第3章

运算符及表达式

思考题

1. C语言的算术表达式与数学中的算式是一样的吗？
2. 如何判断一个数是不是偶数？
3. 如何表示一个数比其他两个数都大？
4. 如何判断一个二进制数的次低位是1？

3.1 基本概念

计算机的基本功能就是计算，判断、智能等都是基于计算实现的。在程序中，各种计算都离不开运算符或操作符(operator)和操作数(operand)。运算符表示进行何种运算，操作数是参与运算的数据。

例如，在下面计算圆周长的程序语句中：

```
c=2*PI*r;
```

其中=、*就是运算符，c、2、PI、r就是操作数。

C语言的运算符分类如表3-1所示。

表3-1 C语言的运算符

No	运算符功能类别	运 算 符
1	算术运算符	+、-、*、/、%
2	关系运算符	>、<、==、>=、<=、!=
3	逻辑运算符	!、&&、||
4	位运算符	<<、>>、~、|、^、&
5	赋值运算符	=、+=、-=、*=、/=、%=、++、--、<<=等
6	条件运算符	?:

续表

No	运算符功能类别	运　算　符
7	逗号运算符	,
8	指针运算符	* 、&
9	求字节数运算符	sizeof
10	强制类型转换运算符	(int)、(double)、(char)等
11	分量运算符	. 、－＞
12	下标运算符	[]
13	其他	函数调用运算符()等

运算符、操作数、函数和圆括号按一定的规则顺序就构成了表达式，表达式最终会计算得到一个数据结果。然后往往赋值给一个变量，或者输出。

当程序中有多个运算符相连出现时，C 语言总是从左至右尽量多地将若干个字符组成一个运算符。如计算机会将 a＝b＋＋＋c 理解为 a＝(b＋＋)＋c 而不是 a＝b＋(＋＋c)。建议编程时加上括号以避免歧义，使程序更易读。

3.1.1　运算符分类

运算符分为单目、双目和三目运算符，表示参与进行该运算的操作数数量。绝大多数是双目运算符，少数几个单目运算符和 1 个三目运算符。

三目运算符：d1? d2：d3。由“?”和“：”共同构成，它们间隔开 3 个操作数。该运算符被称为条件运算符，根据第一个操作数 d1 的值，该运算得到不同的运算结果：如果 d1 是非 0，运算结果是操作数 d2；如果 d1 是 0，则运算结果为操作数 d3。

单目运算符有－(求相反数)、!（逻辑非）、～(位逻辑非)、＋＋(自加 1)、－－(自减 1)、sizeof(求长度)、指针运算符、强制类型转换运算符等。

运算符根据功能分为赋值运算符、算术运算符、关系运算符、逻辑运算符、位运算符等，下面将按此分类详细介绍主要的运算符。

3.1.2　运算符与数据类型

运算符对参与运算的操作数数量有要求，少数运算符对操作数的数据类型也有要求。如求余运算符(％)和位运算符，要求两个操作数都必须是整型；间接寻址单目运算符 * 要求操作数是指针类型。

多数运算符对操作数无类型要求，当一个运算符的几个操作数类型不同时，不同运算符有不同的处理：

(1) 赋值运算会将赋值运算符右边表达式的结果自动转换为赋值运算符左边的变量类型；

(2) 双目算术运算符会将操作数自动转换为整型或实型(大致是将精度低的类型转换为高的类型),参见 2.7 节;三目运算符也是进行这种转换。

(3) 其他运算符不做转换。

运算符的运算结果根据运算符的不同也具有特定的数据类型:

(1) 赋值运算结果为赋值运算符左边的变量类型;

(2) 双目算术运算结果为整型或实型(大致是两操作数中精度高的类型),单目算术运算的结果类型就是操作数的类型;

(3) 逻辑运算、关系运算和求长度运算结果的数据类型都是整型;

(4) 逗号运算的结果数据类型是最右边操作数的类型;

(5) 强制类型转换结果为要求强制转换到的类型;

(6) 指针类型 & 的结果为指针,* 的结果为指针所指类型。

3.1.3 运算符的优先级与结合性

在一个表达式中有多个不同的运算符出现时,先进行哪个运算就取决于各运算符的优先级。传统算术中就有"先乘除后加减"的运算优先级规则。括号的优先级最高,它也是一种强制改变优先级的符号。

在一个表达式中有两个相同优先级的运算符相邻出现时,先进行左边还是右边的运算符运算就取决于各运算符的结合性。如加法、减法运算符+和-就是左结合,赋值运算符=就是右结合。所以表达式 3-2+1 的结果是 2 而不会是 0(假如是右结合,就会先运行 2+1)。运行下面两个语句后,

```
int a,b=1 ;
a=b=2 ;
```

变量 a 的值将会是 2,而不是 1(假如是左结合就会先运行 a=b)。

所有运算符的优先级和结合性见附录 C。准确记住运算符的优先级和结合性对于分析程序是必须的,但对于编程又不是必须的。记住主要的运算符优先级可以使程序简洁,编程时也可以通过括号准确规定优先级。

3.2 算术运算符

算术运算符有+(加)、-(减)、*(乘)、/(除)、%(求余),它们在 C 语言中表示最基本的算术运算。乘、除、求余的优先级比加、减高,都是左结合。求余运算要求两个操作数都是整型数据。

由算术运算符连接起来组成的表达式即为算术表达式。

【注意】

(1) C 语言中没有幂运算符(乘方运算符),幂运算可通过函数 pow 来实现。

(2) C 语言编程中处理除法运算容易出错:

当两个整数进行除法运算时,若不能整除,其结果也一定为整数而舍去小数部分,采取"向零取整"的方式来舍去小数部分。例如,1/2 的结果为 0 而不是 1 或 0.5,-2/3 的结果为 0 而不是-1。

当两个数相除,若除数和被除数中有一个是实型,则进行实型数除法。所以建议编程时遇到整型常量除法,一定将一个操作数加上小数点写为实型常量,如 1./2 而不是 1/2。

比较:1/3 * 3. 的结果为 0,而 1./3 * 3 的结果近似为 1。请分析原因。

(3) 求余运算(%)结果的正负与被除数的符号相同。

例如,20%11 和 20%-11 的结果都是 9,-20%11 和-20%-11 的结果都是-9。

3.3 赋值运算符

赋值运算符就是等号(=),它是 C 语言程序中使用最多的运算符。它表示将其右边的表达式的结果赋值给其左边的变量,是一种数据传递。

赋值运算符优先级很低,仅高于简单列举的逗号运算符。

注意,赋值运算符=不是比较是否相等,比较是否相等是关系运算符==。

赋值运算符左边必须是变量或指针表示的内存空间(统称为左值操作数)。

注意理解语句

```
a=a+2 ;
```

它表示将变量 a 的值取出到 CPU,加上 2 后再赋值给变量 a,即保存回原空间。

C 语言为这一类运算专门设计了运算符,称为复合赋值运算符,就是将算术运算符或位运算符与赋值运算符结合起来,表示将左值操作数的值取出进行算术或位运算后再赋值回原变量。复合赋值运算符有+=、-=、*=、/=、%=、>>=、<<=、&=、^=、|=。

复合赋值运算符仍然是赋值运算符,优先级与赋值运算符一样。

如下面语句执行后,

```
int a=2, b=3 ;
a * =b+4 ;
```

变量 a 的值是 14,而不是 6,因为复合赋值运算 *= 的优先级比算术运算+要低。

C 语言更进一步设计了++和--表示自加 1 和减 1 运算,但它们是单目运算符,优先级要高得多。如语句

```
i++;
++a ;
b--;
--c ;
```

a＋＋与＋＋a 在单独的语句中没有区别，但混合其他运算符时有区别：＋＋a 执行 a＝a＋1，表达式的结果是执行 a＝a＋1 后的结果；a＋＋是先将 a 的值取出作为表达式的结果，同时执行 a＝a＋1。例如，下面程序段运行后

```
int a=1, b=1, c, d ;
c=a++;                                  //c 的值为 1,a 的值为 2
d=++b ;                                 //d 的值为 2,b 的值为 2
```

变量 a、b、c、d 的值分别为 2、2、1、2。

例 3.1 在下面程序执行中，分析各变量中数据的变化。

```
#include<iostream>
using namespace std;
int main(void)
{
    short a=-1;                         //a 为-1,内存中是 11111111 11111111
    unsigned short b;
    int c=1,d=5,m=3,n;
    unsigned k;
    double x;
    n=3.7;                              //n 为 3
    x=1/2*100.;                         //x 为 0.0
    x=1./2*100;                         //x 为 500.0
    n=4/3.*100;                         //n 为 133
    n=(int)a;                           //n 为-1,占 4 字节
    b=(unsigned short)a;                //b 为 65535,内存中是 11111111 11111111
    k=(unsigned)a;                      //k 为 4294967295,内存中是 ffff ffff
    m=c++,n=++d;                        //m 为 1,c 为 2,n 为 6,d 为 6
    m=5%-3,n=-5%-3;                     //m 为 2,n 为-2
    a=b;                                //a 为-1,b 为 65535,内存都是 ffff
    m=b;                                //m 为 65535,4 字节,b 为 65535,2 字节
    return 0;
}
```

例 3.1 中表达式 k＝(unsigned)a 是先将短整型 a 的 2 字节 16 位有符号整型数据进行符号位扩展到 4 字节，再按无符号数读出后赋值给无符号整型变量 k。m＝b 是先将无符号整型 b 的 2 字节 16 位无符号整型数据进行 0 扩展到 4 字节，再赋值给有符号整型变量 a。具体转换方法见 2.7 节。

变量和 const 常量可以在定义时赋值，称为初始化。const 常量不能在此之外再被赋值。

在所有变量被定义后，可以连续赋值。

例 3.2 指出下面程序中的语法错误。

```
#include<iostream>
using namespace std;
int main(void)
{
    int a=b=c=1;                    //错误,变量 b、c 未定义
    int m,n;
    int k=m=n=8;                    //正确,但不规范,全部定义后,再连续赋值更好
    double x,y,z;
    x=y=z=0.;                       //正确且规范
    m=x%3;                          //错误,求余运算%的操作数必须是整型,x 不是
    x/2++;                          //错误,x/2 不是左值操作数,是一个实型数据
    2=m;                            //错误,2 不是左值操作数,是一个整型常量
    y/2=z;                          //错误,y/2 不是左值操作数,是一个实型数据
    return 0;
}
```

3.4 关系运算符

计算机程序能够进行判断就是因为可以进行比较，这种比较运算就是关系运算符。C 语言设计了 6 种关系运算符，都是双目运算符，都是左结合。

C 语言的 6 种关系运算符，按优先级可分为以下两组。

(1) 优先级较高的 4 种：＞(大于)、＞＝(大于或等于)、＜(小于)、＜＝(小于或等于)；

(2) 优先级相对低的 2 种：＝＝(等于)、!＝(不等于)。

由关系运算符连接起来的表达式即为关系表达式，关系表达式的结果都是整型，只有两种结果：0 或 1，分别代表比较的逻辑结果假和真。

例 3.3 分析下面程序中各关系表达式的结果。

```
int a=1,b=5,c=-8,m=0;
double x=0.01;
a>=0;                               //关系表达式的结果为 1
m==0;                               //关系表达式的结果为 1
b==0;                               //关系表达式的结果为 0
b>=c;                               //关系表达式的结果为 1
x>m;                                //关系表达式的结果为 1
a>b+c;                              //关系表达式的结果为 1,算术优先
```

使用关系运算符需注意以下几点。

(1) 无论参与关系运算的操作数的类型如何，关系运算的结果只能是 0(假，false)或 1(真，true)两个整型值之一。C 语言没有逻辑类型，用整型表示。

(2) 关系运算符的“==”运算符(相等的比较)由两个等号“=”组成,注意与赋值运算符“=”区别开。

(3) 字符型数据按其 ASCII 码值的大小进行比较,但字符串常量不能直接用关系运算符比较。如"ABCD"与"AAA"不能直接比较大小,应使用相关的字符串处理函数进行比较。

(4) 不要对实型数据做是否相等或不相等的比较,因为两个实型数据的有效数字位数不一定相同,难以做精确比较。否则可能出现无法预知的结果或出现与预期值背离的结果。应根据两个实型数据的差的绝对值是否小于某个比较小的数来近似判断它们是否相等。

(5) 在 C 语言中,实现数学中的 $3 \leqslant x \leqslant 5$,需要分解为 $3 \leqslant x$ 和 $x \leqslant 5$ 两个关系运算,然后进行逻辑与运算。表达式 3<=x<=5 语法是正确的,但不是与期望的数学逻辑一致,而是先判断 3<=x,结果可能是 0 或 1,再与 5 进行比较,0 或 1 都小于 5,结果永远是 1(真)。正确的表达应该是 3<=x && x<=5。

3.5 逻辑运算符

仅靠关系运算无法表达复杂的逻辑关系,如大于 3 且小于 5、小于 3 或大于 5 等。C 语言为此设计了逻辑运算符。C 语言有 3 种逻辑运算符,优先级、结合性均有差别。由逻辑运算符连接起来组成的表达式即为逻辑表达式,逻辑表达式的结果与关系表达式一样只能是两个整型值之一:0(假)或 1(真)。

(1) 逻辑与运算符:&&,如 a&&b,若 a、b 均为非 0(真),则 a&&b 结果为 1(真);若 a、b 之一为 0(假),则 a&&b 结果为 0(假)。优先级在 3 种逻辑运算中居中,左结合。

(2) 逻辑或运算符:||,如 a||b,若 a、b 之一为非 0(真),则 a||b 的结果为 1(真);若 a、b 均为 0(假),则 a||b 的结果为 0(假)。优先级在 3 种逻辑运算中最低,左结合。

(3) 逻辑非运算符:!,如!a,若 a 为非 0(真),则!a 的结果为 0(假);若 a 为 0(假),则! a 的结果为 1(真)。优先级在 3 种逻辑运算中最高,右结合。

在 C 语言中,逻辑运算的操作数并不要求是逻辑值,允许各种类型的数据参与逻辑运算。对各种数据的逻辑判断规则是数据值为 0 判断为“假”,非 0 都判断为“真”。逻辑运算的结果“真”则保存为 1。

例如,3||2 合法,其结果为 1(true),8&&0 也合法,其结果为 0(false)。

若 a=4,b=5,则 a&&b 的值为 1(true),因为 a 和 b 均为非 0,被当作“真”。

若 a=4,则!a 的结果为 0(false),因 a 的值为非 0 被当作“真”,对“真”进行非运算结果为“假”。

注意,在逻辑运算中,C 语言规定了两种快速处理:

(1) 对于逻辑表达式“(表达式 1)&&(表达式 2)”,如果表达式 1 的值已经是 0,则直接得到逻辑表达式的结果为 0,而不运行计算表达式 2;

(2) 对于逻辑表达式“(表达式 1)||(表达式 2)”,如果表达式 1 的值已经是 1,则直接得到逻辑表达式的结果为 1,而不运行计算表达式 2。

以上两种快速处理对于表达式 2 中有赋值运算时会影响运行结果,分析程序时需要注意,编程则建议避免在表达式 2 中赋值。

例 3.4 将下列逻辑条件用 C 语言的表达式表示。

(1) 三个边长数据 a、b、c 若能够构成三角形,表达式取真值,否则取假值。

【解】 能构成三角形的充要条件是三角形三条边中的任意两条边的边长之和大于第三条边。即 $a+b>c\&\&a+c>b\&\&b+c>a$。

更清晰的写法:

$(a+b)>c\ \&\&\ (a+c)>b\ \&\&\ (b+c)>a$

(2) 判别一个字符(ch)是否是一个字母。

【解】 ch>= 'A' && ch<= 'Z' || ch>= 'a' && ch<= 'z'

(3) 判别某一年(year)是否为闰年。闰年的条件是符合下面两者之一:①能被 4 整除,但不能被 100 整除。②能被 400 整除。如 2008 和 2000 年是闰年,2005 和 2100 年不是闰年。

【解】 (year % 4 == 0 && year % 100 != 0) || year % 400 == 0

当给定 year 为某一整数值时,如果上述表达式值为 1,则 year 为闰年,否则 year 为非闰年。

还可以用“!”运算作用于上述表达式,直接判别非闰年:

!((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)

这时,若表达式值为 1,则 year 为非闰年,否则 year 为闰年。

3.6 位运算符

前面介绍的运算符都是针对 1 字节以上长度的数据处理的,但在对开关量的控制中,二进制数据的 1 位即可采集一个开关量的状况进行输出控制,这样需要对某 1 位或几位数据进行判断或处理,为此,C 语言设计了位运算符。位运算符又分为位逻辑运算符和移位运算符,其操作数要求必须是整型,结果也是整型。

3.6.1 位逻辑运算符

位逻辑运算符有位求反(~)、按位与(&)、按位异或(^)和按位或(|)4 个。其中位求反(~)是单目运算符,其余 3 个是双目运算符。

双目位逻辑运算是将两个整型操作数的每个对应二进制位分别进行逻辑运算,得到结果。单目位逻辑运算是对整型操作数的每个二进制位分别求反得到结果。

注意区别逻辑运算:逻辑运算对操作数类型没有要求,是将操作数当作一个整体进行逻辑运算处理,根据操作数的值是 0 或非 0 得到结果。

例 3.5 判断下面程序的输出结果：

```
#include<iostream>
using namespace std;
int main(void)
{
    short a=-1,b=5;
    unsigned short m=65535,n=0;
    int k=3;
    cout<<(a&b)<<", "<<(a&&b)<<endl;        //5, 1
    cout<<(a|b)<<", "<<(a||b)<<endl;        //-1, 1
    cout<<(m&n)<<", "<<(m&&n)<<endl;        //0, 0
    cout<<(a&m)<<", "<<(a&&m)<<endl;        //65535, 1
    cout<<(a&k)<<", "<<(a&&k)<<endl;        //3, 1
    cout<<~a<<", "<<!a<<endl;               //0, 0
    cout<<~b<<", "<<!b<<endl;               //-6, 0
    cout<<~m<<", "<<!m<<endl;               //-65536, 0
    cout<<~n<<", "<<!n<<endl;               //-1, 1
    cout<<~k<<", "<<!k<<endl;               //-4, 0
    return 0;
}
```

【解析】 分析 5 个变量的二进制值，位逻辑运算时，对短整型、无符号短整型先分别进行符号位扩展，0 扩展到 4 字节 32 位，然后按位进行逻辑运算，最后的结果按整型读出即得到结果。图 3-1 是 5 个变量存储的二进制数据。

a	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
b	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
m	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
n	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

图 3-1 例 3.5 中 5 个变量存储的二进制数据

图 3-1 只画出了整型变量 *k* 的低 16 位数，高 16 位都是 0。

用十六进制数表示，*a* 符号位扩展为 ffffffff，*m* 则扩展为 0000ffff，则 *a*&*m* 的结果为 0000ffff，按整型十进制读出为 65535。

～m 的结果为 ffff0000，按整型读取是补码，原码（对负数补码符号位不变，其余位求反，然后加 1）为 80010000，十进制结果为－65536。

3.6.2 移位运算符

有左移位运算符（<<）和右移位运算符（>>）是双目运算符，要求两个操作数都是

整型或无符号整型，结果也是整型或无符号整型。

$a<<n$，表示将整型数据 a 的二进制数各位依次左移 n 位，左边移出后舍弃，右边移入 0。如果 $n<0$，相当于左移 $32-n$ 位。$a<<n$ 效果大致相当于 a 乘以 2^n。

$a>>n$，表示将整型数据 a 的二进制数各位依次右移 n 位，右边移出后舍弃，整型数左边移入符号位，无符号数左边移入 0。如果 $n<0$，相当于右移 $32+n$ 位。$a>>n$ 的效果大致相当于 a 除以 2^n，但小数部分向下取整。

3.6.3 位运算的应用

在自动控制中，经常需要根据某 1 位或几位的状态判断是否需要做某种处理，而 C 语言的控制语句只能根据总体逻辑值(0 或 1)来进行处理，因此需要将位状态转换为总体逻辑值。

例 3.6 设输入的值在整型变量 a 中，设 a 的最低位称为 b_0，向上各位依次称为 b_1，b_2，…，b_{31}，用 C 语言表达如下。

(1) 判断其 b_3 位是否为 1。

【解】 (a&0b1000)或(a&8)，当 a 的 b_3 位为 1 时，表达式的值为非 0；否则表达式的值为 0。

(2) 判断其 b_3 位是否为 0。

【解】 (~a&0b1000)或(~a&8)，当 a 的 b_3 位为 0 时，表达式的值为非 0；否则表达式的值为 0。

(3) 判断其 b_3 位和 b_5 是否均为 1。

【解】 ((a&0b1000)&&(a&0b100000))或((a&8)&&(a&32))或(a&40==40)，当 a 的 b_3 位和 b_5 位均为 1 时，表达式的值为 1；否则表达式的值为 0。

(4) 判断其 b_3 位是否为 1 且 b_5 为 0，b_7 位为 1、b_8 位为 1。

【解】 ((a&0b110101000)==0b110001000)或(a&424==392)，当 a 的 b_3 位为 1 且 b_5 为 0，b_7 位为 1、b_8 位为 1 时，表达式的值为 1；否则表达式的值为 0。

通过位逻辑运算，屏蔽了不需要的位，将某 1 位或几位的状态转换为了逻辑值，以便于后续的选择控制。

3.7 运算符的优先级

3.1.3 节已经介绍了运算符的优先级和结合性的概念，附录 C 详细列出了所有运算符的优先级和结合性。

为便于记忆，可将运算符的优先级大致归纳成图 3-2 所示的顺序。

其中，逻辑非(!)、位求反(~)认为是单目运算符的优先级，算术运算符中又细分先乘除后加减，算术运算符之后有移位运算符(<<)或(>>)，关系运算符中又细分先比较大小再比较相等，逻辑运算符前还有位逻辑运算符(先位与 &、再位异或^、最后位或|)，逻

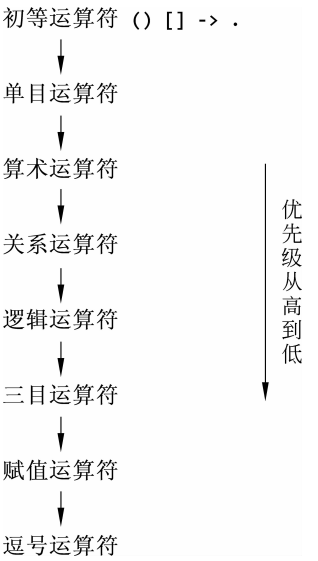


图 3-2　C 语言运算符优先级的粗略顺序

辑运算又细分为先与运算(&&)后或运算(||)。

运算符的结合性绝大多数是左结合，只有 3 类是右结合：单目运算符、三目运算符和赋值运算符。

3.8　小　　结

(1) 运算符是构成表达式并控制计算机进行运算或处理的核心。

(2) C 语言常用的运算符有赋值运算符、算术运算符、关系运算符、逻辑运算符、位运算运算符、条件运算符、逗号运算符等。

(3) 运算符有优先级和结合性。

(4) 根据运算符构成表达式需要操作数的数目，分为单目、双目和三目运算符，多数运算符是双目运算符。

(5) 运算符和操作数构成表达式，表达式完成运算后的结果是一个数据。

习　题　3

1. 分析下面 C 语言表达式的结果。

(1) 1/2 * 4　　(2) 4./8 * 100　　(3) −5%−3

(4) 2&&1　　(5) 2&1　　(6) 2&6

(7) 2||1　　(8) 2|1　　(9) 2|6

2. 将下面各题的条件用 C 语言的关系或逻辑表达式表达。

(1) 整型变量 m 为偶数；

(2) 整型变量 n 为奇数；

(3) 变量 x 满足：$3<x<5$；

(4) 无符号整型变量 m 的最高位为 1 且最低位为 0；

(5) 三角形的三条边长分别为 a、b、c，该三角形是直角三角形。

3. 分别用一个 C 语言的表达式得到下面各题要求的数据。

(1) 无符号整型变量 d 的值的十进制数的个位数；

(2) 无符号整型变量 d 的值的十进制数的十位数；

(3) 无符号整型变量 d 的值的十进制数的百位数；

(4) 无符号整型变量 d 的值用八进制数表示时的最低位数字；

(5) 无符号整型变量 d 的值用八进制数表示时的次低位数字。

4. 指出下面程序语句中的错误。

```
int a,b,c;
double d1,d2,d3,d4,d5;
d1=4ac;
3+=a;
d2+d3=d4;
d2+d3==d4;
d5=d1%2;
a=d1&4;
d2=d3>>3;
```

第4章

程序的选择结构

思考题

1. 如何实现在程序流程中根据逻辑判断选择不同的处理方式？

2. 编程求 x 的绝对值：当 x 小于 0 时，$x=-x$。

3. 编程实现深度网络中的激发函数 ReLU 函数：

$$y=\begin{cases}0, & (x<0)\\ x, & (x\geqslant 0)\end{cases}$$

4. 编程实现转换：根据下面的规则将百分制分数 score($0\leqslant$ score $\leqslant 100$)转换为相应的等级 rank 描述：

$$\text{rank}=\begin{cases}\text{优} & 90\leqslant\text{score}\\ \text{良} & 80\leqslant\text{score}<90\\ \text{中} & 70\leqslant\text{score}<80\\ \text{及格} & 60\leqslant\text{score}<70\\ \text{不及格} & \text{score}<60\end{cases}$$

以上问题的共同特点：虽然有多种可能的情形发生，但在某一时刻，只可能其中一种情形会发生，即各种可能发生的情形之间是一种非此即彼的逻辑关系。从程序的执行逻辑来说，在两条或多条执行线索(操作块)中，只可能依据条件选择其中一条线索来执行，或者某个执行线索只可能依据条件决定其是否执行。这就构成了程序的选择结构，也称为分支结构，其对应的程序执行流程如图 4-1 所示。

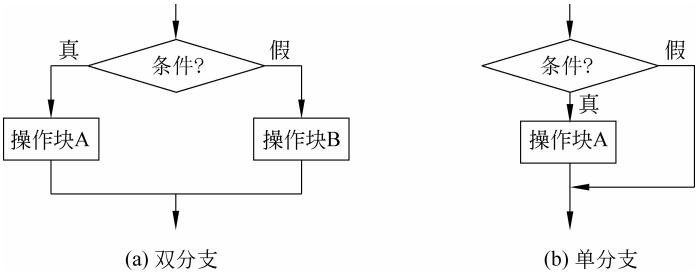
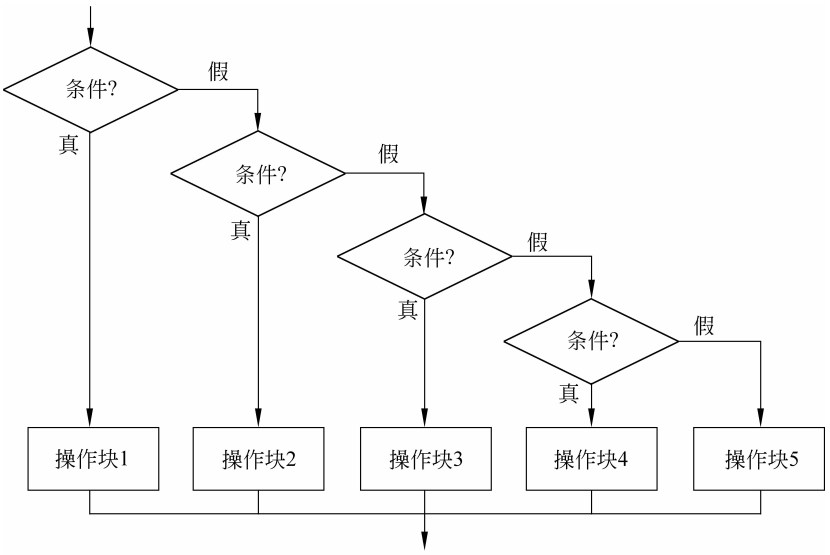


图 4-1　程序选择结构的执行流程示意图



(c) 多分支

图 4-1(续)

从程序执行的流程来看,双分支结构是选择结构的基本形式,单分支结构和多分支结构则是由双分支结构演变而来的。

在 C/C++ 语言中可通过条件表达式、if 语句和 switch 语句建立选择结构的程序。

4.1 双分支选择结构

在两条执行线索中,只可能依据条件选择其中一条线索来执行,这就构成了双分支选择结构的程序。通过 if 语句的基本形式 if-else 语句可实现双分支选择结构的程序。

if-else 语句的形式为:

```
if (表达式)
    语句 1;
else
    语句 2;
```

if-else 语句的程序流程如图 4-2 所示。

双分支语句的执行流程是当表达式的值为"真"时(非 0 值),执行语句 1,否则执行语句 2。语句 1 或语句 2 也可以是复合语句,即用花括号{}括起来的多条语句。

双分支的选择结构是在两个操作块中,依据条件选择其中一个操作块来执行。

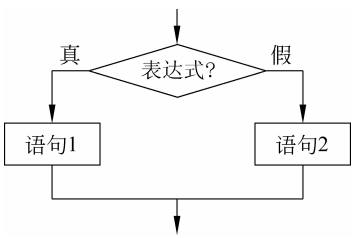


图 4-2 双分支选择结构的程序流程图

【说明】

(1) if 后面括号中的表达式,可以是任何合法的表达式,均可作为 if 语句判断的条件,其依据是表达式的值,非 0 即为“真”,0 值为“假”。

(2) 不要错误地认为 if-else 语句是两个语句。else 是 if 语句中的子句,它不能作为独立的语句单独使用,而必须与 if 配对使用。

例 4.1 编程实现深度网络中的激发函数 ReLU 函数:

$$y=\begin{cases}0, & (x<0)\\ x, & (x\geqslant 0)\end{cases}$$

```
#include<iostream>
using namespace std;
int main(void)
{
    double x,y;
    cout<<"Input a data : ";
    cin>>x;
    if(x<0)
        y=0;
    else
        y=x;
    cout<<y;
    return 0;
}
```

运行程序输入-2.5,得到输出为 0。再次运行程序,输入 3.14,得到输出为 3.14。

【说明】

(1) 作为条件使用的表达式,按照其“真”“假”判断的依据:非 0 为“真”、0 值为“假”,则代码 if(x!=0)的表达式可以简化,写为 if(x),功能一样,但程序的可读性变差了。

(2) 例 4.1 程序中的 if-else 语句还可以改为用条件表达式语句来完成同样的功能:

```
y=(x<0)?0:x;
```

在 if-else 语句中,if-else 两个分支分别代表的执行线索,往往不只是一条语句组成,而是包含一组语句的复合语句。因此,if-else 中的“语句 1”和“语句 2”往往是以复合语句的形式出现。也就是说,if 语句更常用的形式如下:

```
if(表达式)
{
    语句系列 1;
}
else
{
    语句系列 2;
}
```

例 4.2 编程将 x、y 这两个数中较大的数赋给 max，较小的数赋给 min。

```
#include<iostream>
using namespace std;
int main(void)
{
    double x,y,max,min;
    cin>>x>>y;
    if(x>y)
        max=x;
        min=y;
    else
        max=y;
        min=x;
    cout<<"max="<<max<<endl;
    cout<<"min="<<min<<endl;
    return 0;
}
```

程序编译将出现如图 4-3 所示的错误提示信息，提示的错误是其中的 else 子句前面没有配对的 if。为什么会这样呢？

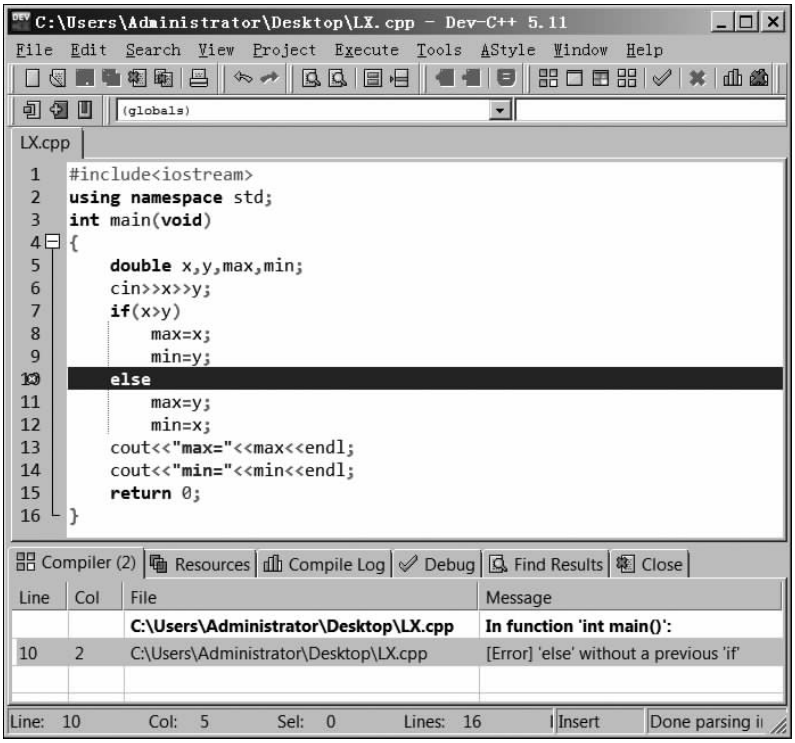


图 4-3 例 4.2 程序编译出错的提示信息

原因是 if-else 语句的两个分支分别只能是一条语句。而上述代码中，if(x>y)后面

紧跟着两条语句“max＝x;”和“min＝y;”。这样一来，“max＝x;”已经是 if(x＞y)的分支所要求的语句，“min＝y;”不属于 if(x＞y)语句的分支范畴，于是后续的 else 子句就找不到前面配对的 if 了。

为了使“max＝x;”和“min＝y;”两条语句都成为 if(x＞y)语句的分支，应该用一对{}把它们复合为一个操作块，相当于一条语句；同理，else 后续的两条语句也应该用一对{}把它们复合为一个操作块。

```
#include<iostream>
using namespace std;
int main(void)
{
    double x,y,max,min;
    cin>>x>>y;
    if(x>y)
    {
        max=x;
        min=y;
    }
    else
    {
        max=y;
        min=x;
    }
    cout<<"max="<<max<<endl;
    cout<<"min="<<min<<endl;
    return 0;
}
```

上述程序也可以用以下方法实现。

```
#include<iostream>
using namespace std;
int main(void)
{
    double x,y,max,min;
    cin>>x>>y;
    if(x>y)
        max=x, min=y;                          //与上面的程序仔细比较下
    else
        max=y, min=x;                          //仔细比较下
    cout<<"max="<<max<<endl;
    cout<<"min="<<min<<endl;
    return 0;
}
```

另外还可以使用条件表达式实现本例题的功能，代码如下：

```
#include<iostream>
using namespace std;
int main(void)
{
    double x,y,max,min;
    cin>>x>>y;
    max=x>y?x:y;
    min=x<y?x:y;
    cout<<"max="<<max<<endl;
    cout<<"min="<<min<<endl;
    return 0;
}
```

例 4.3 编写程序，判断某一年(year)是否为闰年。闰年的条件和逻辑表达式见例3.4第(3)题。

程序为

```
#include<iostream>
using namespace std;
int main(void)
{
    int year, leap;                          //leap用于存储是否闰年的信息:1-是,0-否
    cout<<"Please input year : ";
    cin>>year;
    leap=(year%4==0 && year%100!=0)||(year%400==0);
    if(leap)
        cout<<year<<" is a leap year."<<endl;
    else
        cout<<year<<" is not a leap year."<<endl;
    return 0;
}
```

4.2 单分支选择结构

若某个操作块只能依据条件决定其是否执行，这就构成了单分支选择结构的程序。将 if 语句基本形式中的 else 部分省略，就变成了单分支选择结构，即

```
if(表达式)
    语句;
```

或分支中的语句为复合语句，形如

```
if(表达式)
{
    语句系列;
}
```

单分支选择结构的程序流程如图 4-4 所示，单分支选择结构 if 语句的执行流程是当表达式的值为“真”时(非 0 值)，执行语句，否则不执行。然后继续执行 if 语句后面的语句。

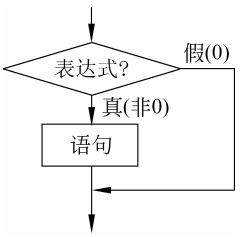


图 4-4　单分支选择结构程序流程图

例 4.4　编程：输入一个字符，判别它是否为大写字母，如果是，将它转换成小写字母，然后输出。

【分析】

(1) 根据条件(是否为大写字母)，决定是否进行大小写转换，这是一个单分支选择结构。

(2) 从 ASCII 码表中可知，大写字母排列在小写字母之前，即大写字母的 ASCII 码值小于小写字母的 ASCII 码值，且对应的大小写字母的 ASCII 码值相差 32。这个 32 的差值不必死记，利用整型与字符型数据通用的特征，用对应大小写字母的字符数据进行相减即可得到。

程序为

```
#include <iostream>
using namespace std;
int main(void)
{
    char ch;
    cin>>ch;
    if(ch>='A'&&ch<='Z') //判断是否是大写字母
        ch=ch+32;        //将大写字母转换成小写字母,也可写为 ch=ch+('a'-'A');
    cout<<ch<<endl;      //('a'-'A')的结果就是 32
    return 0;
}
```

运行程序，输入大写字母，将输出显示对应的小写字母；输入其他字符，则输出不变。

单分支选择结构其实是双分支选择结构在某一分支没有具体操作时的简化结构。如例题 4.4 的问题描述，可以换成与双分支结构对应的描述：“输入一个字符，判别它是否为大写字母，如果是，将它转换成小写字母；如果不是，不转换。然后输出最后得到的字符。”

其对应的双分支选择结构的程序为

```
#include <iostream>
using namespace std;
int main(void)
{
    char ch;
```

```
    cin>>ch;
    if(ch>='A'&&ch<='Z')
        ch=ch+32;
    else
        ;                              //空语句
    cout<<ch<<endl;
    return 0;
}
```

单分支结构相当于一个分支是空语句的双分支结构。

【说明】

(1) 如何在 if-else 语句中表示“双分支选择结构中的没有具体操作的分支”是上述代码的关键。即通过只有一个“;”的空语句来实现双分支选择结构中的一个分支(但它没有具体的操作),从而完整地实现双分支结构。

(2) 上面程序中的 if 语句还可以换成条件表达式语句来完成同样的功能。程序如下:

```
#include <iostream>
using namespace std;
int main(void)
{
    char ch;
    cin>>ch;
    ch=(ch>='A'&&ch<='Z')?(ch+'a'-'A'):ch;
    cout<<ch<<endl;
    return 0;
}
```

例 4.5 不适当使用单分支选择结构的示例如下。

在例 4.1 中,通过双分支选择结构的程序,清晰直观地反映了 ReLU 函数所体现的非此即彼的逻辑关系:

$$y=\begin{cases}0, & (x<0)\\ x, & (x\geqslant 0).\end{cases}$$

若改用以下的单分支结构的程序,也能实现该函数的功能。

```
#include <iostream>
using namespace std;
int main(void)
{
    double x,y;
    cin>>x;
    y=0;
    if(x>=0)
```