可以看到,当输入 i 的值小于或等于 10 时,二者得到的结果相同;而当 i>10 时,二者结果就不同了。这是因为此时对 while 循环来说,一次也不执行循环体(表达式"i<=10"的值为假),而对 do…while 循环语句来说则至少要执行一次循环体。可以得到结论:当 while 后面的表达式的第 1 次的值为"真"时,两种循环得到的结果相同;否则,二者结果不相同(指二者具有相同的循环体的情况)。

5.4 用 for 语句实现循环

除了可以用 while 语句和 do…while 语句实现循环外,C 语言还提供了 for 语句实现循环,而且 for 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,它完全可以代替 while 语句。

例如:

```
for (i=1;i<=100;i++)                    //控制循环次数,i 由 1 变到 100,共循环 100 次
  printf("%d",i);                       //执行循环体,输出 i 的当前值
```

它的执行过程见图 5.7。

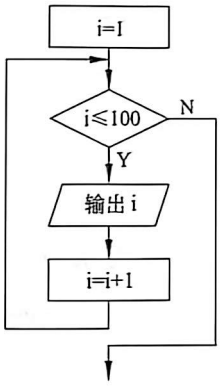

图 5.7

它的作用是:输出 1～100,共 100 个整数。

for 语句的一般形式为

for(表达式 1;表达式 2;表达式 3)

语句

括号中 3 个表达式的主要作用是:

表达式 1:设置初始条件,只执行一次。可以为零个、一个或多个变量设置初值(如 i=1)。

表达式 2:是循环条件表达式,用来判定是否继续循环。在每次执行循环体前先执行此表达式,决定是否继续执行循环。

表达式 3:作为循环的调整,例如使循环变量增值,它是在执行完循环体后才进行的。

最常用的 for 语句形式是:

for(循环变量赋初值;循环条件;循环变量增值)

语句

例如:

```
for(i=1;i<=100;i++)
  sum=sum+i;
```

其中的"i=1"是给循环变量 i 设置初值为 1,"i<=100"是指定循环条件:当循环变量 i 的值小于或等于 100 时,循环继续执行。"i++"的作用是使循环变量 i 的值不断变化,以便最终满足终止循环的条件,使循环结束。也就是:循环变量 i 的初值为 1,循环变量增量为 1,循环变量终值为 100,每执行一次循环,i 的值加 1,直到 i 的值大于 100,就不再执行了。

for 语句的执行过程如下:

(1) 求解表达式 1。本例中把整数 1 赋给变量 i。

(2) 求解表达式 2,若此条件表达式的值为真(非 0),则执行 for 语句中的循环体,然后

执行第(3)步。若为假(0)，则结束循环，转到第(5)步。

上例中，循环条件表达式“i<=100”是一个关系表达式，当 i=1 时，表达式 i<=100 的值为真(非 0)，故执行循环体中的语句，即 printf 语句，输出 i 的当前值 1。然后执行第(3)步。

(3) 求解表达式 3。在本例中，执行 i++，使 i 的值加 1，i 的值变成 2。

(4) 转回步骤(2)继续执行。

由于此时 i=2，表达式 i<=100 的值为真，再次执行循环体中的语句，printf 语句输出 i 的当前值 2。然后再执行步骤(3)。如此反复，直到 i 变到 101，此时表达式 i<=100 的值为假，不再执行循环体，而转到步骤(5)。

可以用图 5.8 来表示 for 语句的执行过程。

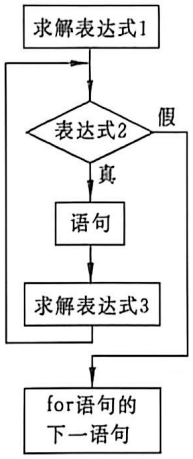

图　5.8

注意：在执行完循环体后，循环变量的值“超过”循环终值，循环结束。例如，在本例中，在执行完循环体后循环变量 i 的值为 101，大于循环终值 100。如果循环变量的增值为负值，如：for(i=100;i>=1;i--)，执行完循环体后循环变量 i 的值为 0，小于循环终值 1。其规律为：循环变量沿着变化的方向“超过”循环终值，循环就结束了。

(5) 循环结束，执行 for 语句下面的一个语句。

上面看到的 for 语句

```
for(i=1;i<=100;i++)
  sum=sum+i;
```

其执行过程与图 5.3 完全一样。它相当于以下语句：

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

显然，用 for 语句简单、方便。

说明：

(1) for 语句的一般形式如下：

for(表达式 1;表达式 2;表达式 3) 语句

可以改写为 while 循环的形式：

```
表达式 1;
while 表达式 2
  {
    语句
    表达式 3
  }
```

二者无条件等价。

(2)“表达式1”可以省略,即不设置初值,但表达式1后的分号不能省略。例如:

```
for(;i<=100;i++) sum=sum+i;                    //for语句中没有表达式1
```

应当注意:由于省略了表达式1,没有对循环变量赋初值,因此,为了能正常执行循环,应在for语句之前给循环变量赋以初值。即

```
i=1;                                           //对循环变量i赋初值
for(;i<=100;i++) sum=sum+i;                    //for语句中没有表达式1
```

执行for语句时,跳过图5.8中的“求解表达式1”这一步。由于在for语句前加了“i=1;”,因此其作用仍然不变。

(3)表达式2也可以省略,即不用表达式2来作为循环条件表达式,不设置和检查循环的条件。如:

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

此时循环无终止地进行下去,也就是认为表达式2始终为真,见图5.9。

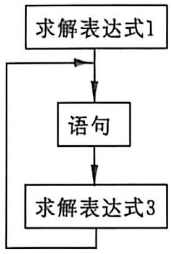

图 5.9

它相当于

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

循环无终止地进行,i的值不断加大,sum的值也不断累加。

(4)表达式3也可以省略,但此时程序设计者应另外设法保证循环能正常结束。例如:

```
for(i=1;i<=100;)                               //没有表达式3
  {
    sum=sum+i;
    i++;                                       //这时可以在循环体中使循环变量增值
  }
```

在上面的for语句中只有表达式1和表达式2,而没有表达式3。i++的操作不放在表达式3的位置,而作为循环体的一部分,效果是一样的,都能使循环正常结束。如果在循环体中无此“i++;”语句,则循环体无止境地执行下去。

(5)如果表达式1和表达式3都没有,只有表达式2,即只给循环条件,情况会怎样?如:

```
for(;i<=100;)                                  //没有表达式1和表达式3,只有表达式2
  {
    sum=sum+i;
    i++;                                       //在循环体中使循环变量增值
  }
```

当然,应当在for语句前给循环变量赋初值,否则循环无法正常执行。即:

```
i=1;                                           //给循环变量赋初值
for(;i<=100;)                                  //没有表达式1和表达式3,只有表达式2
```

```
{
  sum=sum+i;
  i++;                          //在循环体中使循环变量增值
}
```

相当于

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

可见 for 语句比 while 语句功能强，除了可以给出循环条件外，还可以赋初值，使循环变量自动增值等。

(6) 甚至可以将 3 个表达式都可省略，例如：

```
for(; ;) printf("%d\n",i);
```

相当于

```
while(1) printf("%d\n",i);
```

即不设初值，不判断条件(认为表达式 2 为真值)，循环变量也不增值，无终止地执行循环体语句，显然这是没有实用价值的。

(7) 表达式 1 可以是设置循环变量初值的赋值表达式，也可以是与循环变量无关的其他表达式。例如：

```
for (sum=0;i<=100;i++) sum=sum+i;
```

表达式 3 也可以是与循环控制无关的任意表达式。但不论怎样写 for 语句，都必须使循环能正常执行。

(8) 表达式 1 和表达式 3 可以是一个简单的表达式，也可以是逗号表达式，即包含一个以上的简单表达式，中间用逗号间隔。如：

```
for(sum=0,i=1;i<=100;i++) sum=sum+i;
```

或

```
for(i=0,j=100;i<=j;i++,j--) k=i+j;
```

表达式 1 和表达式 3 都是逗号表达式，各包含两个赋值表达式，即同时设两个初值(i=0,j=100)，使两个变量增值(i++,j--)，执行情况见图 5.10。

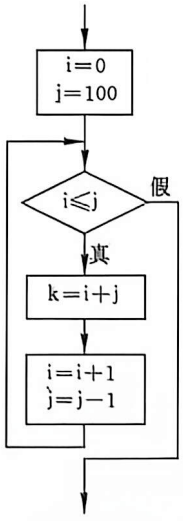

图　5.10

在逗号表达式内按自左至右顺序求解，整个逗号表达式的值为最右边的表达式的值。例如：

```
for(i=1;i<=100;i++,i++) sum=sum+i;
```

相当于

```
for(i=1;i<=100;i=i+2) sum=sum+i;
```

(9) 表达式2一般是关系表达式(如i<=100)或逻辑表达式(如a<b &&x<y),但也可以是数值表达式或字符表达式,只要其值为非零,就执行循环体。分析下面两个例子:

```
① for(i=0;(c=getchar())!='\n';i+=c);
```

在表达式2中先从终端接收一个字符赋给c,然后判断此赋值表达式的值是否不等于'\n'(换行符),如果不等于'\n',就执行循环体。此for语句的执行过程见图5.11,它的作用是不断输入字符,将它们的ASCII码相加,直到输入一个"换行"符为止。

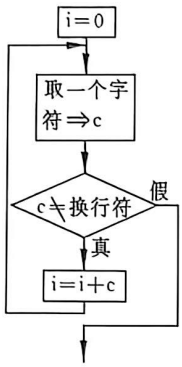

图 5.11

注意:此for语句的循环体为空语句,把本来要在循环体内处理的内容放在表达式3中,作用是一样的。可见for语句功能强,可以在表达式中完成本来应在循环体内完成的操作。

```
② for( ;(c=getchar())!='\n';)
      printf("%c",c);
```

for语句中只有表达式2,而无表达式1和表达式3。其作用是每读入一个字符后立即输出该字符,直到输入一个"换行"为止。

运行情况:

```
Computer↙          (输入)
Computer           (输出)
```

请注意,从终端键盘向计算机输入时,是在按Enter键以后才将一批数据一起送到内存缓冲区中去的。因此不是从终端输入一个字符马上输出一个字符,而是在按Enter键后数据才送入内存缓冲区,然后每次从缓冲区读一个字符,再输出该字符。

(10) C 99允许在for语句的"表达式1"中定义变量并赋初值,如:

```
for(int i=1;i<=100;i++)          //定义循环变量i,同时赋初值1
sum=sum+i;
```

显然,这可以使程序简练,灵活方便。但应注意:所定义的变量的有效范围只限于for循环中,在循环外不能使用此变量。

从上面介绍可以知道,C语言的for语句使用十分灵活,变化多端,可以把循环体和一些与循环控制无关的操作也作为表达式1或表达式3出现,这样程序可以短小简洁。但应注意:过分地利用这一特点会使for语句显得杂乱,可读性降低,最好不要把与循环控制无关的内容放到for语句中。

注意:对以上"说明"中介绍的内容,读者应当了解,以便能看懂别人写的程序,并且在熟练掌握C以后能写出简洁高效的程序,但是,建议初学者开始时不要过于追求技巧而写出别人不易看懂的程序,应当尽量写出清晰易读的程序。

5.5 循环的嵌套

一个循环体内又包含另一个完整的循环结构,称为循环的嵌套。内嵌的循环中还可以嵌套循环,这就是多层循环。各种语言中关于循环的嵌套的概念都是一样的。