^{n-1}-1$	INT_LEASTn_MIN	$-(2^n-1)$
UINT_LEASTn	UINT_LEASTn_MAX	2^{n-1}		0
	INTMAX_MAX	$2^{63}-1$	INTMAX_MIN	$-(2^{63}-1)$

* n 的取值为 8、16、32 或 64,余同。

3.4.4 实数运算

实数类型可以表示小数。由于实数只能存储有限数字位,因此实数运算时通常存在舍入误差。实数的精度取决于尾数部分的位数,尾数部分的位数越多,能够表示的有效数字位越多。例如,单精度数的尾数用 23 位存储,加上默认的小数点前 1 位,$2^{23+1}=16\ 777\ 216$,因为 $10^7<16\ 777\ 216<10^8$,所以单精度浮点数的有效位数是 7 位;双精度的尾数用 52 位存储,$2^{52+1}=9\ 007\ 199\ 254\ 740\ 992$,因为 $10^{16}<9\ 007\ 199\ 254\ 740\ 992<10^{17}$,所以双精度的有效位数为 16 位。

例 3-5 分析程序中实数的舍入误差。

```
#include<float.h>
int main()
{
    float a,b;
    double c,d;
    a=123456.789e5;
    b=a+20;
    c=123456.789e5;
    d=c+20;
    printf("%f\n%f\n",a,b);
    printf("%lf\n%lf\n",c,d);
    return 0;
}
```

运行结果如下:

```
12345678848.000000
12345678848.000000
12345678900.000000
12345678920.000000
```

程序中的 a 和 b 为单精度浮点型,有效位数只有 7 位,其余位相当于无效位,超出有效位数的加减运算没有实际意义;c 和 d 是双精度型,有效位为 16 位,双精度型相对单精度型更接近原数据。

C 语言在头文件 float.h 中定义了各种实数边界常量,常用的符号常量如表 3-2 所示。

表 3-2 实数边界常量

类型	最大边界常量	最大值	最小边界常量	最小值
float	FLT_MAX	3.40282347E+38F	FLT_MIN	1.17549435E−38F
double	DBL_MAX	1.7976931348623157E+308	DBL_MIN	2.2250738585072014E−308

使用时通过预编译指令＃include 包含头文件 float.h。

```
#include<float.h>   或   #include "float.h"
```

3.4.5 复合赋值运算

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。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。复合赋值算术运算符包括＋＝、－＝、*＝、/＝、%＝等。利用复合赋值运算符将一个变量和一个表达式连接起来构成复合赋值表达式。

语法格式：

<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)*/
```

3.5 字 符 运 算

无论表现形式是 ASCII 码字符、宽字节字符还是统一编码字符,字符的本质是使用一个整数标识一个字符。

3.5.1 算术运算

字符数据作为一个整数,可以参与所有整型数据的运算。例如：

```
char c='A'; int n=0;
n=c*100;                  /*c 作为整数参与 c*100 运算*/
printf("\n%d",n);         /*输出 6500*/
c++;                      /*参与 c++运算*/
printf("\n%c",c);         /*输出 B*/
```

需要注意字符数据参与运算后可能超出其使用范围而产生溢出错误。例如：

```
c=c*100;                  /*结果值为 6500,超出 ASCII 码范围*/
```

例 3-6　编写程序，读入一个数字字符，将其转换为对应的整数(例如将字符'1'转换为整数 1)。

```
#include <stdio.h>
int main()                      /*利用字符'1'和字符'0'的 ASCII 码值相差 1 的特点,将
                                  两个字符相减获得整数 1*/
{
    char c='A';
    int n=0;
    c=getchar();                /*读入数字字符*/
    n=c-'0';
    printf("%d\n",n);           /*输出对应整数*/
    return 0;
}
```

3.5.2　字符分类

C 语言的字符处理函数库中提供了实现判定当前字符为何种字符的库函数。有关 ASCII 码字符的分类函数如表 3-3 所示。使用时通过预编译指令 #include 包含头文件 ctype.h：

```
#include <ctype.h>   或   #include "ctype.h"
```

表 3-3　ASCII 字符分类函数

函　数　名	用　　途	函　数　名	用　　途
int isalnum(int c)	是否为英文字母及数字字符	int islower(int c)	是否为小写字母
int isalpha(int c)	是否为英文字母	int isprint(int c)	是否为可打印字符
int isblank(int c)	是否为空白字符	int ispunct(int c)	是否为除了英文字母、数字和空格外的可打印字符
int iscntrl(int c)	是否为控制字符	int isspace(int c)	是否为空格
int isdigit(int c)	是否为数字字符	int isupper(int c)	是否为大写字母
int isgraph(int c)	是否为除空格外的可打印字符	int isxdigit(int c)	是否为十六进制字符

例 3-7　编写程序，如果读入的字符是英文字母或数字字符，输出数字 1，否则输出数字 0。

```
#include <stdio.h>
#include <ctype.h>
int main()
{
    char c='A';
    int n=0;
```

```
    c=getchar();
    printf("%d\n",isalnum(c));  /* 如果输入字符'A',则输出 1;如果输入字符'&',则输出 0 */
    return 0;
}
```

宽字节字符分类函数的功能及用法与表 3-3 所示的函数相对应,但是函数名不同,例如 isalpha 函数对应的宽字节字符分类函数为 iswalpha。使用宽字节字符分类函数时,通过预编译指令 #include 包含头文件 wctype.h。统一字符编码字符属于宽字节字符的一种,因此使用宽字节字符的分类函数。

3.5.3 字符转换

字符转换包括字符大小写之间的转换,转换过程既可以利用字符算术运算实现,也可以通过调用相关库函数实现。针对 ASCII 码字符转换的库函数为 tolower 和 toupper,对应宽字节字符函数为 towlower 和 towupper。算术运算则利用小写字母和对应大写字母之间 ASCII 码值相差 32 的特点,在原字符基础上加/减 32,获得对应字母。

例 3-8 编写程序,将读入的一个大写英文字母转换为一个小写英文字母。

```
/* 方法一:算术计算实现 */
#include <stdio.h>
int main()
{
    char c='A';
    int n=0;
    c=getchar();        /* 读入大写英文字母 */
    n=c+32;
    printf("%c\n",n);  /* 输出小写英文字母 */
    return 0;
}
```

```
/* 方法二:转换函数调用实现 */
#include <stdio.h>
int main()
{
    char c='A';
    int n=0;
    c=getchar();
    n=tolower(c);
    printf("%c\n",n);
    return 0;
}
```

宽字节字符与 ASCII 码字符的转换处理涉及字符串的处理(参见 6.6 节)。

3.6 位 运 算

位运算是 C 语言的一种特殊运算,包括逻辑运算和移位运算。位运算以单独的二进制位作为运算对象,只能对整型或字符型数据进行。

3.6.1 位逻辑运算

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,优先级介于关系运算符与逻辑运算符之

间，结合方向为自左至右；～为单目运算符，优先级高于算术运算符，结合方向为自右至左。

语法格式：

<运算对象>位逻辑运算<运算对象>　　　　(对于 &、|、^)
~<运算对象>

1. 按位与运算

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。当两个操作对象二进制数的相同位均为 1 时，结果数值的相应位为 1，否则为 0。

例如，int 型常量 4 和 7 进行按位与运算表示为 4&7，运算过程为(仅取数据的后两个字节分析，下同)：

```
        4        (0000 0000 0000 0100)
&       7        (0000 0000 0000 0111)
-------------------------------------
        4        (0000 0000 0000 0100)
```

int 型常量－4&7 的运算过程为：

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

按位与运算通常用于对一个数据的某些位清零或保留某些位。例如，保留变量 x 的低 8 位(高 8 位清零)，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

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。当两个运算对象二进制数的相同位均为 0 时，结果数值的相应位为 0，否则为 1。

例如，int 型常量 4 和 7 进行按位或运算表示为 4|7，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，数据的其余位保持不变。例如，将变量 x 的最低位置 1，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

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。当两个二进制数的相同位互不相同的时候，对应位的结果为 1，否则为 0。

例如，int 型常量 4 和 7 进行按位异或运算表示为 4^7，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些位翻转(即原来为1的位变为0,为0的位变为1),其余位不变。例如,将变量x的后4位取反,可以执行运算x&15(15的二进制数为0000 0000 0000 1111)。

4. 按位取反运算

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进行按位取反运算表示为~7,其运算过程如下:

~	7	(0000 0000 0000 0111)
	-8	(1111 1111 1111 1000)

3.6.2 位移运算

位移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,其优先级介于算术运算与关系运算之间,结合方向为自左至右。

语法格式:

<运算对象>位移运算符<移动位数>

运算对象可以是整型或字符型常量、具有确定值的变量或表达式。移动位数表示可以左移或右移的具体位数。

1. 左移运算

左移运算的功能是将<<左侧运算对象的二进制数值逐位左移若干位,左移的位数由<<右侧的数值指定。左侧被移出的位将被舍弃,右侧空出的位补0。例如:

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

由于$(a)_{10}=(5)_{10}=(0000\ 0000\ 0000\ 0101)_2$,a左移2位后的结果$(0000\ 0000\ 0001\ 0100)_2=(20)_{10}$,b的结果为$(20)_{10}$,a左移两位相当于扩大4倍:$b/a=20/5=4=2^2$。

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位相当于原数据乘以2^n。由于位移运算速度比乘法运算速度快很多,处理数据乘法运算时可以采用左移运算。

2. 右移运算

右移运算的功能是将>>左侧运算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。例如:

```
int a=5,b;
b=a>>2;   /*由于(a)10=(5)10=(0000 0000 0000 0101)2,所以b=(0000 0000 0000 0001)2=(1)10*/
```

右移同样会引起数据的变化，右移一位相当于原数据除以 2，右移 n 位则相当于原数据除以 2^n。右移运算时，如果当前的数据为无符号数，左边补 0。如果当前的数据为有符号数，符号位为 0（正数）时，左边补 0；符号位为 1（负数）时，则取决于所使用的系统：补 0 称为“逻辑右移”，补 1 称为“算术右移”。

3.6.3　复合位运算及补位原则

C 语言提供的复合位运算符包括 &=、!=、>>=、<<=和^=。按位取反不存在复合运算。

语法格式：

```
<变量>复合位运算符<表达式>
```

例如：

```
a&=0x11;              /*等价于 a=a&0x11;*/
a>>=2;                /*等价于 a=a>>2;*/
```

按照右端对齐的原则进行不同长度的数据之间的位运算，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，左端补 0 后再进行位运算。补位原则如下：

（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。

3.7　逗 号 运 算

逗号也是 C 语言的一种运算符，称为逗号运算符。逗号表达式由逗号运算符及两个以上的表达式连接而成。

语法格式：

```
<表达式 1>,<表达式 2>,…,<表达式 n>
```

逗号运算符的运算对象可以是任何类型的表达式。在所有运算符中，逗号运算符的优先级最低，结合方向为自左至右。逗号表达式的计算过程是：依次计算＜表达式 1＞，＜表达式 2＞，…，＜表达式 n＞的值，并将＜表达式 n＞的值作为整个表达式的结果值，因此逗号运算又称为顺序求值运算。例如：

```
int a=2,b,c;
float x=5.2;
b=a, 2*a, 2*x;       /*先计算 b=a,a 赋值给 b(b=2),再计算 2*a(其值为 4),最后计算
                       2*x(其值为 10.4),表达式结果为 10.4(最后一个表达式的值)*/
c=(a=10,b=5,a+b);    /*依次计算括号内的逗号表达式 a=10、b=5、10+5,并将 10+5 赋予
```

变量 c * /

多数情况下，使用逗号表达式的目的不是为了获得逗号表达式的最终结果值，而是为了按顺序分别计算每个表达式的结果值，这种计算在循环结构中经常使用。

3.8 强制类型转换

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相同类型的数据后再进行运算。C 语言提供隐式转换和显式转换两种转换方式。隐式转换不需要人工干预，由编译系统自动完成，显式转换则通过强制类型转换运算符明确转换类型。

3.8.1 算术运算中的隐式转换

当不同类型的数据混合进行算术运算时，系统自动转换为同一种数据类型后进行运算。自动转换依据“类型提升”的原则，如图 3-3 所示。

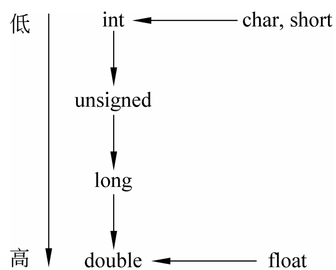

图 3-3 混合运算的数据类型转换规则

按照数据类型由低向高的方向转换，首先将较低类型的数据提升为较高的类型，使两者的数据类型一致(但数值不变)，然后进行计算，其结果是较高类型的数据，以保证不降低精度。数据类型的高低则是根据类型所占存储空间的大小和存储范围判定，占用存储空间越大表示数据的类型越高。算术运算的自动转换规则如下：

(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)自动转换成无符号型数据，整型数据(int)、无符号型数据(unsigned)与长整型数据(long)运算时自动转换成长整型数据(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)都转换成实型数据(double)。

假设 x 是 int 型，y 是 double 型，表达式 x+'a'+y 的计算过程为：首先将字符常量 a 转换为 int 型；按照算术运算规则从左向右计算 x+'a'，计算结果为整型，将 x+'a'的计算结果自动转换为 double 型，与 y 进行加法计算，表达式结果为 double 型。

3.8.2 赋值运算中的隐式转换

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

的数据将转换为左侧变量的类型。即计算出赋值运算符右侧表达式的值后，转换为左侧变量所属的类型，再赋值给左侧的变量。赋值运算的自动转换规则如下：

（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 int 型变量时，由于 char 型数据在运算时根据其 ASCII 码值自动转换为 int 型数据，因此只需将字符数据的 ASCII 码值存储到 int 型变量低 8 位中，高位补 0。将 int 型数据赋给 char 型变量时，只将其低 8 位存入 char 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 int 型变量时，将 unsigned 型变量在内存中的内容原样放入 int 型变量的内存中。将 int 型数据赋给 long int 型变量时，只将 int 型数据放入 long int 型变量的低字节中，高字节全部补充为 int 型数据的符号位。

不同类的型数据进行赋值运算时，如果右侧数据的类型高于左侧，可能造成一部分数据丢失，降低数据的精度；也可能发生数据溢出，导致结果错误。

3.8.3 显式转换

一般情况下，编译系统自动处理数据的类型转换，称为隐式转换。而利用强制类型转换运算符将某一类型的数据强制转换为另外一种类型，则称为显式转换。

语法格式：

```
(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。显式转换的目的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，则必须用括号将其括起来，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*/
```

显式转换后仅产生一个临时的、类型不同的数据继续参与运算，其常量、变量或表达

式的原有类型以及原来的数据值均不改变。例如：

```
int x=5; float y;
y=(float)x/2;     /* x 值被强制转换为实型(5.0)参与运算,运算结果为 2.5,但 x 的数据类
                     型仍为 int */
```

由于类型转换占用系统时间,过多的转换将降低程序的运行效率。因此设计程序时应尽量选择适当的数据类型,以减少不必要的类型转换。

3.9 sizeof 运算

sizeof 运算符用于获取指定数据类型所需要的存储空间大小,其优先级高于双目运算符。

语法格式：

sizeof(类型名) 或 **sizeof(变量名)**

sizeof 运算的结果是无符号整数,表示存储属于类型名或变量名的值所需要的字节数。

当不了解系统中各种数据类型所占存储单元的字节数时,可利用 sizeof 运算符来获取。例如,16 位机的 sizeof(int)的值通常为 2(int 型数据占用 2B 存储空间),32 位机的 sizeof(int)值为 4(int 型数据占用 4B 的存储空间)。sizeof 运算还适用于常量、变量或表达式。例如,float 型变量 x=5.0,其 sizeof(x)的值为 4(float 类型数据占用 4B 存储空间)。此外,sizeof 运算符也可以与其他运算符共同构成复杂表达式,例如 i * sizeof(int)。

尽管 sizeof()的写法与库函数调用写法类似,但是 sizeof 是一个运算符而不是函数。

3.10 标准设备输入输出库

计算机发展到今天,麦克风、摄像头、投影仪、体感传感器等输入输出设备层出不穷,然而键盘、鼠标和显示器仍然是最常用的设备。尽管图形化的输入输出方式已经成为主流,但是文本型的输入输出方式一直得以保留。C 语言为文本型的输入输出提供了标准输入输出函数库,将键盘命名为标准输入设备,用常量 stdin 代表,将显示器命名为标准输出设备,用常量 stdout 代表,实现从键盘输入文本信息和向显示器屏幕输出文本信息。调用标准输入输出库函数时需要在源程序的开始处使用预编译指令 #include 包含头文件 stdio.h。

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

3.10.1 字符输入输出函数

1. ASCII 码字符输入输出

getchar 函数用于单个字符输入，其功能是从标准输入设备(键盘)上读入一个且仅一个字符，并将该字符作为 getchar 函数的返回值。

调用格式：

```
getchar()
```

getchar 函数只能接收一个字符而非一串字符。如果输入"abcde"，getchar 函数也只接收第一个字符'a'。例如：

```
char ch;
ch =getchar();            /* 从键盘读入一个字符并将它赋给 ch */
```

getchar 函数得到的字符可以赋给字符变量或整型变量，也可以不赋给任何变量而作为表达式的一部分。getchar 函数不能显示输入的数据，如果希望显示该数据，必须调用相应的输出函数实现。

putchar 用于对单个字符输出，其功能是将指定表达式的值(输出项)所对应的字符输出到标准输出设备(显示器)上，每次只能输出一个字符。

调用格式：

```
putchar(输出项)
```

putchar 函数必须带输出项，输出项可以是字符型或整型常量、变量、表达式。例如：

```
char ch;
ch=getchar();
putchar(ch);
```

输出字符常量时必须用单引号括起来，如'\n'、'*'等；输出表达式值时，可以是'a'+32 等代表一个确定字符的形式，不能是表达式计算值超出字符存储范围的形式，也不能是字符串形式，'abc'或"abc"都是错误形式。

2. 宽字节字符输入输出

与 getchar 函数对应的 getwchar 函数用于读入宽字节字符，与 putchar 函数对应的 putwchar 函数用于输出宽字节字符。使用时通过预编译指令 #include 包含头文件 wchar.h。例如：

```
wchar_t ch;
ch=getwchar();           /* 读入宽字节字符 */
putwchar(ch);            /* 输出宽字节字符 */
```

3.10.2 格式化输出函数

printf 函数为 ASCII 码字符格式化输出函数，其功能是按用户指定的格式将指定的数据输出到标准输出设备上。

调用格式：

```
printf("格式控制字符串",输出项列表)
```

由双引号括起来的格式控制字符串用于指定数据的输出格式，由格式控制字符(格式转换说明符、标志、域宽、精度)和普通字符组成。格式转换说明符和百分号(%)同时使用，用以说明输出数据的数据类型，标志、宽度和精度为可选项；普通字符则原样输出。

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);    /* 双引号中的%d 和%f 为格式说明符,逗号和\n 为普通字
                                符,输出项为 a 和 x+1 */
```

输出项与格式控制字符在类型和数量上必须一一对应。例如，输出项 a 与%d 对应，表示按 int 型输出变量 a 的值；输出项 x+1 与%f 对应，表示按 float 型输出表达式 x+1 的值。

当 printf 函数没有输出项，而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。例如：

```
printf("Hello C programming!");    /* 输出字符串"Hello C programming!" */
```

printf 函数不会自动换行，如果希望"Hello C programming!"分行输出，则需要使用换行符\n。例如：

```
printf("Hello\nC programming! ");
```

或

```
printf("Hello\n");
printf("C programming! ");
```

输出结果为两行信息：

```
Hello
C programming!
```

换行符'\n'作为输出控制字符，其作用是执行 printf 函数时，其后的输出结果从新一行输出。如果字符串中忘记使用'\n'，即使多次调用 printf 函数，输出的结果也不能换行。例如：

```
printf("Hello");
printf("C programming!");
```

输出结果为一行字符：

```
Hello C programming!
```

由于双引号、单引号、反斜线等在C语言中有特殊用途，如果输出结果中需要包含这些字符，则必须使用转义字符形式输出。例如：

```
printf("\"Hello C programming! \"");
```

输出结果为：

```
"Hello C programming!"
```

当输出程序的计算结果时，需要使用由%开头，后跟若干格式转换说明符的格式控制字符串，用以说明数据输出的类型、长度、位数等。

语法格式：

```
% [修饰符] 格式转换说明符
```

修饰符为可选项，包括标志修饰符、宽度修饰符、精度修饰符、长度修饰符，用于确定输出数据的宽度、精度、对齐方式等，以产生更加规范、整齐、美观的数据输出形式。没有修饰符时，按系统默认设定输出。

1. 格式转换说明符

格式转换说明符规定了对应输出项的输出类型，即输出的数据转换为指定的格式输出。该项不能省略。常用的格式转换说明符及其含义如表3-4所示。

表3-4 格式转换说明符及其含义

格式转换说明符	含　义
c	按字符形式输出单个字符
d,i	按十进制整数形式输出带符号整数(正数不输出符号)
u	按十进制整数形式输出无符号整数
o	按八进制整数形式输出无符号整数(不输出前缀0)
x,X	按十六进制整数形式输出无符号整数(不输出前缀0x)
f,F	按十进制小数形式输出单、双精度实数
e,E	按科学记数法输出单、双精度实数，例如1.2E1
g,G	根据精度设置输出双精度实数
a,A	按照十六进制形式输出实数，格式为[－]0xh.hhhh p±d，例如9.0输出为0x9p+0
p	输出内存地址，详见第8章
s	按字符串形式输出
%	输出%

格式转换说明符必须与%结合使用，表3-4中的字符只有放在%后面才能作为输出的格式转换说明。如果%后的字符未在表3-4中列出，则输出行为不确定。例如：

```
int d=15;
```