第 3 章

程序流程控制

学习目标：

(1) 理解程序设计的流程控制概念。

(2) 掌握分支和多分支语句的语法格式及用法。

(3) 理解循环的概念，掌握 while、do-while、for 3 种循环语句的语法格式、用法及区别。

(4) 掌握选择结构和循环结构嵌套的含义及用法。

编写程序的目的是使用计算机帮助我们完成相关任务，而任务的执行都是有顺序的，这个顺序体现在程序中就是程序的流程。程序流程控制是指在程序设计中控制完成某种功能的次序。无论多复杂的算法，均可通过顺序、选择、循环 3 种基本控制结构构造出来。每种结构仅有一个入口和出口。由这 3 种基本结构组成的多层嵌套程序称为结构化程序。


视频

3.1 顺序结构程序设计

顺序结构是一种线性、有序的结构。顺序结构的程序设计是最简单的，只要按照解决问题的顺序写出相应的语句，执行顺序是自上而下，依次执行。顺序结构可以独立使用构成一个简单的完整程序，但大多数情况下顺序结构都是作为程序的一部分，与其他结构一起构成一个复杂的程序，如分支结构中的复合语句、循环结构中的循环体等。

【例 3-1】 交换两个整型变量的值。

程序代码如下：

```
/* e3_1.c */
#include<stdio.h>
void main(){
    int x=3,y=5,temp;
    temp=x;
    x=y;
    y=temp;
    printf("x=%d,y=%d\n ",x,y);
}
```

程序运行结果：

```
x=5,y=3
```

程序说明：

(1) 本程序是一个顺序结构的程序，程序中的语句按排列次序顺序执行。

(2) 程序通过一个中间变量 temp 对 x 和 y 的值进行交换。首先将 x 的值存储在临时变量 temp 中，将 y 的值赋给 x，再将 temp 的值赋给 y，从而实现 x 和 y 值的互换。

(3) 将中间的 3 条语句修改为：

```
x=x+y;    y=x-y;    x=x-y;
```

也可以实现 x 和 y 值的互换功能，这样可以不需要定义中间变量 temp。

3.2 选择结构程序设计

选择结构是根据条件成立与否选择程序执行的路径，控制程序的流程。选择结构的程序设计方法关键在于构造合适的分支条件和分析程序流程，根据不同的程序流程选择适当的执行语句。选择结构适合于带有逻辑或关系比较等条件判断的计算，设计这类程序时往往要先绘制流程图，然后根据程序流程写出源程序，这样做把程序设计分析与语言分开，使得问题简单化，易于理解。选择结构也称为分支结构，C 语言的选择结构通过 if 语句和 switch 语句实现。

3.2.1 if 语句

1. 单分支 if 语句

基本格式：

```
if(表达式)
   语句
```

执行这一结构时，首先对表达式的值进行判断。如果表达式的值为“真”，则执行其后的语句，否则不执行该语句。其执行流程如图 3.1 所示。

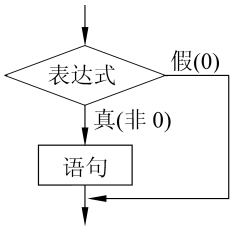

图 3.1 if 结构的流程

【例 3-2】 在两个数中输出较大数。

为了在两个数中找出较大的数，显然需要对两个数进行比较，根据比较结果决定输出哪个数。也就是说，需要根据指定条件来决定输出结果。

程序代码如下：

```
/* e3_2.c */
```

```
#include <stdio.h>
void main(){
    int num1,num2;
    printf("\n input 2 numbers: ");              /* 提示用户输入两个数 */
    scanf("%d%d",&num1,&num2);                   /* 从键盘输入数据 */
    if(num1>=num2)
        printf("max=%d\n",num1);                 /* 若 num1>=num2,输出 num1 */
    if(num1<num2)
        printf("max=%d\n",num2);;                /* 若 num1<num2,输出 num2 */
}
```

程序运行结果：

```
nput 2 numbers: 5 3
max=5
```

程序说明：

显然，程序中的两条 printf 语句只有一条会被执行。

使用 if 语句应注意下列几个问题：

(1) if 语句中，若条件满足需要执行多条语句，则必须用花括号把多条语句括起来组成一条复合语句；若条件满足只执行一条语句，则花括号可以省略。

(2) 条件表达式通常是逻辑表达式或关系表达式。例如：

```
if(x==y&&a==b) printf("x=y,a=b\n");
```

表达式也可以是其他表达式，如赋值表达式等，甚至也可以是一个变量。例如：

```
if(x=3) 语句;
if(a) 语句;
```

都是合法的，只要表达式的值为“真”，则执行其后的语句。

(3) 空语句是允许的，表示什么也不做。例如：

```
if(b>0);
```

(4) 正确使用缩进格式，有助于更好地理解程序，尤其是在使用 if 语句嵌套时。

2. 双分支 if-else 语句

基本格式如下：

```
if(表达式)
   语句 1
else
   语句 2
```

执行该结构时，首先对表达式的值进行判断。如果表达式的值为“真”，则执行语句 1；否则执行语句 2。其执行流程如图 3.2 所示。

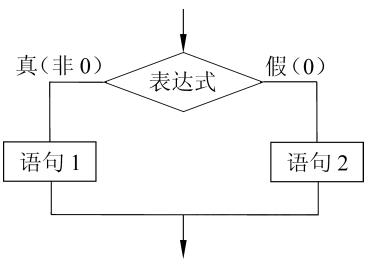

图 3.2　if-else 结构

因此例 3-2 的程序也可以改写为例 3-3 的程序。

【例 3-3】 在两个数中输出较大数。

程序代码如下：

```
/* e3_3.c */
#include<stdio.h>
void main(){
int num1,num2;
    printf("\n input 2 numbers: ");
    scanf("%d%d",&num1,&num2);
    if(num1>num2)
        printf("max=%d\n",num1);
    else
        printf("max=%d\n",num2);
}
```

程序运行结果：

```
input 2 numbers:
5 3
max=5
```

程序说明：

用 if-else 语句实现，只进行一次比较就可以完成处理，比单分支的 if 语句易于理解且格式清晰。

对于条件较少的问题，使用单分支 if 或双分支的 if-else 语句通常都可以解决，但是如果问题中涉及的条件较多，使用单分支 if 或双分支的 if-else 语句处理起来就比较麻烦，这时可以采用多分支的 if-else-if 语句来完成。

3. 多分支 if-else-if 结构

基本格式：

```
if(表达式 1)
   语句 1
else if(表达式 2)
      语句 2
      else if(表达式 n)
               语句 n
          else 语句 n+1
```

执行这一结构时，依次判断表达式的值，当某个表达式值为“真”时，则执行对应的语句，然后跳出整个 if-else-if 语句。如果所有的表达式均为假，则执行语句 n＋1。if-else-if 结构的执行过程如图 3.3 所示。

【例 3-4】 将学生成绩由百分制转化为等级制。规则如下：

(1) 85 分(含)以上为 A 级；

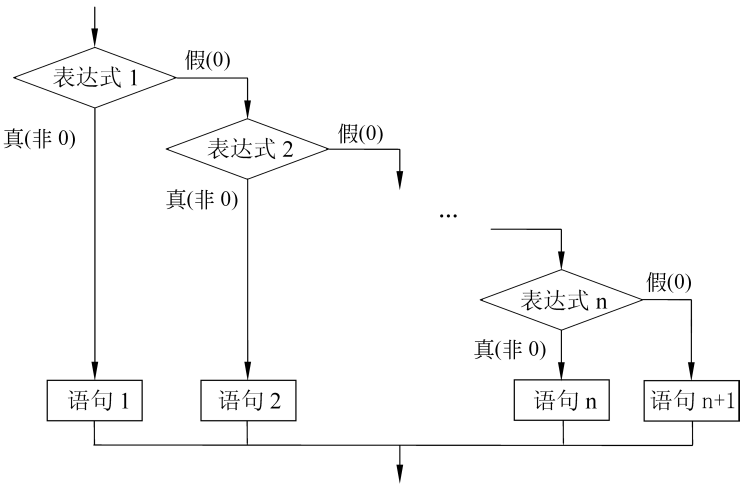

图 3.3　if-else-if 结构的执行过程

(2) 70 分(含)以上且 85 分以下为 B 级；

(3) 60 分(含)以上且 70 分以下为 C 级；

(4) 60 分以下为 D 级。

程序代码如下：

```
/* e3_4.c */
#include <stdio.h>
void main(){
    float score;
    printf("\n please input a score: ");
    scanf("%f",&score);
    if(score>=85)
        printf("the score %f is A \n",score);
    else if(score>=75)
            printf(" the score %f is B \n",score);
        else if(score>=60)
                printf("the score %f is C \n",score);
            else
                printf("the score %f is D \n",score);
}
```

程序运行结果：

```
please input a score: 89
the score 89.000000 is A
```

程序说明：

这是一个多分支选择的问题，需要对学生的成绩按区间分别判断，由于等级区间较多，因此使用多分支的 if-else-if 语句比较方便。

4. if 语句的嵌套

在解决一些问题时，还可能出现在一个条件判断中又包含了其他条件判断的情况，这种情况称为 if 语句的嵌套。

基本格式：

```
if(表达式 1)
    if(表达式 2) 语句 1
    else 语句 2;
else
    if(表达式 3) 语句 3
    else 语句 4
```

根据处理问题的不同，内层的 if-else 结构也可以是一个单分支 if 语句，或在内层的 if-else 结构中包含更多的分支。

【例 3-5】 求一元二次方程 $ax^2+bx+c=0(a\neq0)$的根。

求解一元二次方程的根 x 时有 3 种情况，分别为(记 $\Delta=b^2-4ac$)：

(1) $\Delta>0$，有两个不等的实根；

(2) $\Delta=0$，有两个相等的实根；

(3) $\Delta<0$，无实根。

程序代码如下：

```
/* e3_5.c */
#include<stdio.h>
#include<math.h>
void main(){
    float a,b,c,x1,x2,delta;
    printf("输入 a,b,c 的值: ");
    scanf("%f %f %f", &a, &b, &c);
    delta=b*b-4*a*c;
    if(delta>=0){
       if(delta>0){
          x1=(-b+sqrt(delta))/(2*a);
          x2=(-b-sqrt(delta))/(2*a);
          printf("两个不等的实根: x1=%.2f x2=%.2f\n", x1, x2);
       }
        else{
             x1=-b/(2*a);
             printf("两个相等的实根,x1=x2=%.2f\n", x1);
        }
    }
    else{
         printf("方程无实根!\n");
```

```
    }
}
```

程序运行结果：

```
输入 a,b,c 的值: 2 6 3
两个不等的实根: x1=-0.63 x2=-2.37
```

程序说明：

(1) 外层 if-else 语句处理 delta>=0 和 delta<0 的情况，内层 if-else 语句处理 delta>0 和 delta=0 的情况。这就是典型的 if 语句的嵌套结构。

(2) 在嵌套的 if 语句中，由于有多个 if 和多个 else，此时应注意 else 和 if 如何匹配。匹配的规则是在嵌套 if 语句中，if 和 else 按照“就近配对”的原则配对，即 else 总是与它上面距它最近的且还没有配对的 if 相匹配。

(3) 在嵌套的 if-else 语句中，如果内嵌的是单分支 if 语句，可能在语义上产生二义性。如：

```
if(表达式 1)
   if(表达式 2) 语句 1
else
   if(表达式 3) 语句 2
   else 语句 3
```

以上程序段中，虽然第一个 else 与第一个 if 在书写格式上对齐，但按照匹配规则，与它匹配的应是第二个 if。如果希望第一个 else 与第一个 if 匹配，可以采用两种方法：

① 加花括号。程序段改写为：

```
if(表达式 1)
   {if(表达式 2) 语句 1}
else
   if(表达式 3) 语句 2
   else 语句 3
```

加上花括号后，括号内的 if 语句是一个整体，不再与外部 else 匹配，这样第一个 else 只能与第一个 if 匹配。

② 加空的 else 语句。程序段改写为：

```
if(表达式 1)
   if(表达式 2) 语句 1
   else;
else
      if (表达式 3) 语句 2
      else 语句 3
```

加上空的 else 后，执行结果不变，但由于该 else 与内层 if 匹配，这样原来的第一个 else 就可以与第一个 if 匹配。

3.2.2 switch 语句

多分支选择结构也可以使用 switch 语句，switch 语句也称为标号分支结构。

基本格式：

```
switch(表达式){
    case 常量表达式 1: 语句序列 1
    case 常量表达式 2: 语句序列 2
    ……
    case 常量表达式 n: 语句序列 n
    default: 语句序列 n+1
}
```

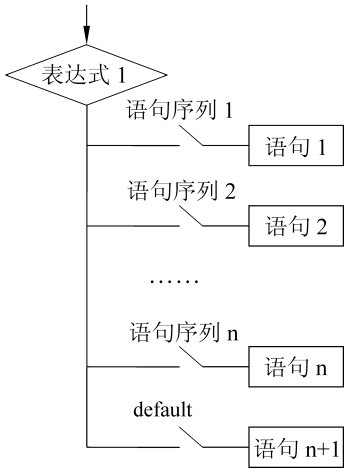

图 3.4　switch 结构的流程

switch 结构在执行时，首先计算 switch 判断表达式的值，并按照计算结果依次寻找 case 子结构中与之相等的常量表达式，若找到，则执行该 case 子结构对应的语句序列；若找不到与之相等的常量表达式，则执行 default 后的语句序列。default 子句不是必需的。若结构中无 default 子结构且没有相符的 case 子结构时，则什么也不做。执行流程如图 3.4 所示。

在使用 switch 结构时，应注意以下几个问题。

(1) case 后的各常量表达式值不能相同。

(2) case 后允许有多个语句，可以不用花括号括起来。

(3) 若 switch 表达式与常量表达式 m 匹配，执行该 case 子结构对应的语句序列 m 后，不是立即退出 switch 结构，而是继续执行语句序列 m+1，直至语句序列 n+1。若需要在执行语句序列 m 后，立即退出 switch 结构，则在每个语句序列后加一条 break 语句，break 语句用于跳出 switch 结构。

(4) switch 结构也可以嵌套。

【例 3-6】 输入 1～7 中的数字，将其转换成相应的星期英文单词。

程序代码如下：

```
/* e3_6.c */
#include<stdio.h>
void main(){
    int num;
    scanf("%d",&num);
    switch(num) {
        case 1: printf("Monday\n"); break;
        case 2: printf("Tuesday\n"); break;
        case 3 : printf("Wednesday\n"); break;
        case 4: printf("Tursday\n"); break;
```

```
        case 5: printf("Friday\n"); break;
        case 6: printf("Saturday\n");break;
        case 7: printf("Sunday\n");break;
        default: printf("Error\n");
    }
}
```

程序运行结果：(输入数字 5)

```
Friday
```

若输入 1～7 之外的数字,则显示 error。

程序说明：

每一个 case 子结构最后都有一条 break 语句,用于跳出 switch 流程。程序中如果每一个 case 子结构无 break 语句,运行时,输入数字 5,则运行结果为：

```
Friday
Saturday
Sunday
Error
```

【例 3-7】 编写程序测试是数字、空白,还是其他字符。

程序代码如下：

```
/* e3_7.c */
#include<stdio.h>
void main(){
    char c;
    scanf("%c",&c);
    switch(c) {
        case '0':
        case '1':
        case '2':
        case '3':
        case '4':
        case '5':
        case '6':
        case '7':
        case '8':
        case '9':printf("this is a digit\n"); break;
        case ' ':
        case '\n':
        case '\t':printf("this is a blank\n"); break;
        default :printf("this is a character\n"); break;
      }
  }
```

程序运行结果：

```
3
this is a digit
```

程序说明：

输入数字 3，应该执行 case '3'后面的语句，但 case '3'后面没有语句，因此会继续执行 case '4'后面的语句，以此类推。执行 case '9'后面的语句，遇到 break 退出 switch 结构。

【例 3-8】 输入某年某月某日，判断这一天是这一年的第几天。

程序代码如下：

```
/* e3_8.c */
#include<stdio.h>
void main(){
    int day,month,year,sum,leap;
    printf("please input year,month,day\n");
    scanf("%d,%d,%d",&year,&month,&day);
    switch(month) {                                  /* 先计算某月以前月份的总天数 */
        case 1:sum=0;break;
        case 2:sum=31;break;
        case 3:sum=59;break;
        case 4:sum=90;break;
        case 5:sum=120;break;
        case 6:sum=151;break;
        case 7:sum=181;break;
        case 8:sum=212;break;
        case 9:sum=243;break;
        case 10:sum=273;break;
        case 11:sum=304;break;
        case 12:sum=334;break;
        default: printf("data error");break;
}
    sum=sum+day;                                     /*再加上某天的天数*/
    if(year%400==0||(year%4==0&&year%100!=0))  /*判断是不是闰年*/
        leap=1;
    else
        leap=0;
    if(leap==1&&month>2)              /*如果是闰年且月份大于 2，总天数应该加一天*/
        sum++;
    printf("It is the %dth day.",sum);
}
```

程序运行结果：

```
please input year,month,day:
2021,9,10
```

```
It is the 253th day.
```

程序说明：

先把前 8 个月的天数加起来，然后再加上 10 天即为本年的第几天。正常 2 月份 28 天，若是闰年且输入月份大于 2 时多加一天。

3.3 循环结构程序设计

循环结构是重复执行一个或几个模块，直到满足某一条件为止。循环结构可以减少源程序重复书写的工作量，是程序设计中最能发挥计算机特长的程序结构。使用循环结构前需要确定以下两个问题，一是重复执行哪些语句？二是重复执行这些语句的条件是什么？这两个问题决定了循环的内容和循环的条件。C 语言提供 4 种循环方式，即 while 循环、do-while 循环、for 循环和 goto 循环。一般情况下 4 种循环可以互相代替，但不提倡用 goto 循环，因为强制改变程序的顺序经常会给程序的运行带来不可预料的错误。特别要注意在循环体内应包含趋于循环结束的语句（即循环变量值的改变），否则可能成为死循环。本书不介绍 goto 循环。

3.3.1 while 语句

while 用来实现“当型”循环结构。

基本格式：

```
while(表达式)
    语句
```

执行 while 语句时，先对表达式进行计算，若值为“真”（非 0），执行循环体语句，否则结束循环。每执行完一次循环体，都要对表达式进行一次计算和是否再次执行循环体的判断。流程如图 3.5 所示。

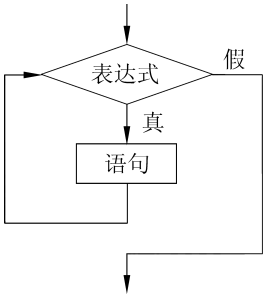

图 3.5 while 结构的流程

【例 3-9】 计算 sum＝1＋2＋3＋4＋…＋100。

本例的算法在 1.4.1 节已经介绍，由于重复加 1～100 的每个数，因此需要使用循环结构。这里使用 while 语句实现循环。

程序代码如下：

```
/* e3_9.c */
#include<stdio.h>
void main(){
    int sum=0,i;
    i=1;
    while(i<=100){
        sum=sum+i;
```

```
        i++;
    }
    printf("sum=%d\n",sum);
}
```

程序运行结果：

```
sum=5050
```

程序说明：

循环变量 i 初值为 1，循环体每执行一次，i 值增 1。当 i=101 时，结束循环体。也就是说，程序执行后，i 最终值是 101，即循环条件共判断了 101 次。


视频

【例 3-10】 猴子吃桃问题。

猴子第一天摘下若干个桃子，当即吃了一半，不过瘾，又多吃了一个。第二天又将剩下的吃了一半，又多吃了一个，以后每天都吃剩下的一半多一个，到第 10 天时，只剩下一个桃子。求第一天共摘了多少个桃子。

算法分析：

设第 n 天的桃子数为 p_n，则算法可描述为：

$$p_{10}=1 \tag{3-1}$$

$$p_n=p_{n-1}/2-1 \tag{3-2}$$

要计算第一天的桃子数，采用逆推法。即从第 10 天开始推算，式(3-1)为赋初值，式(3-2)用 C 语言描述为：

```
p1=2*(p2+1);
p2=p1;
```

这里的 p1 和 p2 分别代表每个“昨天”和每个“今天”桃子数。不断用“ 昨天”的桃数替代“今天”的桃数，这种不断地旧值递推得到新值的过程叫作迭代。迭代要有初值、迭代公式、迭代终止次数。

程序代码如下：

```
/* e3_10.c */
#include<stdio.h>
void main(){
    int p1,p2=1;
    int n=9;
    while(n>0) {
        p1=2*(p2+1);
        p2=p1;
        n--;
    }
    printf("the total is %d\n",p1);
}
```

程序运行结果：

```
the total is 1534
```

程序说明：

迭代初值为1(p2＝1)，即第10天剩下的桃子数；迭代公式为p1＝2＊(p2＋1)，p2＝p1。

使用while循环要注意以下几点。

(1) while循环在语法上整体是一条单独语句。

(2) 若循环体有多条语句，应用花括号将这些语句括起构成一条复合语句；若循环体只有一条语句，花括号可以省略。

(3) while是一个入口条件循环，如果开始条件不成立，则循环体一次也不执行。

(4) 循环体可以是空语句。

如程序片段：

```
int n=0;
while(n++<3);
    printf("n is %d\n",n);
printf("it's over.\n");
```

其执行结果为：

```
n is 4
it's over.
```

循环体只有一条空语句(;)。循环体虽然什么功能也没有实现，但循环体被执行了3遍；由于循环条件被判断了4次，n＋＋被执行4次。

【例3-11】 输入两个正整数m和n，求其最大公约数和最小公倍数。

算法分析：

求最大公约数通常采用“辗转相除法”，又称欧几里得算法。具体算法如下。

(1) 用m和n中的大数m除以n，得余数r(0＜＝r＜＝n)。

(2) 判断余数r是否为0。若r＝0，当前的除数值则为最大公约数，算法结束；否则进行下一步。

(3) 若r!＝0，用当前除数更新被除数，用当前余数更新除数，再返回第(1)步。

程序代码如下：

```
/* e3_11.c */
#include<stdio.h>
void main(){
    int a,b,m,n,t,r;
    printf("please input 2 numbers:\n");
    scanf("%d,%d",&m,&n);
    if(m<n){
        t=m;
        m=n;
        n=t;
```

```
    }
    a=m; b=n;
    while(b!=0){                                    /* 利用辗转相除法,直到 b 为 0 为止 */
        r=a%b;
        a=b;
        b=r;
    }
    printf("greatest common divisor:%d\n",a);
    printf("least common multiple:%d\n",m*n/a);
}
```

程序运行结果：

```
please input 2 numbers:
35,42
greatest common divisor:7
least common multiple:210
```

程序说明：

(1) 在辗转相除之前用 if 语句比较 m、n 的大小，并将较大数存放在 m 中，较小数存在 n 中。然后通过大数对小数的辗转相除得出结果。需要注意最后输出的是变量 a 的值而不是变量 b 的值，这是因为在循环体中辗转相除的除数 b 总是赋值给变量 a，随后 b 被赋值给余数。

(2) 使用变量 a 和 b，是为了对 m 和 n 两个变量进行保护。

3.3.2 do-while 语句

do-while 也是一种循环结构，称为当型循环。

基本格式：

```
do
    循环体语句
while(表达式);
```

当程序流程到达关键字 do 后，立即执行一次循环体，然后对表达式进行判断。若表达式值为“真”，则重复执行循环体；否则结束循环。do-while 结构至少执行一次循环体。执行过程如图 3.6 所示。

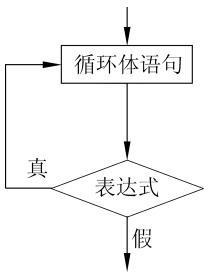

图 3.6 do-while 结构的流程

【例 3-12】 使用 do-while 语句改写例 3-9 的程序。

程序代码如下：

```
/* e3_12.c */
#include <stdio.h>
void main(){
    int sum=0,i;
```

```
    i=1;
    do{
          sum=sum+i;
          i++;
    }while(i<=100);
    printf("sum=%d\n",sum);
}
```

while 循环根据表达式的成立与否来决定是否执行循环语句，所以其循环语句可能一次也不执行；而 do-while 循环是先执行循环体，而后再判断表达式的值，因此 do-while 循环至少执行一次。

3.3.3 for 循环


视频

for 循环是 C 语言中最具特色的循环结构，使用最为灵活方便。

基本格式：

```
for(表达式 1;表达式 2;表达式 3)循环体
```

说明：

表达式 1：初值表达式，循环开始前为循环变量赋初值。

表达式 2：循环控制逻辑表达式，控制循环执行的条件，决定循环的次数。

表达式 3：循环控制变量修改表达式，使循环趋向结束。

先执行表达式 1，表达式 1 在整个循环中只执行一次。接着重复下面的过程：判断表达式 2 的值，若为“真”，执行循环体，然后执行表达式 3，再判断表达式 2，……；如果表达式 2 为“假”，则结束循环。循环体在循环控制条件（表达式 2）成立的情况下被反复执行。执行过程如图 3.7 所示。

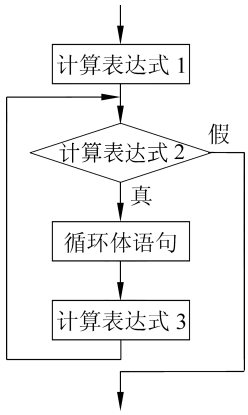

图 3.7　for 循环结构的流程

【例 3-13】 使用 for 循环改写例 3-9 的程序。

程序代码：

```
/* e3_13.c */
#include <stdio.h>
void main(){
int sum=0,i;
    for(i=1;i<=100;i++)
        sum=sum+i;
    printf("sum=%d\n",sum);
}
```

程序说明：

(1) 表达式 1 可以省略，但其后的分号不能省略。如：

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

(2) 表达式 2 可以省略,但其后的分号不能省略。当表达式 2 省略时,循环条件将始终为“真”,此时必须在循环体内加入使程序终止执行的语句,否则循环体将永远反复执行,形成死循环。

(3) 表达式 3 是步长表达式,用来设置循环变量的变化。随着循环变量不断变化,其值趋于使循环条件不成立。表达式 3 也可以省略,此时需要通过其他方式实现循环变量的变化。如:

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

使用 for 循环要注意以下几点。

(1) 整个 for 循环语句在语法上是一条语句。

(2) 若循环体有多条语句,应用花括号将这些语句括起构成一条复合语句;若循环体只有一条语句,花括号可以省略。

(3) 表达式 1 中可以包含多个表达式,此时需用逗号分隔。如:

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

(4) 循环体可以是空语句。如:

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

循环体只有一条空语句“;”,被循环执行 100 次。


视频

【例 3-14】 求 Fibonacci 数列的第 40 项。Fibonacci 数列规律:前两个数为 1,从第三个数开始,每个数都是其前面两个相邻数的和。

算法分析:

Fibonacci 数列用算法可表示为:

$$f1 = 1 \quad (n = 1) \tag{3-3}$$

$$f2 = 1 \quad (n = 2) \tag{3-4}$$

$$f_n = f_{n-1} + f_{n-2} \quad (n >= 3) \tag{3-5}$$

其中式(3-5)为迭代公式。用 C 语言来描述为:

```
f=f1+f2;
f1=f2;
f2=f;
```

程序代码如下:

```
/* e3_14.c */
```

```
#include <stdio.h>
void main(){
    long f,f1,f2;
    int i;
    f1=f2=1;                                    /*初始化前两项的值*/
    for(i=3;i<=40;i++){
        f=f1+f2;                                /*当前第 i 项的值等于前两项值之和*/
        f1=f2;                                  /*将当前 f2 的值赋给 f1*/
        f2=f;                                   /*将当前第 i 项的值赋给 f2*/
    }
    printf("the 40th is %ld\n",f);
}
```

程序运行结果：

```
the 40th is 102334155
```

程序说明：

由于数列前两项的值已确定，因此从第三项开始计算，所以循环变量 i 的初值为 3，i 值通过表达式 3 发生变化。循环体为一条复合语句{f＝f1＋f2;f1＝f2;f2＝f;}，该语句先计算当前第 i 项的值(等于前两项 f1、f2 之和)，然后将当前 f2 的值赋给 f1，再将当前第 i 项的值赋给 f2，通过执行这条复合语句就为计算第 i＋1 项准备好了 f1 和 f2。此处 f、f1、f2 应定义为长整型。

【例 3-15】 打印出所有的“水仙花数”。所谓“水仙花数”是指一个三位数，其各位数字立方和等于该数。

在 100～999 依次取数 m，然后分离出百位数 i、十位数 j、个位数 k，并将三者立方和与 m 相比较，如相等，则 m 为水仙花数。

程序代码如下：

```
/* e3_15.c */
#include<stdio.h>
void main(){
    int i,j,k,m;
    for(m=100;m<1000;m++){
        i=m/100;                                /*m 的百位数*/
        j=m/10%10;                              /*m 的十位数*/
        k=m%10;                                 /*m 的个位数*/
        if(m==i*i*i+j*j*j+k*k*k)
            printf("%5d",m);
    }
}
```

程序运行结果：

```
  153  370  371  407
```

程序说明：

本例采用的是穷举法，对问题中所有可能的值或状态逐一测试。

3.3.4 循环的嵌套

在一个循环体内包含另一个完整循环的结构，称为循环的嵌套。对于一些较为复杂的问题，必须使用循环的嵌套，while、do-while、for 循环都可以相互嵌套。需要注意，执行嵌套的循环时，外层循环每执行一次，内层循环执行一个周期。

【例 3-16】 打印如下图形：

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

打印此类图形，先找出规律性。可以把图形分上下两部分来看，前 4 行规律相同：第 i 行由 2 * i－1 个星号和 6－2 * i 个空格组成；后 3 行规律相同：第 i 行由 7－2 * i 个星号和 2 * i 个空格组成。每行结尾要换行。本例也应使用嵌套的循环实现，外层循环控制行，内层循环控制列。

程序代码如下：

```
/* e3_16.c */
#include<stdio.h>
void main(){
    int i,j,k;
    for(i=1;i<=4;i++){                    /*打印上半部分,共 4 行*/
        for(k=1;k<=8-2*i;k++)
            printf(" ");                  /*输出每行前面的空格*/
        for(j=1;j<=2*i-1;j++)
            printf("*");                  /*输出每行的星号*/
        printf("\n");                     /*输出每行后换行*/
    }
    for(i=1;i<=3;i++){                    /*打印下半部分,共 3 行*/
        for(k=1;k<=2*i;k++)
            printf(" ");                  /*输出每行前面的空格*/
        for(j=1;j<=7-2*i;j++)
            printf("*");                  /*输出每行的星号*/
        printf("\n");                     /*输出每行后换行*/
    }
}
```

程序说明：

内层循环中的 printf("\n")语句是必不可少的，它的作用是在每行结尾换行，否则所有的内容将全部显示在一行上。

【例 3-17】 使用嵌套循环实现打印九九乘法表。

程序代码如下：

```
/* e3_17.c */
#include <stdio.h>
void main(){
    int m,n;
    for(m=1;m<=9;m++){                    /*控制行,共 9 行*/
        for(n=1;n<=m;n++)                 /*在行一定的情况下,循环输出该行的 1～m 列*/
            printf("%d*%d=%d\t",m,n,m*n);
        printf("\n");
    }
}
```

程序运行结果：

```
1*1=1
2*1=2  2*2=4
3*1=3  3*2=6  3*3=9
4*1=4  4*2=8  4*3=12  4*4=16
5*1=5  5*2=10  5*3=15  5*4=20  5*5=25
6*1=6  6*2=12  6*3=18  6*4=24  6*5=30  6*6=36
7*1=7  7*2=14  7*3=21  7*4=28  7*5=35  7*6=42  7*7=49
8*1=8  8*2=16  8*3=24  8*4=32  8*5=40  8*6=48  8*7=56  8*8=64
9*1=9  9*2=18  9*3=27  9*4=36  9*5=45  9*6=54  9*7=63  9*8=72  9*9=81
```

程序说明：

外层循环控制行。内层循环是在行一定的情况下，循环输出该行的 1～m 列。用变量 m 控制行数，同时控制每行的列数。第 m 行输出 m 个表达式，因此内层循环总是通过 n 的变化执行 m 次。如第 3 行输出 3 个，第 4 行输出 4 个，……。

【例 3-18】 百马百担问题。有 100 匹马，驮 100 担货，大马驮 3 担，中马驮 2 担，两匹小马驮 1 担，问大、中、小马各多少匹？

算法分析：这是一个不定方程求解问题。

$$a+b+c=100$$
$$3*a+2*b+c/2=100$$

式中，a、b、c 分别表示大、中、小马。

由题目给出的条件可得到 3 个变量的取值范围：

a：0～33 的整数

b：0～50 的整数

c：0～200 的偶数(两只小马组合才能驮 1 担)

采用穷举法，用 3 层 for 循环来实现。

程序代码如下：

```
/* e3_18.c */
#include <stdio.h>
void main(){
    int a,b,c;
    for(a=0;a<=33;a++)                    /*取所有可能的大马数*/
        for(b=0;b<=50;b++)                /*取所有可能的中马数*/
            for(c=0;c<=200;c+=2)          /*取所有可能的小马数*/
                if((a+b+c==100)&&(3*a+2*b+c/2==100))
                    printf("a=%d,b=%d,c=%d\n",a,b,c);
}
```

程序运行结果：

```
a=2,b=30,c=68
a=5,b=25,c=70
a=8,b=20,c=72
a=11,b=15,c=74
a=14,b=10,c=76
a=17,b=5,c=78
a=20,b=0,c=80
```

程序说明：

(1) 第 3 层循环若改为 for(c＝0;c＜＝200;c＋＋)也是可以的。因两匹小马驮 1 担，使用 c＋＝2 来修订其值使程序更优化。

(2) 若 a 和 b 已知，则 c 的值可以根据 a 和 b 的值求出，因此本例也可以使用两层循环实现，这样程序执行效率更高。循环语句部分可以改写如下：

```
for(a=0;a<=33;a++)                        /*取所有可能的大马数*/
    for(b=0;b<=50;b++){                   /*取所有可能的中马数*/
        c=100-a-b;                        /*小马数根据大马和中马数求出*/
        if((c%2==0)&&(3*a+2*b+c/2==100))
            printf("a=%d,b=%d,c=%d\n",a,b,c);
}
```

3.3.5 几种循环的比较

(1) 3 种循环都可以用来处理同一个问题，一般可以互相代替。

(2) for 循环和 while 循环一样，循环体可能一遍都不执行；do-while 循环的循环体至少执行一遍。

(3) 用 while 和 do-while 循环时，循环变量初始化的操作应在 while 和 do-while 语句之前完成，而 for 语句可以在表达式 1 中实现循环变量的初始化。