第5章 循环结构程序设计

案例导入——计算30名学生“C语言程序设计”课程的平均成绩

【问题描述】 从键盘输入30名学生“C语言程序设计”课程的成绩，计算该课程的平均成绩。

【解题分析】 要处理该问题，首先要了解题目的含义和要求，该题目只需要输入30名学生的成绩，并不需要保存这些成绩，因此只需要输入一个成绩做一次累加，此操作要处理30次，之后用成绩的累加值除以30就可以得到平均成绩。使用之前学过的顺序结构可以处理该问题：

首先定义3个变量分别存储学生成绩score、成绩的和total和平均成绩average，根据目前学校的常规约定，学生成绩score可以定义为整型，成绩的和total定义为整型，平均成绩average定义为单精度浮点型。学生成绩通过键盘输入，不需要给出初值；成绩的和total需要做累加，应该给出初值0；平均成绩average直接被赋值，也不需要给出初值。

```
int score,total=0;
float average;
```

然后输入一个学生的“C语言程序设计”课程的成绩score，并累加到成绩的和total中。

```
scanf("%d",&score);
total+=score;
```

以上两条语句需要再重复29次，才能完成30名学生“C语言程序设计”课程的成绩的输入和累加。

最后利用total除以30得到平均值，并保留两位小数输出。

```
average=total/30.0;          /*total是整型变量、30是整型常量，想得到实型结果则需
                               要将其中一个转化为实型数据*/
printf("average=%.2f\n",average);
```

利用以上方式确实能够完成题目要求，但显然不可取。此题目要求得到30名学生的平均成绩，重复书写30次相同的代码还可以忍受，但如果要求得到3000、30 000名学生

的平均成绩，输入和累加的代码段就需要重复书写 3000、30 000 次，那是难以想象的。工作量太大，也导致程序冗长、可读性差。另外，如果学生人数不确定，只是要求计算一批学生的“C 语言程序设计”课程的平均成绩，使用之前的顺序结构和选择结构根本没办法完成。

实际上，每种计算机的高级语言都提供了循环控制结构，用来处理需要重复操作的运算。

在 C 语言中，可以使用循环语句完成该题目：

```
#include <stdio.h>
int main()
{
    int score,i,total=0;
    float average;
    i=0;                                    //变量 i 初值设置为 0
    while(i<30)                             //当 i 的值小于 30 时执行花括号内的语句
    {
        scanf("%d",&score);                 //输入一名学生的成绩
        total+=score;                       //将该学生的成绩累加到 total 中
        i++;                                //每执行一次循环 i 的值加 1
    }
    average=total/30.0;                     //循环结束后计算平均值赋给变量 average
    printf("average=%.2f\n",average);       //输出 30 名学生的平均成绩
    return 0;
}
```

可以看出，用一个循环语句（while 语句）就可以解决代码段重复书写 30 次的问题。

循环结构的理解

导学与自测

扫二维码观看本章的预习视频——循环结构的理解，并完成以下课前自测题目。

【自测题 1】 下面程序的输出结果是(　　)。

```
#include <stdio.h>
int main()
{
    int i,sum=0;
    i=1;
    while(i<11)
    {
        sum+=i;
        i++;
    }
    printf("sum=%d\n",sum);
    return 0;
}
```

A. 55　　B. sum＝55　　C. sum＝66　　D. sum＝65

【自测题 2】 下面程序的输出结果是(　　)。

```
#include <stdio.h>
int main()
{
    int i,sum=0;
```

```
    i=1;
    while(i<11)
    {
        i++;
        sum+=i;
    }
    printf("sum=%d\n",sum);
    return 0;
}
```

A. 55　　B. sum＝55　　C. sum＝65　　D. sum＝66

【自测题 3】　下面程序段中 while 循环的执行次数是(　　)。

```
int t=10;
while(t=1)
{
    t++;
}
```

A. 一次也不执行　　B. 执行 10 次

C. 无限次　　D. 语法错误

我们在处理实际问题时，有些问题的处理过程是采用重复的动作来完成的。例如，要统计某班学生某一学期某门课程的平均成绩，就需要对每名学生的成绩进行输入和累加，这个输入和累加的过程是需要不断重复进行的。这种重复执行的过程在计算机的程序设计中称为循环。

循环结构是结构化程序设计的 3 种基本结构之一，也是程序流程控制中一种很重要的结构，几乎在所有的解决实际问题的应用程序中都要用到循环控制结构。它与顺序结构、选择结构一起实现复杂结构的程序设计。

现实生活中所遇到的循环形式根据实际问题的不同，其表现形式也不同，有的是已知重复(循环)次数，如上面提到的统计某班学生某一学期某门课程的平均成绩，学生人数定了，重复计算(循环)的次数也就确定了；有的则不能预知重复(循环)次数，如从 1 开始，多少个自然数的和刚好能超过 10 000。为解决上述两类循环问题，C 语言提供了 3 种实现循环结构的语句，分别是 while 语句(当型循环)、do…while 语句(直到型循环)和 for 语句。

本章首先介绍 C 语言实现循环结构的这 3 种循环语句的语法及使用方法，然后对能够改变循环体中语句执行顺序的 continue 语句和 break 语句做简单介绍，最后介绍使用循环结构解决的一些实际问题。

5.1　while 语句

while 语句的一般格式为

```
while(循环条件表达式)
{循环体语句}
```

语句的执行过程：在执行 while 语句时，先对循环条件表达式进行计算，若其值为真(非 0)，则执行循环体语句，然后重复上述过程，直到循环条件表达式的值为假(0)时，循环结束，程序控制转至 while 语句的下一个语句。这种先判断循环条件的循环称为当型循环。语句的执行过程如图 5.1 所示。

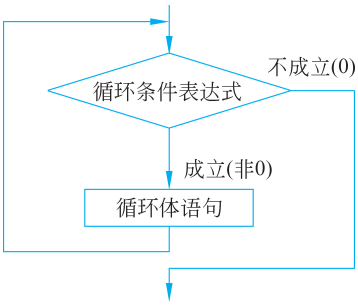

图 5.1　while 语句流程图

使用 while 语句时，应注意以下 4 个问题。

(1) while 语句的特点是“先判断，后执行”，如果循环条件表达式的值一开始就是 0，则循环体一次也不执行，但循环条件表达式是要执行的。

(2) while 语句中的循环条件表达式一般是关系表达式或逻辑表达式，但也可以是数值表达式或字符表达式，只要能判断真假即可。

(3) 循环体如果是一条语句，则花括号可以省略。

(4) 在循环体中，必须有使循环条件趋向于不成立(假)的语句。如果没有使循环条件趋向于不成立(假)的语句，则循环永远不能结束，称为死循环。

例 5.1　计算 $\sum_{n=1}^{100} n$ 。即计算 1＋2＋3＋…＋100，也就是求自然数 1～100 之累加和。

问题分析：这是一个累加(求和)问题，其计算过程如下。第一次计算 0＋1；第二次用第一次的求和结果加上 2，实际计算的是前两项的和；以此类推，第 n 次用第 $n-1$ 次的求和结果加上 n，即为自然数 1～n 之和，当 n 的取值为 100 时，即得到自然数 1～100 之和。

按照这一思路，可以构建如下算法。

(1) 声明一个变量(sum)存放加法的和，并设置初值为 0。

(2) 将 1 加入 sum。

(3) 将 2 加入 sum。

(4) 将 3 加入 sum。

…

(101) 将 100 加入 sum。

(102) 输出 sum 的值。

可以看出，步骤(2)～(101)描述的是相同的动作，因此，完成这种重复的操作可以利用 C 语言提供的循环结构来实现。

因此，程序可如下设计。

(1) 声明一个变量 sum，初值为 0。

(2) 设置变量 n，初值为 l。

(3) 将 n 加入 sum。

(4) 让 n 的值加 1。

(5) 当 n＜＝100 成立时，重复执行步骤(3)和(4)；当 n＞100 时，执行步骤(6)。

(6) 输出 sum 的值。

从上面的描述可以看出，步骤(3)和(4)要重复执行 100 次。

程序如下：

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

程序运行结果：

```
sum=5050
```

从例 5.1 中可以看出，当给定条件成立时，反复执行某几个步骤（程序段），直到条件不成立为止。给定的条件称为循环条件，反复执行的程序段称为循环体。其中，变量 n 称为循环控制变量，在循环体外的赋值 n=1 中的 1 称为循环控制变量的初值，在 while(n<=100)中的 100 称为循环控制变量的终值，在循环体内的 n++称为循环控制变量的增量。为了保证循环能正常退出，循环体内的循环控制变量的增量一定是使循环控制变量的值从初值慢慢接近终值，直到循环条件为假。

例 5.2　现有某班若干学生的 C 语言成绩，求该班学生的 C 语言的平均成绩。


计算平均值

问题分析：本例仍然是一个累加（求和）问题。但本题没有确定学生人数，不知道应该从键盘输入多少名学生的 C 语言成绩，即不知道循环多少次。这种类型的题目的解决办法可以采用以下两种方式。第一种方式是定义一个整型变量 count，用来存储学生人数，即循环次数，并从键盘输入；之后利用循环每输入一个成绩 score 就进行一次累加，当所有 count 名学生的成绩输入完成后，此时 total 的值即为 count 名学生的 C 语言成绩的累加和，用成绩的累加和除以 count 即得到该门课程的平均成绩。

因此，程序可如下设计（学生成绩为实型数据）。

（1）声明整型变量 count，存放学生人数；声明实型变量 total，初值为 0，存放成绩的累加和；声明实型变量 score，存放每名学生的成绩；声明实型变量 average，存放平均成绩。

（2）声明一个整型变量 i 作为循环控制变量，初值为 0。

（3）输入学生人数，即为变量 count 赋值。

（4）当 i<count 成立时，重复执行步骤（5）～（7）；当 i>=count 时，执行步骤（8）。

（5）输入每名学生的成绩，即为变量 score 赋值。

（6）将 score 加入 total 中。

（7）循环控制变量 i 加 1。

（8）计算 average=total/count，输出 average 的值（平均值保留 1 位小数）。

程序如下：

```
#include <stdio.h>
int main( )
{
    int count;
    float  total=0,score,average;
    int  i=0;
    printf("请输入学生人数:");
    scanf("%d",&count);
    printf("请输入学生成绩:");
    while(i<count)
    {   scanf("%f",&score);
        total=total+score;
        i++;
    }
    average=total/count;
    printf("%d名学生的C语言平均成绩:% .1f\n",count,average);
    return  0;
}
```

程序运行结果 1：

```
请输入学生人数：20
请输入学生成绩：23  89  67  78  94  56  34  83  90  67  48  72  81  65  74  87  73  67  90  98
20名学生的C语言平均成绩：  71.8
```

程序运行结果 2：

```
请输入学生人数：10
请输入学生成绩：68 78 86 66 51 90 97 61 72 81
10名学生的C语言平均成绩：  75.0
```

第二种方式是根据题目的描述找出处理问题的方法，本例处理的是一门课程的学生成绩，而学生成绩的取值范围是大于或等于 0 并且小于或等于 100 的。可以利用这个条件设置循环，即当输入的数据满足条件时进行累加；否则退出循环，并计算平均成绩。

因此，程序可如下设计(学生成绩为实型数据)。

(1) 声明整型变量 count，初值为 0，用来统计学生人数；声明实型变量 total，初值为 0，存放成绩的累加和；声明实型变量 score，存放学生的成绩；声明实型变量 average，存放平均成绩。

(2) 输入第一名学生的 C 语言成绩，即为变量 score 赋值。

(3) 当 score 满足大于或等于 0 并且小于或等于 100 时，重复执行步骤(4)～(6)；当 score 小于 0 或者大于 100 时，执行步骤(7)。

(4) 将 score 加入 total 中。

(5) 统计学生人数的变量 count 加 1。

(6) 输入下一名学生的成绩，即为变量 score 赋值。

(7) 计算 average＝total/count，输出 average 的值(平均值保留 1 位小数)。

程序如下：

```
#include <stdio.h>
```

```
int main( )
{
    int count=0;
    float  total=0,score,average;
    printf("请输入学生成绩:");
    scanf("%f",&score);
    while(score>=0 && score<=100)
    {
        total=total+score;
        count++;
        printf("请输入学生成绩:");
        scanf("%f",&score);
    }
    average=total/count;
    printf("%d名学生的C语言平均成绩:%.1f\n",count,average);
    return  0;
}
```

程序运行结果 1：

```
请输入学生成绩：85.6
请输入学生成绩：78.5
请输入学生成绩：89
请输入学生成绩：68
请输入学生成绩：75.5
请输入学生成绩：-2
5名学生的C语言平均成绩：79.3
```

程序运行结果 2：

```
请输入学生成绩：88.5
请输入学生成绩：79.5
请输入学生成绩：92
请输入学生成绩：101
3名学生的C语言平均成绩：86.7
```

例 5.3　设 $s=1\times2\times3\times\cdots\times n$，求 s 不大于 400 000 时最大的 n。

问题分析：

(1) 本例实际是计算前 n 个自然数的乘积。即用每次的乘积(s)与 400 000 比较，并记录乘积次数。所以它是一个不定次数的循环，循环体(计算乘积)是否执行，由 s 的值决定。

(2) 当 $s\leqslant400\ 000$ 时，需要计算自然数的乘积，即需要执行循环体。

(3) 当 $s>400\ 000$ 时，退出循环体。但此时 s 已经超过 400 000，所以所要的 n 值应该是实际求出的 n 值减 1。

因此，程序可如下设计。

(1) 定义所需的整型变量 n 存放参与乘积运算的自然数，整型变量 s 存放乘积，它们初值均为 1。

(2) 判断条件 s<=400 000 是否成立。若成立，则转步骤(3)；若不成立，则转步骤(4)。

(3) 计算每项的值 n，并用该值与前 n－1 项的乘积进行相乘运算，回到步骤(2)。

(4) 循环结束后，输出 n－1 的值，该值即为前 n 个自然数的乘积不大于 400 000 时最大的 n。不大于 400 000 的 s 值应该是退出循环时的 s 值除以 n。

程序如下：

```
#include <stdio.h>
int  main( )
{
    int  n=1;
    int  s=1;
    while(s<=400000)
    {
        n=n+1;
        s=s*n;
    }
    printf("不大于 400000 时最大的 n 为%d\ns 值为%d\n",n-1,s/n);
    return  0;
}
```

程序运行结果：

```
不大于400000时最大的n为  9
s值为 362880
```


素数的判断

例 5.4 从键盘输入一个非负整数，判断 m 是不是素数。

问题分析：

(1) 素数是指除 1 以外的只能被 1 和其自身整除的自然数，如 2、3、5、7、11 都是素数。所以，判断一个正整数 m 是不是素数所采用的方法如下：用$[2,m-1]$的所有整数除 m，但实际上用$[2,\sqrt{m}]$的所有整数除 m 即可，只要$[2,\sqrt{m}]$中有一个数能整除 m，m 就不是素数；若 m 不能被$[2,\sqrt{m}]$中的任何一个数整除，则 m 才是素数。

(2) 由(1) 的分析可以看出，判断一个正整数是素数还是非素数，判定的次数是不同的，也就是说，循环次数是不固定的。因此，可以采用设置标志的循环，一旦确定了某个正整数不是素数，就改变标志的初值。根据标志的值，即可区分素数和非素数。

(3) 循环体主要是判断$[2,\sqrt{m}]$中的某个整数 i 是否能整除 m，若能整除，则置标志变量 flag 为 1(在开始前先置 flag 为 0)。因此，循环采用当型循环结构实现，执行循环体的条件是 i<=sqrt(m)&&flag==0 的结果为真。退出循环有两种可能：一种是对于$[2,\sqrt{m}]$的所有整数都已判断完毕，且均不能整除 m，即 flag 仍为 0，在此情况下就认为 m 是素数；另一种是标志变量 flag 已改为 1，这表示在$[2,\sqrt{m}]$中已发现了一个整数 i 能整除 m，即说明 m 不是素数。

因此，程序可如下设计。

(1) 定义所需的整型变量 m，标志 flag=0，循环变量 i=2。

(2) 从键盘输入变量 m 的值(2 是最小的素数)。

(3) 如果 m<2，输出“输入错误!”，程序结束；否则执行步骤(4)～(6)。

(4) 判断条件 i<=sqrt(m) && flag==0 是否成立，若成立，执行循环体；若不成立，结束循环。

(5) 在循环体中判断条件 m%i==0 是否成立，若成立，m 不是素数，flag=1；若不

成立，i＋＋。

（6）循环结束后，判断 flag＝＝0 是否成立，若成立，m 是素数；若不成立，m 不是素数。

程序如下：

```
#include <math.h>
#include <stdio.h>
int  main()
{
    int  m,flag=0,i=2;
    printf("请输入一个非负整数: \n");
    scanf("%d",&m);
    if(m<2)
        printf("输入错误!\n");
    else
    {
        while(i<=sqrt(m) && flag==0)
        {
            if(m%i==0)
                flag=1;   //m 不是素数,修改 flag 的值
            else
                i++;
        }
        if(flag==0)
            printf("%d 是素数。\n",m);
        else
            printf("%d 不是素数。\n",m);
    }
    return 0;
}
```

程序运行结果 1：

```
请输入一个非负整数:
17
17是素数。
```

程序运行结果 2：

```
请输入一个非负整数:
297
297不是素数。
```

例 5.5　求两个非负整数 m 和 n 的最大公约数和最小公倍数。

问题分析：

（1）两个非负整数 m 和 n 的最大公约数一定小于或等于 m 和 n 中的较小数，所以将 m 和 n 中的较小数存入变量 t 中。若 m 不能被 t 整除或者 n 不能被 t 整除，则 t 一定不是 m 和 n 的最大公约数。此时，应使 t 的值减 1，再重复上述过程，直到 m 和 n 能同时被 t 整除为止，如果 m 和 n 是两个互质的非负整数，则 t 最终会减为 1。

（2）两个非负整数 m 和 n 的最小公倍数一定大于或等于 m 和 n 中的较大数，所以将 m 和 n 中的较大数存入变量 t 中。若 t 不能被 m 整除或者 t 不能被 n 整除，则 t 一定不是 m 和 n 的最小公倍数。此时，应使 t 的值增 1，再重复上述过程，直到 t 能同时被 m 和

n 整除为止，如果 m 和 n 是两个互质的非负整数，则 t 最终会增至 m 和 n 的乘积。

因此，程序可如下设计。

(1) 定义所需的变量 m、n、t。

(2) 从键盘输入变量 m、n 的值。

(3) 将 m、n 中的较小数赋值给 t。

(4) 判断条件 m%t!=0||n%t!=0 是否成立。若成立，t 不是最大公约数，计算 t--，重复执行步骤(4)；若不成立，则说明 m%t、n%t 均为 0，即求得最大公约数 t。

(5) 将 m、n 中的较大数赋值给 t。

(6) 判断条件 t%m!=0||t%n!=0 是否成立。若成立，t 不是最小公倍数，计算 t++，重复执行步骤(6)；若不成立，则说明 t%m、t%n 均为 0，即求得最小公倍数 t。

程序如下：

```
#include <stdio.h>
int  main( )
{  int  m,n,t;
    scanf("%d%d",&m,&n);
    t=(m<=n)?m:n;
    while (m%t!=0||n%t!=0)        //t 能否整除 m、n
          t--;
    printf("最大公约数为%d\n",t);
    t=(m>n)?m:n;
    while (t%m!=0||t%n!=0)        //m、n 能否整除 t
          t++;
    printf("最小公倍数为%d\n",t);
    return  0;
}
```

程序运行结果 1：

```
请输入两个正整数：12 8
最大公约数为  4
最小公倍数为  24
```

程序运行结果 2：

```
请输入两个正整数：7 25
最大公约数为  1
最小公倍数为  175
```

5.2 do…while 语句

do…while 语句的一般格式为

```
do {
       循环体语句
}while(循环条件表达式);
```

语句的执行过程：先执行循环体语句，然后对循环条件表达式进行计算，若其值为真