第5章

程序流程控制——循环

计算机对于数据的强大处理能力常常是通过循环进行相同的操作而获得的。循环处理是程序设计的必备流程控制结构。

5.1 循环指令的种类

重复处理可以形成首尾相接的动作称为循环 (loop)，其构成有四要素：

1. 循环内要进行的动作。
2. 决定要不要继续重复操作的判断式。
3. 判断式的初始参数设定。
4. 让循环能够停止的参数改变方式。

缺一不可。

循环的指令有三种:

while 循环，for 循环和 do-while 循环，

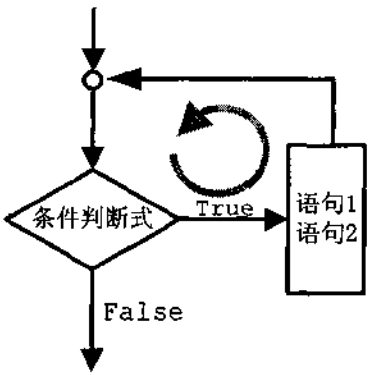
其中 while 和 for 循环都是先判断后再执行，称之为执行前测试循环 (pretest loop) 或进入控制循环 (entrance-controlled loop)。

而 do-while 循环则是先执行再判断要不要继续，称之为执行后测试循环 (post-test loop) 或跳出控制循环 (exit-controlled loop)，循环内的处理动作至少会执行一次。

5.2 while 循环

只要条件判断式的结果为 true(或是任何非零的数值)，while 循环 (while loop)就会重复执行语句，其语法及流程图如下所示：

```
while(条件判断式)
{
   语句1
   语句2
}
```



在流程图中我们也注意到循环 (loop) 的存在 (图中浅色带有箭头的环)，代表循环进行的重复步骤．由于 C++ 的区块 (又称为复合语句) 可以视为单一语句，因此上述语法可以进一步写成以下的精简语法：

```
while (条件判断式)
    语句
```

范例程序

程序 Sum.cpp 通过 while 循环计算从 1 到使用者输入的整数和 (例如：输入 100，计算从 1 到 100 的整数和)。

```
// Sum.cpp
#include <iostream>
using std::cin;
using std::cout;
using std::endl;
// ------ 主程序 --------------------
int main()
{
  int N, Sum, Count ;

  cout << "请输入要累计的数目:" << endl;
  cin  >> N;
  Count = 1 ;
  Sum = 0;
  while ( Count <= N )
  {
    Sum += Count ;
    Count++;
  }
  cout << "总和是: " << Sum << endl ;
  return 0;
}
```

典型的操作过程

```
请输入要累计的数目：
90
总和是： 4095
```

讨论

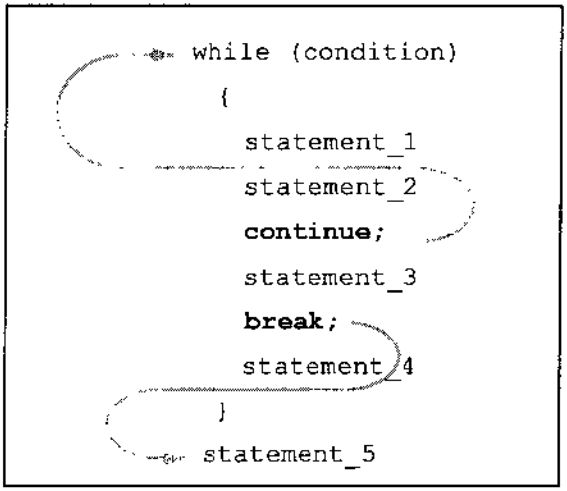
1. Count 是 while 循环内部所要使用的累计数据，而 Sum 则是累计的总和，两者都要给予适当的初值。

2. “Count <= N”是 while 循环的条件判断式。如果其判断结果是 0(逻辑上的“false”)，则循环内的语句不会执行，如果其判断结果不是 0(逻辑上的“true”)，则循环内的语句就会执行。一定要安排让循环的条件判断式的值有可能是 0，否则循环无法停止。本程序是以 Count 累加（亦即“Count++;”）让条件判断式最后变为 0（亦即“Count > N”），以结束 while 循环。

5.3 continue 和 break

所有的重复处理结构，包括 while，for，和 do-while 三种循环都支持 continue 和 break 两个指令。

continue 让程序忽略其后的语句，直接跳回到循环的开头，而 break 则直接跳离循环，如下图所示：



范例程序

在程序 Sum2.cpp 中，我们使用语句“break;”改写 5.2 节的 Sum.cpp：

```
// Sum2.cpp
#include <iostream>
using std::cin;
using std::cout;
using std::endl;
// ------ 主程序 --------------------
int main()
{
  int N, Sum, Count ;

  cout << "请输入要累计的数目:" << endl;
  cin  >> N;
  Count = 1 ;
  Sum = 0;
  while ( true )
  {
    Sum += Count ;
    Count++;
```

此为试读，需要完整PDF请访问：www.ertongbook.com

```
        if ( Count > N ) break;
      }
      cout << "总和是: " << Sum << endl ;
      return 0;
    }
```

不过，如同 goto 语句一样，break 和 continue 都容易让程序脱离控制结构的规范，导致不易调试，必须小心使用。

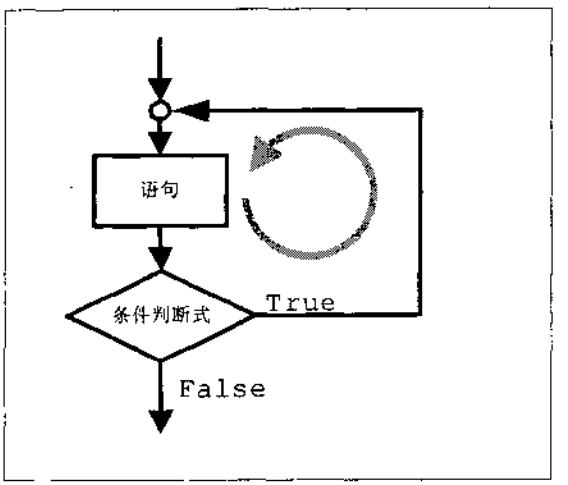
5.4 do-while 循环

do-while 循环 (do-while loop) 是先执行再判断的语句，所以其内含的语句至少会执行一次。其语法为：

```
do
   语句
while (条件判断式);
```

注意 do-while 循环必须以“;”结尾。

do-while 循环的流程图可以画成下面的样子：



范例程序

在程序 Guess.cpp 里，要求使用者从 A 至 E 里猜一个字母，只有与程序内部预设的字母（在这里暂时设为 D）吻合才能结束此程序：

```
// Guess.cpp
#include <iostream>
using std::cin;
using std::cout;
using std::endl;
// ------ 主程序 --------------------
int main()
{
  char Secret1 = 'D', Secret2 = 'd';
  char Guess;

  do
  {
    cout << "请输入一个从 A 至 E 的字母："
         << endl ;
    cin  >> Guess;
  } while ((Guess != Secret1) && (Guess != Secret2));

  cout << "你猜对了，就是" << Secret1 << endl ;
}
```

操作结果

```
请输入一个从 A 至 E 的字母：
b
请输入一个从 A 至 E 的字母：
c
请输入一个从 A 至 E 的字母：
d
你猜对了，就是 D
```

范例程序

上述程序 Guess.cpp 也可以直接使用 while 循环完成，如下列程序 Guess2.cpp：

```
// Guess2.cpp
#include <iostream>
using std::cin;
using std::cout;
using std::endl;
// ------ 主程序 --------------------
int main()
{
  char Secret1 = 'D', Secret2 = 'd';
  char Guess   = 'z';

  while((Guess != Secret1) && (Guess != Secret2))
  {
     cout << "请输入一个从 A 至 Z 的字母："
          << endl ;
     cin  >> Guess;
   }

  cout << "你猜对了，就是" << Secret1
       << endl ;
}
```

讨 论

在这个程序中，由于一开始就面临判断，因此 Guess 需要有初值，例如语句“**char Guess = 'z';**”。

5.5　for 循环

for 循环 (for loop) 可以说是最常用的重复处理语法.它是 while 循环的一种紧凑的写法，将分散各处的初始值设定式，条件判断式和参数改变式都写在一起，用小括号包起来．for 循环的语法如下：

```
for (初始值设定式; 条件判断式; 参数改变式 )
    语句
```

其中小括号内由分号区隔的三个表达式 (expressions) 的内容分别是：

- 初始值设定式：设定 for 循环内部各变量在整个 for 循环开始运算前的初值．
- 条件判断式：决定循环继续运算的条件．此条件如果满足(表达式的值不为 0)，则循环继续运转，否则循环停止，接着处理 for 循环之后的语句．
- 参数改变式：每次循环结束前执行的语句．通常会通过循环内某一个特定参数的递增或递减 (这类参数都是整数，而且名称通常为 i，j，或 Count 等)，以使每次循环都会产生不同的结果．

例如:

```
for ( int i = 0 ; i < 10 ; i++ )
    cout << i ;
```

在这个典型的程序片断中，上述“循环内某一个特定参数”即这里的整数 i:

1. 上述 i 的声明方式让它成为局部变量 (local variable)，只能在循环内适用．整数 i 也可以在整个 for 循环之前声明，例如

```
int i;
for ( i = 0 ; i < 10 ; i++ )
    cout << i ;
```

在这种写法下，i 在进入循环前就会被给予初值 (在这个例子中为 0)，因此不会影响结果。

2. 条件判断式为“i<10”而参数改变式为“i++”，因此，整数 i 从 0 开始，每个循环都会增加 1，一直到 9 才会结束，输出为：

```
0123456789
```

3. 由于参数改变式在每次循环结束前才执行，因此，整数 i 的递增式写成“i++”(后置) 或“++i”(前置) 的效果都一样。

对于 for 循环这种特殊的语法，它的流程图可以用一个六角形的判断符号很精简地表示成如图 5.5.1 的形式：

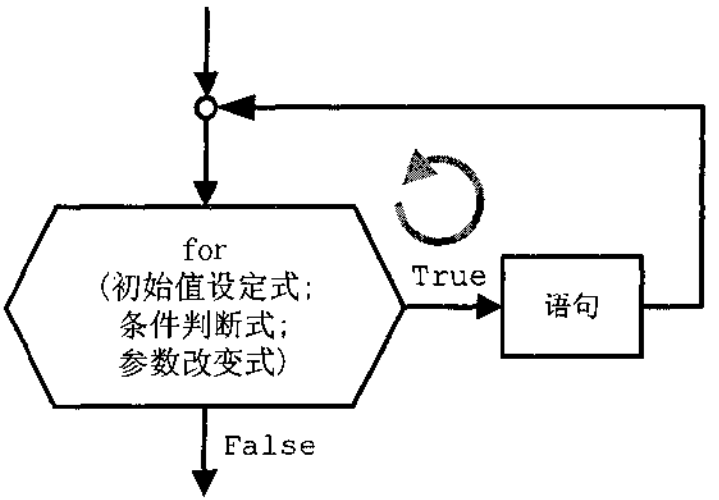


图 5.5.1　for 循环的流程图

范例程序

我们可以将程序 Sum.cpp 以 for 循环很精简地改写成以下程序：

```
// SumFor.cpp
#include <iostream>
using std::cin;
using std::cout;
using std::endl;
// ------ 主程序 --------------------
int main()
{
  int N, Sum, Count ;
  cout << "请输入要累计的数目:" << endl;
  cin  >> N;
  Sum = 0;
  for ( Count = 1 ; Count <= N ; Count++ )
        Sum += Count ;
  cout << "总和是: " << Sum << endl ;
  return 0;
}
```

for 循环的 continue 和 break

for 循环也支持 continue 和 break 两个指令，和 while、do-while 循环一样。但是，continue 指令在 for 循环之内的特性比较特殊，它在执行时并不是直接跳到循环开始的地方，而是在跳回之前先执行参数改变式。事实上，这是必然的结果，因为如果不是如此的话，下一个循环将会有一样的参数，for 循环也就进入无穷循环，无法跳出来。

我们在以下程序 CandB.cpp 的 for 循环内使用 continue 和 break 以验证上述的讲法：

此为试读，需要完整PDF请访问：www.ertongbook.com

范例程序 CandB.cpp

```
// CandB.cpp
#include <iomanip>
using std::cin;
using std::cout;
using std::endl;
// ------ 主程序 --------------------
int main()
{
  for ( int i = 2 ; i <= 20 ; cout << "C ", i=i+2 )
  {
    cout << i;
    if (i==6)  continue;
    cout << "A";
    if (i==12) break;
    cout << "B";
  }
  cout << endl ;
}
```

操作结果

```
2ABC 4A3C 6C 8ABC 10ABC 12A
```

讨论

1. 这个 for 循环内的参数 i 从 2 开始，每个循环的循环增加 2。
2. “参数改变式”在上述 for 循环内包括“cout <<"C "”和“i=i+2”两个表达式，且以逗号“，”分开。
3. 当 i 等于 6 时执行“**continue;**”，因此跳过“cout <<"A";”和“cout <<"B";”，但是会执行上述参数改变式，以致“cout <<"C "”仍会执行。
4. 当 i 等于 12 时执行“**break;**”，因此跳过“cout <<"B";”，直接离开 for 循环。

for 循环的其它写法

有趣的是，for 循环也可使用类似 while 循环的写法，只是这个时候必须保有原来 for 之后小括号内的两个分号。

例如，程序 SumFor.cpp 的 for 循环：

```
for ( Count = 1 ; Count <= N ; Count++ )
      Sum += Count ;
```

可以改成下列任何一种写法：

(1) 将 Count 的初始值设定移到 for 循环之前

```
Count = 1;
for ( ; Count <= N ; Count++ )
      Sum += Count;
```

(2) 将参数改变式移到 for 循环的本体复合语句之内

```
Count = 1;
for ( ; Count <= N ; )
{
  Count++ ;
  Sum += Count ;
}
```

(3) 将循环内容移到参数改变式之内。这时 for 循环的本体语句就只剩下空语句 (null statement) ";" 而已。

```
Count = 1;
for ( ; Count <= N ; Sum += Count, Count++);
```

虽然上述三种写法都是正确的，也可以得到完全相同的结果，但仍以原来的写法较容易

理解。

for 循环最常用于需要将某些处理重复执行固定次数的场合。例如，要执行语句 8 次，则可以写成

```
for ( int i = 0 ; i < 8 ; i++ )
  要执行的语句
```

或

```
for ( int i = 1 ; i <= 8 ; i++ )
  要执行的语句
```

范例程序 Temp.cpp

以下我们使用 for 循环以产生一个摄氏与华氏温度的对照表：

```
// Temp.cpp
#include <iomanip>
using std::cin;
using std::cout;
using std::endl;
using std::setw;
// ------ 主程序 --------------------
int main()
{
  float C;
  cout << "  摄氏    华氏 " << endl ;
  cout << "-----------------"  << endl ;

  for ( int i = 1 ; i <= 10 ; i++ )
  {
    C = 10.0*i;
    cout << setw(5) << C << "  " << setw(5)
         << C*9.0/5.0 + 32.0     << endl ;
  }
  cout << "-----------------"  << endl ;
}
```

操作结果

```
  摄氏     华氏
----------------
  10       50
  20       68
  30       86
  40      104
  50      122
  60      140
  70      158
  80      176
  90      194
 100      212
----------------
```

讨论

1. 为了输出数字的对齐，我们在程序 temp.cpp 中使用了函数 setw() 来设定字段宽度 (setw 为 set width 的缩写)。使用这个输出格式的设定函数需要在程序开头的地方加入头文件 <iomanip>。我们将在第 13 章详细说明输出格式的设定。
2. 计算摄氏温度的时候，计算式 "C = 10.0*i;" 使用了隐式数据类型转换 (implicit type conversion)，将 i 先转成数据类型 float 再和 10.0 相乘。我们也可明白地写成 "C = 10.0*float(i);"。

5.6 嵌套循环

由于循环本身也是语句，因此可以放在其它循环内部，形成循环内有循环的嵌套循环(nested loops)，常用于矩阵的处理。

我们在下列程序 DisIntM.cpp 中使用嵌套循环以显示出一个矩阵的内容。

范例程序 DisIntM.cpp

```
// DisIntM.cpp
#include <iomanip>
using std::cin;
using std::cout;
using std::endl;
using std::setw;
// ------ 主程序 --------------------
int main()
{
  int M=3, N=5;
  cout << M <<" 行 " << N
       << " 列矩阵:" << endl;
  for (int i = 1; i <= M; i++)
  {
    for(int j = 1; j <= N ; j++)
      cout << setw(5)
           << i << j ;
    cout << endl;            // 断行
  }
  return 0;
}
```

操作结果

```
3 行 5 列矩阵：
   11    12    13    14    15
   21    22    23    24    25
   31    32    33    34    35
```

讨论

1. 这个矩阵是由整数组成，各元素相当于由行数接着列数两个数字的组合。
2. 在每一个内循环结束后，都加了语句“cout << endl; ”来断行，让结果更清楚。
3. 为了输出数字的对齐，我们在程序中使用了函数 setw()来设定字段宽度。使用这个输出格式的设定函数需要在程序开头的地方加入头文件 <iomanip> 和 5.5 节 Temp.cpp 程序的用法相同。

5.7　常犯的错误

1. 和上一章相同，循环处理的结构中所使用的条件判断式容易混淆逻辑判断的相等符号“==” 和赋值运算子 “=”。

2. 多放置了分号。

(1) 如果在 while 循环中的关键词 while 之后多了分号：

```
while (判断式);
  语句
```

则其语法相当于

```
while (判断式)
```

此为试读，需要完整PDF请访问：www.ertongbook.com