第 5 章

循环结构程序设计

所谓循环(Loop),就是重复地执行同一段代码,例如要计算 1+2+3+…+99+100 的值,就要重复进行 99 次加法运算。

在日常生活中,经常会遇到一些需要反复处理和解决的问题。无论是家务琐事、工作中的任务,还是在学习过程中遇到的难题,重复性的工作总是不可避免的。同样,在编写和运行程序时,也会遇到许多需要重复处理的问题。这些重复性问题可能涉及数据的批量处理、循环执行某些操作,或者是在特定条件下反复进行某些计算。无论是现实生活中的重复任务,还是程序中的循环处理,掌握有效的处理方法和技巧都是非常重要的,这有助于提高效率,减少错误,确保任务能够顺利完成。

5.1 while 语句

while 语句是一种用于实现循环操作的语句,其一般形式如下:

```
while(<条件表达式>)
{
    <循环语句>
    <循环变量表达式>
}
```

其中,花括号所包含的<循环语句>与<循环变量表达式>共同构成了循环体语句。该语句的执行遵循以下两个步骤。

(1) 首先,计算<条件表达式>的值,若结果为“假”,则终止循环,继续执行循环体之后的语句;若结果为“真”,则执行循环体内的语句。

(2) 随后,重复步骤(1)。

while 语句的流程图如图 5.1 所示,其中虚线框内为循环体语句。

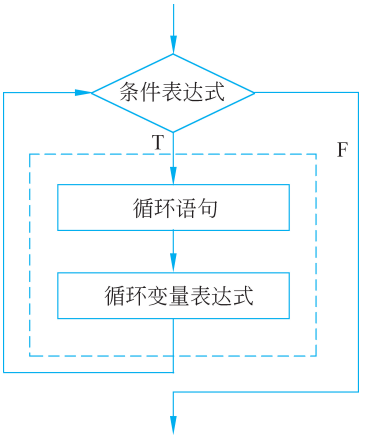

图 5.1 while 语句的流程图

说明:

(1) <循环变量表达式>用于更新循环变量的值。

(2) 通常情况下,<循环语句>应由一对花括号包围,形成复合语句,且该复合语句至少应包含两条语句。

(3) 若循环体只包含一条语句,花括号可以省略。

(4) 若循环体语句中缺少<循环变量表达式>,仅包含<循环语句>,则可能导致程序陷入无限循环,从而引发错误。

（5）鉴于 while 循环是先对<条件表达式>进行判断，再决定是否执行<循环语句>，当<条件表达式>的值为假时，存在循环语句一次也不被执行的可能性。

例 5.1 求 1＋2＋3＋…＋100。

分析：

（1）这是一个累加的问题，需要先后将 100 个数相加。如果用变量 i 表示要加的数，i 的初始值为 1；用 sum 表示累加的和，初始值为 0。

（2）累加 i：sum＋＝i。

（3）i 自增 1，i＋＋。

（4）重复步骤(2)和步骤(3)，直到 i 超过 100 时结束。

（5）i 超过 100 为中止累加的条件，进行累加的条件与之相反，即 i≤100，这就是进行循环累加的条件。

根据分析所得流程图如图 5.2 所示，其中虚线框内为 while 循环结构。程序如下：

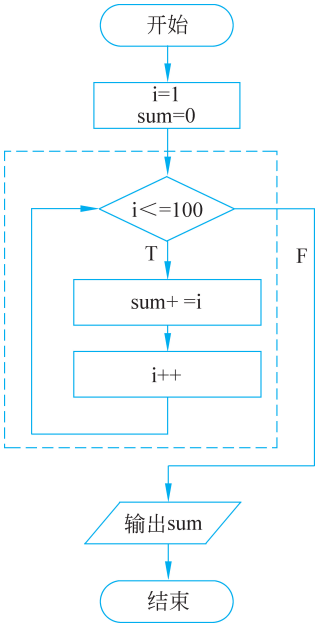

图 5.2 例 5.1 的流程图

```
#include <stdio.h>

int main()
{
    int i,sum;                      //定义变量
    i=1;                            //循环变量赋初值
    sum=0;                          //赋初值
    while(i<=100)                   //满足条件 i<=100 时运行循环体语句
    {
        sum+=i;                     //累加
        i++;                        //自增 1
    }
    printf("sum=%d\n",sum);         //输出累加和
    return 0;
}
```

例 5.2　从键盘上输入一些字符，遇到回车符时结束，统计这些字符中的英文字母个数。

分析：

（1）用变量 ch 表示从键盘上输入的每一个字符，当为回车符时结束输入，循环的条件为 ch !='\n'。

（2）英文字母可能是小写字母，也可能是大写字母，判断是否为英文字母的条件为：

```
(ch>='a' && ch<='z') || (ch>='A' && ch<='Z')
```

（3）使用变量 count 用来统计，其初始值为 0，当 ch 为英文字母时，count 加 1。

流程图如图 5.3 所示，程序如下：

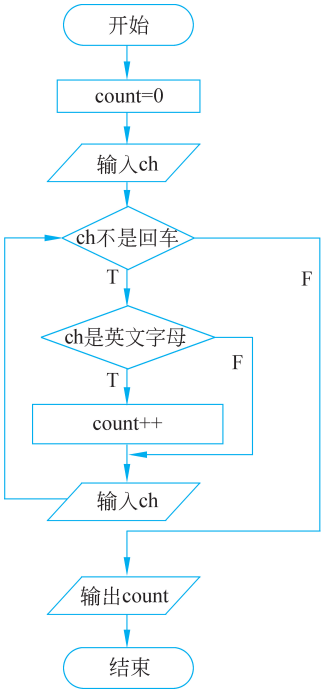

图 5.3　例 5.2 的流程图

```
#include <stdio.h>
int main()
{
    int count=0;
    char ch;
    ch=getchar();
    while(ch!='\n')
    {
        if((ch>='a'&&ch<='z')||(ch>='A'&&ch<='Z'))
        {
            count++;
        }
        ch=getchar();
    }
    printf("count=%d\n",count);
    return 0;
}
```

在程序中，也可把 ch＝getchar()语句置于 while 循环的条件表达式之内；经过修改，

while 循环体内仅包含一条 if 语句，且该 if 语句也只有一条语句。因此，可以省略 while 循环体与 if 语句的花括号，程序如下：

```
#include <stdio.h>
int main()
{
    int count=0;
    char ch;
    while((ch=getchar())!='\n')
        if((ch>='a' && ch<='z') || (ch>='A' && ch<='Z'))
            count++;
    printf("count=%d\n",count);
    return 0;
}
```

5.2 do-while 语句

与 while 语句相似，do-while 语句同样能够实现循环操作。然而，其执行流程与 while 语句存在差异：do-while 语句首先执行循环体，随后才对条件进行判断，若条件满足，则再次执行循环体。

do-while 语句的一般形式为：

```
do
{
    <循环语句>
}while(<条件表达式>);
```

该语句的执行包含两个阶段。

(1) 执行<循环语句>。若<循环语句>只有一条语句，可不加花括号。

(2) 计算<条件表达式>，若表达式结果为“真”，则返回执行(1)；若结果为“假”，则终止循环语句的执行，并继续执行后续语句。

do-while 语句的流程图如图 5.4 所示。

例 5.3 用 do-while 语句求 1+2+3+…+100。

流程图如图 5.5 所示，程序如下：

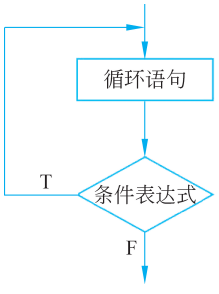

图 5.4 do-while 语句的流程图

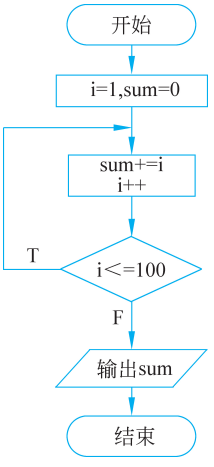

图 5.5 例 5.3 的流程图

```
#include <stdio.h>
int main()
{
    int i=1,sum=0;
    do
    {
        sum+=i;
        i++;
    }while(i<=100);
    printf("sum=%d\n",sum);
    return 0;
}
```

5.3　for 语句

在C语言中，for语句是最为常用的循环结构之一，其循环次数由循环变量控制，故此循环也称为**计数循环**。计数循环由3个关键部分构成：首先，对循环控制变量进行初始化；其次，对循环条件进行测试；最后，更新循环控制变量的值。

for语句的一般形式为：

```
for(<初始表达式>; <条件表达式>; <循环变量表达式>)
{
    <循环语句>
}
```

for语句的执行有以下4个步骤。

(1) 计算<初始表达式>的数值。该表达式通常用于初始化循环控制变量。

(2) 判断<条件表达式>的值。该表达式作为循环条件，若其值为“假”，则终止循环并继续执行循环体之后的语句；若其值为“真”，则执行<循环语句>。若<循环语句>仅包含一条语句，则可以省略花括号。

(3) 计算<循环变量表达式>的值。该表达式负责更新循环控制变量的数值。

(4) 转到步骤(2)。

例5.4　用for语句求1＋2＋3＋…＋100。

流程图如图5.6所示，程序如下：

```
#include <stdio.h>

int main()
{
    int i,sum;
    for(i=1,sum=0; i<=100; i++)
    {
        sum+=i;
    }
    printf("sum=%d\n",sum);
    return 0;
}
```

可见，for语句与while语句的流程图结构相同。因此，for语句能够替代while语句。使用for语句解决例5.2的程序如下：

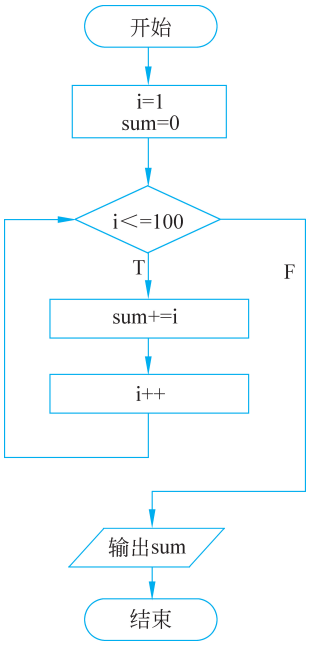

图 5.6　例 5.4 的流程图

```
#include <stdio.h>
int main()
{
    int count;
    char ch;
    for(count=0; (ch=getchar())!='\n'; )
        if((ch>='a' && ch<='z' ) || (ch>='A' && ch<='Z' ) )
            count++;
    printf("count=%d\n",count);
    return 0;
}
```

在上述程序中，<循环变量表达式>部分被省略。在 for 语句的构成中，<初始表达式>、<条件表达式>、<循环变量表达式>以及<循环语句>均可选择性地省略，但 3 个表达式之间的分号不可省略。

例如，for 语句求 1＋2＋3＋…＋100 的关键语句为：

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

可改写为：

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

（1）省略<初始表达式>，可把这个表达式的内容放在 for 语句之前。

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

（2）省略<循环变量表达式>，可把这个表达式的内容放在<循环语句>里面。

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

（3）省略<循环语句>，可以把这个表达式的内容放在<循环变量表达式>里面，语句之间用逗号间隔。此时，<循环语句>为空，如果不加花括号，必须以分号结尾。以下两种代码是等效的。

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

（4）省略<条件表达式>，可以把这个表达式的内容放在＜循环语句＞里面，加上中止循环的语句即可。

```
i=1,sum=0;
for( ;  ; )
{
    if(i>100)
    {
        break;
    }
    sum+=i;
    i++;
}
```

例 5.5　从键盘上输入两个正整数 m 和 n(m＜＝n)，计算 m＋(m＋1)＋(m＋2)＋…＋n。使用 3 种语句的解决的程序如下：

while 语句

```
#include <stdio.h>
int main()
{
    int m,n,sum=0,s;
    scanf("%d,%d",&m,&n);
    s=m;
    while(s<=n)
    {
        sum+=s;     s++;
    }
    printf("%d\n",sum);
    return 0;
}
```

do-while 语句

```
#include <stdio.h>
int main()
{
    int m,n,sum=0,s;
    scanf("%d,%d",&m,&n);
    s=m;
    do
    {
        sum+=s;     s++;
    }while(s<=n);
    printf("%d\n",sum);
    return 0;
}
```

for 语句

```
#include <stdio.h>
int main()
{
    int m,n,sum=0,s;
    scanf("%d,%d",&m,&n);
    for(s=m; s<=n; s++)
    {
        sum+=s;
    }
    printf("%d\n",sum);
    return 0;
}
```

5.4　改变循环执行的状态

在特定情境下，可能需要提前终止正在执行的循环。以慈善募捐为例，一旦募捐金额达到既定目标，即可停止。

5.4.1　break 语句

使用 break 语句可以使得程序流程跳出 switch 结构，并继续执行紧随其后的语句。此外，break 语句也可用于从循环体中退出，即**提前终止循环**，然后继续执行循环之后的代码。

例 5.6　在某学院的 600 个学生中开展了一场慈善募捐活动，目标是筹集 2 万元善款。

当募捐总额达到这一目标时，募捐活动随即结束。请统计并报告参与捐款的学生人数以及平均每人捐款的金额。

分析：可以通过循环结构来处理捐款事宜。由于事先无法确定循环的具体次数，可以将其设定为一个最大值600，即假设有最多600人参与捐款。在循环体内，将累加每次的捐款金额，并通过if语句判断累计总额是否已经达到了2万元。一旦达到这个目标，循环将停止执行，不再继续累加，并且将计算出平均每人捐款的金额。涉及的变量有：每次捐款金额amount、捐款总金额total、人均捐款金额average、循环计数器i。

流程图如图5.7所示，程序如下：

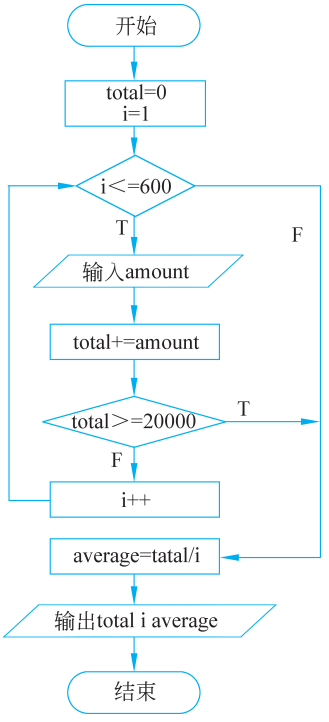

图5.7　例5.6的流程图

```
#include <stdio.h>
int main()
{
    int amount,total,i;
    float average;
    total=0;
    for(i=1; i<=600; i++)
    {
        printf("请输入第%d人的捐款:",i);
        scanf("%d",&amount);
        total+=amount;
        if(total>=20000)
        {    break;    }
    }
    average=total*1.0/i;
    printf("捐款总额:%d,捐款人数:%d,平均每人捐款:%.2f\n",total,i,average);
    return 0;
}
```

5.4.2　continue 语句

continue 语句具备调整循环执行流程之功能，能够使程序跳过当前循环的余下部分，直接过渡至下一轮循环。

例 5.7　编写程序，输出 200～600 中不能被 3 整除的数字，输出时每个数字的最小宽度为 5。

流程图如图 5.8 所示，程序如下：

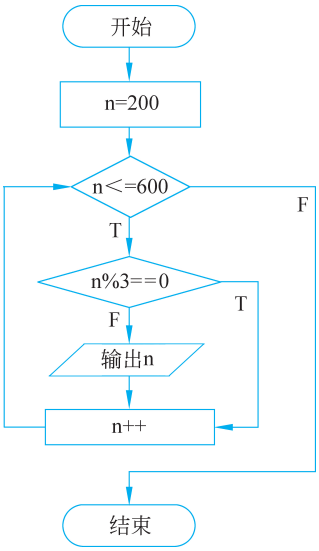

图 5.8　例 5.7 的流程图

```
#include <stdio.h>
int main()
{
    int n;
    for(n=200; n<=600; n++)
    {
        if(n%3==0)
        {
            continue;
        }
        printf("%5d",n);
    }
    return 0;
}
```

从流程图中可以清晰地观察到，continue 语句在其中仅作为一条改变循环路径的流程线存在，当 n 能够被 3 整除时，它使得程序跳过循环体内 continue 后面的“输出 n”的操作，即“printf("%5d",n);”语句，而直接转向执行 n 的自增操作（n＋＋，即 for 循环语句中的<循环变量表达式>）。

5.4.3　break 语句与 continue 语句的区别

continue 语句的作用是终止当前循环的剩余部分，并立即开始下一次循环迭代。它并

不中断整个循环结构的执行。相对地，break 语句用于完全终止循环，不再进行任何后续的循环操作，也不再判断循环条件是否满足。如果有以下两个循环结构及其流程图，程序(1)的流程如图 5.9 所示，而程序(2)的流程如图 5.10 所示。

```
(1) while(表达式 1)
{
    语句 1
    if(表达式 2)break;
    语句 2
}
(2) while(表达式 1)
{
    语句 1
    if(表达式 2)continue;
    语句 2
}
```

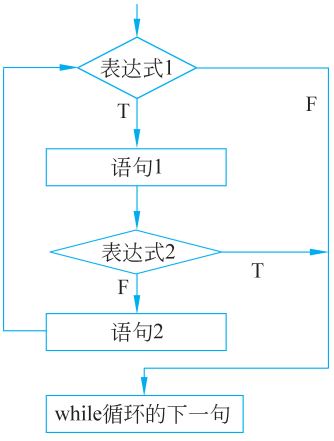

图 5.9 break 语句的流程图

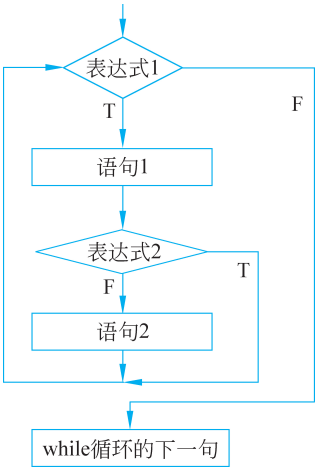

图 5.10 continue 语句的流程图

5.5 循环结构的嵌套

在编程领域，当一个循环结构(A)内嵌包含另一个完整的循环(B)，这种结构称为循环嵌套。其中，A 循环被称为**外层循环或第一层循环**，而 B 循环则称为**内层循环或第二层循环**。

嵌套循环可以进一步包含其他循环，形成多层嵌套的结构。从外向内的循环依次称为第一层、第二层、第三层循环等。第一层循环通常指的是最外层的循环结构，它控制着整个循环过程的起始和终止，而内层循环负责执行最精细的操作。

可以利用多层循环来解决各种复杂的问题，无论是数据处理、算法实现还是其他需要多重迭代的场景。每一层循环都扮演着重要的角色，确保程序能够按照预定的逻辑顺序执行，最终达到预期的结果。

3 种循环(while 循环、do-while 循环和 for 循环)可以相互嵌套。例如，以下几种都是合法的嵌套形式：