# 第5章 循环控制语句

**本章学习目标**

- 理解循环控制语句的结构；
- 掌握 while、do…while 循环语句的使用；
- 掌握 break、continue、goto 语句的使用；
- 掌握 for 循环语句的使用；
- 掌握循环控制语句的应用。

选择条件语句实现逻辑控制处理，而在 C 语言程序中，除了顺序结构和选择结构，还有循环结构。循环结构用来重复执行某一个或多个任务，其特点是：当满足判定条件时，反复执行一个语句序列。本章将主要介绍循环语句的两种重要分支，即 while 语句和 for 语句。除此之外，本章还将介绍一些转移语句的使用，这些语句与循环语句配合使用，可以实现更加复杂的程序需求。

## 5.1 while 循环语句

### 5.1.1 while 循环的基本形式

while 循环语句是循环结构的一种，其一般的形式如下所示。

```
while(循环条件){
    循环体
}
```

while 语句首先会检验括号中的循环条件，当条件为真时，执行其后的循环体。每执行一遍循环，程序都将回到 while 语句处，重新检验条件是否满足。如果一开始条件就不满足，则不执行循环体，直接跳过该段代码。如果第一次检验时条件满足，那么在第一次或其后的循环过程中，必须有使得条件为假的操作，否则循环将无法终止。

while 循环语句的执行流程如图 5.1 所示。

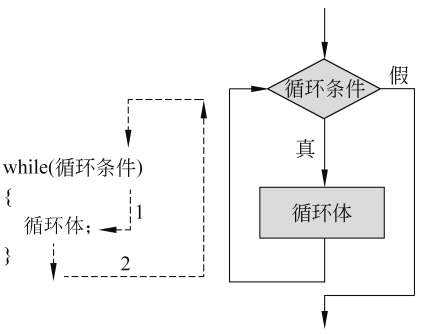

图 5.1 while 循环语句的执行流程

**! 注意：**

无法终止的循环被称为死循环或无限循环。

while 循环的使用如下所示。

```
while(sum <100){
    sum +=1;
}
```



如上述代码操作,while 语句先判断 sum 变量是否小于 100,如果小于 100,则条件为真,执行其后的语句,即 sum＋＝1,如果不小于 100,则条件为假,跳过其后面的语句。如果开始时的 sum 小于 100,则执行循环,每次循环都使得 sum 加 1,直到 sum 不满足小于 100 时,循环结束。

通过 while 循环实现计算 1 累加到 100 的结果,具体如例 5.1 所示。

**例 5.1** 计算累加和。

```
1   #include <stdio.h>
2
3   int main(int argc, const char * argv[])
4   {
5       int Sum =0;
6       int Number =1;
7
8       while(Number <=100){
9           Sum =Sum +Number;
10          Number++;
11      }
12
13      printf("The Result is %d\n", Sum);
14      return 0;
15  }
```

**输出:**

```
The Result is 5050
```

**分析:**

上述示例中,因为要计算 1～100 的累加结果,所以要定义两个变量,Sum 为计算的结果,Number 表示 1～100 的数字。

第 8～11 行:使用 while 语句判断 Number 是否小于等于 100,如果条件为真,则执行其后语句块中的内容,如果为假,则跳过语句块执行后面的内容,初始 Number 的值为 1,判断条件为真,执行循环语句。

第 9～10 行:总和 Sum 等于上一轮循环得到的总和加上本轮 Number 表示的数字,完成累加操作,Number 自加 1 后,while 再次判断新的 Number 值,当 Number 值大于 100 时,循环操作结束,将结果 Sum 输出。

### 5.1.2 do…while 语句

do…while 语句与 while 语句类似,它们之间的区别在于:while 语句是先判断循环条件的真假,再决定是否执行循环体。而 do…while 语句则先执行循环体,然后再判断循环条

件的真假，因此 do…while 语句中的循环体至少要被执行一次。do…while 语句的一般形式如下所示。

```
do{
    循环体
}while(循环条件);
```

do…while 语句的执行流程如图 5.2 所示。

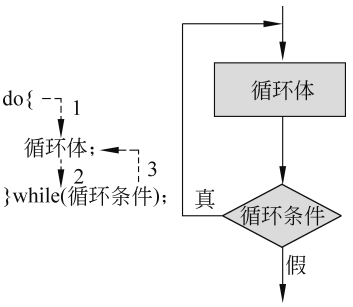


图 5.2　do…while 语句的执行流程

如图 5.2 所示，do…while 语句首先会执行一次循环体中的语句，然后判断表达式，当表达式的值为真时，返回重新执行循环体语句，执行循环，直到表达式的判断为假为止，此时循环结束。

do…while 语句的使用具体如例 5.2 所示。

**例 5.2**　do…while 语句。

```
1   #include <stdio.h>
2
3   int main(int argc, const char * argv[])
4   {
5       int Sum = 0;
6       int Num = 1;
7
8       do{
9           Sum = Sum + Num;
10          Num++;
11      }while(Num <= 100);
12
13      printf("The result is %d\n", Sum);
14
15      return 0;
16  }
```

**输出：**

```
The result is 5050
```

**分析：**

上述示例中，Num 表示 1～100 的数字，Sum 表示计算的总和。

第 8～11 行：程序先执行 do 后面的循环体语句，在循环体语句中，总和 Sum 等于上一轮循环得到的总和加上本轮 Num 表示的数字，完成累加操作。

Num 自加 1 后，while 再次判断新的 Num 值，如果 Num 值小于或等于 100，则继续执行 do 后面的循环体语句，否则跳出循环。

do…while 语句与 while 语句最大的不同是：前者是先执行再判断，后者是先判断后执行。

## 5.2 for 循环语句

### 5.2.1 for 循环的基本形式

在 C 语言中，除了使用 while 和 do…while 实现循环外，for 循环也是常见的循环结构，而且其语句更为灵活，不仅可以用于循环次数已经确定的情况，还可以用于循环次数不确定而只给出循环结束条件的情况，完全可以代替 while 语句，其语法格式如下所示。

```
for(赋初始值;循环条件;迭代语句){
    语句 1;
    ...
    语句 n;
}
```

当执行 for 循环语句时，程序首先指定赋初始值操作，接着执行循环条件，如果循环条件的值为真，程序执行循环体内的语句，如果循环条件的值为假，程序则直接跳出循环。执行完循环体内的语句后，程序会执行迭代语句，然后再执行循环条件并判断，如果为真，则继续执行循环体内的语句，如此反复，直到循环条件判断为假，退出循环。

for 循环语句的执行流程如图 5.3 所示。

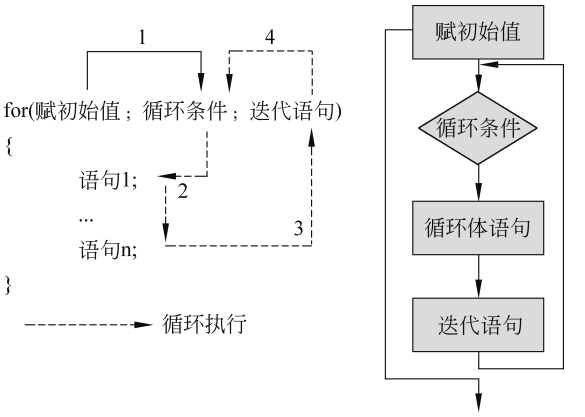


图 5.3　for 循环语句的执行流程

while 循环中限定循环的次数会比较麻烦，需要在循环体内对控制循环次数的变量进行自增或自减，而 for 循环则不需要，具体如例 5.3 所示。

**例 5.3**　for 循环语句。

```
1   #include <stdio.h>
2
3   int main(int argc, const char * argv[])
4   {
5       int Sum = 0;
6       int i;
7
8       for(i = 1; i <= 10; i++){
9           Sum += i;
10      }
11
12      printf("%d\n", Sum);
13      return 0;
14  }
```

**输出：**

```
55
```

**分析：**

第 8～10 行：先执行 i=1，再执行 i<=10，判断表达式的值为真，因此执行 for 循环的内嵌语句(循环体)，执行完成后，执行 i++操作，再次转到 i<=10，判断表达式的值，如此反复，直到执行完 i++后，判断表达式的值为假，则结束循环。

**注意：**

C90 标准规定，循环变量声明必须在循环语句之前，在 for 小括号中声明和定义循环变量是语法错误的，如 for(int i=0;i<10;i++)。由于大部分 C 编译器不能很好地支持 C99 标准，为了程序的通用，建议遵循 C90 标准。

由此可知，应写为如下格式。

```
int i;                          //循环变量 i 在 for 循环之前声明
for(i=0;i<10;i++);              //语法正确
```

### 5.2.2 多循环变量的 for 循环

for 循环也可以对多个循环变量进行赋值和增减运算。每个赋值表达式和增减表达式之间使用逗号分隔。接下来通过示例演示，具体如例 5.4 所示。

**例 5.4** 多循环变量。

```
1   #include <stdio.h>
2
3   int main(int argc, const char * argv[])
4   {
5       int i,j,k;
6
7       for(i=5,j=5,k=5; i>0; i--,j--,k--){
```

```
8           printf("i=%d j=%d k=%d\n",i,j,k);
9       }
10
11      return 0;
12  }
```

**输出：**

```
i=5 j=5 k=5
i=4 j=4 k=4
i=3 j=3 k=3
i=2 j=2 k=2
i=1 j=1 k=1
```

**分析：**

第 7～9 行：程序先执行 i=5,j=5,k=5,再执行 i>0 判断,判断条件为真,则执行输出,i、j、k 的值为 5。

执行 i－－,j－－,k－－,再次执行 i>0 判断,判断条件为真,继续输出 i、j、k 的值为 4,如此反复,直到判断 i>0 为假时,跳出循环。

各个变量的赋值表达式与自减表达式使用逗号隔开。

### 5.2.3 for 循环的变体

在 for 循环中,每个表达式都可以省略,省略不同的表达式会出现不同的效果,省略赋值表达式,需要在 for 语句之前给变量赋初值;省略判断表达式,即循环条件表达式始终为真,循环会无终止地进行;省略增减表达式,为了保证程序的正常结束,需要在循环体中添加操作表达式。

**1. 省略赋值表达式**

for 循环语句中的第一个表达式的功能是对循环变量赋初值。因此,如果省略了赋值表达式,就会跳过这一步操作,则应在 for 语句之前为循环变量赋值。省略赋值表达式的格式如下所示。

```
for( ;Num<10;Num++)
```

接下来通过示例展示省略赋值表达式的 for 语句的使用,如例 5.5 所示。

**例 5.5** 省略赋值表达式。

```
1   #include <stdio.h>
2
3   int main(int argc, const char * argv[])
4   {
5       int Num =1;
6       int Sum =0;
7
8       for( ; Num <=100; Num++){
```

```
9           Sum +=Num;
10      }
11
12      printf("The result is %d\n", Sum);
13      return 0;
14  }
```

**输出：**

```
The result is 5050
```

**分析：**

上述示例中，程序的功能为计算 1～100 的累加和。

第 8～10 行：for 循环语句中的赋值表达式被省略，因此循环变量需要在执行 for 语句之前完成赋值(对变量 Num 进行赋值)。

**2. 省略判断表达式**

省略判断表达式，即不判断循环条件，则循环无终止地进行下去，也可以认为判断表达式始终为真。省略判断表达式的格式如下所示。

```
for(Num=1; ;Num++)
```

上述表达式省略第 2 个表达式，其操作等同于如下情况。

```
Num=1;
while(1){
    Num++;
}
```

**3. 省略增减表达式**

省略 for 语句中的最后一个表达式可能会导致程序无法正常结束。因此，在程序设计时需要在循环体语句中补充该表达式的操作，具体如例 5.6 所示。

**例 5.6** 省略增减表达式。

```
1   #include <stdio.h>
2
3   int main(int argc, const char * argv[])
4   {
5       int Num, Sum =0;
6
7       for(Num =1; Num <=100; ){
8           Sum +=Num;
9           Num++;
10      }
11
12      printf("The result is %d\n", Sum);
13      return 0;
14  }
```

输出：

```
The result is 5050
```

分析：

上述示例中，程序的功能为计算 1～100 的累加和。

第 7～10 行：for 循环语句中的增减表达式被省略，因此在循环体执行语句中需要补充该表达式的操作(即 Num＋＋)，避免循环无限执行。

综上所述，for 循环语句中省略任一表达式都是可行的，也可以同时省略两个及以上的表达式，当表达式被全部省略时，表示无终止地执行循环，如下所示。

```
for( ; ; ) <==>while(1)
```

### 5.2.4 for 循环的嵌套

很多情况下，单层循环不能解决实际的问题，此时则需要使用多层循环嵌套。嵌套指的是在一个循环中包含另一个循环，外层的循环每执行一次，内层的循环需要全部完整地执行一次。

无论是 for 循环、while 循环还是 do…while 循环，它们的内部都可以嵌套新的循环。最常见的例子就是多重 for 循环。接下来通过示例展示 for 循环嵌套的情况，具体如例 5.7 所示。

**例 5.7** for 循环的嵌套。

```
1   #include <stdio.h>
2
3   int main(int argc, const char * argv[])
4   {
5       int i, j;
6
7       for(i =0; i <5; i++){
8           for(j =0; j <5-i; j++){
9               printf("* ");
10          }
11          printf("\n"); /* 输出换行 */
12      }
13      return 0;
14  }
```

输出：

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

**分析：**

第 7～12 行：在 for 循环语句中嵌套另一个 for 循环语句，外层的 for 循环语句执行一次，内层的 for 循环语句完整执行一次（完成该 for 语句的全部循环操作），即执行循环体语句 5 次。

## 5.3 转移语句

### 5.3.1 break 语句

break 语句用来终止并跳出循环，继续执行后面的代码，其一般的形式如下所示。

```
break;
```

break 语句不能用于循环语句和 switch 语句之外的任何其他语句中。

**注意：**

上述 break 语句与 switch…case 分支结构中的 break 语句的作用是不同的。

break 语句的使用如例 5.8 所示。

**例 5.8** break 语句。

```
1   #include <stdio.h>
2
3   int main(int argc, const char * argv[])
4   {
5       int i;
6
7       for(i =0; i <10; i++){
8           if(i ==5) /* 当 i 等于 5 时,跳过本轮循环 */
9           break;
10          printf("i =%d\n", i);
11      }
12
13      return 0;
14  }
```

**输出：**

```
i =0
i =1
i =2
i =3
i =4
```

**分析：**

第 7～11 行：for 循环执行的条件为变量 i 的值小于 10，变量 i 的初始值为 0，执行判断后满足条件，进入循环，其值变为 1，执行 if 判断，不满足条件，不执行 break 语句，最后通过

printf()函数输出 i 的值为 1。

在 for 循环中,执行 i 自加操作,继续执行下一次循环,变量 i 的值变为 2,执行 if 判断,仍然不满足条件,输出 i 的值为 2。以此类推,直到 i 增加到 5 时,执行 break 语句,跳出整个循环,程序结束,不输出 i 的值。因此,i 输出的最大值为 4。

### 5.3.2 continue 语句

有时在循环语句中,当满足某个条件时,希望结束本次循环,即跳过本次循环中尚未执行的部分,继续执行下一次循环操作,而非直接跳出全部循环。在 C 语言中使用 continue 语句实现这一需求。continue 语句的一般形式如下所示。

```
continue;
```

**注意:**

continue 语句与 break 语句的区别是:continue 语句只结束本次循环,而不是终止整个循环的执行,break 语句则是结束整个循环过程,不再判断执行循环的条件是否成立。

continue 语句的使用如例 5.9 所示。

**例 5.9** continue 语句。

```
1   #include <stdio.h>
2
3   int main(int argc, const char * argv[])
4   {
5       int i;
6
7       for(i =0; i <10; i++){
8           if(i ==5) /* 当 i 等于 5 时,跳过本轮循环 */
9           continue;
10          printf("i =%d\n", i);
11      }
12
13      return 0;
14  }
```

**输出:**

```
i =0
i =1
i =2
i =3
i =4
i =6
i =7
i =8
i =9
```