第3章

选择结构及相关表达式

程序由3种基本结构组成：顺序结构、选择结构和循环结构。前面介绍了顺序结构，本章将介绍选择结构。选择结构又称分支结构，它在程序执行中根据条件判断的结果来控制程序的流程。

3.1 选择结构

程序执行过程中，经常需要进行条件判断，根据判断结果进行不同的处理，形成多个分支。假如一门课程的及格分数定为60分，那么当输入一个学生的分数后，成绩如果大于或等于60分，则显示“通过”，否则显示“不通过”。程序流程图如图3-1所示。

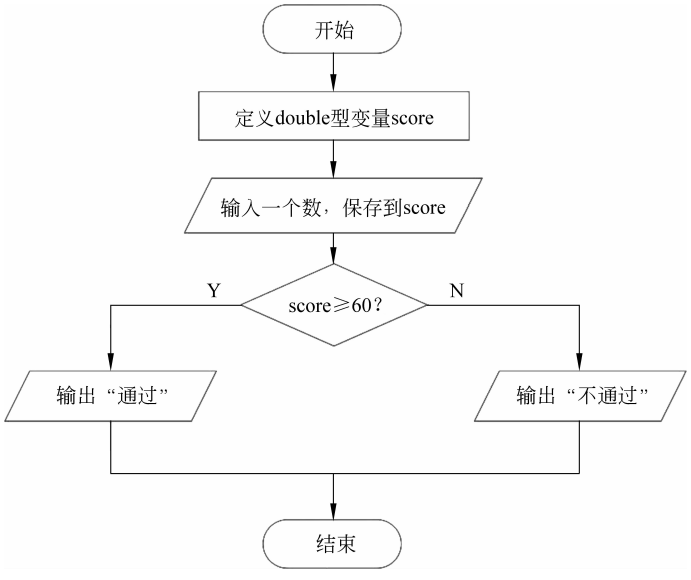

图3-1 选择结构程序流程图示例

选择结构中的判断条件通常是使用关系运算符或逻辑运算符的表达式，因此本章先学习关系运算符和逻辑运算符，再学习用if语句或switch语句来控制程序的流程。

3.2　关系运算符和关系表达式

3.2.1　关系运算符

比较两个值的运算符称为关系运算符。程序中经常需要比较两个值的大小，根据结果决定程序的下一步。例如比较成绩与 60 的大小，若成绩≥60 输出“通过”，若成绩＜60 输出“不通过”。

C&C++ 的关系运算符有 6 个，如表 3-1 所示。

表 3-1　关系运算符

关系运算符	说　明	优　先　级
＞	大于	相同 低于＋、－、＊、/、% 高于＝ 高于＝＝、！＝
＞＝	大于或等于	
＜	小于	
＜＝	小于或等于	
＝＝	等于	相同 低于＋、－、＊、/、% 高于＝ 低于＞、＞＝、＜、＜＝
！＝	不等于	

关系运算符是双目运算符，结合性为左结合。＜、＜＝、＞、＞＝ 的优先级相同，高于＝＝和！＝；＝＝和！＝的优先级相同。关系运算符的优先级低于算术运算符，高于赋值运算符。

例如：

(1) c ＞ a ＋ b 等价于 c ＞ (a ＋ b)。

(2) a ＞ b ＝＝ c 等价于 (a ＞ b) ＝＝ c。

(3) a ＝＝ b ＜ c 等价于 a ＝＝ (b ＜ c)。

(4) a ＝ b ＞ c 等价于 a ＝ (b ＞ c)。

3.2.2　关系表达式及应用

关系表达式用关系运算符将表达式连接起来。一般形式为

表达式　　关系运算符　　表达式

任何表达式都可以用关系运算符连接。例如，下面都是合法的关系表达式：

```
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,真对应整数 1 或布尔型值 true,假对应整数 0 或布尔型值 false。

例如:

(1) 5 > 0。值为真,表达式的结果为 1(true)。

(2) (a = 3) > (b = 5)。第一个小括号内表达式的值是 a 的值,将 3 赋给 a(注意,=是赋值运算符);第二个小括号内表达式的值是 b 的值,将 b 赋给 5;然后判断 3>5,值为假,该表达式的结果为 0 (false) 。

3.2.3 陷阱:关系表达式的常见问题

在实际应用中,要注意以下几个与数学应用不同的情况:

(1) 10 <= a <= 20。它在数学中的含义是 a 的值在 10 到 20 之间且包含 10 和 20,但是在程序中这个表达式永远为真。按照计算机的运算规则,首先计算表达式 10<=a,得到的值要么为真(1)要么为假(0),而无论 1 或 0 都一定小于或等于 20。要想表达 10 <= a 和 a <= 20 两个条件同时成立,要使用逻辑运算符 && 连接这两个条件,正确的表达式是 10 <= a && a <= 20,逻辑运算符见 3.3 节。

(2) ==与=的区别。例如,a==2 是一个关系表达式,判断 a 与 2 是否相等,结果为真或假;而 a=2 是一个赋值表达式,将 2 赋值给=左边的变量,表达式的值为 a 的值,即 2。程序认为 a=2 和 a==2 的语法都是正确的,所以书写时要注意,判断相等时要用两个等号(==),赋值表达式使用一个等号(=)。

(3) 判断一个实数是否等于 0,不要用==或!=,而是要判断是否在精度范围内。例如:

```
double a=0;
double eps=0.000001;          //定义一个精度
fabs(a)<eps
```

为真表示 a 的值为 0(a 的值与 0 的差在精度范围内)。

3.3 逻辑运算符和逻辑表达式

3.3.1 逻辑运算符

现实生活中判断的条件可能是复杂的,需要对多个条件综合判断后才能得到结果。例如,闰年的判断条件是“四年一闰,百年不闰,四百年再闰”,即闰年要满足下面两个条件之一:①能被 4 整除但是不能被 100 整除;②能被 400 整除。判断能否整除要用关系表达式,几个关系表达式之间存在着“并且”“或”的关系,这些关系就要用逻辑运算符表示。C&C++ 中提供了 3 种逻辑运算符,如表 3-2 所示。

表 3-2　逻辑运算符

逻辑运算符	说　明	举　　例
&&	逻辑与	a>b && a>c(a 比 b 大并且 a 比 c 大时为“真”,否则为“假”)
||	逻辑或	a>0 || b>0(a 大于 0 或者 b 大于 0 时为“真”,否则为“假”)
!	逻辑非	! 0(0 是“假”,则! 0 的值是“真”)

逻辑与运算符 && 和逻辑或运算符||均为双目运算符,具有左结合性。逻辑非运算符! 为单目运算符,具有右结合性。逻辑运算符和其他运算符的优先级从高到低是

! → 算术运算符 → 关系运算符→ && → || → =

例如:

(1) a > b && c > d 等价于 (a > b) && (c > d)。

(2) ! b == c || d < a 等价于 ((! b) == c) || (d < a)。

(3) a + b > c && x + y < b 等价于 ((a + b) > c) && ((x + y) < b)。

3.3.2　逻辑表达式及应用

将两个表达式用逻辑运算符连接起来就成为一个逻辑表达式,3.3.1 节的几个式子就是逻辑表达式。逻辑表达式的一般形式可以表示为

表达式　逻辑运算符　表达式

逻辑表达式的值是逻辑真或假,真对应整数 1 或布尔型值 true,假对应整数 0 或布尔型值 false。关系表达式、赋值表达式、算术表达式、函数调用表达式等任何表达式都可以用逻辑运算符连接。判断一个条件是真还是假时,非 0 值代表真,0 值代表假。例如,在 3+2 == 5 && 2 + 3 中,3+2==5 是关系表达式,值为 1(true),2+3 是算术表达式,值为 5(非 0 值),在逻辑运算中取值为 1(true),所以表达式 3+2 == 5 && 2 + 3 的值为 1(true)。

逻辑运算时,计算机一旦能够得到表达式的结果,就会结束计算。例如 0 && 3>2,0 与任何值逻辑与的结果都是假,则计算机不会判断 3>2 的结果。

例如,若 a=4,b=5,则:

(1) ! a 值为 0。因为 a 的值是非 0,为真,非真是假。

(2) a && b 值为 1。因为 a 和 b 的值都是非 0(真),逻辑与的结果也是真。

(3) a - b || a + b,值为 1。因为 a - b 的值是非 0(真),逻辑或的结果也是真(不用看 a+b 的值)。

(4) ! a || b 值为 1。因为! a 的值为真,逻辑或的结果也是真(不用看 b 的值)。

(5) 4 && 0 || 2 值为 1。首先看 4&&0 的值,为假,即 0(false),然后 0 || 2 的结果为真。

【例 3-1】 写出判断某一年(year)是否为闰年的条件表达式。

闰年要满足下面两个条件之一:①能被 4 整除但是不能被 100 整除;②能被 400 整

除。例如,2000 年、2004 年是闰年,而 1900 年、2005 年不是闰年。

可以用一个逻辑表达式来表示上面两个条件,表达式的值为 1(true)则 year 是闰年,否则不是闰年:

```
( 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真(即表达式值为 1),则 year 是闰年;否则 year 不是闰年。

【例 3-2】 计算 3 个点组成的三角形面积。

用户输入 3 个点的坐标,判断它们能否组成三角形。如果能,计算三角形面积并输出;如果不能,输出错误提示。

程序代码如下:

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    double a,b,c;
    cin >>a >>b >>c;
    if ( a+b>c && a+c>b && b+c>a && a>0 && b>0 && c>0 )
    {
        double s =(a +b +c ) / 2;
        double area =sqrt ( s * ( s -a ) * ( s -b ) * ( s -c ) );
        cout <<area;
    }
    else
        cout <<"不能组成三角形";
    return 0;
}
```

3.4 条件运算符及条件表达式

条件运算符(?:)要求有 3 个运算对象,因此为三目(元)运算符,它是 C&C++ 中唯一的三目运算符,可以把 3 个表达式连接起来,构成一个条件表达式,条件表达式的一般形式如下:

表达式 1 ? 表达式 2 : 表达式 3

条件表达式的执行顺序如下:

(1) 求解表达式 1。

(2) 若表达式 1 的值为非 0(真),则条件表达式的值是表达式 2 的值。

(3) 若表达式 1 的值为 0(假),则条件表达式的值是表达式 3 的值。

例如：

```
max = ( a >b ) ? a : b;
```

相当于下面的 if 语句：

```
if ( a >b)
    max =a;
else
    max =b;
```

条件运算符优先于赋值运算符，因此上面赋值表达式的求解过程是：先求解条件表达式，再将它的值赋给 max。条件表达式（a ＞ b）？a ：b 的执行结果是 a 和 b 中较大的值。

在条件表达式中，允许表达式 1 的类型与另外两个表达式的类型不同。例如，对于表达式 x ? 'a': 'b'，如果已定义 x 为整型变量，若 x 的值为 0，则条件表达式的值为字符'b'。

表达式 2 和表达式 3 的类型也可以不同，此时条件表达式的值是二者中较高的数据类型。例如，条件表达式 x ＞ y ? 1 ：1.5，由于 1.5 是 double 型，精度要比 1 的 int 型高，因此，将 1 转换成双精度数，条件表达式的值是双精度浮点型值 1.0 或 1.5。

【例 3-3】 输入一个字符，判断它是否为大写字母。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母。

算法分析：

(1) 小写字母的 ASCII 码值比对应的大写字母的 ASCII 码值大 32。

(2) 判断一个字符变量 ch 的值是否是大写字母，就是判断'A'≤ch≤'Z'，程序中用逻辑表达式表示为 ch ＞＝ 'A' &&. ch ＜＝ 'Z'。常量'A'也可以用 65 表示(65 是'A'的 ASCII 码)。

代码如下：

```
#include <iostream>
using namespace std;
int main()
{
    char ch;
    cin >>ch;
    ch = (ch >='A' && ch <='Z') ? (ch +32) : ch;     //若 ch 是大写字母则转换为小写字母
    cout <<ch;
    return 0;
}
```

运行结果如下：

```
A
a
```

字符转换也可以使用标准库中的字符函数 isupper 来实现，在 C++ 程序中需要包含

std 命名空间的 cctype 文件,在 C 语言程序中是 ctype.h。

代码如下:

```
#include <iostream>
#include <cctype>
using namespace std;
int main()
{
    char ch;
    cin >>ch;
    ch =isupper(ch) ? ( ch +32 ) : ch;     //若 ch 是大写字母则转换为小写字母
    cout <<ch;
    return 0;
}
```

3.5 C99&C++ 的布尔型常量与变量

C++ 提供布尔型(bool)常量 true 和 false,分别用来表示逻辑值的真和假。C 语言中没有布尔型常量,一般用整数 1 或非 0 值表示 true,用 0 表示 false。

C++ 中提供布尔型变量用来保存逻辑值:true(真)和 false(假),对应整数值 1 和 0。

C 语言默认无布尔型,一般使用 int 型变量保存逻辑值对应的整数 1 和 0。从 C99 开始支持布尔型,但需要增加 #include <stdbool.h>语句(布尔型定义在 stdbool.h 中)。因此使用布尔型变量时要注意编译环境是否支持 C99 或 C++ 。

在逻辑运算表达式中,非 0 值表示 true,0 值表示 false。

3.6 if 语句

实现选择结构主要使用 if 语句。if 语句根据表达式的值来决定执行哪一个分支。

3.6.1 标准 if…else 语句

标准的 if…else 语句的一般使用形式如下:

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

if…else 语句在执行时首先判断 if 后面表达式的值,为真则执行语句 1,为假则执行语句 2,如图 3-2 所示。if 和 else 后面只能是逻辑上的一条语句(有多条语句时要加上大括号变成复合语句)。

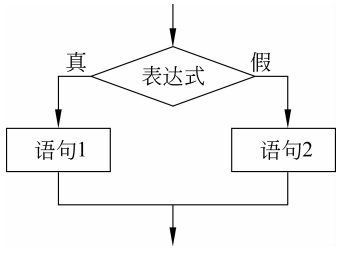

图 3-2 if…else 语句执行过程的流程图

书写 if…else 语句时要注意几个问题:

(1) if 后面的表达式可以是任何一种表达式,表达式的值非 0 则为真,0 为假。

(2) if 后面的表达式要用一对小括号括起来。

(3) if 后面的语句只能是逻辑上的一条语句,可以是空语句,也可以是复合语句。一般为了防止出错,即使一条语句也建议加上一对大括号。

(4) else 后面没有表达式。else 后面的语句只能是逻辑上的一条语句,可以是空语句,也可以是复合语句。一般为了防止出错,即使是一条语句,也建议加上一对大括号。

【例 3-4】 输入一个分数(0～100),如分数不低于 60,提示 Pass,否则提示 Fail。

一共有两个分支,使用 if…else 结构。源代码如下:

```
#include <iostream>
using namespace std;
int main()
{
    int grade;
    cout <<"Enter a grade(0～100): ";
    cin >>grade;
    if ( grade >=60 )
        cout <<"Pass\n";
    else
        cout <<"Fail\n";
    return 0;
}
```

运行示例如下:

```
Enter a grade(0～100): 75
Pass
```

注意,条件表达式一定要写到 if 后面的小括号内,如果条件表达式单独作为一条语句出现,是不能控制流程的。例如,下面是错误的写法:

```
 1  #include <iostream>
 2  using namespace std;
 3  int main()
 4  {
 5      int grade;
 6      cout <<"Enter a grade(0～100): ";
 7      cin >>grade;
 8      grade >=60 ;
 9      if ( grade )
10          cout <<"Pass\n";
11      else
12          cout <<"Fail\n";
13      return 0;
14  }
```

程序执行到第 8 行,表达式的结果无论是真还是假,语句都结束了,并没有记录 grade >=60 的结果。接下来的 if 语句中判断条件是 grade,也就是说,grade 的值非 0 就为真,0 为假,这与判断 grade 大于或等于 60 的要求不符。

【例 3-5】 根据 3 条边的长求三角形的面积;如无法组成三角形,输出错误信息。

算法分析:

(1) 3 个边长 a、b、c 组成三角形的条件是"任意两条边长之和大于第三边"。表达式为

a + b > c && a + c > b && b + c > a && a > 0 && b > 0 && c > 0

(2) 3 个边长 a、b、c 组成的三角形的面积公式是

$$s = \frac{a+b+c}{2}$$

$$\text{area} = \sqrt{s*(s-a)*(s-b)*(s-c)}$$

(3) 求平方根需要调用 sqrt 函数,它定义在 std 命名空间的 cmath 头文件中。格式化输出需要包含 std 命名空间的 iomanip 头文件。

源代码如下:

```
#include <iostream>
#include <cmath>                                  //使用数学函数时要包含头文件 cmath
using namespace std;
int main()
{
    double a,b,c;
    cout <<"Please Enter a,b,c: ";
    cin >>a >>b >>c;
    if ( a +b >c && a +c >b && b +c >a && a >0 && b >0 && c >0 )
    {                                             //复合语句开始,计算并输出面积
        double s, area ;                          //在复合语句内定义变量
        s =( a +b +c ) / 2;
        area =sqrt ( s * ( s -a ) * ( s -b ) * ( s -c ) ) ;
        cout <<"area=" <<area <<endl ;            //在复合语句内输出变量的值
    }                                             //复合语句结束
    else
       cout<<"It is not a triangle!"<<endl;
    return 0;
}
```

运行示例如下:

```
Please enter a, b, c: 2.45 3.67 4.89
area =4.35652
```

变量 s 和 area 只在复合语句内用到,因此在复合语句内定义,它的作用范围为从定义变量开始到复合语句结束。将某些变量局限在某一范围内,与外界隔离,可避免在其他

地方被误调用。

如果在复合语句外使用 s 和 area，则会在编译时出错。例如，在 return 0；前面增加一行 cout ＜＜ area；编译时会出现错误信息，如[Error]'area' was not declared in this scope，系统认为 area 变量未定义。

3.6.2　简单的 if 语句

if 语句可以只有 if 一个分支，当不满足条件时就什么都不做。语法形式为

```
if(表达式)
    语句
```

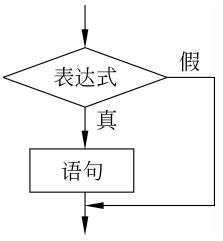

图 3-3　简单 if 语句执行过程的流程图

与标准 if…else 语句类似，if 后面只能是逻辑上的一条语句。其执行过程如图 3-3 所示。

例如，若 x ＞ y 为真，则执行语句输出 x 的值，否则就不输出 x 的值。if 语句如下：

```
if ( x >y )
    cout <<x <<endl;
```

3.6.3　复杂的 if…else if … else 语句

当有多个分支选择时，可采用一条 if…else if…else 语句，else if 允许有多个，每个 else if 后面都要有表达式。其一般形式为

```
if (表达式 1)
    语句 1
else if (表达式 2)
    语句 2
⋮
else if (表达式 n)
    语句 n
else
    语句 n+1
```

if…else if…else 语句在执行时，首先判断 if 后面的表达式 1 的值，若值为真，执行语句 1，语句结束；若值为假，判断表达式 2 的值，若值为真，执行语句 2，语句结束……如果所有 if 后面的条件都为假，执行最后的 else 后面的语句 $n+1$。同样地，if 与 else 后面的所有语句都只能是逻辑上的一条语句。

if…else if…else 语句的执行过程如图 3-4 所示。

【例 3-6】 铅笔价格如下：每支铅笔单价 2 元，购买量大于或等于 10 支且小于 50 支时对超出部分打 9.5 折，当购买量大于或等于 50 支时对超出部分打 9 折。根据用户输入的铅笔数量计算顾客应付金额(保留到元，角与分忽略)。

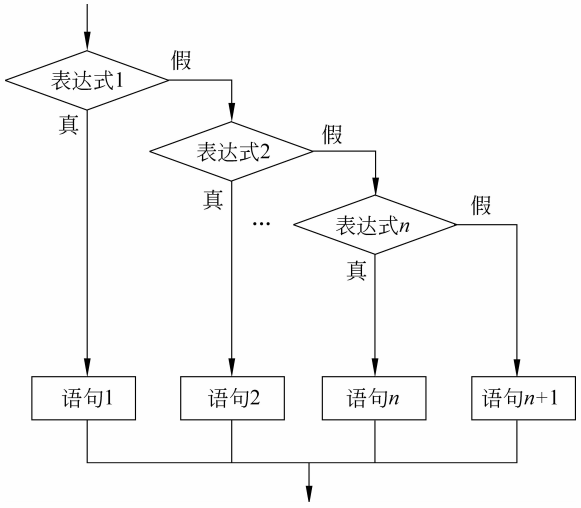

图 3-4　if…else if…else 语句执行过程的流程图

算法分析：定义符号常量 UNIT 值为 2，定义整型变量 x 保存用户输入的购买量，浮点型变量 y 为应付金额。

由于有多个分支，使用 if…else if…else 来实现，注意表达式的写法(见 3.2 节中的类似例子)。

源代码如下：

```
#include <iostream>
using namespace std;
const double UNIT =2;                          //每支铅笔单价为 2 元
int main()
{
    int x;
    int y =0;
    cout <<"请输入支数: " <<endl;
    cin >>x;
    if ( x <0 )
        y =0;
    else if ( x <10 )                          //等价于 x>0 && x<10
        y =x * UNIT;
    else if ( x <50 )                          //等价于 x>=10 && x<50
        y =10 * UNIT +( x-10 ) * UNIT * 0.95;
    else
        y =10 * UNIT +( 50 -10 ) * UNIT * 0.95 +( x -50 ) * UNIT * 0.9 ;
        cout <<"应付款=" <<y ;
    return 0;
}
```

运行 3 次，输入数据与输出结果如下：

```
请输入支数:
2
应付款=4
请输入支数:
11
应付款=21
请输入支数:
51
应付款=97
```

3.6.4　if 语句的嵌套

if 语句中执行分支的语句可以是任何一条语句，如果这条语句是另一个 if 语句，则称为 if 语句的嵌套。其一般形式如下：

```
if( 表达式 1)
    if(表达式 2 )
        语句 1
    else
        语句 2
else
    if(表达式 3 )
        语句 3
else
      语句 4
```

注意：

(1) 外层的 if 语句必须完整地包含内层的选择语句，缩进书写有助于看清包含关系。

(2) else 总是和离它最近且未和任何 else 配对的 if 进行配对，和代码的缩进无关。缩进的作用只是使代码富有层次感，美观易读。

(3) if 语句可以是标准的 if…else 语句，也可以是简单的 if 语句，还可以是复杂的 if…else if…else 语句。最好为每一层内嵌的 if 语句都加上大括号，变成复合语句，这样可以避免很多不必要的错误。例如：

```
if(表达式 1)
{
    if (表达式 2)
        //if 语句的内嵌 if 语句,即表达式 1 与表达式 2 都成立时执行语句 1
            语句 1
}
else
    //与第一个 if 配对,即在表达式 1 不成立时,执行语句 2
    语句 2
```

【例 3-7】 编写程序，判断某一年是否为闰年。

前面已经分析过闰年的条件：(year % 4 == 0 && year % 100 ! = 0) || (year % 400 == 0),如果对表达式逐个进行判断,当 year % 4 == 0 为真时,再判断 year % 100 ! = 0,也为真,就确认是闰年;否则判断 year % 400 == 0,若为真,就确认是闰年,为假则不是闰年;如果 year % 4 == 0 为假就不是闰年。流程图如图 3-5 所示。

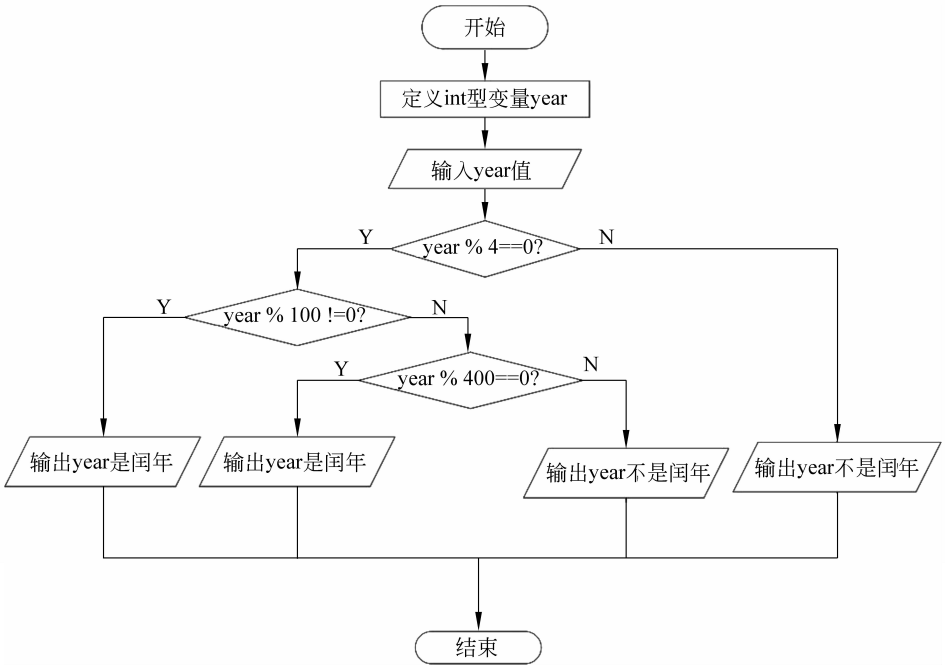

图 3-5　判断年份是否是闰年的程序流程图

代码如下：

```
#include <iostream>
using namespace std;
int main()
{
    int year;
    bool leap =false;                    //bool 型变量 leap 保存是否是闰年,初值为 false
    cout <<"please enter year: " ;       //输出提示
    cin >>year;                          //输入年份
    if ( year %4 ==0 )                   //年份能被 4 整除
    {
        if ( year %100 !=0 )             //年份能被 4 但是不能被 100 整除
            leap =true;                  //是闰年,leap 赋值 true
     else                                //年份能被 4 和 100 整除
     {
           if ( year %400 ==0)           //年份能被 4、100 和 400 整除
            leap =true;                  //是闰年,leap 赋值 true
            else                          //年份能被 4 和 100 整除,但不能被 400
```

```
整除
            leap =false;                    //非闰年,leap 赋值 false
        }
    }
    else                                    //年份不能被 4 整除
        leap=false;

    if ( leap )                             //若 leap 为 true,输出年份和 is
        cout <<year <<" is " ;
    else                                    //若 leap 为 false,输出年份和 is not
        cout <<year <<" is not " ;
    cout <<" a leap year." <<endl;          //输出“ a leap year.”
    return 0;
}
```

运行两次的输入数据和输出结果如下：

```
2004
2004 is a leap year.
1900
1900 is not a leap year.
```

也可以将程序中嵌套的 if 语句修改为 if…else if…else 语句，例如：

```
if ( year %4 !=0 )
    leap =false;
else if ( year %100 !=0 )
    leap =true;
else if ( year %400 ==0)
    leap=true;
else
    leap=false;
```

3.7　switch 语句

3.7.1　switch 语句实现的多分支结构

switch 语句也是一条实现多分支选择结构的语句，它在有些情况下要比 if 语句更清晰。它的一般形式如下：

```
switch ( 表达式 )
{
    case 常量表达式 1: 语句 1;[break;]
    case 常量表达式 2: 语句 2;[break;]
    ⋮
    case 常量表达式 n: 语句 n;[break;]
```

```
    default: 语句 n+1
}
```

说明：

(1) switch 后的表达式必须是整型或字符型；如果是双精度型，必须强制将类型转换为整型。

(2) case 后的常量表达式必须是整型或字符型，如 case 1：、case 'A'：、case 1＋2：等，不能有变量。

(3) 各个 case 后面表达式的值是唯一的，如果相同会出错。例如，switch 中有 case 1||2：与 case 3||4：，则编译出错，因为 1||2 与 3||4 的值都是逻辑真。

(4) 执行 switch 语句时，计算 switch 括号中的表达式的值，然后逐个与 case 后面的常量表达式的值比较。如果相同，从此 case 子句开始一直执行下去(后面的 case 子句不再做判断)；如果与所有的 case 后面的常量表达式的值都不同，就执行 default 后面的语句。

(5) break 是可选语句，如果执行匹配的 case 标记的语句，然后就希望结束 switch 语句(不执行后面的 case 语句)时，需要在该 case 子句的最后加上一条 break 语句，语法是 break；。

(6) case 和 default 的出现次序不影响执行结果。

(7) 每个 case 后面允许有多条语句，不需要加大括号。

【例 3-8】 输入两个整数的四则运算表达式，输出结果。

例如，输入 2＋3，结果为 2＋3＝5；输入 3/2，结果为 3/2＝1(取整数)。

算法分析：要定义 3 个变量；两个整型变量，记录整数；一个字符型变量，记录运算符。然后根据运算符对两个整数执行相应的运算并输出结果。由于有'＋'、'－'、'＊'、'/' 4 个分支，可以使用 switch 语句来实现，也可以使用 if…else if… else if … else 语句来实现。

参考代码如下：

```
#include <iostream>
using namespace std;
int main()
{
    char op;
    int a, b, c =0;
    cin >>a >>op >>b;
    switch ( op )
    {
        case '+' :                          //如果运算符是字符常量'+',就执行后面的语句
            c =a +b;                        //将加法运算的结果赋给 c
            break;              //结束 switch 语句,若不写,则会继续向下执行 c=a-b;语句
        case '-':
            c =a -b;
```

```
            break;
        case '*':
            c =a * b;
            break;
        case '/':
            c =(b!=0) ? (a / b) : 0 ;          //除数取整,且 b 为 0 时结果为 0
            break;
        defalut:
            break;
    }
    cout <<a <<op <<b <<" =" <<c;
    return 0;
}
```

运行示例如下：

```
3+2
3+2 =5
```

说明：用户从键盘输入字符常量时，直接写字符本身，如＋，进入内存时是字符常量'＋'的 ASCII 码。进行条件判断时，要判断是否等于字符常量，如'＋'。而语句中的＋是加法运算符，要注意两者的区别。

本例的 switch 语句也可以用 if 语句实现。代码段如下：

```
if ( op =='+' )
    c =a +b;                                  //将加法运算的结果赋给 c
else if ( op =='-' )
    c =a -b;
else if ( op =='*' )
    c =a * b;
else if ( op =='/' )
    c =(b!=0) ? (a / b) : 0 ;                 //除数取整,且 b 为 0 时结果为 0
```

3.7.2　break 语句的合理使用

一般情况下，switch 的每个 case 中都有一条 break 语句，用来终止 switch 语句。如果程序中省略了 case 中的 break 语句，那么计算机在执行完一个 case 中的代码后，会继续执行下一个 case 中的代码。利用这个特点，在有些 case 后面不加 break 语句，使得同一段代码可以适用于多个 case 标记。

【例 3-9】 用 switch 语句改写例 3-6，价格如下：每支铅笔单价 2 元，购买量大于或等于 10 支且小于 50 支时对超出部分打 9.5 折，当购买量大于或等于 50 支时打 9 折。根据用户输入的铅笔数量计算顾客应付金额(保留到角)。

算法分析：case 后是一个常量值，不能是关系表达式或逻辑表达式，因此要对用户购买铅笔的数量进行分段，对于一个大于或等于 0 的数量 x，除以 10 取整之后，可能的取值

是 0,1,2,…,可以分段计算应付金额,如表 3-3 所示。

表 3-3 按铅笔数量分段计算应付金额

铅笔数量 x 的范围	顾客应付金额 y	x/10
x<0	0	负整数
0≤ x < 10	x * UNIT	0
10 ≤ x < 50	10 * UNIT + (x−10) * UNIT * 0.95	1,2,3,4
x ≥ 50	10 * UNIT + (50 − 10) * UNIT * 0.95 + (x − 50) * UNIT * 0.9	5,6,7,…

当 case 后面的值是 1、2、3、4 时,执行相同的计算 y 的公式,因此可以将语句写到第 4 个 case 的后面,形式为

```
case 1:
case 2:
case 3:
case 4: 语句; break;
```

这样,当 switch 中表达式的值为 1 时,跳到 case 1: 的行,开始向下执行,执行 case 4: 后面的语句,然后执行 break 退出 switch 语句;当 switch 中表达式的值为 2 时类似,跳到 case 2: 的行开始向下执行。

x<0 的值与 x>=50 的值范围太大。可以对 x<0 使用 if 语句判断。x>=50 时可以用 default 处理。

完整的代码如下:

```
#include <iostream>
#include <iomanip>
using namespace std;
const double UNIT =2;                          //每支单价为 2 元
int main()
{
    int x;
    double y =0;
    cout <<"请输入支数: " <<endl;
    cin >>x;
    if ( x <0 )
        y =0;
    else                                       //x 大于或等于 0 时计算价格
    {
        switch ( x / 10 )                      //计算数量除以 10 的可能值
        {
            case 0: y =x * UNIT; break;
            case 1:
```

```
            case 2:
            case 3:
            case 4: y =10 * UNIT +( x-10 ) * UNIT * 0.95; break;
            default: y =10 * UNIT +( 50 -10 ) * UNIT * 0.95 +( x -50 ) * UNIT * 0.
9 ;
        }
    }
    cout <<"应付款=" <<setiosflags(ios::fixed) <<setprecision(2)<<y ;
    return 0;
}
```

3.8　实用知识：生成随机数函数——rand 等函数

产生随机数常用的是 rand、srand 和 time 函数，见表 3-4。

表 3-4　生成随机数的常用函数

函数原型声明	功　能	头　文　件	说　明
int rand();	产生[0，RAND_MAX)区间的一个随机数	#include <cstdlib> using namespace std; 或 # include < stdlib.h>	RAND_MAX 定义在头文件中。通过%等数学运算可以生成指定范围内的随机数，如 rand()%10 生成 0～9 的随机数
void srand (unsigned int);	用来设置 rand 产生随机数时的随机数种子		相同的随机数种子产生的随机数是一样的，因此程序最开始一般用 srand 初始化一个不同的随机数种子，参数值一般是 time 函数返回值（当前系统时间）
time_t time (time_t *);	time(0)的返回值是从 1970 年 1 月 1 日至今所经历的时间（以秒为单位）	#include <ctime> using namespace std; 或 # include < time.h>	time(0)作为 srand 函数的参数，每次运行程序时 time(0)的值都不同，保证了随机数种子不同，进而保证 rand 函数的随机性

【例 3-10】 简单的两位数加法练习游戏。

执行程序时，显示器输出一个两位随机整数([10,99])的加法表达式，用户输入计算结果后，程序判断结果是否正确。

算法分析：程序的关键点是随机数的范围，随机数范围为[10,99]，共 90 个整数。首先用 rand()%(100－10)能得到 0～89 的一个随机整数，再加上 10，就得到 10～99 的整数。

代码如下：

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
```

```
const int MIN =10;                          //符号常量 MIN 保存随机数的最小值 10
const int MAX =99;                          //符号常量 MAX 保存随机数的最大值 99
int main()
{
    //产生[MIN,maxValue]间的随机数(包括 MIN 和 maxValue)
    srand((unsigned)time(0));               //依据系统时间初始化随机种子
    int num1 =MIN +rand() %( MAX -MIN +1 );
    int num2 =MIN +rand() %( MAX -MIN +1 );
    cout <<num1 <<"+" <<num2 <<"=";
    int val =0;
    cin >>val;
    if(num1+num2 ==val)
        cout <<"正确!";
    else
        cout <<"错误!";
    return 0;
}
```

运行两次,示例如下:

第 1 次(用户输入 24):

```
10+14=24
正确!
```

第 2 次(用户输入 80):

```
15+66=80
错误!
```

3.9 选择结构算法及应用

3.9.1 判断整数 *m* 是否能被 *n* 整除

【例 3-11】 输入两个整数 m 和 n,输出 m 是否能被 n 整除。

算法分析:如果 m%n 的结果为 0,则说明 m 能够被 n 整除,否则不能。

代码如下:

```
#include <iostream>
using namespace std;
int main()
{
    int m, n;
    cin >>m >>n;
    if ( m %n ==0 )
        cout <<m <<"能被" <<n <<"整除";
```

```
    else
        cout <<m <<"不能被" <<n <<"整除";
    return 0;
}
```

运行示例如下：

```
6 3
6 能被 3 整除
```

对除法运算一定要注意：如果 n 为 0，则 m%n 会出现异常，退出程序。例如运行程序时输入

```
5 0
```

得到的输出结果是

```
--------------------------------
Process exited after 3.599 seconds with return value 3221225620
请按任意键继续. . .
```

也就是说，没有执行到 main 函数的 return 0；函数返回了一个随机值。为了防止程序异常中止，要考虑到特殊数值的处理。代码修改如下：

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    int m,n;
    cin >>m >>n;
    if ( n ==0 )
        cout <<"除数不能为 0!";
    else if ( n !=0 && m %n ==0 )
        cout <<m <<"能被" <<n <<"整除!";
    else
        cout <<m <<"不能被" <<n <<"整除!";
    return 0;
}
```

3.9.2　判断一个浮点数的值是否等于 0

【例 3-12】 输入一个数给浮点型变量，写出判断变量是否为 0 的表达式。

算法分析：如果是一个整数 n，可以用关系运算符==组成表达式 n==0，如果表达式的值为 true 则说明 n 等于 0，如果表达式的值为 false 则说明 n 不等于 0。

一个浮点数 0.0 在内存中的值可能是 0.00000015，用关系运算符==的结果可能是 false，因为达不到运算符要求的精度。