第5章 流程控制结构

在结构化程序设计中，任何复杂的功能都可以由顺序结构、选择结构和循环结构3种基本结构组合来实现。前面介绍的简单程序设计中，程序中的所有语句按排列顺序自上而下依次执行，是典型的顺序结构，但仅有顺序结构很难实现复杂的功能。为此，作为支持结构化程序设计的C++语言提供了流程控制语句，用于实现选择结构和循环结构。

5.1 选择结构语句

为了实现选择结构，C++语言提供了if语句和switch语句两种类型的选择语句。尽管使用if语句可以实现所有的选择结构，但是在特定情况下，使用switch语句可以更好地实现程序功能并提高程序的可读性。

5.1.1 if语句

if语句也称为条件语句，它的功能是计算表达式的值并根据计算结果选择要执行的操作。根据语句构成，if语句可以分为3种形式，即单分支形式、双分支形式和多分支形式。

1. 单分支形式

单分支形式用于决定是否执行某个语句，其语法格式如下：

```
if(<表达式>) 语句 S
```

其中，<表达式>可以是任意符合C++语言语法规则的表达式，通常为算术表达式、关系表达式、逻辑表达式或逗号表达式；语句S是一个单一语句，如果有多条语句，则必须使用“{}”括起来构成复合语句。

单选择结构的执行流程如图5-1所示。

执行时先计算表达式的值，如果表达式的值为真，则执行语句S，否则直接执行if语句后面的语句。任何非0值均被认为是逻辑真(true)，表示表达式成立；而0则表示逻辑假(false)，表示表达式不成立。

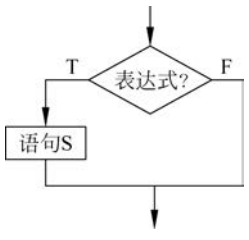

图5-1 单选择结构的执行流程

【例5.1】 输入一个字符，判别它是否为小写字母。如果是，则将其转换成大写字母；如果不是，则不转换，并输出最后得到的字符。

问题分析：用单分支if语句来处理，由于大小写字母的ASCII值相差32，因此将小写字母减去32即可转换成大写字母。其程序流程如图5-2所示。

程序如下：

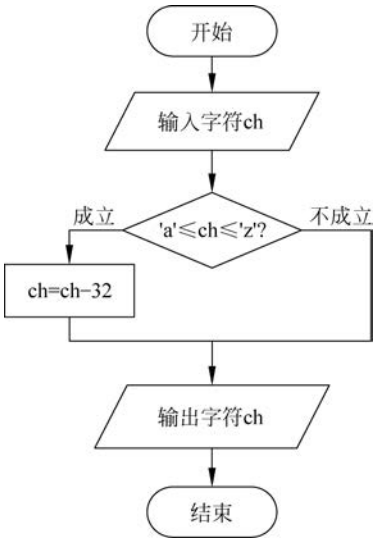

图 5-2　例 5.1 程序流程

```
#include <iostream>
using namespace std;
int main()
{    char ch;
     cout <<"请输入一个字符:";
     cin.get(ch);
     if(ch >= 'a'&& ch <= 'z')                    //判断是否是小写字母
       ch = ch - 32;                              //转换成大写字母
     cout <<"输出的字符是:"<< ch << endl;
     return 0;
}
```

若输入小写字母'a',则程序运行结果如下:

```
请输入一个字符:  a↙
输出的字符是:A
```

若输入大写字母'B',则程序运行结果如下:

```
请输入一个字符:  B↙
输出的字符是:B
```

2. 双分支形式

双分支形式用于在两个语句中选择其中一个执行,其语法格式如下:

```
if (<表达式>) 语句 S1
else 语句 S2
```

该语句通常被称为 if-else 语句,前面介绍的 if 单分支形式是它的特例。其中,表达式可以是任意符合 C++语言语法规则的表达式;语句 Sl 和语句 S2 均为单一语句或复合语句,可以是任意合法语句。若表达式的值不为 0,则执行语句 S1;否则(为 0),执行语句 S2。通常把前者称为 if 分支,而把后者称为 else 分支。

双分支结构的执行流程如图 5-3 所示。

【例 5.2】 输入一个大于 0 的整数，判断该数的奇偶性，并输出。

问题分析：

(1) 因为整数可以分为奇数和偶数两类，所以本问题可以用双分支结构实现。

(2) 从键盘输入一个整数，判断其是否能被 2 整除，如果可以，则说明它为偶数，否则为奇数。

程序流程如图 5-4 所示。

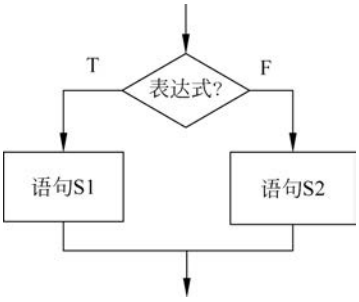

图 5-3 双分支结构的执行流程

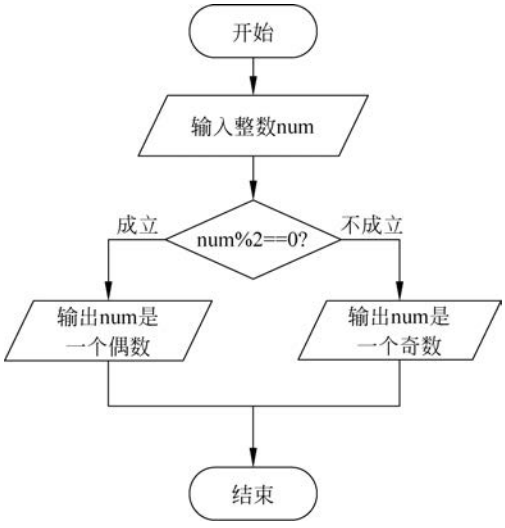

图 5-4 例 5.2 程序流程

程序如下：

```
#include <iostream>
using namespace std;
int main()
{    int num;
     cout <<"请输入一个整数:";
     cin >> num;
     if ((num % 2) == 0)
          cout << num <<"是一个偶数."<< endl;
     else
          cout << num <<"是一个奇数."<< endl;
     return 0;
}
```

若输入整数 13，则程序运行结果如下：

```
请输入一个整数:  13 ↙
13 是一个奇数.
```

若输入整数 16，则程序运行结果如下：

```
请输入一个整数:  16 ↙
16 是一个偶数.
```

3. 多分支形式

if-else 语句可以在两个分支中选择一个执行，若要在多个（超过两个）分支中选择一个执行，这时可以使用多个单分支语句实现，但这会增加程序执行时的判断次数，导致程序效率不高，同时也会降低程序的可读性。此时可以用多分支形式，其语法格式如下：

```
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
```

首先求出表达式 1 的值，若其值不为 0，则执行“语句 1”；否则求解表达式 2 的值，若不为 0，则执行“语句 2”；否则求解表达式 3 的值；以此类推，直至“语句 n＋1”。根据实际情况，最后的“else 语句”可以省略。多分支 if 语句的执行流程如图 5-5 所示。

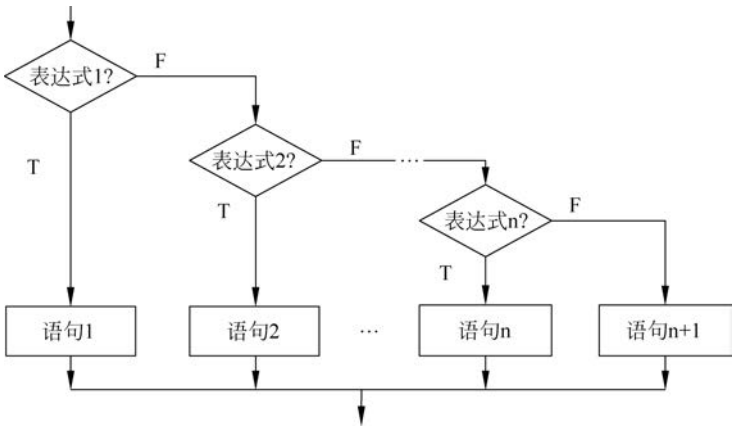

图 5-5　多分支 if 语句的执行流程

【例 5.3】 设计程序，将百分制成绩转换成相应的五分制成绩。其转换规则如下：

90～100 分：优秀；

80～89 分：良好；

70～79 分：中等；

60～69 分：及格；

60 分以下：不及格。

问题分析：

（1）该问题可以用多分支结构实现。

（2）从键盘输入一个数，根据其值的范围输出相应的五分制成绩。

程序流程如图 5-6 所示。

程序如下：

```
#include <iostream>
using namespace std;
int main()
{    int score;
     cout <<"请输入一个成绩(0－100):";
```

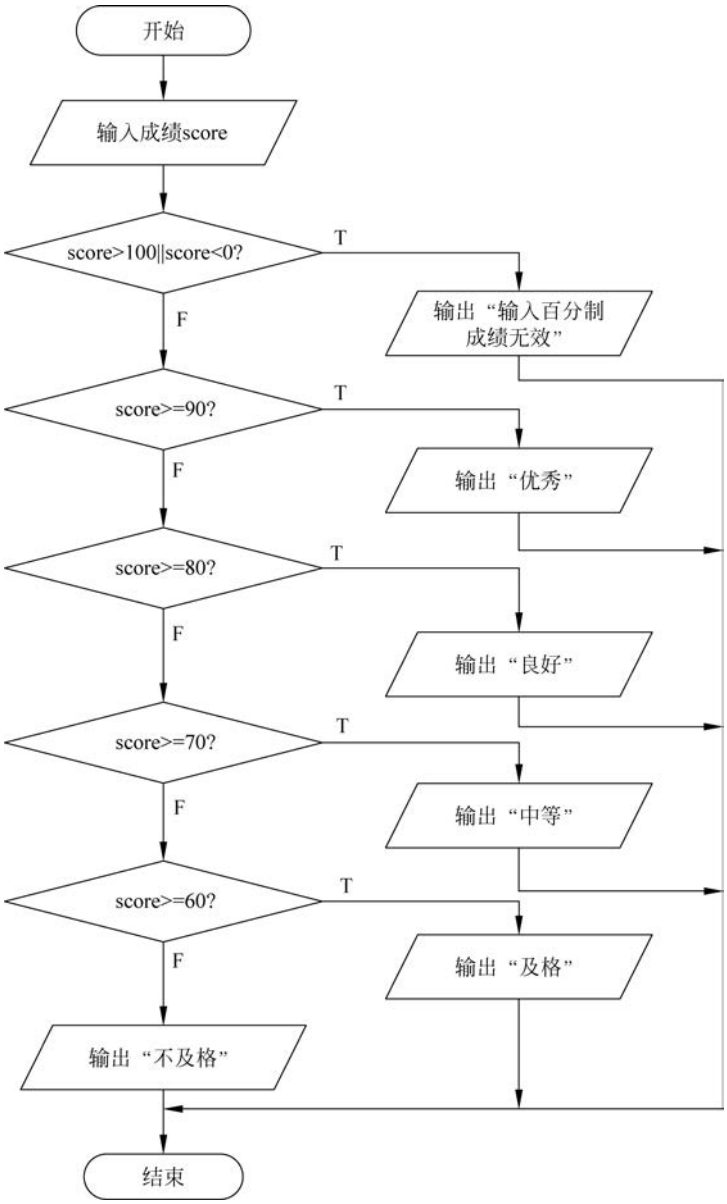

图 5-6　例 5.3 程序流程

```
cin >> score;
if (score > 100 ||score < 0)
{    cout <<"输入百分制成绩无效\n";
     return 1;
}
cout <<"等级为";
if (score > = 90) cout <<"优秀"<< endl;
else if(score > = 80)cout <<"良好"<< endl;
else if(score > = 70)cout <<"中等"<< endl;
else if(score > = 60) cout <<"及格"<< endl;
```

```
    else cout <<"不及格"<< endl;
    return 0;
}
```


若输入成绩为 91,则程序运行结果如下:

```
请输入一个成绩(0 - 100): 91 ↙
等级为优秀
```

若输入成绩为 56,则程序运行结果如下:

```
请输入一个成绩(0 - 100): 56 ↙
等级为不及格
```

在使用 if 语句时,以下问题需要特别注意。

(1) if 后面的表达式必须用“()”括起来,如下面的语句:

```
if x > 0 y = x * x + 1;
```

是错误的,应该写为

```
if (x > 0) y = x * x + 1;
```

(2) if 分支和 else 分支后面的语句均为单一语句,若为多个语句,则必须用“{}”括起来构成复合语句。例如,对两个整数 a 和 b 进行比较,如果 a>b,则交换 a 和 b。使用下面的语句:

```
if (a > b) t = a; a = b; b = t;
```

不能完成该功能,应该写为

```
if (a > b) {t = a; a = b; b = t;}
```

才能完成相应的功能。

(3) 由于浮点数在计算机中存储时通常会有误差,因此表达式中通常不对浮点数进行相等比较。例如,对于两个浮点数 a 和 b,如果两者相等,则输出“相等”,否则输出“不相等”,通常不使用下面的语句:

```
if (a == b) cout <<"相等"<< endl;
else cout <<"不相等"<< endl;
```

而是使用下面的语句:

```
if (fabs(a - b)< 1e - 8) cout <<"相等"<< endl;
else cout <<"不相等"<< endl;
```

即当 2 个浮点数的差的绝对值小于一个很小的正数时,就认为它们相等。

(4) if 语句包含 if 分支和 else 分支两部分,if 分支可以单独使用,但 else 分支必须与 if 分支配对使用,不能单独使用。例如,下面的语句:

```
else y = x + 2;
```

单独出现是错误的。

(5) 如果 if 分支和/或 else 分支中的语句又是 if 语句,则称为 if 语句的嵌套。在 if 语句的嵌套中,else 分支必须与 if 分支正确地配对才能完成指定的功能。C++语言中规定 else 分支与 if 分支的配对规则如下:else 分支总是与其前面最近的处于同一个语句中还没有配对的 if 分支配对。可以通过使用复合语句改变与 else 配对的 if 分支。

例如,按以下方式添加"{}"构成复合语句后,可以实现 else_2 与 if_1 配对。

```
if (x>0)                    //if_1
{   if(x>3)y=1;             //if_2
    else if (x<3) y=2;      //if_3,else_1
}
else y=3;                   //else_2
```

(6) 在 if 语句嵌套时,完成同一个功能可使用不同的嵌套方式实现。例如,例 5.3 的程序中,if 语句的嵌套部分还可以写为

```
if (score<60) cout<<"不及格"<<endl;
else if (score<70) cout<<"及格"<<endl;
else if (score<80) cout<<"中等"<<endl;
else if (score<90) cout<<"良好"<<endl;
else cout<<"优秀"<<endl;
```

或者:

```
if (score>=70)
     if (score<80) cout<<"中等"<<endl;
     else if (score<90) cout<<"良好"<<endl;
     else cout<<"优秀"<<endl;
else if (score<60) cout<<"不及格"<<endl;
else cout<<"及格"<<endl;
```

或者:

```
if (score>=90) cout<<"优秀"<<endl;
if (score<90&&score>=80) cout<<"良好"<<endl;
if (score<80&&score>=70) cout<<"中等"<<endl;
if (score<70&&score>=60) cout<<"及格"<<endl;
if (score<60)cout<<"不及格"<<endl;
```

这些语句(程序段)都可以实现例 5.3 所需的功能。但是,对于同一个输入的成绩,它们的表达式求值次数是不同的,即程序的执行效率不同。

5.1.2 switch 语句

尽管使用多个 if-else 语句或 if 语句的嵌套可以实现多个分支的选择,但是程序的可读性并不好。为此,C++语言还提供了另一种多分支选择语句——switch 语句,又称为开关语句。在特定情况下,使用 switch 语句代替 if 语句实现多分支可以有效地提高程序的可

读性。

switch 语句的语法格式如下：

```
switch(<表达式>)
{   case <常量表达式 1>: [<语句序列 1>]
    case <常量表达式 2>: [<语句序列 2>]
    …
    case <常量表达式 n>: [<语句序列 n>]
    [default: <语句序列 n + 1>]
}
```

其中，<表达式>是任意符合 C++语言语法规则的表达式，但其值只能是字符型或整型；<常量表达式>只能是由常量组成的表达式，其值也只能是字符型常量或整型常量；所有<语句序列>均是可选的，它可由一个或多个语句组成；default 分支也是可选项，尽管其可放在 switch 语句中的任何位置，但实际编程时通常将其作为 switch 语句的最后一个分支。当<表达式>值与所有<常量表达式>的值均不同时，执行 default 分支的语句序列。

switch 语句的执行流程如图 5-7 所示。

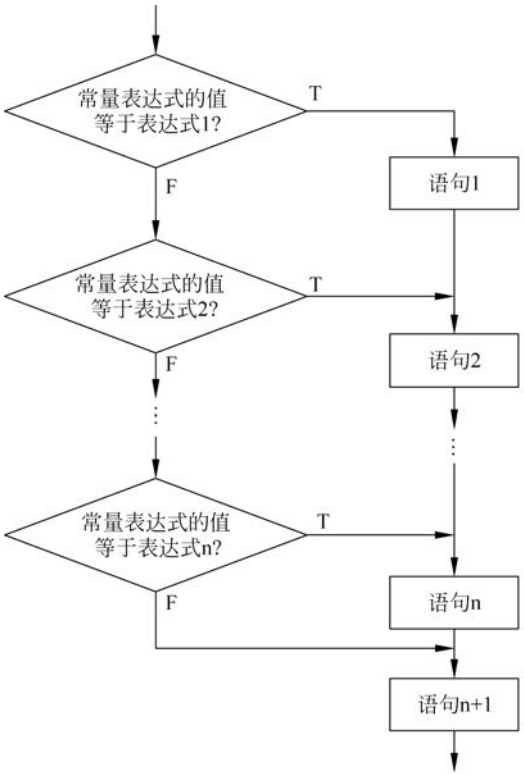

图 5-7　switch 语句的执行流程

执行 switch 语句时，首先计算表达式的值，然后顺序地与 case 子句中所列出的各个常量表达式进行比较。若表达式的值与某个常量表达式的值相等，则执行其后的语句序列，并依次执行该 case 子句后面所有 case 语句中的语句序列，遇到 case 和 default 也不再进行判断，直至 switch 语句结束。

【例 5.4】 输入日期，格式为“年/月/日”，如 2022/5/28，计算该日期是这一年的第几天。

问题分析：要计算某日期是该年的第几天，只需要将“日”的值加上该“月”之前所有月的天数即可。除了 2 月以外，其他各月的天数是固定的，可以使用 case 语句依次求“月”之前所有月的天数之和。闰年的 2 月是 29 天，其他年份的 2 月均为 28 天。当年份可以被 400 整除或能够被 4 整除但不能被 100 整除时，该年份为闰年。

程序如下：

```
#include <iostream>
using namespace std;
int main()
{   int year, month, day, total = 0;
    cout <<"请输入日期,格式为年/月/日:\n";
    char ch;
    cin >> year >> ch >> month >> ch >> day;
    switch (month)
    {   case 12:  total += 30;
        case 11:  total += 31;
        case 10:  total += 30;
        case 9:   total += 31;
        case 8:   total += 31;
        case 7:   total += 30;
        case 6:   total += 31;
        case 5:   total += 30;
        case 4:   total += 31;
        case 3:   if (year % 400 == 0 || (year % 4 == 0 && year % 100 != 0)) total += 29;
                  else total += 28;
        case 2:   total += 31;
        case 1:;
    }
    total += day;
    cout << year <<'/'<< month <<'/'<< day <<"是"<< year <<"年第"<< total <<"天"<< endl;
    return 0;
}
```

程序运行结果如下：

```
请输入日期,格式为年/月/日:
2022/5/28↙
2022/5/28 是 2022 年第 148 天
```

从程序运行结果来看，month＝5，因此依次执行了 case 5、case 4、case 3、case 2、case 1 后面的语句，这与前面 if 语句的多分支有明显的差别。如果执行完某个 case 后面的语句序列后不再执行后面其他 case 及 default 后面的语句序列，则可以在语句序列后添加 break 语句。break 可以直接跳出 switch 语句，接着执行 switch 后面的语句。

【例 5.5】 设计简单的学生管理系统菜单界面程序。设计一个程序，实现图 5-8 所示的菜单界面，程序执行时，

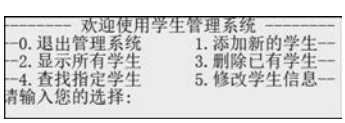

图 5-8 例 5.5 菜单结构

输入编号(0～5),显示所选择的菜单项。

问题分析:由于菜单编号是用整数表示的,因此可以使用 switch 语句来实现。在每个 case 子句中以编号值作为常量表达式,以输出相应菜单项作为语句,并在每个语句后添加 break 语句即可。

程序如下:

```
#include<iostream>
using namespace std;
int main()
{   int x;
    cout<<" --------- 欢迎使用学生管理系统 ---------\n";
    cout<<" -- 0.退出管理系统        1.添加新的学生 --\n";
    cout<<" -- 2.显示所有学生        3.删除已有学生 --\n";
    cout<<" -- 4.查找指定学生        5.修改学生信息 --\n";
    cout<<"请输入您的选择:";
    cin>>x;
    switch(x)
    {    case 0: cout<<"您选择退出管理系统\n";break;
         case 1: cout<<"您选择添加新的学生\n";break;
         case 2: cout<<"您选择显示所有学生\n";break;
         case 3: cout<<"您选择删除已有学生\n";break;
         case 4: cout<<"您选择查找指定学生\n";break;
         case 5: cout<<"您选择修改学生信息\n";break;
         default:cout<<"输入有误\n";
    }
    return 0;
}
```

程序运行结果如下:

```
--------- 欢迎使用学生管理系统 ---------
-- 0.退出管理系统        1.添加新的学生 --
-- 2.显示所有学生        3.删除已有学生 --
-- 4.查找指定学生        5.修改学生信息 --
请输入您的选择: 3↙
您选择删除已有学生
```

从格式可知,switch 语句结构清晰,易理解。在实际应用中,所有 switch 语句均可用 if 语句来实现,但反之不然。这是因为 switch 语句中限定了表达式的取值类型为整型或字符型,而 if 语句中的条件表达式的值可为任意类型。

使用 switch 语句时,需要注意以下问题:

(1) switch 语句中,<表达式>的值只能是字符型或整型;<常量表达式>只能是由常量组成的表达式,其值也只能是字符型常量或整型常量。

(2) 每一个 case 子句中的常量表达式的值必须互不相同,否则会出现自相矛盾的现象(一个值有两种执行方案)。

(3) default 子句可以省略,这时,若不满足条件则不执行任何语句。

(4) case 子句和 default 子句在语句中可以按任意顺序排列,但由于 switch 语句在执行时若匹配到某个子句后,将依次执行该子句之后的所有语句序列,因此不同的排列顺序可能

需要在适当的位置添加 break 语句。例如，对于下列语句：

```
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; break;
     case 3: cout << a <<'/'<< b <<' = '; c = a/b;
}
```

如果把 case 3 子句放到最前面作为第一个 case 子句，则需要在其语句序列中添加 break 语句，即改为“case 3：cout << a <<'/'<< b <<'='；c=a/b；break;”。

（5）若一个子句“:”后面没有语句序列，则该子句与其后面的子句共用语句序列。例如：

```
switch(ch)
{    case 'A':
     case 'B':
     case 'C': cout <<"pass!\n";
}
```

当 ch 为 A、B 和 C 时均执行“cout <<"pass! \n";”。但是，需要特别注意的是，最后一个子句必须有语句序列，即“}”前必须有“;”。下面的语句在编译时将出现错误：

```
switch(ch)
{    case 'A':
     case 'B': cout <<"pass!\n";
     case 'C':
}
```

5.2 循环结构语句

循环结构是结构化程序设计中的 3 种基本结构之一，用于实现反复执行某些操作。结构化程序设计中，循环分为当型循环和直到型循环，前者先判断循环控制条件，当条件成立时执行循环体；后者则先执行循环体再判断循环控制条件。C++语言提供了 3 种循环结构语句，即 while 循环语句、do-while 循环语句和 for 循环语句。while 循环语句是典型的当型循环；do-while 循环语句用于实现直到型循环，但其与典型直到型循环的“执行循环体直到循环控制条件成立”不同的是，do-while 循环语句是“执行循环体直到循环控制条件不成立”；for 循环语句主要用来实现循环体执行次数已知的循环，但实际上，该语句可以实现任何类型的循环。

5.2.1 while 循环语句

while 循环语句的语法格式如下：

```
while (<表达式>)
    循环体
```

其中，while 是 C++语言的关键字；表达式是循环控制条件，可以是任意符合 C++语言语法

规则的表达式；循环体语句部分可以是一条语句，也可以是由“{}”括起来的多条语句。

while 循环语句中，必须有使表达式趋于不成立的语句，从而使循环体在执行一段时间后能够结束。表达式一直成立，从而导致循环语句不能运行结束的情况（称为“死循环”）是必须要避免的。

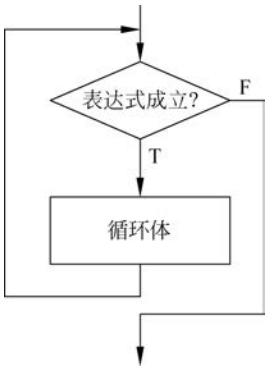

图 5-9　while 循环语句的执行流程

while 循环语句的执行流程如图 5-9 所示，先计算表达式的值，若表达式成立（值不为 0），则执行循环体一次，再计算表达式的值。重复以上过程，直到表达式的值为 0，则结束 while 循环语句的执行，继续执行其后的语句。

while 循环语句的特点是先判断循环控制条件（计算表达式的值），后执行循环体，如果第一次判断时循环控制条件就不成立，则循环体一次也不执行。

【例 5.6】 求 1～100 之间奇数之和 sum＝1＋3＋5＋…＋99。

问题分析：用变量 sum 存储计算结果，给它赋初值为 0，变量 i 表示每一个奇数值，赋初值为 1，每次增加 2。重复执行的操作（循环体）为“sum＝sum＋i;i＝i＋2;”，当条件“i＜＝99”成立时执行循环体。

程序如下：

```
#include <iostream>
using namespace std;
int main()
{    int i = 1,sum = 0;
     while (i <= 99)
     {    sum = sum + i;
          i = i + 2;
     }
     cout <<"sum = "<< sum << endl;
     return 0;
}
```

程序运行结果如下：

```
sum = 2500
```

【例 5.7】 从键盘输入一个正整数，编程求出它的各位数字之和。

问题分析：从键盘输入一个正整数，赋给变量 x，sum 赋初值为 0，则 x 和 10 求余得到个位数，加到 sum 中，n 和 10 整除去掉个位数。重复这两个操作，一直到 x 为 0 结束，即循环体为“sum＝sum＋x％10；x＝x/10;”。

程序如下：

```
#include <iostream>
using namespace std;
int main()
{    int x, sum = 0;
     cout <<"请输入一个正整数:";
     cin >> x;
```

```
    while (x!= 0)
    {    sum = sum + x % 10;
         x = x/10;
    }
    cout <<"它的各位数字之和为"<< sum << endl;
    return 0;
}
```

程序运行结果如下：

```
请输入一个正整数: 123 ↙
它的各位数字之和为 6
```

5.2.2 do-while 循环语句

do-while 循环语句的语法格式如下：

```
do
  <循环体>
while(<表达式>);
```

其中，do 是 C++语言的关键字，必须和 while 联合使用，不能单独出现。do-while 循环语句中的表达式和循环体的定义规则与 while 循环语句相同。

do-while 循环语句的执行流程如图 5-10 所示，先执行循环体，然后计算表达式的值，若表达式成立（值不为 0），再执行循环体。重复以上过程，直到表达式的值为 0，则结束 do-while 循环语句的执行，继续执行其后的语句。从执行流程可以看出，do-while 循环语句的循环体至少执行一次。

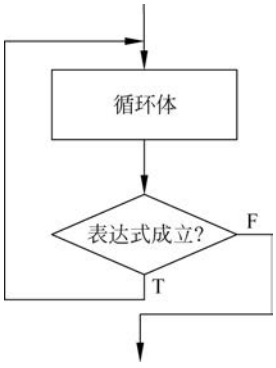

图 5-10　do-while 循环语句的执行流程

【例 5.8】 用 do-while 循环语句求 1～100 之间奇数之和 sum＝1＋3＋5＋…＋99。

问题分析：从例 5.6 的执行过程来看，累加过程会执行多次。因此，直接将其中的 while 循环语句改成 do-while 循环语句即可实现相应的功能。

程序如下：

```
#include <iostream>
using namespace std;
int main()
{    int i = 1, sum = 0;
     do
     {    sum = sum + i;
          i = i + 2;
     } while (i <= 99); //while 条件后的分号不能少
     cout <<"sum = "<< sum << endl;
     return 0;
}
```

程序运行结果如下：

```
sum = 2500
```

使用 while 循环语句和 do-while 循环语句时，必须注意以下问题：

(1) 条件表达式不可以为空，若为空，编译程序将会报告错误。

(2) 循环体是一个语句，若为多个语句，则需要使用“{}”括起来构成复合语句。例如，下列程序段中，循环体是“sum＋＝n;”，而不是“sum＋＝n; n＋＋;”：

```
while(sum < 100)
     sum += n;
n++;
```

要将“sum＋＝n; n＋＋;”作为循环体，应写为

```
while(sum < 100)
{    sum += n;
     n++;
}
```

(3) 在 while 循环语句和 do-while 循环语句之前，通常需要对表达式及循环体中所使用的变量进行初始化。例如，对于(2)中的程序段，要使程序能够正确执行，在 while 循环语句之间应有诸如“sum＝0; n＝1;”的初始化语句(序列)。

(4) 循环体中需要有使循环控制条件(表达式)趋于不成立的语句，否则会出现“死循环”。例如：

```
while(x!= 0)
    y += x;
```

若在 while 循环语句执行前，x 的值不为 0，则该循环为“死循环”。所以，在循环体中应该有类似“x＋＋;”“x＋＝2;”“x－－;”等语句，使得 x 的值在经过若干次(≥0 次)循环后值变为 0，以结束循环。

(5) while 循环语句的循环体可能一次也不执行，而 do-while 循环语句循环体至少执行一次，这是二者仅有的区别。因此，在循环体至少执行一次时，二者可以互换。

【例 5.9】 输入两个正整数 m 和 n，求其最大公约数。

问题分析：可以用欧几里得算法(辗转相除法)来求解。

设有两个正整数 m、n：

(1) m 被 n 除，得到余数 r(0≤r≤n)，即 r＝m％n; n→m; r→n。

(2) 若 r＝0，则算法结束，m 为最大公约数，否则转到(1)。

程序如下：

```
#include <iostream>
using namespace std;
int main()
{    int m, n, r;
     cout <<"请输入两个正整数:";
     cin >> m >> n;
```

```
    do
    {    r = m % n; m = n; n = r;
    }while(r!= 0);
    cout <<"最大公约数为"<< m << endl;
    return 0;
}
```

程序运行结果如下：

```
请输入两个正整数: 6 8 ↙
最大公约数为 2
```

5.2.3 for 循环语句

while 循环语句和 do-while 循环语句中循环体的执行次数由循环控制条件来决定，而循环控制条件在循环体中的语句改变，通常用于循环体执行次数未知的循环。在很多问题中，循环次数是已知的，虽然也可以用 while 循环语句和 do-while 循环语句来实现，但是程序逻辑并不是很清晰。为此，C++语言提供了 for 循环语句，for 循环也被称为“计数循环”。for 循环语句的语法格式如下：

```
for(表达式 1;表达式 2;表达式 3)
    循环体
```

for 循环语句的执行流程如下：

(1) 计算表达式 1 的值。

(2) 计算并判断表达式 2 的值，若表达式 2 成立(值不为 0)，则执行第(3)步；否则跳转到第(4)步。

(3) 执行循环体，计算表达式 3 的值，返回第(2)步。

(4) 结束循环，接着执行其后面的语句。

for 循环语句的执行流程如图 5-11 所示。

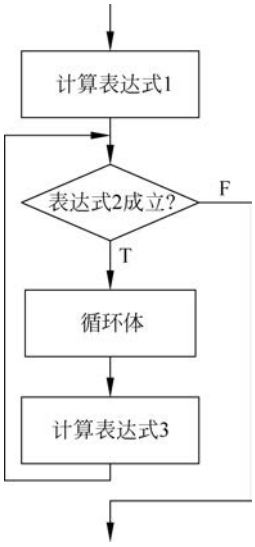

图 5-11　for 循环语句的执行流程

其中，表达式 1～3 均可以为任意符合 C++语言语法规则的表达式，循环体也是单个语句，若为多个语句，则必须用“{}”括起来构成复合语句。从执行流程看，for 循环语句和 while 循环语句本质上是相同的。

【例 5.10】 用 for 循环语句求 1～100 之间奇数之和 sum＝1＋3＋5＋…＋99。

程序如下：

```
#include <iostream>
using namespace std;
int main()
{    int i, sum = 0;
    for(i = 1; i <= 99; i = i + 2)
      sum = sum + i;
    cout <<"sum = "<< sum << endl;
    return 0;
}
```

程序运行结果如下：

```
sum = 2500
```


【例 5.11】 Fibonacci 数列是这样一个数列：1、1、2、3、5、8、13、21、34、55、…，该数列是意大利数学家 Leonardo Fibonacci 由兔子繁殖过程为原型而引入，故又称为“兔子数列”。同时，由于该数列后一项与前一项的比例是趋于黄金分割数的，因此其又被称“黄金分割数列”。该数列前 2 项均为 1，从第 3 项开始，每一项均为该项之前的两项之和。编写程序，输出 Fibonacci 数列的前 20 项，每行显示 5 项。

问题分析：Fibonacci 数列的前两项是已知的，因此可以直接输出。从第 3 项开始，每一次均为其前面两项之和，可以直接通过求和运算得到并输出。本问题要求 Fibonacci 数列的前 20 项，循环次数已知，因此可以使用 for 循环语句来实现。程序流程如图 5-12 所示。

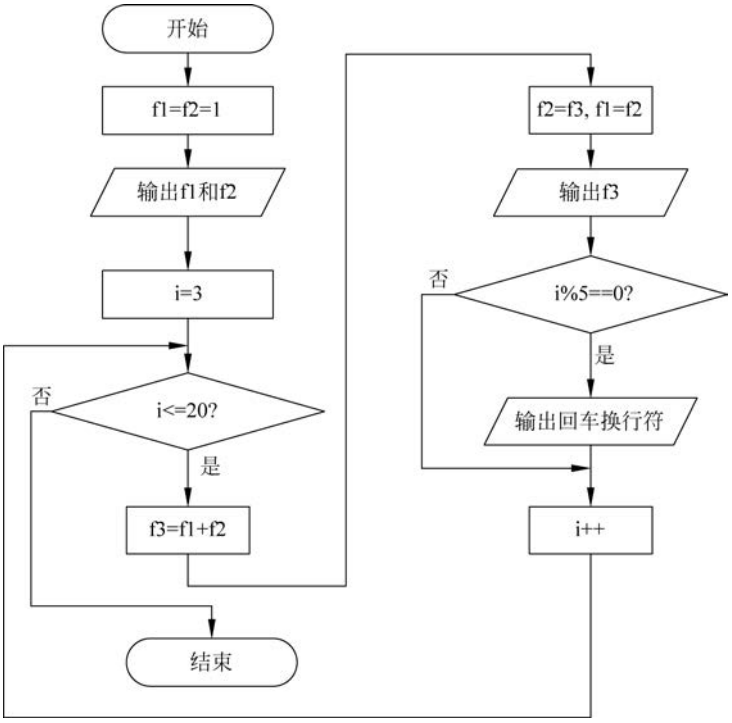

图 5-12　例 5.11 程序流程

程序如下：

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{    int f1, f2, f3;
     int i;
     f1 = f2 = 1;
     cout << setw(10)<< f1 << setw(10)<< f2;
     for (i = 3;i <= 20;i++)
```

```
    {   f3 = f2 + f1;
        f1 = f2;
        f2 = f3;
        cout << setw(10)<< f3;
        if (i % 5 == 0) cout << endl;
    }
    return 0;
}
```

程序运行结果如下：

```
1         1         2         3         5
8         13        21        34        55
89        144       233       377       610
987       1597      2584      4181      6765
```

使用 for 循环语句实现循环时需要注意以下问题：

(1) 一般情况下，for 循环语句中，表达式 1 用作循环控制变量初始化，表达式 2 用作对循环控制变量的值进行判断，表达式 3 用作改变循环控制变量的值。例如，下列求自然数 1～10 之和的程序段中，表达式 1“i＝1”对循环控制变量赋初值为 1，表达式 2“i＜＝10”用于判断循环控制变量 i 是否不大于 10，表达式 3“i＋＋”用于每次执行循环体后对循环控制变量 i 加 1。

```
sum = 0;
for(i = 1; i <= 10; i++)
     sum += i;
```

(2) for 循环语句中，表达式 1～3 均可以为任何符合 C++语言语法规则的表达式，甚至可以为空。例如，(1)中的程序段可以写为

```
for(sum = 0, i = 1; i <= 10; i++)
     sum += i;
```

也可以写为

```
for(sum = 0, i = 1; i <= 10; sum += i, i++);
```

甚至可以写为

```
sum = 0;
i = 1;
for(; i <= 10;)
{   sum += i;
    i++;
}
```

当表达式为空时，中间起分隔作用的“;”不能省略。若表达式 2 被省略，则在循环体中必须有使循环能够结束的语句，否则会出现“死循环”。例如，上面的程序段可以写为

```
sum = 0;
i = 1;
for(;;)
{   sum += i;
    i++;
    if (i > 10) break;
}
```

(3) 尽管表达式 1～3 均可以为空，也可以为任何符合 C++语言语法规则的表达式，但为了提高程序的可读性，建议把与循环控制变量无关的操作放在语句之前或循环体中。for 循环语句的常用语法格式如下，其中“步长”为循环控制变量每次增加或减小值。

```
for(循环控制变量 = 初值;循环控制变量<= 终值;循环控制变量 += 步长)
    循环体
```

或者：

```
for(循环控制变量 = 初值;循环控制变量>= 终值;循环控制变量 -= 步长)
    循环体
```

(4) for 循环语句可以使用 while 循环语句或 do-while 循环语句来代替。例如，(1)中程序段可以写为

```
sum = 0;
i = 1;
while (i <= 10)
{   sum += i;
    i++;
}
```

或者：

```
sum = 0;
i = 1;
do
{   sum += i;
    i++;
}while(i <= 10);
```

5.2.4 循环嵌套

while 循环语句、do-while 循环语句和 for 循环语句 3 种循环语句的循环体可以是任意符合 C++语言语法规则的语句，当然也可以是循环语句，或者是包含循环语句的复合语句。如果一个循环语句的循环体中又包含完整的循环语句，则称其为循环嵌套，也被称为多重循环。

【例 5.12】 我国北魏时期数学家张丘建在《张丘建算经》一书中曾提出过著名的“百钱买百鸡”问题，该问题描述如下：鸡翁一，值钱五；鸡母一，值钱三；鸡雏三，值钱一；百钱买百鸡，问翁、母、雏各几何？编写程序，输出该问题的所有解。

问题分析：百钱最多可以买鸡翁 100/5＝20 只，鸡母 100/3＝33 只，而鸡的总数为 100，

因此鸡雏的数量最多也只能是 100 只。只需要对鸡翁的数量(0～20)、鸡母的数量(0～33)和鸡雏的数量(0～100)的各种组合进行判断,若一个组合中鸡的总数为 100 且总钱数也为 100,同时鸡雏的数量是 3 的倍数,则该组合即为问题的一个解。该问题可以用 for 循环语句的嵌套来实现。

程序如下:

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{    int cock, hen, chick;
     cout << setw(10)<<"鸡翁"<< setw(10)<<"鸡母"<< setw(10)<<"鸡雏"<< endl;
     for( cock = 0; cock <= 20; cock++)
          for(hen = 0; hen <= 33; hen++)
               for( chick = 0; chick <= 100; chick++)
               {     if(cock * 5 + hen * 3 + chick/3 == 100 && chick % 3 == 0 &&
                        cock + hen + chick == 100 )
                     { cout << setw(8)<< cock << setw(11)<< hen << setw(10)<< chick << endl;
                     }
               }
     return 0;
}
```

程序运行结果如下:

```
鸡翁     鸡母     鸡雏
0        25       75
4        18       78
8        11       81
12       4        84
```

5.3 其他流程控制语句

程序设计语言是否应该支持诸如 goto 语句这样的非结构化语句的问题,在二十世纪六七十年代曾经引发激烈的争论。结构化程序设计支持者强调程序只能由顺序、选择和循环 3 种基本结构构成,且每个结构应该是“单入口单出口”,这样编写的程序可读性好;非结构化程序设计支持者则认为结构化程序设计限制了程序设计者的自由,且使用非结构化程序设计可以编写出执行效率更高的程序。作为妥协的产物,现在支持结构化程序设计的程序设计语言中大多保留了部分不符合结构化程序设计思想的语句。C++语言中,这类语句包括 break 语句、continue 语句和 goto 语句。

5.3.1 break 语句

break 语句在前面 switch 语句中已经提到过,它用于跳出 switch 语句中 break 语句之后的分支,转去执行 switch 语句后面的语句。break 语句还可以用于循环体中,用于跳出当前循环,转到循环语句后面的语句执行。break 语句的语法格式如下:

```
break;
```

【例 5.13】 输入一个大于 1 的正整数，判断其是否是素数。

问题分析：

（1）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。

（2）判断一个数 x 是否为素数，可以依次用 2～x－1 作为除数，判断 x 能否被其整除。只要能被其中之一整除，则说明 x 不是素数，可以提前结束循环。

（3）进一步，由于 x 不可能被大于 x/2 的数整除，因此可将循环次数降低，x 只需被 2～x/2 整除即可。

程序流程如图 5-13 所示。

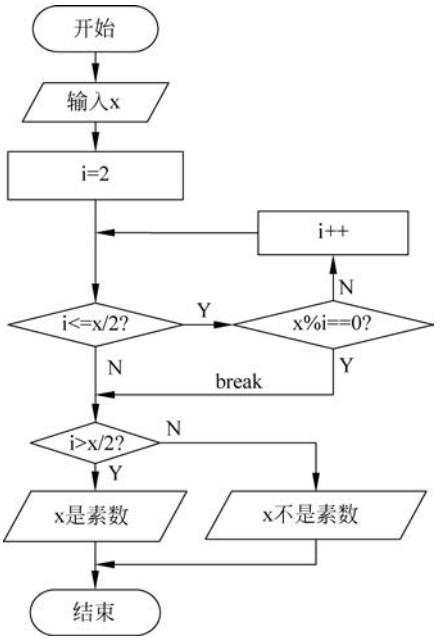

图 5-13　例 5.13 程序流程

程序如下：

```
#include<iostream>
using namespace std;
int main()
{    int x, i;
     cout<<"请输入一个大于 1 的正整数:";
     cin>>x;
     for (i=2; i<=x/2; i++)                    //i 作为除数,从 2～x/2 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,则不必再判断其他因子
     if (i>x/2)                                //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;
}
```

程序运行结果如下：

```
请输入一个大于 1 的正整数：  21 ↙
21 不是素数
请输入一个大于 1 的正整数：  29 ↙
29 是素数
```

程序中，当 x 能被 i 整除时，则表明 x 不是素数，不需要再判断其他因子，这时可用 break 语句提前结束循环，直接执行本循环后的语句。需要注意的是，在多重循环中，一个 break 语句只能结束一层循环。

5.3.2 continue 语句

continue 语句只能用在循环体中，用于跳过本层循环体中 continue 语句之后的语句，直接进行下一轮循环的判断。continue 语句的语法格式如下：

```
continue;
```

【例 5.14】 编程求整数 10～30 中 5 的倍数之和。

问题分析：需要对 10～30 中的每一个整数进行检查，如果能被 5 整除，就累加；否则，就检查下一个整数是否符合要求。

程序如下：

```
#include <iostream>
using namespace std;
int main()
{    int x, sum = 0;
     for(x = 10; x <= 30; x++)
     {     if (x % 5!= 0) continue;
           sum = sum + x;
     }
     cout <<"10～30 中 5 的倍数之和为"<< sum << endl;
     return 0;
}
```

程序运行结果如下：

```
10～30 中 5 的倍数之和为 100
```

5.3.3 * goto 语句

goto 语句是一种无条件转移语句，用于将程序执行流程转移到指定语句，而不是该语句后面的语句。goto 语句的语法格式如下：

```
goto 语句标号;
```

其中，语句标号用于标注语句地址。

带标号的语句形式如下：

```
语句标号：语句
```

语句标号的定义与变量名的定义规则相同。

goto 语句通常与 if 语句配合使用，以实现有条件转移。

【例 5.15】 用 goto 语句和 if 语句实现求整数 10～30 中 5 的倍数之和。

```
#include <iostream>
using namespace std;
int main()
{   int x = 10, sum = 0;
    loop:                       //loop 为语句标号
    if (x <= 30)
    {    if (x % 5 == 0) sum = sum + x;
         x++;
         goto loop;
    }
    cout <<"10～30 中 5 的倍数之和为"<< sum << endl;
    return 0;
}
```

程序运行结果如下：

```
10～30 中 5 的倍数之和为 100
```

5.4 程 序 举 例

【例 5.16】 用生成伪随机数的库函数 rand()设计一个自动出题程序，输出两位正整数的四则运算表达式，并对输入的计算结果的正确性进行判断。

问题分析：C++语言提供的库函数 rand()产生的是一串固定序列的随机整数，要使每次运行产生不一样的值，就需要使用 srand()函数。srand()函数用来选择初始位置，称为初始化随机数种子。一般用当前时间初始化随机数种子，这样产生的序列更接近真正的随机数。

常用以下语句产生随机数：

```
srand(time(NULL)); //初始化种子
x = rand() % (终值 - 初值 + 1) + 初值
```

(1) 用 rand()函数生成 10～99 的两位正整数。

```
rand() % 90 + 10
```

(2) 所做运算有 4 种，可用 0～3 表示，用 switch 语句实现选择。

程序如下：

```
#include <iostream>
#include <ctime>
using namespace std;
int main()
{   int a, b, c, d, op;
    /*a、b 保存随机生成的两操作数，c 保存程序计算结果，d 保存答题者的答案，
    op 保存随机生成的运算类型 */
    srand((unsigned)time(NULL));                    //初始化随机数种子
```

```
    a = rand() % 90 + 10;
    b = rand() % 90 + 10;
    op = rand() % 4;                              //共有 4 种运算,分别用 0、1、2、3 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; break;
         case 3: cout << a <<'/'<< b <<' = '; c = a/b;
    }
    cin >> d;                                     //输入计算结果
    if (d == c)
         cout <<"正确!\n";
    else
         cout <<"错误,请继续努力!\n";
    return 0;
}
```

程序运行结果如下：

```
51 * 27 = 1377 ↙
正确!
51 + 27 = 77 ↙
错误,请继续努力!
```

说明：初始化随机数种子如果写成“srand(time(NULL));”，编译系统会给出一个警告错误，原因是 VC 2010 版中，time()函数的返回值 time_t(绝对秒数)是 64 位的，而 srand()的定义是 void srand(unsigned int seed)，其参数是 32 位的 unsigned int，所以会丢失数据。需要将其改成“srand((unsigned)time(NULL));”，强制转换 time_t 到 unsigned int。

【例 5.17】 自然常数 e 是数学中常用的一个常数，它是无限不循环小数，也被称为欧拉常数或纳皮尔常数。可以通过下式求得 e 的近似值：

$$e \approx 1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\cdots+\frac{1}{n!}+\cdots$$

编写程序求 e 的近似值，直到最后一项小于 10^{-6}。

问题分析：本题属于累加求和问题，可用循环语句解决。

(1) 当前通项值 t 是其前一项乘以 1/n。

(2) 重复的操作：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。

程序如下：

```
#include <iostream>
using namespace std;
int main()
{    double e = 1, t = 1, n = 0;
     while(1.0/t >= 1e-6)
     {    n = n + 1;
          t *= n;
          e = e + 1.0/t;
     }
     cout <<"e = "<< e << endl;
```

```
    return 0;
}
```


程序运行结果如下：

```
e = 2.71828
```

【例 5.18】 牛顿迭代法是在实数域和复数域上求解方程近似根的重要方法之一。它从任意一个初值开始，通过迭代求得方程的近似根。该方法的具体过程如下：对方程 $f(x)=0$，给定初值 x_1，计算 $f(x_1)$的值，过点$(x_1,f(x_1))$作曲线 $y=f(x)$的切线交 x 轴于 x_2，再计算 $f(x_2)$，过点$(x_2,f(x_2))$作曲线 $y=f(x)$的切线交 x 轴于 x_3，…，这样一直进行下去，所得到的值 x_n 将越来越接近于方程的根(图 5-14)。当相邻两次所求值的差足够小时，x_n 就可以作为方程的近似根。显然，曲线 $y=f(x)$上过点$(x_1,f(x_1))$的切线的斜率为

$$f'(x_1)=f(x_1)/(x_1-x_2)$$

由此可得

$$x_2=x_1-f(x_1)/f'(x_1)$$

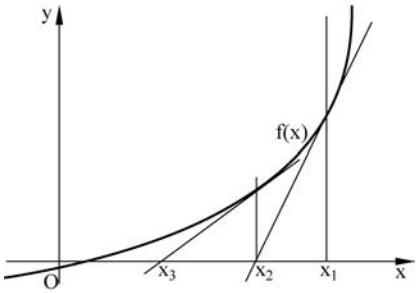

图 5-14　牛顿迭代法

以此类推，可得牛顿迭代法的迭代公式为

$$x_n=x_{n-1}-f(x_{n-1})/f'(x_{n-1})$$

编程求方程 $x^2+2x+e^x-4=0$ 在 $x=5$ 附近的近似根，要求相邻两次迭代误差小于 10^{-8}。

问题分析：这是典型的迭代算法。由给定的初值 x_1，根据公式求得 x_2，再由 x_2 根据公式可求得 x_3，…。由于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值，因此仅使用两个变量即可。也就是说，将前一次计算的结果 x_2 作为本次计算的初值 x_1，再根据公式求得新的 x_2，不断重复以上过程，一直到$|x_2-x_1|<10^{-8}$ 为止，这时 x_1 或 x_2 为此方程的近似根。其中，f(x)的导数为 $2x+2+e^x$，C++语言提供了数学库函数 exp(x)用来求 e^x。

程序如下：

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{   double x1, x2 = 5, fx, dx;
```

```
    do
    {   x1 = x2;
        fx = x1 * x1 + 2 * x1 + exp(x1) - 4;
        dx = 2 * x1 + 2 + exp(x1);
        x2 = x1 - fx/dx;
    }while(fabs(x2 - x1)> 1e - 8);
    cout <<"方程的近似根为"<< x2 << endl;
    return 0;
}
```

程序运行结果如下：

```
方程的近似根为 0.717662
```

【例 5.19】 传说古印度有一位名叫舍罕的国王因为一位大臣发明了国际象棋而打算重奖他，这位聪明的大臣跪在国王面前说："陛下，请您在这张棋盘的第一个小格内放一粒麦子，在第二个小格内放两粒，在第三个小格内放四粒，照这样下去，每一小格内都比前一小格增加一倍。陛下啊，把这样摆满棋盘上所有 64 格的麦粒，都赏给您的仆人吧！"国王想都没想就答应了。若每粒麦子的质量是 0.015g，编写程序计算放满整个棋盘需要多少吨麦子。

问题分析：这是一个求累加和的问题。循环次数已知，因而可以用 for 语句来实现。sum 的初值为 0，通项 t 初值为 1，后项是前项的 2 倍，则重复执行的操作（循环体）为"sum＝sum＋t；t＝t＊2；"。

程序如下：

```
#include < iostream >
using namespace std;
int main()
{   double sum = 0,t = 1;
    int i;
    for(i = 1;i < = 64; i++)
    {   sum = sum + t;                    //sum 表示总的数量
        t = t * 2;
    }
    sum * = 0.015;
    sum/ = 1e6;                           //1t 等于 1000kg
    cout <<"共重"<< sum <<"吨！"<< endl;
    return 0;
}
```

程序运行结果如下：

```
共重 2.76701e + 011 吨!
```

习　　题

一、选择题

1. if 语句后的表达式应该是_____。

A. 赋值表达式

B. 关系表达式

C. 任意符合 C++语言语法规则的表达式

D. 算术表达式

2. 运行下列程序段后，x 的值为_____。

```
float x = 1, y = 3, z = 5;
if (x + y > z)
      if (y < z) x = y;
      else x = z;
```

A. 1　　B. 3　　C. 4　　D. 5

3. 在嵌套使用 if 语句时，C++语言规定 else 总是_____。

A. 和之前与其具有相同缩进位置的 if 配对

B. 和之前与其最近的 if 配对

C. 和之前与其最近的且未配对的 if 配对

D. 和之后的第一个 if 配对

4. 运行下列程序段时，若从键盘输入 1，则 y 的值为_____。

```
int x, y = 0;
cin >> x;
switch(x)
{    case 1: y++;
     case 2: y++;
     default: y++;
}
```

A. 0　　B. 1　　C. 2　　D. 3

5. 下列程序段运行后的输出结果是_____。

```
int x = 1;
switch(x + 1)
{    case 1: cout <<"One";break;
     case 2: cout <<"Two";break;
     case 3: cout <<"Three";break
     default: cout <<"Error";break;
}
```

A. One　　B. Two　　C. TwoThree　　D. TwoThreeError

6. 下列程序段运行后的输出结果是_____。

```
int x = 15;
if (x % 2 == 0) cout << x/2;
else cout << x/2 + 1;
```

A. 7　　B. 7.5　　C. 8　　D. 8.5

7. 下列 while 循环的执行次数为_____。

```
int k = 2;
while (k = 1) k -- ;
```

A. 0　　B. 1　　C. 2　　D. 无限

8. 下列 while 循环的执行次数为_____。

```
int k = 2;
while (k == 1) k -- ;
```

A. 0　　B. 1　　C. 2　　D. 无限

9. 语句“while(e);”中的条件 e 等价于_____。

A. e==0　　B. e!=1　　C. e==1　　D. e!=0

10. 下列程序段运行后的输出结果是_____。

```
int n = 9;
while (n > 6) cout << -- n;
```

A. 987　　B. 876　　C. 8765　　D. 9876

11. 下列有关 break 语句和 continue 语句的叙述正确的是_____。

A. 前者用于循环语句,后者用于 switch 语句

B. 前者用于循环语句或 switch 语句,后者用于循环语句

C. 前者用于 switch 语句,后者用于循环语句

D. 前者用于循环语句,后者用于循环语句或 switch 语句

12. 下列程序段运行后的输出结果是_____。

```
int a = -1,b = 0;
while(a++) ++b;
cout << a <<'\t'<< b << endl;
```

A. 0 1　　B. 1 1　　C. 1 2　　D. 2 3

13. 下列程序段运行后的输出结果是_____。

```
x = 3;
do{   y = x -- ;
      if(!y)
      {   cout <<'x';
          continue;
      }
      cout <<'#';
}while(x >= 1 && x <= 2);
```

A. 将输出##　　B. 是死循环

C. 将输出###　　D. 含有不合法的控制表达式

二、填空题

1. 下列程序片段的运行结果是_____。

```
for(int i = 0;i < 5;i++)
     if (i % 2) cout << i <<" ";
```

2. 下列程序的运行结果是_____。

```
#include <iostream>
using namespace std;
int main()
{    int n = 351, s = 0;
     do{   s += n % 10;
           n /= 10;
       }while(n);
     cout << s << endl;
     return 0;
}
```

3. 下列程序片段的运行结果是_____。

```
int a, b;
for(a = 0, b = 0; a <= 5; a++)
{    if (b >= 8) break;
     if (a % 2 == 1) { b += 7; continue; }
     b -= 3;
}
cout << a <<',' << b << endl;
```

4. 下列程序的运行结果是_____。

```
#include <iostream>
using namespace std;
int main(){
for(int i = -1;i < 4;i++)
  cout <<(i?'0':'*');
return 0;
}
```

5. 下列程序片段的运行结果是_____。

```
int i = 0, j = 0, k = 6;
if (++i > 0 || ++j > 0) k++;
cout << i <<',' << j <<',' << k << endl;
```

三、编程题

1. 编写程序，根据下列公式，由键盘输入 x 的值，计算 y 的值。x、y 均为 double 类型。

$$y=\begin{cases} x-1, & -5\leqslant x\leqslant 5 \\ x+1, & 5<x\leqslant 10 \\ 15.6, & \text{其他} \end{cases}$$

2. 编写程序，求出所有的水仙花数。如果一个 3 位数每个位上的数字的 3 次幂之和等于它本身，则称该数为水仙花数。例如，$153=1^3+5^3+3^3$，所以 153 是水仙花数。

3. 编写程序，输出 100～200 所有的素数，每行输出 5 个。

4. 用迭代法求 $x=\sqrt{a}$ 的近似值，设迭代初值为 a/2。要求相邻两次求出的 x 的差的绝对值小于 10^{-5}。迭代公式如下：

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

5. 编程求一元二次方程 $ax^2+bx+c=0$ 的实数解，系数 a、b、c 从键盘输入。

6. 征税的办法如下：收入在 800 元以下（含 800 元）的不征税；收入在 800 元以上、1200 元以下者，超过 800 元的部分按 5%的税率收税；收入在 1200 元以上、2000 元以下者，超过 1200 元部分按 8%的税率收税；收入在 2000 元以上者，2000 元以上部分按 20%的税率收税。编写按收入计算税费的程序。

7. 编写程序，输出从公元 1800 年～公元 2000 年中所有闰年的年份，一行输出 8 个。判断公元年份是否为闰年的条件如下：①年份如能被 4 整除，而不能被 100 整除，则为闰年；②年份能被 400 整除也是闰年。

8. 编写程序，输入一行字符，分别统计数字字符、字母字符和其他字符的个数。