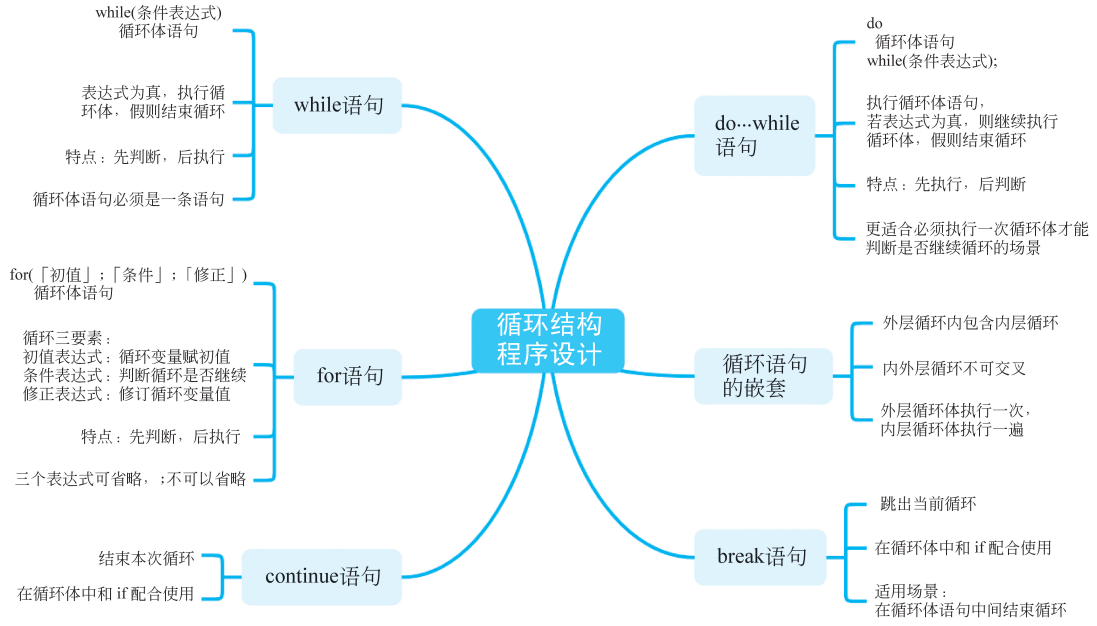
第5章 循环结构程序设计



在编程界有一句名言：懒惰是程序员美德的第一要素。这里所谓的“懒惰”，并不是不爱劳动不学习、不思进取，而是指写尽量少的代码达到同样甚至更好的目标。因此，懒惰的程序员更善于归纳总结，寻找规律，利用“循环”解决问题。

例如：某学生的开发作品参加某计算机设计大赛，共 5 位评委，请计算该作品的最终平均得分。勤快的程序员可能这样设计：

扫一扫

```
#include <stdio.h>
int main()
{
```

```
    float sum=0,score;          //sum 存放累加和,score 存放作品得分
    scanf("%f",&score);         //第 1 个评委打分
    sum=sum+score;              //计算累加和
    scanf("%f",&score);         //第 2 个评委打分
    sum=sum+score;              //计算累加和
    scanf("%f",&score);         //第 3 个评委打分
    sum=sum+score;              //计算累加和
    scanf("%f",&score);         //第 4 个评委打分
    sum=sum+score;              //计算累加和
    scanf("%f",&score);         //第 5 个评委打分
    sum=sum+score;              //计算累加和
    printf("sum=%f\n",sum/5);
}
```

这种编程思路的确也是一种解决方案,但其中的两条语句:

```
scanf("%f",&scroe);
sum=sum+score;
```

因为有 5 位评委而重复了 5 次。如果是大众评委,人数骤增到 1000 人时,程序岂不长到难以忍受的地步?并且随着代码的增长,程序出错概率增高,排错难度、维护量增大,这是懒惰的程序员万万不能容忍的。所以,他们会利用循环结构写出最简洁的代码,完成计算总分的任务。

循环结构是指在满足一定条件的情况下,重复执行某程序段的结构。其中,需要满足的一定条件称为循环条件,重复执行的程序段称为循环体。C 语言提供了 while 循环、do-while 循环和 for 循环三种,同样是循环思想,但适用场景不同的循环语句,又提供了与循环语句巧妙配合的 break 和 continue 语句。

另外,用 goto(无条件跳转)语句和 if 语句结合也可以构成循环结构,但会破坏结构化设计风格,导致程序可读性差。并且,上述三种循环结构足以解决各种实际问题,因此建议不要也不必使用 goto 语句。

5.1 while 语句

while 语句是“先判断,后执行”型循环语句,其一般应用形式如下:

```
while(条件表达式)
    循环体语句
```

其执行流程如图 5-1 所示:

① 判断条件表达式的值,如果值为真(非 0),则转到第②步;否则转到第③步。

② 执行“循环体语句”,转到第①步继续执行。

③ 终止循环,即执行循环体后面的语句。

while 语句的特点是“先判断,后执行”,如果第一次判断条件表达式的值即为假,则循

环体语句一次也不会执行。

【例 5-1】 编程实现，某同学第 1 天学习一个单词，以后每天都比前一天多学习一个单词，请问 30 天后该同学共学习了多少单词？

程序分析：第 1 天学习一个单词，则第 n 天学习 n 个单词，因此天数和单词数正好相等，因此可以省去一个表示天数的变量，用单词数控制循环，算法流程图如图 5-2 所示。

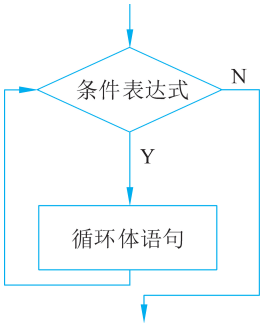


图 5-1 while 语句执行流程图

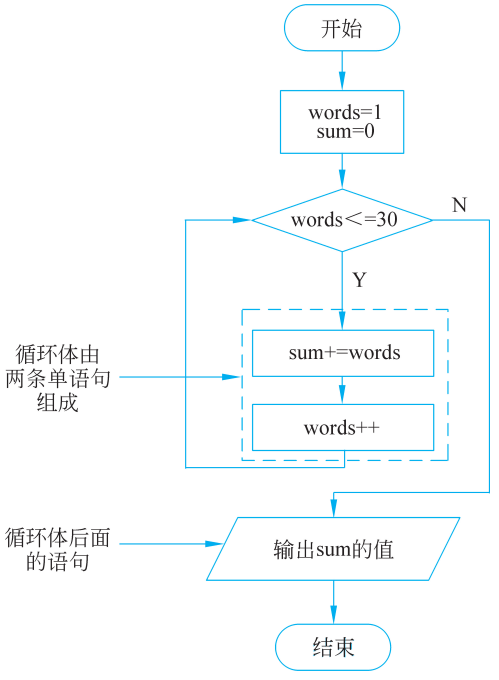


图 5-2 例 5-1 算法流程图

```
#include <stdio.h>
int main()
{
    int words=1,sum=0;          //words 存放单词个数,sum 存放单词累加和
    while(words<=30)            //循环条件"words<=30"必须用()括起来,后面没有";"
    {
        sum+=words;             //计算累加和
        words++;                //单词数比前一天多一个
    }
    printf("sum=%d\n",sum);     //输出单词总数,循环结束后执行此语句
}
```

循环体有两条单语句，要用{}括起来形成一条复合语句

程序执行结果为：

```
sum=465
```

while 语句注意事项如下。

(1) 循环变量的初值、循环条件和循环变量的修正称为循环的三要素，是循环控制的关键。参与循环条件的变量称为循环变量，如例 5-1 中的 words。“words＝1”是为循环变量

赋初值为1，循环条件“words＜＝30”控制循环何时结束，“words＋＋”修正循环变量的值，使循环条件逐步靠近假值。三要素中的任何一个因素发生改变，都会影响循环的执行次数。例如，缺少 words＋＋则会使程序陷入死循环中，因为 words 中的值不发生改变，循环条件“words＜＝30”永远为真，循环永远不会结束。

（2）严格遵守 while 语句的语法格式，例如循环条件“words＜＝30”需要用一对圆括号“()”括起来，括号不能省略；例如循环条件不能空着，即使是想表示死循环，也要写成类似“while(1) ”的形式，当然，一般死循环没有意义，除非循环体内有 break 或 exit 等强制退出方式；例如循环条件“(words＜＝30)”后面没有分号或逗号等。

（3）循环体语句可以是表达式语句、空语句、流程控制语句、复合语句中的任何一种，但必须是一条语句。如果循环体是多条语句，一定要用花括号{}括起来组成一条复合语句。例如，将例 5-1 中源代码去掉花括号：

```
while(words<=30)
    sum+=words;
    words++;
```

一条循环语句，循环体为sum+=words;

循环后面的语句，与while 并列

程序运行时，光标会一直闪烁，但就是不出结果。这是因为循环体变成了“sum＋＝words;”，“words＋＋;”被排除在循环体外，变成了循环后面的语句。执行循环时，循环变量 words 不发生任何改变，就导致循环条件 words＜＝30 永远为真，陷入死循环中。

【例 5-2】 编程实现，统计一行字符中单词的个数，单词间用若干空格分开。

程序分析：

（1）一行字符是指输入若干字符，以回车符“\n”结束。

（2）如果当前位置不是，但前一个位置是空格，则说明有一个新单词，单词数加 1。

```
#include <stdio.h>
int main()
{
    char ch;                          //用于存放字符
    //num 用于存放单词个数，space 为 1 表示是空格，为 0 表示不是空格
    int num=0,space=1;                //space 的初值为 1，为统计第一个单词做好准备
    while((ch=getchar())!='\n')       //从键盘上接收一个字符赋给 ch，遇回车符结束
        if(ch==' ') space=1; //ch是空格则space赋为1
        else if(space==1) //ch不是空格，且前一个是空格
        { /*space赋为0，为判断下一个字符做准备*/
            space=0;
            num++;  //单词数加1
        }
    printf("一共有%d 个单词\n",num);
}
```

循环体是一条 if 语句

程序运行结果为：

```
程序输入：I'll do my best!<CR>
输出结果：一共有 4 个单词
```

注意：

(1) 循环条件“(ch=getchar())!='\n'”的含义为：用 getchar 函数接收一个字符，将其赋给 ch，然后比较 ch 中的值是否与回车符相等。ch=getchar()两端必须用圆括号括起来，因为=的优先级低于!=，如果写成“ch=getchar()!='\n'”，则表示“ch=(getchar()!='\n')”，即 getchar 函数接收的字符先和回车符做!=关系运算，运算结果赋给 ch。因为关系运算的结果只有 1 或 0，所以循环体内 ch=='\n'的判断条件永远不会成立，会导致无论多少个单词，最终结果都是一个单词的逻辑错误。

(2) getchar 函数是一个缓冲输入函数，也就是从键盘上输入若干字符先放入缓冲区中，直到遇到回车符，getchar 函数才从缓冲区中依次读取字符。因此单步执行或调试此程序时，需要在第一次执行 getchar 函数时一次性输入一行字符，以回车符结束。以后每执行一次循环，会依次在缓冲区中读取一个字符。

扫一扫

5.2 for 语句

for 循环语句与 while 循环语句一样，也是“先判断，后执行”型循环结构，其一般应用形式如下：

```
for(「初值表达式」;「条件表达式」;「修正表达式」)
    循环体语句
```

其执行流程如图 5-3 所示。

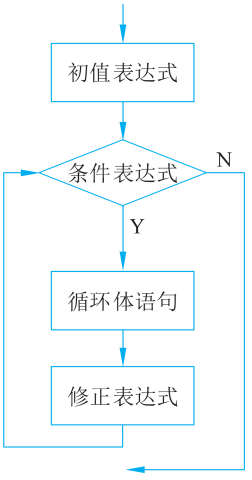


图 5-3 for 语句执行流程图

① 计算“初值表达式”的值。

② 计算“条件表达式”的值，若值为真(非 0)，则转到第③步；若值为假(0)，则转到第⑤步。

③ 执行一次循环体。

④ 计算“修正表达式”，转到第②步继续执行。

⑤ 循环结束，执行 for 语句后面的语句。

【例 5-3】 利用 for 语句完成例 5-1 的功能，即某同学第 1 天学习一个单词，以后每天都比前一天多学习一个单词，请问 30 天后该同学共学习了多少单词？

```
#include <stdio.h>
int main()
{
    int words,sum=0;
    for(words=1;words<=30;words++)        //循环三要素用分号隔开
        sum+=words;          //循环体只有一条单语句，不用加括号形成一条复合语句
    printf("sum=%d\n",sum);
}
```

此例采用 for 循环完成例 5-1 的功能，通过与 while 实现代码比较发现，for 与 while 都是“先判断，后执行”型循环，可以互相代替，但 for 循环将循环的三要素：循环变量赋初值、循环条件和循环变量修正集中放在了一起，结构更紧凑，条理更清晰，因而更受欢迎，成为应用最广泛的循环语句。

for 语句注意事项如下。

(1) 三个表达式是循环的三要素，均可以省略，但表达式之间的分号“;”不能省略。例如，程序中的关键代码也可以写成：

```
words=1;                //将初值表达式提到循环前，加分号形成语句
for(;words<=30;)        //没有初值表达式和修正表达式，但必须有;
{
    sum+=words;
    words++;            //将修正表达式移至原循环体的尾部，加分号形成语句
}
```

但不建议这样写，因为这将循环三要素分散开来，让 for 语句“结构紧凑，条理清晰”的优点荡然无存。

(2) 循环三要素中的条件表达式是 for 语句的循环条件，一般为逻辑表达式或关系表达式，但不限于此，表达式结果为非 0，则判定为“真”，为 0，则判断为“假”，甚至可以省略。例如：

```
for(;1;)
for(;;)
```

都是条件表达式为永真的死循环，循环体中存在 break 或 exit 的强制退出才有意义。

(3) 初值表达式和修正表达式都可以是多个表达式，但需用逗号分开，例如：

```
for(words=1,sum=0;words<=30;words++,sum+=words)
    ;     //必须有循环体，如果没有可执行语句可用空语句
```

但不建议这样写，因为 sum=0 和 sum+=words 不是循环的三要素，放置于三要素位置会影响 for 语句的条理性。

【例 5-4】 编程实现，某学生的开发作品参加某计算机设计大赛，共 5 位评委，计算该作品的最终平均得分。

```
#define N 5                                  //定义符号常量 N 表示评委人数
int main()
{
    float sum=0,score;                       //sum 存放累计总分，score 存放评委评分
    int judges;                              //judges 表示评委人数
    for(judges=1;judges<=N;judges++)         //控制评委人数不超过 5 人
    {   //输出提示信息，明确是第几位评委
        printf("请输入第%d 位评委的评分: ",judges);
        scanf("%f",&score);                  //输入第 judges 位评委的评分
        sum=sum+score;                       //把当前评分累加到 sum 中
    }
    printf("sum=%.2f\n",sum/5);              //循环后的语句
}
```


扫一扫

5.3 do…while 语句

do…while 语句是“先执行，后判断”型循环结构，其一般应用形式如下：

```
do
    循环体语句
while(条件表达式);
```

其执行流程如图 5-4 所示。

① 执行循环体语句。

② 判断条件表达式的值，若值为真(非 0)，则转到第①步，若值为假(0)，则转到第③步。

③ 循环结束，执行 do…while 后面的语句。

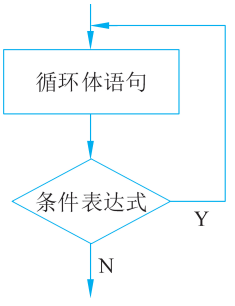


图 5-4 do…while 语句执行流程图

【例 5-5】 利用 do…while 语句完成例 5-1 的功能，即某同学第 1 天学习一个单词，以后每天都比前一天多学习一个单词，请问 30 天后该同学共学习了多少单词？

```
#include <stdio.h>
int main()
{
    int words=1,sum=0;
    do
    {   //循环体有两条单语句，必须用花括号括起来形成一条复合语句
        sum+=words;
        words++;
    }while(words<=30);        //;必不可少
    printf("sum=%d\n",sum);
}
```

对比例 5-1 和例 5-5 源代码发现：do…while 和 while 的核心思想一致，只是条件判断和循环体的执行次序不一样，do…while 是“先执行，后判断”，因此循环体至少执行一次；而 while 是“先判断，后执行”，循环体可能一次也不执行。两者在绝大多数情况下可以互相代替，只是 while 循环更适合循环体有可能一次也不执行的场景。例如，例 5-2 统计一行字符中单词的个数，有可能单词个数为 0，用 do…while 则会出错；do…while 循环更适合必须执行一次循环体才能判断是否继续循环的场景，例如，例 5-6 终极密码游戏必须进行一次猜测，才能判断是否要继续循环。

【例 5-6】 编程实现，编程模拟终极密码猜数小游戏。

程序分析： 猜数小游戏的规则是由程序随机生成一个 1～100 的整数密码，用户输入猜测数据。如果猜大了，则修改提示范围上限；如果猜小了，则修改提示范围下限；如果猜对了，则提示猜对了，程序结束。

例如，密码为 78，提示用户输入 1～100 的数；如果用户输入 48，则提示用户输入 48～100 的数；用户输入 80，则提示用户输入 48～80 的数；用户输入 78，则提示猜对了，程序结束。

```
#include <stdio.h>
#include <time.h>                       //time 函数包含在 time.h 中
#include <stdlib.h>                     //srand 和 rand 函数包含在 stdlib.中
int main()
{
    int password,guessdata,high=100,low=1;
    //用当前系统时间做 srand 函数的种子,保证程序多次运行产生的随机数不一样
    srand((unsigned)time(NULL));
    password=rand()%100+1;              //将产生的随机数,即密码限制在 1~100
    do
    {   //提示用户输入数据范围
        printf("请输入[%d-%d]之间的整数:\n",low,high);
        scanf("%d",&guessdata);         //用户输入一个猜测数
        if (guessdata>password)         //猜高了,则用猜测数代替范围上限
            high=guessdata;
        else                            //猜低了,则用猜测数代替范围下限
            low=guessdata;
    }while(guessdata!=password);        //猜不中就继续循环
    printf("恭喜你,猜中啦!\n");          //退出循环一定是猜中了
}
```

此例更适合用 do…while 实现，但用 while 循环也可以实现，例如核心代码修改为：

```
printf("请输入[%d-%d]之间的整数:\n",low,high);       //第一次提示范围
scanf("%d",&guessdata);                 //用户第一次猜测
while(guessdata!=password)              //必须先猜测一次才能判断是否继续猜测
{
    if (guessdata>password)
        high=guessdata;
    else
        low=guessdata;
    printf("请输入[%d-%d]之间的整数:\n",low,high);
    scanf("%d",&guessdata);
}
```

通过 do…while 和 while 实现代码的比较可以看出，do…while 实现方式更简洁，条理更清晰。因为此例的循环条件“guessdata！＝password”中的循环变量 guessdata 需要先赋值才能进行判断，因此当用 while 实现时，必须在循环前先猜测一次。这就导致语句：

```
printf("请输入[%d-%d]之间的整数:\n",low,high);
scanf("%d",&guessdata);
```

在循环前和循环体中重复出现了两次。因为第一次猜测提到循环前，循环体中的语句顺序也发生了变化。

思维拓展：①读者在测试例 5-6 时，可以添加一条输出语句，将密码输出到屏幕上，然后根据猜高、猜低、猜中三种情况分别进行测试。测试成功后再把输出语句注释掉，这样可以提高编程效率。②如果添加统计用户猜测的次数功能，应该如何修改程序呢？③如果用户猜测数据不在提示范围内，例如提示范围为[40～100]，但用户输入 20，则提示用户超出了范围，重新猜测，应该如何修改程序呢？


扫一扫

5.4 循环语句的嵌套

在一个循环（称为外层循环）体内又包含了另一个完整的循环语句（称为内层循环），称为循环的嵌套。内层循环体内又包含一个完整的循环语句，则构成多层循环。从理论上讲，循环的层数没有限制，但层数过多会影响程序的可读性，建议不超过 3 层。

前面介绍的 while、for 和 do…while 都可以相互嵌套。下面仅以 for 循环为例，介绍循环语句的嵌套应用。

【例 5-7】 编程实现，模拟一个简单的分秒计时器。

程序分析：分秒计时器由分和秒两个值组成，最多记录 59 分 59 秒。秒值每满 60，分值加 1。

```
#include <stdio.h>
#include <windows.h>                    //Sleep 函数所在头文件
int main()
{
    int minute,second;                  //定义分、秒变量
    for(minute=0;minute<60;minute++)    //分从0变化到59
        for(second=0;second<60;second++)    //秒从0变化到59
        {
            printf("\b\b\b\b\b\b");//输出退格键，新时间覆盖原时间
            printf("%02d:%02d",minute,second);//输出时间，不足两位用0补充
            Sleep(1000);    //睡眠1秒,S首字母大写
        }
}
```

外层循环 → for(minute=0;minute<60;minute++)

内层循环 → for(second=0;second<60;second++)

例 5-7 是一个双层循环，内层循环（虚线内）是外层循环体的一个组成语句，被完整地包裹在外层循环体中，两者不能相互交叉。执行时，外层循环的循环体执行一次，内层循环体执行一遍，其具体执行流程如图 5-5 所示。

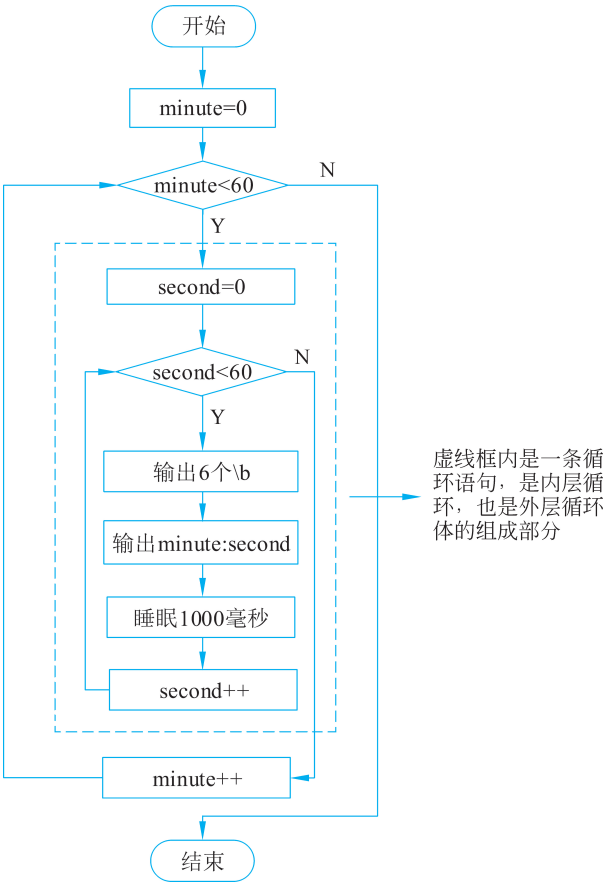


图 5-5　双层嵌套循环的执行流程图

① 程序开始。

② minute 赋值为 0。

③ 判断表达式“minute＜60”的值，若值为真（非 0），则转到第④步继续执行，否则转到第⑨步继续执行，即外层循环结束，转到外层 for 循环后面的语句继续执行。

④ second 赋值为 0。

⑤ 判断表达式“second＜60”的值，若值为真（非 0），则转到第⑥步继续执行，否则内层循环结束，转到第⑧步继续执行，即内层循环结束，转到内层循环后面的语句继续执行。

⑥ 执行内层循环体语句“输出 6 个\b，输出 minute：second 的值，睡眠 1000 毫秒”。

⑦ 执行内层循环修订表达式“second＋＋”，转到第⑤步继续执行。

⑧ 执行外层循环修订表达式“minute＋＋”，转到第③步继续执行。

⑨ 程序结束。

知识拓展：计时器功能可以从两方面优化。

（1）程序运行后，提示用户“请按任意键开始计时”，而不是程序一运行就开始计时。在外层循环前加如下代码即可：

```
printf("请按任意键开始计时");getch();
```

（2）程序开始计时后，按任意键结束计时，而不是被动地等程序计时至 59：59 结束。可