第3章 基本流程控制结构

3.1 结构化程序设计

介绍结构化程序设计方法的基本思想以及C++的基本控制结构。

3.1.1 结构化程序设计

C++源程序由若干函数构成，而函数又是由语句构成的。对程序员来说，编写程序的一个主要内容就是如何将解决一个应用问题所使用的算法用C++的语句和函数来描述。换句话说，就是如何组织C++程序的结构。

结构化程序设计诞生于20世纪60年代，发展到20世纪80年代，已经成为当时程序设计的主流方法。它的产生和发展形成了现代软件工程的基础，也是目前流行的面向对象的程序设计方法的基础。结构化程序设计的基本思想是采用“自顶向下，逐步求精”的程序设计方法和“单入口单出口”的控制结构。自顶向下、逐步求精的程序设计方法从问题本身开始，经过逐步细化，将解决问题的步骤分解为由基本程序结构模块组成的结构化程序框图。“单入口单出口”的思想认为，一个复杂的程序，如果它仅是由顺序、选择和循环三种基本程序结构通过组合、嵌套构成，则构造的程序一定是一个单入口单出口的程序，据此就很容易编写出结构良好、易于调试的程序来。

结构化设计方法是以模块化设计为中心，将待开发的软件系统划分为若干个相互独立的模块，使每一个模块的工作变得单纯而明确，为设计较大的软件打下良好的基础。由于模块相互独立，因此在设计其中一个模块时，不会受到其他模块的牵连，因而可将原来较为复杂的问题简化为一系列简单模块的设计。模块的独立性还为扩充已有的系统、建立新系统带来了不少的方便。按照结构化设计方法设计出的程序具有结构清晰、可读性好、易于修改和容易验证的优点。C++是一种支持结构化程序设计思想的程序设计语言，使用C++编写程序时，应该遵循结构化程序设计方法。

在结构化程序设计方法中，模块是一个基本概念。一个模块可以是一条语句、一段程序、一个函数等。在流程图中，模块用一个矩形框表示，如图3.1所示。模块的基本特征是其仅有一个入口和一个出口，即要执行该模块的功能，只能从该模块的入口处开始执行，执行完该模块的功能后，从模块的出口转向执行其他模块的功能。即使模块中包含多条语句，也不能随意从其他语句开始执行，或提前退出模块。

图3.1 程序模块

3.1.2 基本控制结构

按照结构化程序设计的观点，任何算法功能都可以通过由程序模块组成的三种基本程序结构的组合来实现。

(1) 顺序结构由两个程序模块串接构成，如图 3.2 左部虚线框所示，这两个程序模块是顺序执行的，即首先执行“程序模块 1”，然后执行“程序模块 2”。顺序结构中的两个程序模块可以合并成一个等效的程序模块，即将图 3.2 中左部虚线框部分整个看成一个程序模块，如图 3.2 的右部所示。通过这种方法，可以将许多顺序执行的语句合并成一个比较大的程序模块。

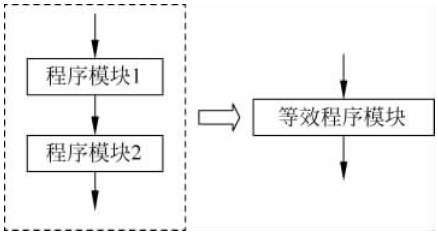

图 3.2 顺序结构

顺序结构是最常见的程序结构形式，在一般程序中大量存在。但并不是所有程序都可以只使用顺序结构编写。在求解实际问题时，常常要根据输入数据的实际情况进行逻辑判断，对不同的判断结果分别进行不同的处理；或者需要反复执行某些程序段，以避免多次重复编写结构相似的程序段带来程序结构上的臃肿。这就需要在程序中引入选择结构和循环结构。一个结构化程序正是由这三种基本程序结构交替综合而构成的。

(2) 选择结构如图 3.3 左部虚线框所示。从图中可以看出，根据逻辑条件成立与否，分别选择执行“程序模块 1”或者“程序模块 2”。虽然选择结构比顺序结构稍微复杂了一点，但是仍然可以将其整个作为一个等效的程序模块，如图 3.3 右部所示。一个入口 (从顶部进入模块开始判断)，一个出口(无论执行了“程序模块 1”还是“程序模块 2”，都应从选择结构框的底部出去)。在编程过程中，还可能遇到选择结构中的一个分支没有实际操作的情况，如图 3.4 左部虚线框所示。这种形式的选择结构可以看成是图 3.3 中的选择结构的特例。

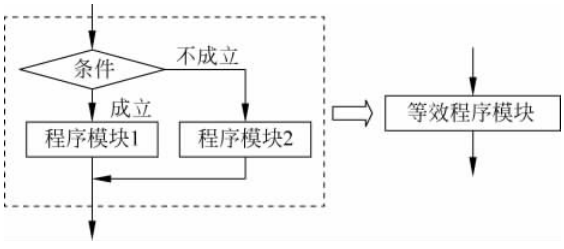

图 3.3 选择结构

(3) 循环结构如图 3.5 左部虚线框所示。在进入循环结构后首先判断条件是否成立，如果成立则执行“程序模块”，反之则退出循环结构。执行完“程序模块”后再去判断条件，如果条件仍然成立则再次执行内嵌的“程序模块”，循环往复，直至条件不成立时退出循环结

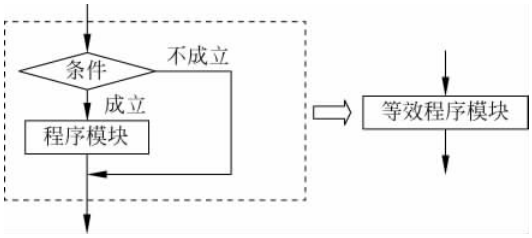

图 3.4　单分支选择结构

构。循环结构也可以抽象为一个等效的程序模块,如图 3.5 右部所示。图 3.5 中的循环结构可以描述为“当条件成立时反复执行程序模块”,故又称为当型循环(while 型循环)结构。

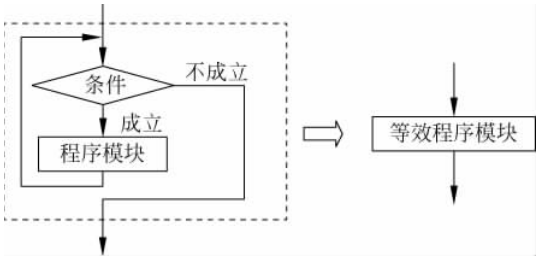

图 3.5　当型循环结构

除了当型循环外,还有直到型循环(do-while 型循环)结构,如图 3.6 所示,其特点是进入循环结构后首先执行“程序模块”,然后再判断条件是否成立,如果成立则再次执行“程序模块”,直到条件不成立时退出循环结构。

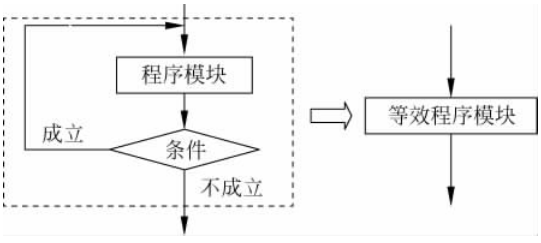

图 3.6　直到型循环结构

3.1.3　C++语言的语句分类

语句是组成 C++程序的基本单元,其标志是结束时以分号“;”结束。分号是一个 C++语句的组成部分。C++语句可以分为以下六类。

1) 说明语句

在 C++中,把完成对数据结构的定义和描述、对变量的定义性说明统称为说明语句,它可放在函数中允许出现语句的任何位置,也可以放在函数定义之外。例如:

```
int  m;                    //说明语句,定义了整型变量 m
```

2）控制语句

完成一定流程控制功能的语句，即有可能改变程序执行顺序的语句称为控制语句。控制语句包括选择结构的实现语句（条件语句、开关语句、分支选择语句）、循环结构的实现语句（循环语句）、转向语句、从函数中返回语句等。

3）函数调用语句

在一次函数的调用后加上一个分号所构成的语句，称为函数调用语句。例如：

```
sin(x);                    //函数调用语句
```

4）表达式语句

在任一表达式的后面加上一个分号，就构成一个表达式语句。例如：

```
i = i + 1;                 //表达式语句
```

5）空语句

只由一个分号所构成的语句称为空语句，它不执行任何操作。

6）复合语句

用花括号{ }把一条或多条语句括起来后构成一个语句，称为复合语句。它可以出现在只允许出现一个语句的任何位置。花括号是C++中的一个标点符号，左花括号“{”标明了复合语句的起始位置，右花括号“}”标明了复合语句的结束，所以右花括号后边的分号就不需要了。复合语句主要用于控制语句中。

3.2 选择结构语句

3.2.1 if 语句

if语句也称为条件语句，它的功能是用来判定由表达式所表达的条件是否满足，根据判定的结果（真或假）决定执行哪个操作。C++中提供了三种形式的if语句，其格式如下：

1）单选条件语句格式

```
if(表达式) 语句
```

这种单选if语句的执行过程见图3.4，执行时首先求出表达式的值，若表达式的值不等于0，则执行“语句”，否则，跳过该if语句，直接执行后继的语句。例如：

```
if(x > y)   cout << x <'\n';
```

执行该语句时，当 x > y 时，屏幕上显示 x 的值；当 x≤y 时，该语句不被执行。

2）二选一条件语句格式

```
if(表达式) 语句 1
else 语句 2
```

这种二选一条件语句的执行过程见图3.3，执行时首先求出表达式的值，若表达式的值不等于0，则执行“语句1”，否则，执行“语句2”。例如：

```
if(x > y) cout << x <'\n';
```

```
else cout << y <<'\n';
```

执行该语句时,当 x > y 时,屏幕上显示 x 的值;当 x≤y 时,屏幕上显示 y 的值。

3) 多选一条件语句格式

```
if(表达式 1) 语句 1
else if(表达式 2) 语句 2
else if(表达式 3) 语句 3
⋮
else if(表达式 n) 语句 n
else 语句 m
```

这种多选一条件语句的执行过程见图 3.7,执行时首先求出表达式 1 的值,若表达式 1 的值不等于 0,则执行"语句 1";若表达式 1 的值等于 0,则再求表达式 2 的值,若表达式 2 的值不等于 0,则执行"语句 2";若表达式 2 的值等于 0,则再求表达式 3 的值……以此类推,直至"语句 m"。根据实际情况,最后的"else 语句 m"也可省略。

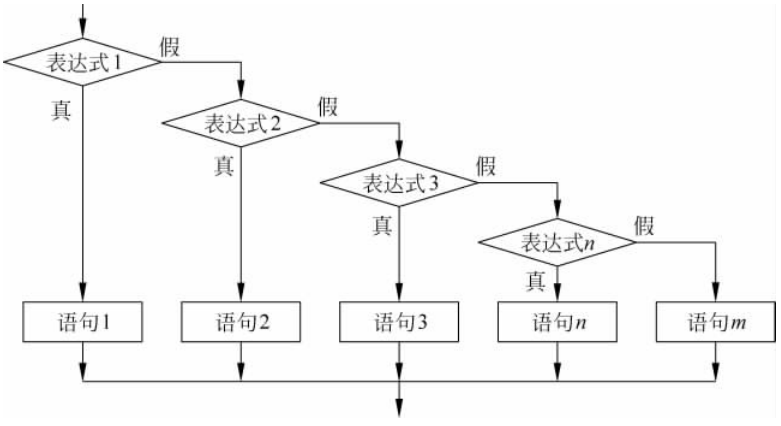

图 3.7　多选一条件语句执行流程

说明:

(1) 三种形式的 if 语句中的"表达式"可以是符合 C++语法规则的任一表达式,一般为逻辑表达式、关系表达式或算术表达式。

(2) 三种形式的 if 语句中的"语句"可以是一个单一语句,也可以是一个复合语句。通常把该"语句"称为条件语句的内嵌语句。

(3) 第二、第三种形式的 if 语句中,在每个 else 前面有一分号,整个语句结束处有一个分号。例如:

```
if(x>0)
  cout<<x<<'\n';
else
  cout<<(-x)<<'\n';
```

各有一个分号

这是由于分号是 C++语句中不可缺少的部分,这个分号是 if 语句中的内嵌语句所要求的。如果内嵌语句是复合语句,则复合语句的结束符"}"也作为内嵌语句的结束符,不需要再添加分号了。例如:

```
配对使用
if(delta)
{ cout<<"方程有两个不同的实根:\n";
  cout<<"x1="<<(-b+delta)/2/a<<'\n';
  cout<<"x2="<<(-b-delta)/2/a<<'\n';
}
else
{ cout<<"方程有两个相等的实根:\n";
  cout<<"x1=x2="<<-b/2/a<<'\n';
}
```
没有分号

(4) if 语句中,if 后内嵌语句(又称为 if 子句)与 else 后内嵌语句(又称为 else 子句)不要误认为是两个语句,它们都属于同一个 if 语句。else 子句不能作为语句单独使用,它必须是 if 语句的一部分,与 if 配对使用。

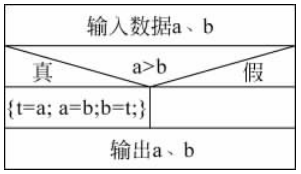

图 3.8　由小到大依次输出两个数

【例 3.1】 输入两个实数,按代数值由小到大依次输出这两个数。

算法分析:先输入两个数据,如果前一个数据大于后一个数据,则将两个数据互换,最后依次输出两个数据。注意,互换两个数据要用到第三个数据变量作为桥梁,否则两个数据相互覆盖,达不到互换的目的。其算法流程图如图 3.8 所示。

源程序如下:

```
#include<iostream>
using namespace std;
int main()
{   float a,b,t;                          //定义变量
    cout<<"请输入两个实数:\n";            //在屏幕上输出提示信息
    cin>>a>>b;                            //给变量赋值
    if(a>b)                               //如果前一个变量大于后一个变量
    {   t=a; a=b; b=t; }                  //利用第三个变量交换数据
    cout<<a<<'\t'<<b<<endl;               //依次输出变量
    return 0;
}
```

程序的运行情况如下:

```
请输入两个实数:
7  5.4↙
5.4      7
```

其中,if 条件成立后执行的内嵌语句是复合语句,一定要用花括号“{}”括起,如果没有花括号“{}”,那么 if 的控制范围只到条件表达式后的第一个分号处。

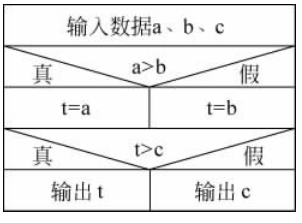

图 3.9　求三个数中的大者

【例 3.2】 从键盘上输入三个整数,输出其中的最大数。

算法分析:定义三个整型变量 a、b、c,分别表示从键盘上输入的三个整数。先比较 a 与 b,求出其大者,然后再将该数与 c 比较,求出二者中的大数,该大数就是三个整数中的最大数。其算法流程图如图 3.9 所示。

源程序如下：

```
#include<iostream>
using namespace std;
int main()
{   int a, b, c, t;
    cout<<"输入三个整数:  ";
    cin>>a>>b>>c;
    cout<<"a="<<a<<'\t'<<"b="<<b<<'\t'<<"c="<<c<<'\n';
    if(a>b) t=a;
    else  t=b;
    cout<<"最大数是:";
    if(t>c) cout<<t<<'\n';
    else cout<<c<<'\n';
    return 0;
}
```

程序的运行情况如下：

```
输入三个整数: 2  1  5↙
a=2    b=1    c=5
最大数是:5
```

【例 3.3】 判断从键盘输入字符的种类。字符可分为五类：数字、大写字母、小写字母、控制字符(ASCII 的编码小于 32)和其他字符。

算法分析：从键盘输入的某字符属于五种类型中的一种，因此可用多选一条件语句实现分类。其算法流程图如图 3.10 所示。

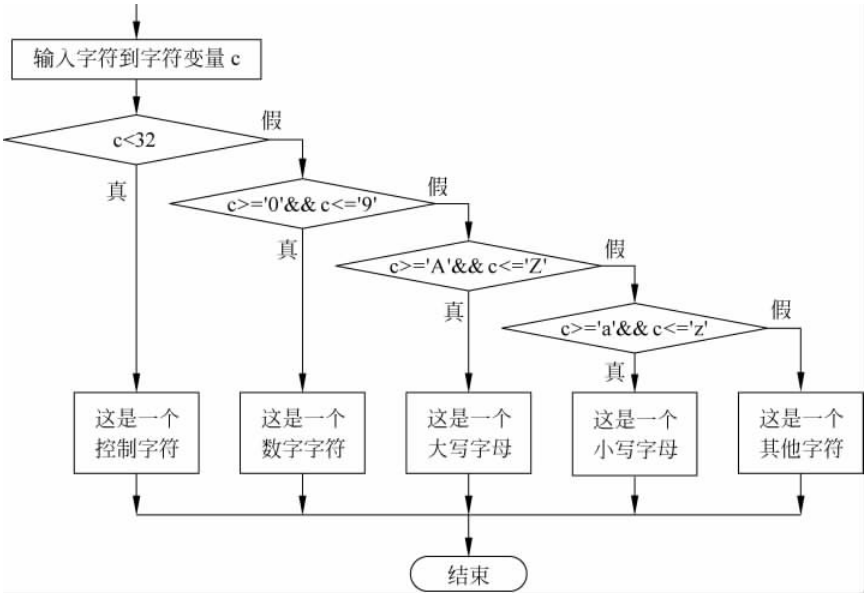

图 3.10　判断输入字符的种类

源程序如下：

```
#include<iostream>
using namespace std;
int main()
{   char c;
    cout<<"输入一个字符:";
    cin.get(c);                              //A
    if(c<32)
        cout<<"这是一个控制字符。\n";
    else if(c>='0'&&c<='9')
        cout<<"这是一个数字字符。\n";
    else if(c>='A'&&c<='Z')
        cout<<"这是一个大写字母。\n";
    else if(c>='a'&&c<='z')
        cout<<"这是一个小写字母。\n";
    else  cout<<"这是一个其他字符。\n";
    return 0;
}
```

程序的运行情况(运行三次)如下：

```
输入一个字符: a↙
这是一个小写字母。
```

```
输入一个字符: ↙
这是一个控制字符。
```

```
输入一个字符: +↙
这是一个其他字符。
```

本例程序中 A 行使用 cin.get(c)可接收键盘输入的任意字符，包括空格和回车键。若使用 cin >> c 输入方式，则不能接收键盘输入的空格和回车字符。

【例 3.4】 编程求一元二次方程 $ax^2+bx+c=0$ 的解，系数 a、b、c 从键盘上输入。

算法分析：方程系数应为实型数据，输入系数 a、b、c 的值后，若 $b^2-4ac<0$ 时，则方程无实根；若 $b^2-4ac>0$ 时，则方程存在两个不等的实根；$b^2-4ac=0$ 时，则方程存在两个相等的实根。其算法流程图如图 3.11 所示。

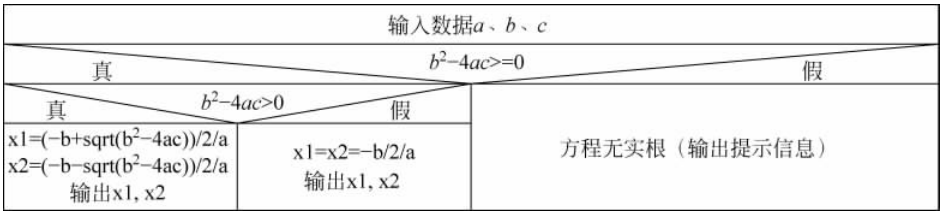

图 3.11　求一元二次方程的根

源程序如下：

```
#include<iostream>
#include<cmath>                          //A
using namespace std;
int main()
{   float a, b, c, delta;
    cout<<"输入三个系数:";
    cin>>a>>b>>c;
    cout<<"a = "<<a<<'\t'<<"b = "<<b<<'\t'<<"c = "<<c<<'\n';
    delta = b*b-4*a*c;
    if(delta<0)
        cout<<"方程无实根!\n";
    else{                                //B
        delta = sqrt(delta);             //C
        if(delta){                       //D
            cout<<"方程有两个不等的实根:\n";
            cout<<"x1 = "<<(-b+delta)/2/a<<'\n';
            cout<<"x2 = "<<(-b-delta)/2/a<<'\n';
        }
        else{                            //E
            cout<<"方程有两个相等的实根:\n";
            cout<<"x1 = x2 = "<<-b/2/a<<'\n';
        }
    }
    return 0;
}
```

程序的运行情况如下：

```
输入三个系数: 1 3 2↙
a = 1     b = 3     c = 2
方程有两个不等的实根:
x1 = -1
x2 = -2
```

在程序中，B行的else分支使用了复合语句，并且在复合语句中又包含了二选一条件语句(D、E行)，这是if语句的嵌套。使用嵌套的if语句时，应注意if与else的配对关系。else总是与它上面最近的、未配对的if配对，如E行的else就与D行的if配对。程序中C行用到了开平方函数sqrt()，这是一个C++的库函数，它返回的开平方根值是一个double类型的双精度实数。当使用到C++提供的常用数学函数时，要包含头文件cmath，如程序中的A行所示。有关函数的用法，将在第4章介绍。D行中if(delta)等价于if(delta!=0)。

3.2.2 条件运算符"?:"

若在if语句中，当被判别的表达式的值为"真"或"假"时，都执行一个赋值语句且向同一个变量赋值时，可以用一个条件运算符来处理。例如，有以下if语句：

```
if(a>b)
```

```
    max = a;
else
    max = b;
```

其功能是：当 a > b 时，将 a 的值赋给 max；当 a≤b 时，将 b 的值赋给 max。可以看到，无论 a > b 是否满足，都是向同一个变量 max 赋值。在这种情况下，也可以用下面的条件运算符来处理：

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

其中，“(a > b) ? a : b”是一个由条件运算符构成的“条件表达式”。它是这样执行的：如果(a > b)条件为真，则条件表达式取值 a，否则取值 b。

条件运算符使用格式为：

表达式 1? 表达式 2 : 表达式 3

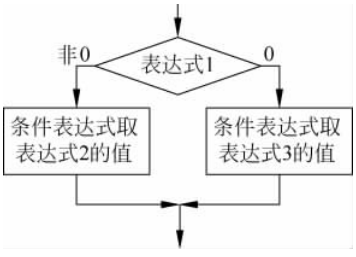

图 3.12　条件表达式执行流程

条件运算符是 C++语言中唯一的一个要求有三个操作数对象的运算符，称为三目(元)运算符，其中的三个表达式可以是任意的符合 C++语法规则的表达式。条件表达式的执行过程见图 3.12，运算时先求出表达式 1 的值，若其值不等于 0 时，则求出表达式 2 的值(不求表达式 3 的值)，并把该值作为运算的结果；若表达式 1 的值等于 0 时，则求出表达式 3 的值(不求表达式 2 的值)，并把它作为运算的结果。

说明：

(1) 条件运算符“? :”的优先级仅高于赋值运算符、复合赋值运算符和逗号运算符，而低于其他的算术、逻辑、关系等运算符。因此

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

中括号可以不要，可以直接写成：

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

同样，如果有语句

```
max = a>b ? a : b+1 ;
```

则它等效于

```
max = a>b ? a : ( b+1 ) ;
```

而不等效于

```
max = ( a>b ? a : b ) +1 ;
```

(2) 条件运算符的结合方向为“自右至左”。例如，表达式

```
a>b ? a : c>d ? c : d
```

相当于

```
a>b?a:(c>d?c:d)
```

(3) 条件运算符的三个表达式的类型可不同,此时条件表达式取转换级别较高的类型(见图 2.16)。例如:

```
cout<<(3>2?'a':20);
```

此时,程序输出 97 而不是 a,因为表达式 3 为 20 是整型,比字符'a'的类型转换级别高,故将字符'a'转换成整型。

【例 3.5】 输入一个字符,判别它是否是大写字母,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。

算法分析:关于大小写字母之间的转换方法,在本书例 2.8 中已做了介绍,因此,可直接编写程序。

```
#include<iostream>
using namespace std;
int main()
{   char ch;
    cout<<"请输入一个字符:  ";
    cin>>ch;
    ch=(ch>='A'&&ch<='Z')?(ch+32):ch;
    cout<<ch<<'\n';
    return 0;
}
```

程序的运行情况如下:

```
请输入一个字符:  Y↙
y
```

程序中条件表达式"(ch>='A'&&ch<='Z')?(ch+32):ch"的作用是:如果字符变量 ch 的值为大写字母,则条件表达式的值为(ch+32),即相应的小写字母,32 是小写字母和大写字母 ASCII 码的差值。如果 ch 的值不是大写字母,则条件表达式的值为 ch,即不进行转换。

3.2.3 switch 语句

switch 语句又称为开关语句,它是一种多分支选择语句,其格式为:

```
switch(表达式)
{
    case 常量表达式1:   语句1
    case 常量表达式2:   语句2
    …
    case 常量表达式n:   语句n
    default:            语句n+1
}
```

其中,switch 后的表达式可以是任意的符合 C++语法规则的表达式,但其值只能是整型

或字符型。各常量表达式只能由常量组成，其值也只能是整型或字符型。case 后的语句均是任选的，它可以由一个或多个语句组成。default 子句也可省略。

switch 语句的执行过程是：先计算表达式的值，然后将其顺序地与 case 后面的常量表达式进行比较，若表达式的值与常量表达式的值相等，就由此开始进入相应的 case 后的语句执行，然后依次执行其后每个 case 后的语句，遇到 case 和 default 也不再进行判断，直至 switch 语句结束。如果要使其在执行完相应的语句后中止执行下面的语句，则应在相应语句后加 break 语句，以使流程跳出 switch 语句。

【例 3.6】 编写程序，将百分制的学生成绩转换为优秀、良好、中等、及格和不及格五级制成绩。标准如下。

优秀：90～100 分；

良好：80～89 分；

中等：70～79 分；

及格：60～69 分；

不及格：60 分以下。

算法分析：使用 switch 语句构成的多分支选择结构编写程序。成绩转换时是根据分数范围进行的。因此，构造一个整型表达式 grade/10 用于将分数段化为单个整数值。例如，对于分数段 60～69 中的各分数值，表达式的值均为 6。用该整数值配合 switch 语句中各 case 后常量表达式，并灵活运用 break 语句，即可编写出所需转换程序。程序如下：

```
#include<iostream>
using namespace std;
int main()
{   int grade;                          //百分制成绩
    cout<<"输入百分制成绩：";
    cin>>grade;
    cout<<"五级制成绩：";
    switch ( grade/10 )
    {
    case 10:                            //A
    case 9: cout<<"优秀\n";
          break;
    case 8: cout<<"良好\n";               //B
          break;                        //C
    case 7: cout<<"中等\n";
          break;
    case 6: cout<<"及格\n";
          break;
    default : cout<<"不及格\n";
    }
    return 0;
}
```

程序的运行情况如下：

```
输入百分制成绩：89↙
五级制成绩：良好
```

说明：

(1) switch 语句中“case 常量表达式”只起语句标号作用，并不是在该处进行条件判断。在执行 switch 语句时，根据 switch 后面表达式的值找到匹配的入口标号，就从此标号开始执行下去，不再进行判断。若遇到 break 语句，则终止程序流程，立即退出 switch 语句。如执行例 3.5 程序时输入 89，则表达式 grade/10 的值为 8，此值与 case 8 相匹配，因此执行 B、C 行后程序就结束了。

(2) 每一个 case 的常量表达式的值必须互不相同，否则就会出现互相矛盾的现象(对表达式的同一个值，有两种或多种执行方案)。

(3) 当省略 case 后面的语句时，则可实现多个入口共用一组语句。例如，例 3.6 中，若输入 100，则 grade/10 的值等于 10，从 case 10 进入执行程序。此时 A 行中没有语句，则顺序执行 case 9 后两条语句，即 case 10 和 case 9 共用了语句组“cout <<"优秀\n"; break;”。

(4) 各个 case 和 default 出现的前后次序不影响执行结果。

【例 3.7】 指出下面程序段中的错误。

```
float x = 2.5;
int a,b;
a = 3; b = 4;
switch ( x * 2 )                    //D
{
case 2.5 :  ...                     //E
case a + b :  ...                   //F
case 1 , 2 , 3 : ...                //G
}
```

解：该程序段中多处存在不符合语法规则的现象。D 行中的表达式的值为实数，这是不允许的，但写成(int)(x * 2)就符合语法规则了。E 行中 case 后的常量表达式的值是实型，不符合语法规则。F 行中 case 后的表达式不是常量表达式，也不符合语法规则。而 G 行中 case 后的表达方式也是不允许的。若要表达共用同一语句，则 G 行应改写成：

```
case 1 :
case 2 :
case 3 :  ...
```

注意：在实际应用中，任一 switch 语句均可以用条件语句来实现，但并不是任何条件语句均可用 switch 语句来实现。这是因为 switch 语句中限定了表达式的取值类型为整型或是字符型，而条件语句中的条件表达式可取任意类型的值。

【例 3.8】 设计程序，实现简单的计算器功能，即能完成加、减、乘和除算术运算。如输入：

```
2.5 * 4
```

则输出为

```
2.5 * 4 = 10
```

算法分析：运算符为字符类型，取值为＋、－、*、/，可用 switch 语句中四个 case 来区分这四种情况。输入两个操作数及运算符后，根据运算符的种类，利用相应 case 后的语句

完成相应的运算。源程序如下：

```
#include <iostream>
using namespace std;
int main()
{   float data1,data2;
    char op;
    cout <<"输入数据,格式为: 操作数 1  运算符  操作数 2 : \n";
    cin >> data1 >> op >> data2;              //从键盘输入的运算符存放在字符型变量 op 中
    switch (op )
    {
    case '+' : cout << data1 << op << data2 <<" = "<< data1 + data2 <<'\n';
           break;
    case '-' : cout << data1 << op << data2 <<" = "<< data1 - data2 <<'\n';
           break;
    case '*' : cout << data1 << op << data2 <<" = "<< data1 * data2 <<'\n';
           break;
    case '/' : if(data2)
                  cout << data1 << op << data2 <<" = "<< data1/data2 <<'\n';
               else cout <<"除数为 0!\n";
           break;
    default : cout << op <<"是一个无效的运算符!\n";
    }
    return 0;
}
```

程序的运行情况如下：

```
输入数据,格式为: 操作数 1  运算符  操作数 2:
2.5 * 4↙
2.5 * 4 = 10
```

用条件语句也可实现例 3.8 中的功能。如下面程序：

```
#include <iostream>
using namespace std;
int main()
{   float data1,data2;
    char op;
    cout <<"输入数据,格式为: 操作数 1  运算符  操作数 2 \n";
    cin >> data1 >> op >> data2;
    if(op == '+')
        cout << data1 << op << data2 <<" = "<< data1 + data2 <<'\n';
    else if(op == '-')
        cout << data1 << op << data2 <<" = "<< data1 - data2 <<'\n';
    else if(op == '*')
        cout << data1 << op << data2 <<" = "<< data1 * data2 <<'\n';
    else if(op == '/'){
        if(data2)
            cout << data1 << op << data2 <<" = "<< data1/data2 <<'\n';
        else cout <<"除数为 0!\n";
```

```
    }
    else cout << op <<"是一个无效的运算符!\n";
    return 0;
}
```


3.3 循环结构语句

循环结构是结构化程序设计的基本结构之一,用它来实现在某一条件成立时重复执行某一些操作的功能。例如,求 1～100 之间整数之和:

$$s=1+2+3+\cdots+100$$

显然,在程序中不可能依次列出 1～100 个数,要完成以上的求和,可按以下步骤操作:

(1) 给整型变量 s 赋初值 0,变量 i 赋初值 1;

(2) 令 s=s+i,i=i+1;

(3) 若 i≤100,则重复步骤(2);

(4) 输出 s 的值。

在以上步骤中,步骤(2)和(3)是要重复执行的操作。把这种重复执行的操作称为循环体。i≤100 是重复执行的条件,称为循环条件。C++语言提供了三种实现循环结构的语句:while 语句、do-while 语句和 for 语句。

3.3.1 while 语句

while 语句可以实现“当型”循环结构。其一般格式为:

```
while ( 表达式 ) 语句
```

其中,表达式是循环条件,它可以是 C++中任一符合语法规则的表达式。“语句”为循环体,可以是 C++中的任一语句。while 语句的执行过程是:先计算表达式的值,若表达式的值不等于 0,则执行“语句”,再计算表达式的值,重复以上过程,直到表达式的值等于 0 为止,执行流程见图 3.13。

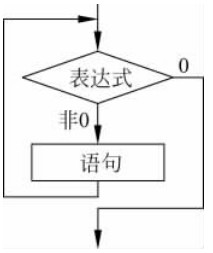

图 3.13 while 语句执行流程

【例 3.9】 求 1～100 之间整数之和 $s=1+2+3+\cdots+100$。

算法分析:按照 3.3 节给出的 4 个步骤,程序如下。

```
#include < iostream >
using namespace std;
int main()
{   int i = 1,s = 0;                          //循环变量赋初值
    while(i < = 100)                          //循环条件
    {   s = s + i;
        i++;                                  //改变循环变量
    }
    cout <<"s = "<< s <<'\n';
    return 0;
}
```

程序的运行结果如下：

```
s = 5050
```

说明：

(1) while 语句的执行是先判断循环条件后执行循环体，所以循环体可能执行若干次，也可能一次都不执行。例如：

```
int n = 0;
while (n)   cout <<" *** "<<'\n';
```

该程序段中由于循环条件为假，循环体不被执行，所以屏幕上没有显示“ *** ”。

(2) 在循环体中或表达式内应有使循环趋向于结束的成分。例如，本例中循环结束的条件是“i > 100”，因此在循环体中应该有使 i 增值以最终导致 i > 100 的语句，例中用“i＋＋；”语句来达到此目的。如果无此语句，则 i 的值始终不改变，循环永不结束(称为死循环)。

(3) 当循环体有多个语句时，必须用{}把循环体括起来构成一个复合语句。如本例中，循环体应执行两条语句“s＝s＋i；i＋＋；”，故将这两条语句用{}括起。如果没有{}，则 while 语句只能控制到其后的第一个分号处。

【例 3.10】 编程求表达式 $s=1+\frac{1}{3}+\frac{1}{5}+\frac{1}{7}+\cdots+\frac{1}{99}$。

算法分析：与例 3.9 类似，该题也是求多个数据的累加和，即为“s＝s＋t;”的形式，但通项 t 的形式与例 3.9 不同，是一个分母为 n，分子为 1 的浮点数，所以该题的关键是找出通项 t 的迭代规则。也就是说，在每次循环操作中，用前面一项推导出后面一项。具体地说，就是用$\frac{1}{3}$推导出$\frac{1}{5}$，用$\frac{1}{5}$推导出$\frac{1}{7}$……直到达到循环结束的要求为止。源程序如下：

```
#include < iostream >
using namespace std;
int main()
{   int i = 1;
    float s = 0,t;                          //累加和为浮点数
    while(i < 100)
    {   t = 1.0/i;                          //通项的表示
        i = i + 2;                          //分母进行迭代
        s = s + t;                          //计算累加和
    }
    cout <<"s = "<< s <<'\n';
    return 0;
}
```

程序的运行结果如下：

```
s = 2.93778
```

【例 3.11】 用公式 $\pi/4=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots$ 求 π 的近似值，直到最后一项的绝对值小

于 10^{-6} 为止。

算法分析：本题与例 3.10 类似，求多个数据的累加和，但是通项 t 的符号为一正一负，即每一项数据的符号都与前一项相反。在通项迭代时，设置一个符号标志 sign＝1，使其在每次循环中都乘以－1，从而达到数据项一正一反的目的。源程序如下：

```
#include<iostream>
#include<cmath>
using namespace std;
int main()
{   int i=1,sign=1;                    //设置符号标志
    float s=0,t=1;                     //累加和为浮点数
    while(fabs(t)>1e-6)
    {   t=sign*1.0/i;                  //通项的表示
        sign=-sign;                    //符号取反
        i=i+2;                         //分母进行迭代
        s=s+t;                         //计算累加和
    }
    cout<<"pi="<<4*s<<'\n';
    return 0;
}
```

程序的运行结果如下：

```
pi=3.1416
```

3.3.2 do-while 语句

do-while 语句可以实现“直到型”循环结构，其一般格式为：

```
do
    循环体语句
while( 表达式 );
```

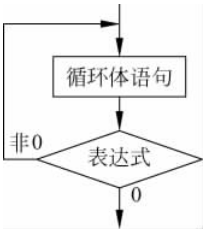

图 3.14　do-while 语句执行流程

其中，表达式是循环条件，可以是 C++中任一符合语法规则的表达式。循环体语句可以是 C++中的任一语句。do-while 语句的执行过程是：先执行循环体语句，然后再计算表达式的值，若表达式的值等于 0，则退出 do-while 语句；若表达式的值不等于 0，则继续执行循环体语句，直到表达式的值等于 0 时为止。因此，do-while 循环至少要执行一次循环体语句。图 3.14 给出了该语句的执行流程图。

【例 3.12】 用 do-while 语句编程，求 1～100 之间整数之和 $s=1+2+3+\cdots+100$。

源程序如下：

```
#include<iostream>
using namespace std;
int main()
```

```
{    int i = 1,s = 0;
     do
     {    s = s + i;
          i++;
     }while ( i <= 100 ) ;                    //注意 while 条件后的分号不能少
     cout <<"s = "<< s <<'\n';
     return 0;
}
```

程序的运行结果如下：

```
s = 5050
```

说明：

(1) 一般情况下，用 while 语句和 do-while 语句处理同一问题时，若二者的循环体部分是一样的，它们的结果也一样。如例 3.9 和本例程序中的循环体是相同的，得到结果也相同。但是如果 while 后面的表达式一开始就为假(0 值)时，两种循环的结果则不同。

(2) do-while 语句中 while 后面的分号“;”不可缺少，否则，会出现语法错误。

【例 3.13】 用迭代法求 $x=\sqrt{a}$ 的近似值。求平方根的迭代公式为

$$x_{n+1}=\frac{1}{2}\left(x_n+\frac{a}{x_n}\right)$$

要求前后两次求出的 x 的差的绝对值小于 10^{-5}。

算法分析：这是典型的迭代算法。

首先给定一个 a 的平方根的初值 $x_0=a$ 或 $x_0=a/2$，然后根据上述公式求解：

$x_1=\frac{1}{2}\left(x_0+\frac{a}{x_0}\right)$，若 $|x_1-x_0|>10^{-5}$，继续利用公式求解：

$x_2=\frac{1}{2}\left(x_1+\frac{a}{x_1}\right)$，若 $|x_2-x_1|>10^{-5}$，继续利用公式求解：

$x_3=\frac{1}{2}\left(x_2+\frac{a}{x_2}\right)$，若 $|x_3-x_2|>10^{-5}$，继续利用公式求解：

……直到

$x_{n+1}=\frac{1}{2}\left(x_n+\frac{a}{x_n}\right)$，当 $|x_{n+1}-x_n|<10^{-5}$ 时，x_{n+1} 就是 a 的平方根。

编写程序完成这个计算。因为每次迭代的算法是一致的，都是将前一次求解出的结果作为本次迭代的初值，再根据公式求出新的结果。因此，仅使用两个变量就可以描述迭代过程。在利用公式求解新值前，将前一次计算的结果 x_1 作为本次计算的初值 x_0，再根据公式求出新的 x_1，也就是说，不断重复以下的循环体：

```
{    x0 = x1;                            //将 x1 作为新的 x0
     x1 = (x0 + a/x0)/2;                 //求出新的 x1
}
```

直到满足 $|x_1-x_0|<10^{-5}$ 为止。源程序如下：

```
#include < iostream >
```

```
#include<cmath>                          //A
using namespace std;
int main()
{   float x0,x1,a;
    cout<<"输入一个正数:";
    cin>>a;
    if ( a<0 )  cout<<a<<"不能开平方!";
    else {   x1=a/2;                     //迭代初值
             do {  x0=x1;
                   x1=(x0+a/x0)/2;
             }while ( fabs( x1-x0 )>1e-5 );  //B
             cout<<a<<"的平方根等于: "<<x1<<'\n';
    }
    return 0;
}
```

程序的运行情况如下:

```
输入一个正数:3↙
3的平方根等于: 1.73205
```

注意:程序中B行用到了C++提供的数学库函数fabs( ),其功能是求实数的绝对值,使用时要包含数学库文件cmath,如A行所示。

3.3.3 for语句

在C++语言中,for语句使用最为灵活,它可以取代前两种循环语句。for循环语句的一般格式为:

for (表达式1 ; 表达式2 ; 表达式3) 循环体语句

其中,三个表达式都可以是C++中的任一符合语法规则的表达式,循环体语句可以是任一C++的语句。for语句的执行流程见图3.15,其执行过程如下:

(1) 先求解表达式1;

(2) 求解表达式2,若其值为非0,则执行循环体语句,然后执行下面步骤(3);若其值为0,则结束循环,转到步骤(5)。

(3) 求解表达式3。

(4) 转回步骤(2)继续执行。

(5) 循环结束,执行for语句下面的语句。

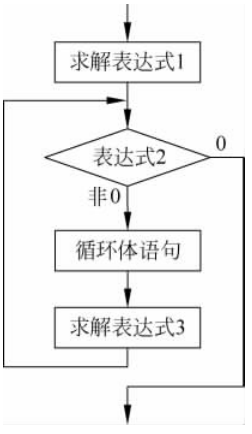

图3.15 for语句执行流程

【例3.14】 用for语句编程,求1～100之间整数之和$s=1+2+3+\cdots+100$。

源程序如下:

```
#include<iostream>
using namespace std;
int main()
{   int i , s =0 ;
```

```
    for ( i = 1 ; i <= 100 ; i++)
       s = s + i;
    cout <<"s = "<< s <<'\n';
    return 0;
}
```

程序的运行结果如下：

```
S = 5050
```

显然，从运行结果看，例 3.9、例 3.12 和本例是一样的。

说明：

(1) for 语句中三个表达式的功能如下：

for (循环变量赋初值 ; 循环条件 ; 循环变量增量) 循环体语句

其中，循环变量赋初值(表达式 1)用来给循环控制变量赋初值；循环条件(表达式 2)决定什么时候退出循环；循环变量增量(表达式 3)规定循环控制变量每循环一次后按什么方式变化。这三个部分之间必须用分号“;”分开。

(2) for 循环语句中的三个表达式都是可选择项，即可以缺省，但分号“;”不能缺省。也就是说，for 语句中必须有两个分号。

(3) 省略“表达式 1”，表示不对循环控制变量赋初值(此前循环控制变量已经赋过初值时，可以省略“表达式 1”)。如例 3.14 中程序可写为：

```
#include <iostream>
using namespace std;
int main()
{   int i = 0 , s = 0 ;
    for (  ; i <= 100 ; i++)
         s = s + i;
    cout <<"s = "<< s <<'\n';
    return 0;
}
```

(4) 省略“表达式 2”，表示循环条件一直为真(非 0)。此时，for 语句等同于：

```
for (  ; 1 ;  ) { … }
```

这种情况下，若不在循环体中做终止循环处理，便成为死循环。例如：

```
for ( i = 1 ;  ; i++) s += i ;
```

该循环为死循环，即程序不停地执行 s＋＝i；i＋＋；这两条语句，不会自动终止。

(5) 省略“表达式 3”，则不对循环控制变量进行操作。此时可在循环体中加入修改循环控制变量的语句。如例 3.14 中程序可写为：

```
#include <iostream>
using namespace std;
int main()
```

```
{   int i , s = 0 ;
    for ( i = 1 ; i <= 100 ; )
    {   s = s + i;
        i++;
    }
    cout <<"s = "<< s <<'\n';
    return 0;
}
```

(6) 表达式 1 可以是设置循环变量初值的赋值表达式,也可以是其他表达式,同时还可以在此处定义变量。但需要注意的是,在表达式中定义的变量,只在循环体中有效。例如,例 3.14 中程序可写为:

```
#include < iostream >
using namespace std;
int main()
{   for ( int i = 1 , s = 0 ; i <= 100 ; i++)  //A
          s = s + i;
    cout <<"s = "<< s <<'\n';                  //B
    return 0;
}
```

3.3.4 三种循环的比较及适用场合

对于任何一种重复结构的程序段,均可用这三种循环语句中的任何一个来实现。但对不同的重复结构,使用不同的循环语句,不仅可优化程序的结构,还可精简程序。它们之间的关系如下:

(1) 用 while 和 do-while 循环时,循环变量初始化的操作应在 while 和 do-while 语句之前完成,而 for 语句可以在表达式 1 中实现循环变量的初始化。

(2) 用 while 和 do-while 循环时,循环体或 while 后表达式中应包括使循环趋于结束的操作,而 for 语句中使循环趋于结束的操作由表达式 3 给出。

(3) for 语句和 while 语句都是先判断循环条件,并根据循环条件决定是否要执行循环体,循环体有可能执行若干次,也可能一次都不执行。而 do-while 语句是先执行循环体,后判断循环条件,所以循环体至少要执行一次。

因此,对于至少要执行一次重复结构的场合,可以使用 do-while 语句。对于执行重复结构次数要在运行过程中才能确定的场合,使用 while 或 do-while 语句比较方便。由于 for 语句功能最强,可以完成其他类型的循环功能,所以,一般在循环变量初值、循环变量增量给定及重复结构执行次数确定的情况下,使用 for 语句比较方便。

3.3.5 多重循环

一个循环体内又包含另一完整的循环结构,称为多重(层)循环,又称为循环的嵌套。内嵌的循环体中还可以嵌套循环。上述三种循环均可以互相嵌套,且 C++ 中对嵌套的层数没有限制。

【例 3.15】 打印出以下图案。

```
    *
  * * *
* * * * *
  * * *
    *
```

算法分析：该图案可看成由两部分构成：上面 4 行规律相同，由星号组成了一个正三角形；下面 3 行规律相同，由星号组成了一个倒三角形。首先输出前 4 行正三角形部分，其组成规律如表 3.1 所示。

表 3.1　正三角形图案的组成规律

上面四行	第 1 行	第 2 行	第 3 行	第 4 行	第 i 行
空格(␣)	3	2	1	0	$4-i$
星号(*)	1	3	5	7	$2\times i-1$

对每一行来说，既要输出空格“␣”，又要输出星号“*”，且每次只能输出一个空格或星号。因此，如果设定外层循环是行数 i，内层就有两个并列的循环，依次输出 $4-i$ 个空格和 $2\times i-1$ 个星号，之后输出换行符'\n'，结束该行的输出。

如果用 j 表示空格数，用 k 表示星号数，则程序段为：

```
for(i=1;i<=4;i++)                    //i 为行数,依次是 1,2,3,4
{   for(j=1;j<=4-i;j++)              //对每一行,输出的空格数不同
        cout<<' ';                   //执行一次循环,输出一个空格
    for(k=1;k<=2*i-1;k++)            //对每一行,输出的星号数不同
        cout<<'*';                   //执行一次循环,输出一个星号
    cout<<'\n';                      //输出完一行的空格和星号后输出换行符
}
```

同样，图案的后半部分倒三角形的组成规律见表 3.2，其输出方法与上部分相同。

表 3.2　倒三角形图案的组成规律

下面 3 行	第 1 行	第 2 行	第 3 行	第 i 行
空格(␣)	1	2	3	i
星号(*)	5	3	1	$2\times(3-i)+1$

源程序如下：

```
#include<iostream>
using namespace std;
int main()
{   int i,j,k;
    for(i=1;i<=4;i++){                    //控制上面 4 行
        for(j=1;j<=4-i;j++)               //控制 * 前的空格数
            cout<<' ';                    //输出一个空格
        for(k=1;k<=2*i-1;k++)             //控制 * 个数
```

```
            cout<<'*';
        cout<<'\n';                                    //输出完一行后换行
    }
    for ( i=1 ; i<=3 ; i++){                           //控制下面3行
        for ( j=1 ; j<=i ; j++)                        //控制*前的空格数
            cout<<' ';                                 //输出一个空格
        for ( k=1 ; k<=2*(3-i)+1 ; k++)                //控制*个数
            cout<<'*';
        cout<<'\n';
    }
    return 0;
}
```

程序运行情况如图 3.16 所示。

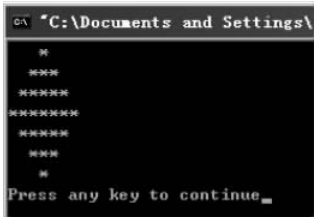

图 3.16　两重循环实现打印菱形图案

3.4　控制执行顺序的语句

前面已介绍的 C++语句都是根据其在程序中的先后次序,从主函数 main 开始,依次顺序执行的。从表面上看,循环语句或条件语句也改变了程序的执行顺序,但由于整个循环可以看成是一个语句体(条件语句也一样),因此它仍是顺序执行的。当要改变程序的执行顺序,即不依次执行紧跟着的语句,而是跳到另一个语句处接着执行时,就要用到下面介绍的一些控制语句。

3.4.1　break 语句

break 语句的使用格式为:

```
break;
```

break 语句只能用在循环语句和 switch 语句(开关语句)中。当 break 用于 switch 语句中时,可使程序跳出 switch 语句而执行 switch 以后的语句。break 在 switch 语句中的用法已在前面介绍过,这里不再举例。当 break 语句用在循环语句体中并被执行时,可使程序终止循环,不再进行循环条件的判断,而直接跳出循环体,执行循环后面的语句。通常,break 语句总是与 if 语句联合在一起使用,即满足条件时便跳出循环体。

【例 3.16】　输入三个整数,求这三个整数的最小公倍数。

算法分析:根据定义,三个数 a、b、c 的最小公倍数是可以整除这三个数的最小的那一个数。因此,设定最小公倍数 x 的初值为 a(也可为 b 或 c),终值为 $a\times b\times c$,从小到大逐一

判断，直到满足选择条件，此时的 x 就是最小公倍数。因为三个数的公倍数可能不止一个，第一个满足选择条件的就是最小的那个数，所以此时程序必须用 break 中止循环。源程序如下：

```
#include<iostream>
using namespace std;
int main()
{   int a,b,c,x;
    cout<<"请输入三个整数:  ";
    cin>>a>>b>>c;
    for(x=a; x<a*b*c; x++)                          //x从小到大逐一测试
        if(x%a==0&&x%b==0&&x%c==0)                  //公倍数的选择条件
            break;                                  //首先满足条件的是最小公倍数
    cout<<a<<","<<b<<","<<c<<"的最小公倍数是:"<<x<<endl;
    return 0;
}
```

程序的运行情况如下：

```
请输入三个整数:  8  6  16↙
8,6,16的最小公倍数是:48
```

注意：

(1) break 语句对 if 语句不起作用。

(2) 在多重循环中，一个 break 语句只能向外跳出一层。

【例 3.17】 输入一个整数，判断其是否为素数。

算法分析：素数是指除了 1 和该数本身之外，不能被其他任何整数整除的数。判断一个数 x 是否为素数，可以使 2、3、…、$x-1$ 依次作为除数，循环判断其能否被 x 整除，如果在循环过程中有一个数 i 可以被 x 整除，那么 x 就不是素数，其余比 i 大的数也不必再循环判断；如果上述所有除数循环结束，仍没有能被 x 整除的数，则 x 就是素数。源程序如下：

```
#include<iostream>
using namespace std;
int main()
{   int x,i;
    cout<<"请输入一个整数:";
    cin>>x;
    for(i=2;i<x;i++)                    //i作为除数,从2～(x-1)循环
        if(x%i==0)                      //判断i是否为x的因子
            break;                      //如果i为因子,x不是素数,不必再判断其他因子
    if(i>=x)                            //条件成立,从i<x退出循环,是素数
        cout<<x<<"是素数\n";
    else                                //从break退出循环,不是素数
        cout<<x<<"不是素数\n";
    return 0;
}
```

程序的运行情况如下(运行两次)：

```
请输入一个整数: 35 ↙
35 不是素数
```



```
请输入一个整数: 37 ↙
37 是素数
```

实际上,x 只需被 2～$x/2$ 间的整数甚至只需被 2～$\sqrt{x}$ 之间的整数除即可,这样可以减少运算次数。因此,源程序还可以这样修改:

```
#include<iostream>
using namespace std;
int main()
{   int x,i;
    cout<<"请输入一个整数: ";
    cin>>x;
    for(i=2;i<x/2;i++)
        if(x%i==0)
            break;
    if(i>=x/2)
        cout<<x<<"是素数\n";
    else
        cout<<x<<"不是素数\n";
    return 0;
}
```

3.4.2 continue 语句

continue 语句的使用格式为:

```
continue;
```

该语句只能用在循环语句的循环体中,作用是结束本次循环,即跳过循环体中剩余的语句,转到判断循环条件的位置,直接判断循环条件,决定是否重新开始下一次循环。continue 语句通常与 if 语句联合在一起使用,即满足某一条件时便终止该次循环。

【例 3.18】 输出 10～20 之间不是 3 的倍数的数。

算法分析:使用 for 循环结构对 10～20 之间的每一个整数进行检测,当能被 3 整除时,执行 continue 语句,不进行输出。只有当不能被 3 整除时才执行输出。程序如下:

```
#include<iostream>
using namespace std;
int main()
{   int i;
    for(i=10;i<=20;i++)
    {   if(i%3==0) continue;
        cout<<i<<'\t';
    }
    cout<<'\n';
```

```
    return 0;
}
```

程序的运行结果如下：

```
 10  11  13  14  16  17  19  20
```

注意：break 语句与 continue 语句两者之间的区别，前者是结束本层循环，而后者是结束本次循环。结束本次循环后，是否继续循环，要看循环条件表达式的值，由该值决定是否要开始下一次的循环。

*3.4.3 goto 语句

goto 语句是一种无条件转移语句，其使用格式为：

```
goto  语句标号;
```

其中，语句标号是一个有效的标识符，这个标识符加上一个冒号“:”一起出现在程序中的某处，执行 goto 语句后，程序将跳转到该标号处并执行其后的语句。

【例 3.19】 用 goto 语句和 if 语句构成循环，计算 1～100 之间整数之和。

程序如下：

```
#include<iostream>
using namespace std;
int main()
{   int i=1,s=0;
loop:if(i<=100)                    //A
     {   s=s+i;
         i++;
         goto loop;                //B
     }
     cout<<"s="<<s<<'\n';
     return 0;
}
```

程序中 A 行的 loop 是语句标号。当执行到 B 行时，程序流程跳转到 loop 标示的 if 语句处，执行 if 语句。当 i 的值增加到 101 时，if 语句的表达式的值为 0，if 语句的执行结束。显然，if 语句与 goto 语句联合使用相当于“当型循环结构”。

结构化程序设计方法主张限制使用 goto 语句，因为滥用 goto 语句将使程序流程无规律、可读性差。但也不是绝对禁止使用 goto 语句。一般来说，有两种用途：

(1) 与 if 语句一起构成循环结构，如例 3.19。

(2) 从循环体中跳转到循环体外。但在 C++语言中可以用 break 语句和 continue 语句跳出本层循环和结束本次循环。goto 语句的使用机会已大大减少，只是需要从多层循环的内层循环跳转到外层循环外时才用到 goto 语句。但是这种用法不符合结构化程序设计原则，一般不宜采用，只有在不得已时(如能大大提高效率)才使用。

*3.4.4 exit 和 abort 函数

exit 和 abort 函数都是 C++的库函数,其功能都是终止程序的执行,将流程控制返回给操作系统。通常,前者用于正常终止程序的执行,而后者用于异常终止程序的执行。

1) exit 函数

exit 函数的格式为:

```
exit( 表达式 );
```

其中,表达式的值只能是整型数。通常把表达式的值作为终止程序执行的原因。执行该函数时,将无条件地终止程序的执行而不管该函数处于程序中的什么位置,并将控制返回给操作系统。通常表达式的取值为一个常数:用 0 表示正常退出,而用其他的整数值作为异常退出的原因。

当执行 exit 函数时,系统要做终止程序执行前的收尾工作,如关闭该程序打开的文件、释放变量所占用的存储空间(不包括动态分配的存储空间)等。

2) abort 函数

abort 函数的格式为:

```
abort();
```

调用该函数时,括号内不能有任何参数。其作用是向标准错误流(std::cerr)发送程序异常终止的消息,然后终止程序。在执行该函数时,系统不做结束程序前的收尾工作,直接终止程序的执行。

除了上面介绍的语句或库函数可以改变程序的执行顺序外,还有 return 语句也可改变程序的执行顺序。return 语句将在第 4 章中介绍。

3.5 综合应用举例

【例 3.20】 用公式 $e=1+\frac{1}{1!}+\frac{1}{2!}+\cdots+\frac{1}{n!}+\cdots$ 求 e 的近似值,要求计算到通项 $\frac{1}{n!}<10^{-7}$ 时为止。

算法分析:定义变量 e 存放已计算出的近似结果,n 存放当前项序号,t 存放当前通项值,其初始值分别为 e=1.0、n=1、t=1.0。当前通项值 t 是其前一项乘以 $\frac{1}{n}$,即 $t=t*\frac{1}{n}$。由于级数求和时项数不确定,故可采用 while 循环语句。重复的操作是:将 t 加到 e 中,当计算完一项后,当前项序号 n 增 1,为下一项做准备。重复以上操作,直到满足精度要求。源程序如下:

```
#include<iostream>
using namespace std;
int main()
{    double e=1.0, t=1.0;
     int n=1;
     while( t>=1e-7 )
```

```
    {    t = t/n ;
         e = e + t ;
         n++;
    }
    cout <<"e = "<< e <<'\n';
    return 0;
}
```

程序的运行结果如下：

```
e = 2.71828
```

该程序也可以使用 do-while 语句编写，同时也可以显示计算的级数项数。程序如下：

```
#include < iostream >
using namespace std;
int main ()
{  double e = 1.0 , t = 1.0 ;
   int n = 1 ;
   do
   {  t = t/n ;
      e = e + t ;
      n++;
   }while ( t > = 1.0e - 7 );
   cout <<"e = "<< e <<" (n = "<< n <<")"<<'\n';
    return 0;
}
```

程序的运行结果如下：

```
e = 2.71828 (n = 12)
```

【例 3.21】 利用 C++中产生随机数的库函数 rand，设计一个自动出题的程序，要求可给出加、减、乘三种运算；做何种运算也由随机数来确定；运算时的两个操作数的取值范围为 0～9；共出 10 题，每题 10 分，最后给出总的得分。

算法分析：

(1) 定义变量 a、b、op 分别表示两个操作数和运算符；定义变量 c、d 分别表示计算机的计算结果和用户的答案，用户的答案由键盘输入；定义循环变量 i，表示出题数；定义变量 sum，用以累计得分，初值为 0。

(2) 由于出题数确定，采用 for 循环语句控制出题，循环变量 i 取值 1～10。

(3) 在每一次出题时(即每一次循环时)，所做运算有三种，可用 switch 语句实现选择。

源程序如下：

```
#include < iostream >
#include < ctime >
using namespace std;
int main()
```

```
{   int i , a , b , sum = 0 ;
    int op , c , d ;
    srand(time(NULL));
    for ( i = 1 ; i <= 10 ; i++){
        a = rand() % 10 ;
        b = rand() % 10 ;
        op = rand() % 3 ;                    //共有三种运算,分别用 0、1、2 来表示
        switch (op){
        case 0: cout << a <<' + '<< b <<' = ' ;
                c = a + b ; break ;
        case 1: cout << a <<' - '<< b <<' = ' ;
                c = a - b ; break ;
        case 2: cout << a <<' * '<< b <<' = ' ;
                c = a * b ;
        }
        cin >> d ;                           //接收用户从键盘输入的答案
        if(d == c){                          //将用户输入的答案 d 与计算机运算的答案 c 比较
            cout <<"正确!\n";
            sum += 10;
        }
        else cout <<"错误!\n";
    }
    cout <<"10 题中答对: "<< sum/10 <<"题,"<<'\t'<<"得分:"<< sum <<'\n';
    return 0;
}
```

程序的运行结果如下:

```
2 - 0 = 2 ↙
正确!
6 + 9 = 15 ↙
正确!
3 + 6 = 9 ↙
正确!
3 * 4 = 12 ↙
正确!
3 - 4 = -1 ↙
正确!
7 - 3 = 4 ↙
正确!
4 * 6 = 24 ↙
正确!
6 + 1 = 7 ↙
正确!
8 + 8 = 16 ↙
正确!
3 * 5 = 15 ↙
正确!
10 题中答对: 10 题,        得分:100
```

其中，10 道四则运算的题目是随机产生的。

提示：随机数的产生方法。

C++函数库中有专门产生随机数的函数 rand，该函数产生的是一串固定序列的随机整数。因为随机数序列的顺序是固定的，如果每一次都从一个固定的位置开始输出这个序列，那么每次产生的随机数都是一样的，也就失去了“随机数”的意义了。因此，要产生真正意义上的随机数，关键是每次要从不同的位置处开始输出这个序列。函数 srand(n)便是用来选择初始位置的，称为“初始化随机数种子”。srand(100)是从序列的第 100 个数起开始输出，srand(1000)是从第 1000 个数起开始输出……一般用当前时间初始化随机数种子，因为每时每刻的当前时间都是不相同的，这样产生出的序列更接近真正的随机数。常用以下语句产生随机数：

```
#include<ctime>
using namespace std;
…
srand(time(NULL));                    //初始化种子
x = rand()%10 ;                       //产生不大于 10 的数
```

程序中取该随机数除以 10 的余数作为操作数，从而保证操作数的取值范围为 0～9。

【例 3.22】 编程求出 2～100 之间的所有素数。

算法分析：与例 3.17 类似，将 x 由键盘输入改成循环赋值 2～100，然后输出其中的素数即可。当然，素数的算法还可以在例 3.17 的基础上再优化。如对于 2～100 之间的数，只需判断奇数是否是素数。

对任意一个奇数 x，其因子的范围为 $2\sim\sqrt{x}$，如果 x 能被 $2\sim\sqrt{x}$ 之间任何一个整数 i 整除，则提前结束循环，此时 i 的值必然小于或等于 $\sqrt{x}$；如果 x 不能被 $2\sim\sqrt{x}$ 之间的任一整数 i 整除，则在完成最后一次循环后，i 的值还要增 1，此时 $i=\sqrt{x}+1$。在循环之后判别 i 的值是否大于或等于 $\sqrt{x}+1$ 就成为 x 是否为素数的标准，若是，则表明 x 未曾被 $2\sim\sqrt{x}$ 之间任一整数整除过，因此 x 是素数。源程序如下：

```
#include<iostream>
#include<cmath>
#include<iomanip>
using namespace std;
int main()
{   int x , m , k , i ;
    cout<<setw(8)<<2<<setw(8)<<3 ;      //先输出两个素数：2 和 3
    for ( k=2 , x=5 ; x<100 ; x+=2 )    //检测 5～100 间奇数
    {   m=sqrt((double)x) ;
        for ( i=3 ; i<=m ; i++)
            if ( x%i==0 ) break ;       //检测 x 是否为素数
         if ( i>=m+1 )                  //条件成立，则 x 为素数，需输出；否则不输出
         {  cout<<setw(8)<<x ;
            k++;                        //统计输出素数的个数
            if ( k%5==0 )cout<<'\n' ;   //控制每行输出 5 个素数
         }
     }
```

```
        cout<<'\n';
        return 0;
    }
```


程序的运行结果如下：

```
 2    3    5    7   11
13   17   19   23   29
31   37   41   43   47
53   59   61   67   71
73   79   83   89   97
```

练　习　题

一、选择题

1. 对下面三条语句(其中 s1 和 s2 为内嵌语句),正确的论断是_____。

```
if( a )s1 ; else s2 ;          //①
if( a == 0 )s2 ; else s1 ;     //②
if( a!= 0 )s1 ; else s2 ;      //③
```

A. 三者相互等价　　B. ①和②等价,但与③不等价

C. 三者互不等价　　D. ①和③等价,但与②不等价

2. 若有定义“int a=1，b=2，c=3，d=4；”,则表达式 a>b? a:c>d? c:d 的值为_____。

A. 1　　B. 2　　C. 3　　D. 4

3. 若有定义语句“int x=4，y=5;”,则表达式“y>x++? x--:y++”的值是_____。

A. 3　　B. 4　　C. 5　　D. 6

4. 以下选项中与“if(a==1) a=b; else a++;”语句功能不同的 switch 语句是_____。

A. switch(a==1) {case 0: a=b; break; case 1: a++;}

B. switch(a==1) {case 1: a=b; break; default: a++;}

C. switch(a==1) { default: a++; break; case 1: a=b;}

D. switch(a==1){ case 1: a=b; break; case 0: a++;}

5. 执行下列语句序列：

```
int i = 0;
while(i<25) i += 3;
cout<<i;
```

输出结果是_____。

A. 24　　B. 25　　C. 27　　D. 28

6. 有如下程序段：

```
int i = 5;
while(i = 0){cout <<' * '; i -- ;}
```

运行时输出“*”的个数是_____。

A. 0　　B. 1　　C. 5　　D. 无穷

7. 程序段“int x＝3；do{ cout << x－－；}while（!x）；”中循环体的执行次数是_____。

A. 3　　B. 2　　C. 1　　D. 死循环

8. 下列循环语句中有语法错误的是_____。

A. int i; for(i＝1;i＜10;i＋＋)　cout <<' * ';

B. int i,j; for(i＝1,j＝0;i＜10; i＋＋) cout <<' * ';

C. int i＝0; for(; i＜10;i＋＋) cout <<' * ';

D. for(1) cout <<' * ';

9. 有如下程序段：

```
int y = 9;
for( ; y > 0; y -- )
    if(y % 3 == 0)  cout <<-- y;
```

运行后输出结果是_____。

A. 963　　B. 852　　C. 741　　D. 875421

10. 有如下程序段：

```
char b = 'a', c = 'A';
for(int i = 0;i < 6;i++)
    if(i % 2)  cout <<(char)(i + b);
    else cout <<(char)(i + c);
```

运行后输出结果是_____。

A. ABCDEF　　B. aBcDeF　　C. abcdef　　D. AbCdEf

11. 以下程序段中的变量已正确定义

```
for(i = 0;i < 4;i++,i++)
    for(k = 1;k < 3;k++);
cout <<' * ';
```

运行后输出结果是_____。

A. ********　　B. ****　　C. **　　D. *

12. 有如下程序段：

```
int i,j,m = 55;
for(i = 1;i <= 3;i++)
    for(j = 3;j <= i;j++)  m = m % j;
cout << m << endl;
```

运行后输出结果是_____。

A. 0　　B. 1　　C. 2　　D. 3

13. 在循环语句体中使用 break 和 continue 语句的作用是_____ 。

A. 结束循环和结束本次循环　　B. 结束本次循环和结束循环

C. 两语句都结束本次循环　　D. 两语句都结束循环

14. 有如下程序段:

```
int i = 1;
while(1)
{   i++;
    if(i == 10)     break;
    if(i % 2 == 0)  cout <<' * ';
}
```

运行这个程序段,输出“*”的个数是_____。

A. 10　　B. 3　　C. 4　　D. 5

二、填空题

1. 以下程序的运行结果是_____。

```
#include < iostream >
using namespace std;
int main()
{   int x = 1, y = 2 , z = 3 ;
    x += y += z ;
    cout << ( x < y?x++ : y++)<<'\n' ;
    return 0;
}
```

2. 以下程序的运行结果是_____。

```
#include < iostream >
using namespace std;
int main()
{   int a = 1,b = 2,c = 3,d = 0;
    if(a == 1&&b++ == 2)
        if(b!= 2||c -- != 3)
            cout << a <<'\t'<< b <<'\t'<< c << endl;
        else
            cout << a <<'\t'<< b <<'\t'<< c << endl;
    return 0;
}
```

3. 以下程序的运行结果是_____。

```
#include < iostream >
using namespace std;
int main()
{   int k = 5,n = 0;
    do
    {   switch(k)
        {
        case 1:
```

```
        case 3: n += 1;k -- ;break;
        default: n = 0; k -- ;
        case 2:
        case 4: n += 2;  k -- ; break;
        }
        cout << n;
    }while(k > 0&&n < 5);
    return 0;
}
```

4. 从键盘输入 abcz↙时，则下列程序的输出是_____。

```
#include < iostream >
using namespace std;
int main()
{   char  ch;
    cin >> ch;
    while ( ch!= 'z' )
    {   switch ( ch )
        {   case 'a':
            case 'b':  cout <<'1'<< endl ;  break ;
            case 'c':  cout <<'3'<< endl ;
            default:  cout <<"default"<< endl ;
         }
         cin >> ch ;
     }
    return 0;
}
```

5. 以下程序的运行结果是_____。

```
#include < iostream >
using namespace std;
int main()
{   int i = 5;
    do
    {   switch ( i % 2 )
        {
        case 0:  i -- ; break ;
        case 1:  i -- ; continue ;
        }
        i -- ;
        cout << i ;
    }while ( i > 0 ) ;
    cout <<'\n';
    return 0;
}
```

三、编程题

1. 根据下面函数表达式编写一个程序，实现输入 x 的值后，能计算并输出相应的函数值 y。

$$y=\begin{cases}x^2 & x<0\\ 2.5x-1 & 0\leqslant x<1\\ 3x+1 & x\geqslant 1\end{cases}$$

2. 设计一个程序,要求对从键盘输入的一个不多于 5 位的正整数,能输出它的位数并输出它的各位数字之和。

3. 编程计算 $s=1!+2!+3!+\cdots+n!$,n 的值从键盘输入,要求输出 n 和 s 的值。

4. 编程求 $S_n=a+aa+aaa+\cdots+\underbrace{aa\cdots a}_{n个a}$之值,其中 a 是一个数字,n 表示 S_n 的项数和其最后一项的位数。例如 2+22+222+2222+22222(此时 $n=5$)。

5. 利用公式

$$\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots$$

编程计算 π 的近似值,要求计算到最后一项的绝对值小于10^{-5}为止。

6. 编程求出所有的“水仙花数”。所谓“水仙花数”是指一个 3 位数,其各位数字立方和等于该数本身。例如,153 是水仙花数,因为 $153=1^3+5^3+3^3$。

7. 编程求满足以下条件的三位整数 n,它除以 11 所得到的商等于 n 的各位数字的平方和,且其中至少有两位数字相同。例如,131 除以 11 的商为 11,各位数字的平方和为 11,所以它是满足条件的三位数。

8. 设计一个求两正整数 m 和 n 的最大公约数的程序,m 和 n 的值由键盘输入。