第 3 章　程序控制结构

C 语言的程序控制结构主要有顺序结构、选择结构和循环结构三种。通过组合三种程序控制结构，可以解决各种复杂的实际问题，使得 C 语言具有强大的编程能力。本章将重点介绍顺序结构、选择结构和循环结构的基本语法与使用方法。

3.1　顺序结构

C 语言中的顺序结构主要由说明语句、表达式语句、空语句以及复合语句组成。在顺序结构程序中，各语句（或命令）是按照位置的先后顺序执行的，并且每条语句都会被执行到。可用如图 3-1 所示的顺序语句结构表示顺序结构流程图。

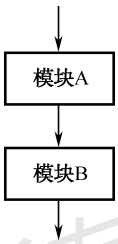

图 3-1　顺序语句结构

顺序结构的程序主体是依次执行具体功能的各条语句，主要包括：① 提供数据的语句。② 运算语句。③ 输出语句。

【例 3-1】 输入两个整数，用两种方法完成两数的交换。

```
//程序一
#include <stdio.h>
int main()
{
    int  a, b, t;
    scanf("%d%d", &a, &b);        //提供数据
    t=a;
    a=b;
    b=t;                          //运算
    printf("%d %d\n", a, b);      //输出
    return 0;
}
//程序二
#include <stdio.h>
int main()
{
    int  a, b;
    scanf("%d%d", &a, &b);
    a=a+b;
    b=a-b;
    a=a-b;
    printf("%d %d\n", a, b);
    return 0;
}
```

运行结果为：

```
100 200
200 100
```

程序一和程序二都利用了三条顺序执行的赋值语句，其中，程序一用变量 t 作为交换中间变量，此法在实际编程中常用；程序二利用加减法运算，实现了交换功能。不管用哪种方法，三条赋值语句的顺序都不可随意改变，否则不能达到目的。

3.2 选择结构

选择结构是实现结构化程序设计的基本成分之一，它所要解决的问题是根据“条件”判断的结果决定程序执行的流向，因此该结构也称为判断结构。程序执行的流向是根据条件表达式的值为 0 或非 0 来决定的。非 0 代表条件为“真”，即条件成立；0 代表条件为“假”，即条件不成立。

使用选择结构，需要考虑两个方面的问题：一是在 C 语言中如何表示条件，二是在 C 语言中用什么语句实现选择结构。在 C 语言中表示条件，可以用任意表达式，但一般用关系表达式或逻辑表达式。实现选择结构用条件（if）语句或分支（switch）语句。下面详细介绍这些语句。

3.2.1 if 语句

1．简单的 if 语句

在简单的 if 语句中，关键词 if 后跟随一个用圆括号括起的表达式，随后是用花括号括起的一条或多条语句。语法格式如下：

```
if(表达式)  语句
```

这里的“表达式”就是决定程序流向的“条件”，当表达式的值为非 0 时执行“语句”，否则不执行。该语句的执行过程如图 3-2 所示。

【例 3-2】 输入任意三个整数 num1、num2 和 num3，求三个数中的最大值。

```
#include <stdio.h>
int main()
{
    int  num1, num2, num3, max;
    printf("Please input three numbers:");
    scanf("%d,%d,%d", &num1, &num2, &num3);
    max=num1;
    if (num2>max)    max=num2;
    if (num3>max)    max=num3;
    printf("The three numbers are:%d,%d,%d\n", num1, num2, num3);
    printf("max=%d\n", max);
return 0;
}
```

运行结果为：

```
Please input three numbers:10,30,20
The three numbers are:10,30,20
max=30
```

2．if-else 语句

if-else 语句的语法格式如下：

```
if(表达式)
    语句1
else
    语句2
```

if 后面的“表达式”，通常是能产生“真”、“假”结果的关系表达式或逻辑表达式，也允许为其他类型的数据，如整型、浮点型、字符型等。它的执行流程是：当 if 后表达式的值为真（非 0）时，执行语句 1，否则执行语句 2。这里的语句 1 和语句 2 可以是简单语句，也可以是复合语句。该语句的执行过程如图 3-3 所示。

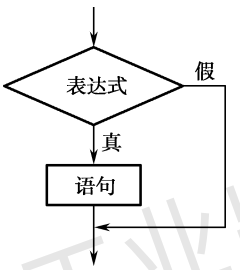

图 3-2　简单的 if 语句

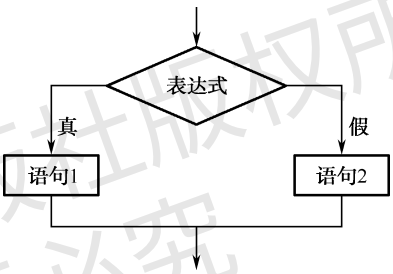

图 3-3　if-else 语句

【例 3-3】 输入任意一个整数，输出该整数的绝对值。

```
#include <stdio.h>
int main()
{
    int  n;
    scanf("%d", &n);
    if(n>=0)
       printf("%d\n", n);
    else
       printf("%d\n", -n);
    return 0;
}
```

运行结果为：

```
-35
35
```

这种形式的 if-else 语句也可以用条件运算符改写，上例中的程序可改写成：

```
#include <stdio.h>
int main()
{
    int  n;
    scanf("%d", &n);
    printf("%d\n", n>=0?n:-n);
    return 0;
}
```

3．带 else if 语句的 if 语句

带 else if 语句的 if 语句是一种多分支选择结构，语法格式如下：

```
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
```

该语句的执行过程如图 3-4 所示，在 n 个条件中，如果满足其中某个条件（表达式的值为非 0），则执行相应的语句，并跳出整个 if 结构执行该结构后面的语句；如果一个条件也不满足，则执行语句 n+1。如果没有语句 n+1，那么最后一个 else 可以省略，此时该 if 结构在 n 个条件都不满足时，将不执行任何操作。同样，这里的语句均可以是用一对{}括起来的复合语句。

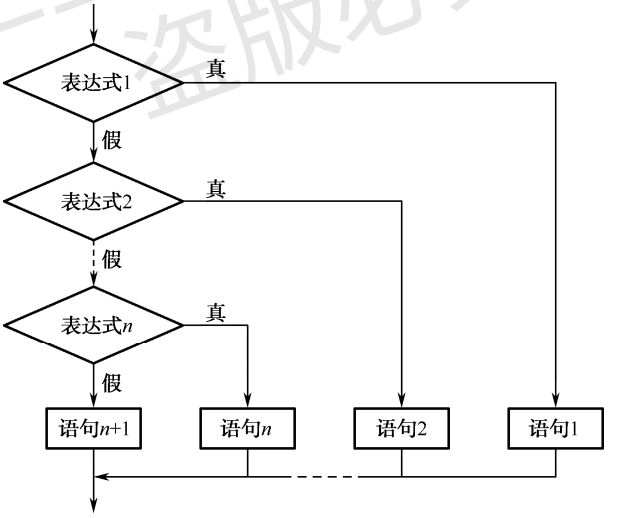

图 3-4　带 else if 语句的 if 语句

【例 3-4】 用带 else if 语句的 if 语句求解一元二次方程 $ax^2+bx+c=0$ 的根，系数 a、b、c 的值从键盘输入。

分析：按照数学上的理论，此一元二次方程的根由三个系数 a、b、c 的值决定，不同的系数组合可能有不同的根，因此程序中要充分考虑各种情况。

程序如下（程序中使用的符号都在英文状态下输入，后续程序都遵循此规范要求）：

```
#include <stdio.h>
#include <math.h>
int main()
{
    float  a, b, c, p, x1, x2, real, imag;
    scanf("%f%f%f", &a, &b, &c);
    if(a==0&&b==0&&c==0)//三者都为0，有无穷解
        printf("Infinite roots!\n");
    else if(a==0&&b==0&&c!=0)//a、b为0，c不为0，无解
        printf("No roots!\n");
    else if(a==0&&b!=0)//a为0，b不为0，无论c值如何，都只有一个根
        printf("Single root:%f\n", -c/b);
    else if(a!=0)
    {
        p=b*b-4*a*c;
        real = -b/(2*a);
        imag = sqrt(fabs(p))/(2*a);
        if(p==0)//b*b-4*a*c==0，有两个相同解，一个实数根
            printf("Single root:%f\n", real);
        else if(p<0)//b*b-4*a*c<0，有两个虚数根
        {
            printf("Complex roots:");
            printf("%f+%fi, %f-%fi\n", real, imag, real, imag);
        }
        else//b*b-4*a*c>0，有两个不同的实数根
        {
            x1 = real+imag;
            x2 = real-imag;
            printf("Real roots:%f and %f\n", x1, x2);
        }
    }
return 0;
}
```

运行结果（程序分别执行5次）：

```
0  0  0
Infinite roots!
1  4  4
Single root:-2.000000
```

```
4  2  1
Complex roots:-0.250000+0.433013i, -0.250000-0.433013i
2  4  1
Real roots:-0.292893 and -1.707107
0  0  2
No roots!
```

关于使用 if 语句，需要说明以下三点。

（1）if 语句可以嵌套使用。例 3-4 中关于一元二次方程求根问题，按数学上的理论，$a\neq0$ 是方程众多求根条件之一，但在此条件下又分两种情况：当 $b^2-4ac\geqslant0$ 时，方程有两个实数根；当 $b^2-4ac<0$ 时，方程有两个虚数根。所以在 else if (a!=0)语句下嵌套使用 if-else 语句。

（2）在 if 结构中，语句可以是简单语句，也可以是复合语句，复合语句一定要加{}以表示 if 或 else if 条件的作用域范围，即当 if 后的条件成立时应该执行的所有语句。例如：

```
if(x>y)  x=y;y=z;z=x;//不适当的写法
```

在该程序段中，当 x>y 时执行“x=y;”语句，然后依次执行“y=z;z=x;”两条语句。当 x<=y 时不执行“x=y;”语句，但后面两条语句仍然要执行，因为它们不在 if 条件的作用域范围内，所以，如果希望满足条件 x<=y 时三条语句都不执行，那么程序应写成：

```
if(x>y) {x=y;y=z;z=x;}//使用复合语句
```

（3）当程序中有众多 if 和 else 时，else 总是跟它上面最近的 if 配对，并且 else 必须和 if 配对使用。分析以下程序（右边是实际编程中利用缩进格式来显示的逻辑结构）：

```
int a=1,b=3,c=5,d=4,x;
if(a<b)
if(c<d)  x=1;
else
if(b<d)  x=2;
else  x=3;
else  x=6;
```

```
if(a<b)
    if(c<d)  x=1;
    else
        if(b<d)  x=2;
        else  x=3;
else  x=6;
```

该程序的运行结果应为 x=2，流程图如图 3-5 所示。

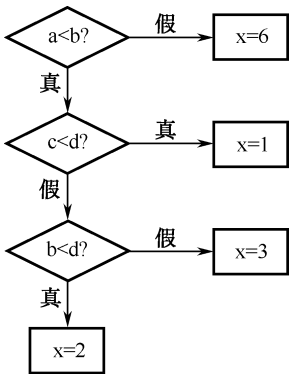

图 3-5　流程图

3.2.2 switch 语句

分支语句即指 switch 语句，也称为开关语句，它也是一种多分支选择结构。switch 是关键字，后面跟一个表达式，这个表达式里包含的某些变量在具体的问题模型里通常可能取不同的常量值。switch 语句能够根据表达式值的不同，使得程序转入不同的模块执行。

switch 语句的一般形式如下：

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

这种结构的语句，在其他高级语言中也称为 case 语句。它的执行流程是：当 switch 表达式的值与某个 case 后面的常量表达式的值相同时，程序就执行这个 case 后面的语句，并接着执行这个 case 后面的后续语句，直到 switch 语句的最后。其执行过程可以用图 3-6 来表示，其中，e 表示 switch 表达式，e1, e2, …, en 分别表示常量表达式 1、常量表达式 2、……、常量表达式 *n*。图 3-6 为不带 break 语句的 switch 语句。通常，程序员并不希望出现这种情况，一般各个 case 分支语句之间是相互排斥的，所以在每组 case 分支语句后可以用 break 语句结尾。break 语句的作用是使程序在执行匹配的 case 分支语句后直接跳出 switch 语句，接着执行 switch 语句后面的语句。

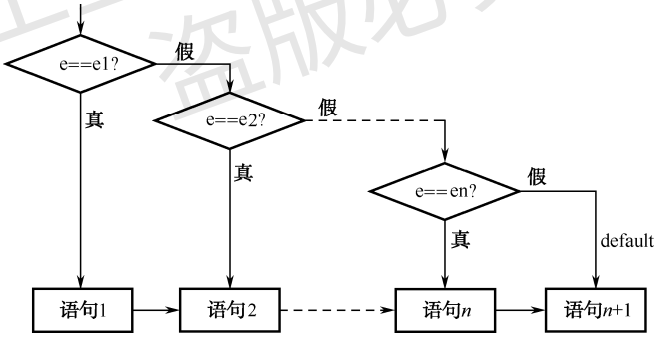

图 3-6　不带 break 语句的 switch 语句

关于 switch 语句的几点说明：

（1）每个 case 后面的常量表达式的值必须互不相同，以免程序执行的流程产生矛盾。

（2）switch 表达式可以是整型表达式、字符表达式等。

（3）多个 case 可以公用一组执行语句，例如：

```
……
case  4:
case  5:
case  6:
case  7:  d=8;
```

表示，当 switch 表达式的值为 4、5、6 或 7 时，都执行同一组语句：“d=8;”。

（4）若要在执行一条 case 分支语句后，使程序执行流程退出 switch 语句，那么可以加入 break 语句。常用的带 break 语句的 switch 语句的语法格式如下：

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

【例 3-5】 编写程序输入一个 5 位或 5 位以下的正整数，逆序输出该数。

分析：正整数对 10 取余后能得到其个位上的数，如 1234%10 等于 4，因此，如果某个数第一次对 10 取余得到个位数后，再除以 10 并对 10 取余，那么将得到原来十位上的数，如 1234/10%10 等于 3，其余类推，可得到整数各位上的数。本例利用无 break 语句的 switch 语句，避免了用带 else if 语句的 if 语句编写程序产生的相对冗长的代码，读者可自行用 if 语句写出程序进行比较。

```
#include <stdio.h>
int main()
{
    int  a,n;
    scanf("%d", &a);
    if(a>0&&a<100000)
    {
        n=a<10?1:a<100?2:a<1000?3:a<10000?4:a<100000?5:0;
        printf("%d digits, inversed number: ", n);
        switch(n)
        {
            case 5:
                printf("%d", a%10);  a=a/10;
            case 4:
                printf("%d", a%10);  a=a/10;
            case 3:
                printf("%d", a%10);  a=a/10;
            case 2:
                printf("%d", a%10);  a=a/10;
            case 1:
                printf("%d\n", a%10);
        }
    }
```

```
    else printf("The number is not a valid num!\n");
    return 0;
}
```

运行结果为：

```
54321
5 digits, inversed number: 12345
```

【例 3-6】 用 switch 语句编写一个可以处理四则运算的程序。

分析：本例使用带 break 语句的 switch 语句。通过对运算符的分析，我们希望对每种运算符的式子执行相匹配的运算，而不执行其他多余的运算，因此每种运算完成后均用 break 语句跳出 switch 语句避免前述情况的发生。

```
#include <stdio.h>
int main()
{
    float  v1,v2;
    char  op;
    printf("Please type your expression: ");
    scanf("%f%c%f", &v1, &op, &v2);
    switch(op)
    {
        case  '+':
            printf("%f+%f=%f\n", v1, v2, v1+v2);  break;
        case  '-':
            printf("%f-%f=%f\n", v1, v2, v1-v2);  break;
        case  '*':
            printf("%f*%f=%f\n", v1, v2, v1*v2);  break;
        case  '/':
            if(v2==0)
                printf("division by zero!\n");
            else
                printf("%f/%f=%f\n", v1, v2, v1/v2);
            break;
        default:
            printf("unknown operator.\n");
    }
    return 0;
}
```

运行结果（程序分别执行 4 次）如下：

```
Please type your expression: 2.1+3↓
2.100000+3.000000=5.100000
Please type your expression: 2.1-3
```

```
2.100000-3.000000=-0.900000
Please type your expression: 2.1*3
2.100000*3.000000=6.300000
Please type your expression: 2.1/3
2.100000/3.000000=0.700000
```

例 3-5 和例 3-6 分别是不带 break 语句和带 break 语句的有关 switch 语句的例子，通过实例可以看出，可以利用 switch 语句不带 break 语句的特点编写一些程序特例。

另外，带 break 语句的 switch 语句跟 else if 语句可以处理相同的问题，它们都是多分支选择结构，都需要逐个判断分支条件，当分支条件为真时执行该条件作用域下的语句块。但它们又有所不同，if 语句针对具体问题划分出具体的条件区间，写成 C 语言条件表达式，而 switch 语句则针对某个具体表达式的值展开讨论，尽可能列出所有可能出现的结果分别进行处理。有些问题既可以用 else if 语句处理，也可以用 switch 语句处理，但在写程序时还是要多考虑，才能写出更简单、实用、正确的程序。举个例子，我们要根据学生的分数区间输出相应的等级，90～100 分的等级为 A，80～89 分的等级为 B，70～79 分的等级为 C，60～69 分的等级为 D，低于 60 分的等级为 F，其他情况提示错误字样。定义学生的分数变量为 grade，用 else if 语句可写成如下程序：

```
if(grade>=0&&grade<=100)
    if(grade>=90)       printf("A");
    else if(grade>=80)  printf("B");
    else if(grade>=70)  printf("C");
    else if(grade>=60)  printf("D");
    else   printf("F");
else
     printf("Error");
```

但这个问题如果要用 switch 语句来解决，switch 表达式显然不能简单地写成 switch (grade)，因为 grade 可能出现的值实在太多了。即便分数只能是 0～100 之间的所有整数，根据 grade 的可能出现的常量值进行判断，也需要 101 条 case 语句才能完成，这样会使程序显得太过冗长。观察上面提到的分数区间会发现，每个分数区间除以 10 后得到的结果就是有限的了，用 switch 语句改写的程序如下：

```
if(grade>=0&&grade<=100)
    switch ((int)grade/10)
    {
       case  10:
       case  9:     printf("A);  break;
       case  8:     printf("B");  break;
       case  7:     printf("C");  break;
       case  6:     printf("D");  break;
       default:     printf("F");
    }
else
     printf("Error");
```

可见，对于类似的情况，我们可以采取某些运算让原来 if 语句中的区间表示方法中的变量得到一个或几个固定值，从而可以运用 switch 语句。

3.3 循环结构

C 语言的循环结构也是结构化程序设计的基本成分之一。循环结构解决的问题是在某个条件下，要求程序重复执行某些语句或某个模块。循环的实现一般包括 4 个部分：初始化，条件控制，重复的操作语句，以及通过改变循环变量值最终改变条件的真假值，使循环能正常结束。循环条件所用的表达式，可以是算术表达式、关系表达式、逻辑表达式或最终能得到非 0 值或 0 值的其他任意表达式。重复执行的语句或模块，称为循环体。

C 语言中，实现循环结构主要有以下三种循环语句：

- while 语句
- do-while 语句
- for 语句

此外，C 语言还提供了两个无条件控制语句：break 语句和 continue 语句，这两条辅助语句一般用来控制程序中的某个循环结构是继续执行还是跳出循环结构。

3.3.1 while 语句

while 语句的语法格式如下：

```
while(表达式){
    循环体;
}
```

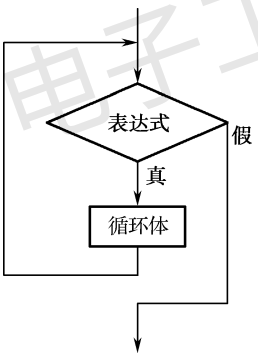

图 3-7 while 语句

当执行 while 语句时，先判断表达式的值，若为非 0（真）值，则执行循环体语句，每执行一次循环体后，都要再判断一下表达式的值，如果仍然是非 0 值，则再一次执行循环体，如此循环，一直到表达式的值为 0（假）时，循环终止，转而执行 while 语句后面的语句。其中，表达式的作用是进行条件判断，通常为关系表达式或逻辑表达式；循环体可以是简单语句也可以是复合语句。while 语句的执行流程如图 3-7 所示。

while 语句先判断条件再执行循环体。因此，while 语句的作用是，当条件成立时，使语句（即循环体）反复执行。为此，在循环体中应该增加对循环变量进行修改的语句，使循环趋于结束，否则将使程序陷入死循环。

【例 3-7】 用 while 语句写一个程序，统计从键盘输入的数字字符出现的次数，并把其中的数字字符依次输出。

```
#include <stdio.h>
int main()
{
    char  c;
    int  ct=0;
    while ((c=getchar())!='\n')
```

```
        {
            if(c>='0'&&c<='9')
            {
                ct++;
                printf("%c ", c);
            }
        }
        printf("\nThere are %d digits!\n", ct);
        return 0;
    }
```

运行结果为：

```
shanghai12345china5678asian
1 2 3 4 5 5 6 7 8
There are 9 digits!
```

若表达式只用来表示等于零或不等于零的关系时，可以简化成如下形式：

```
while (x!=0)    可写成   while (x)
while (x= =0)   可写成   while (!x)
```

表达式里可以嵌套赋值表达式。例如，while((c=getchar())!='\n')表示从键盘读入一个字符并将它赋给变量 c，然后判断 c 是否是回车符，如果是回车符则中断循环，否则继续读字符并对它重新进行判断。

while 语句中的循环体可以为空语句，也可以为简单语句，或者为复合语句。注意，复合语句一定要用一对花括号{}括起来。下面两条 while 语句是不一样的：前者当 x 成立时，语句不循环执行；后者当 x 成立时，语句循环执行。

```
while(x);   //循环体语句为空语句
{//复合语句
    语句;
}
```

与

```
while(x){   //进入循环
    语句;
}
```

在循环体中，循环变量的值可以被使用，但最好不要对循环变量重新赋值，否则程序有可能陷入死循环。

3.3.2 do-while 语句

do-while 语句的语法格式如下：

```
do{
    循环体;
}while(表达式);
```

do-while 语句先执行循环体，然后再判断表达式是否成立，若表达式成立，则继续执行循

环体，接着重新计算表达式中的值并判断真假，直到表达式的值为0（假）时，终止循环。所以，无论循环条件的值如何，至少会执行循环体一次。它的执行流程如图3-8所示。

do-while语句适用于先执行循环体，后判断循环条件的情况。它每执行一次循环体后，再判断条件，以决定是否进行下一个循环。

【例3-8】 编写程序，输入一个5位或5位以下的正整数，逆序输出该数并计算它是几位数。

```
#include <stdio.h>
int main()
{   int  num, n=0;
    printf("Please input the number:");
    scanf("%d", &num);
    printf("Inversed number is: ");
    do
    {
        printf("%d", num%10);
        n++;
    } while(num=num/10);
    printf("\nIt has %d bits.\n", n);
    return 0;
}
```

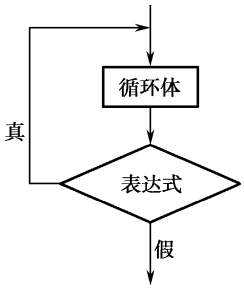

图3-8　do-while语句

运行结果为：

```
Please input the number:3453
Inversed number is: 3543
It has 4 bits.
```

此类问题也可以先逆序构成一个数后再输出（假设所有变量已经正确定义）：

```
m=0;   //m表示num的逆序数
while(num){
    m=m*10+num%10;
    num/=10;
    n++;  //n表示位数
}
```

当然，while和do-while语句可以用来处理同一个问题，但有一点必须注意，while语句可以一次也不执行循环体（循环条件一开始就不满足时），而do-while语句则不同，程序执行到do-while语句时，至少要执行一次循环体后才会去判断循环条件。所以在使用这两种循环语句相互替代时，应考虑它们的异同。

3.3.3　for语句

for语句的语法格式如下：

```
for (表达式1;表达式2;表达式3){
    循环体;
}
```

for 后面括号内的三个表达式用分号隔开，它们的功能分别是：

表达式 1 为初始化表达式，通常用来设定循环变量的初值或者循环体中任何变量的初值，可用逗号作为分隔符设置多个变量的值。

表达式 2 为循环条件表达式。

表达式 3 为增量表达式。执行一次循环体后，要求解一次增量表达式的值，目的是对循环条件表达式产生影响，使得循环条件表达式的值可能产生变化，从而终止循环的执行。表达式 3 也可以写成以逗号分隔的多个表达式，也可以包含一些本来可以放在循环体中执行的其他表达式。

for 语句的执行过程如图 3-9 所示。从流程图中可以看出，首先求表达式 1 的值，而后判断表达式 2 的值是真还是假，如果表达式 2 的值为真，则执行循环体后再去求表达式 3 的值，接着重新判断表达式 2 的值是真还是假，如此循环直到表达式 2 的值为假时，立即终止循环而继续执行循环结构外面的语句。

【例 3-9】 求 1～100 间奇数之和，即求 1+3+5+…+99。

```
#include "stdio.h"
int main()
{
    int sum=0,i;
    for(i=1;i<=100;i++)
      if(i%2!=0)
          sum+=i;
    printf("sum=%d\n", sum);
    return 0;
}
```

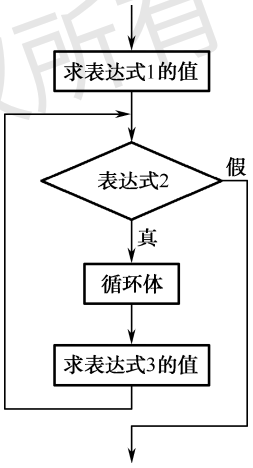

图 3-9　for 语句

运行结果为：

```
sum=2500
```

从 for 语句的流程图可以看出，for 语句中表达式 1 的初始化工作是在执行循环之前完成的，因此可以写在 for 的前面；而表达式 3 即增量表达式在每次执行完循环体后再求值，因此可以写到循环体后面，即 for 语句可以写成如下形式：

```
表达式1
for(;表达式2;)
{
    语句                    //原循环体
    表达式3
}
```

因此，求解 1～100 间奇数之和的算法可以转换成：

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

```
    int sum=0;
    int i=1;
    for(;i<=100;)
    {
       if(i%2!=0)
          sum+=i;
       i++;
    }
    printf("sum=%d\n", sum);
    return 0;
}
```

虽然 while 语句与 for 语句可以完全相互替换，但 while 语句更倾向于用来表示循环次数未知的情况，而 for 语句更倾向于用来实现带步长的循环，故也称为计数循环。

另外，for 语句的表达式 1 和表达式 3 可以是逗号表达式，用来设置多个变量的值或求解多个增量表达式，例如：

```
int i, j, s1=0, s2=0, n;
for(i=1, j=2, n=0; n<=20; n++, i+=2, j+=2)
{
    s1+=i;
    s2+=j;
}
```

该程序用来分别计算正整数的前 20 个偶数之和以及前 20 个奇数之和。

3.3.4 break 语句与 continue 语句

break 语句由关键字 break 后加分号（;）组成。在这里，break 语句被用在循环结构中，其作用是跳出它所在的循环体，提前结束循环，使程序的执行流程无条件地转移到循环结构的下一条语句继续执行。例如，for(;;)语句中表达式 2 位置上为空，它表示条件永远为真，所以在循环体中添加一句 if 语句使得条件满足时执行 break 语句，让程序能够在适当的时候结束循环。需要说明的是，虽然 for 语句中的表达式都是可以省略的，但为了保持语法结构的完整性，分号不能省。

continue 语句由 continue 后面加分号（;）构成，它的作用是结束本次循环，使程序回到循环条件处，判断是否提前进入下一次循环。注意，continue 语句用在 for 语句中将转去执行表达式 3。

需要注意 break 语句与 continue 语句之间的区别，continue 语句只结束本次循环，而不是终止整个循环的执行；而 break 语句则是结束循环，不再进行条件判断。有以下两种循环结构：

```
(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;
}
```

在循环结构（1）中，如果表达式 2 成立，则执行 break 语句，并且不再判断表达式 1 是否成立，直接跳出 while 循环。在循环结构（2）中，如果表达式 2 成立，则执行 continue 语句，意味着当前循环中的语句 2 被跳过，需要重新判断表达式 1 是否成立，以决定循环是否继续。

3.3.5 循环结构的嵌套

在一个循环体内又包含另一个或多个完整的循环结构，称为嵌套循环。例如，for 语句的二层嵌套循环结构语法如下：

```
for(i=0; i<n; i++)
    for(j=0; j<m; j++)
      语句;
```

在这个程序段中，外层循环一共循环 *n* 次，内层循环则循环 *m*×*n* 次。内层循环体的增量总是比外层循环体的增量变化得“快”一些。

再如，用 for 语句嵌套的三层循环结构语法如下：

```
for(i=0; i<n; i++)
    for(j=0; j<m; j++)
        for(k=0; k<l; k++)
           语句;
```

在这个程序段中，外层循环一共循环 *n* 次，中间层内循环则循环 *m*×*n* 次，最内层循环则循环 *m*×*n*×l 次。

【例 3-10】 输入 *n*，计算 *n*×*n* 的乘法表。

```
#include <stdio.h>
int main(){
    int i,j,n;
    scanf("%d",&n);
    for(i=1;i<=n;i++){
        for(j=1;j<=n;j++)
            printf("%4d",i*j);     //计算 i*j 的值
        printf("\n");              //回车换行
    }
    return 0;
}
```

运行结果为：

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

分析：这是两重循环，i 为外层循环变量，j 为内层循环变量。首先，i 固定在一个数值上，

然后执行内层循环，j 变化一个轮次。i+1 后，重新执行内层循环，j 再变化一个轮次。因此，内、外层循环变量不能相同，本程序中分别用 i 和 j。语句“printf("%4d",i*j);”一共执行了 $n \times n$ 次。但如果修改两重循环如下：

```
for(i=1;i<=n;i++){
    for(j=1;j<=i;j++)
       printf("%4d",i*j);  //计算 i*j 的值
    printf("\n");        //回车换行
}
```

语句“printf("%4d",i*j);”将一共执行 $1+2+\cdots+n$ 次，主要是内层循环 j<=i 条件起到了限制作用。读者可自行修改程序并查看运行结果。

【例 3-11】 计算 1!+2!+3!+…+100!的值，要求使用嵌套循环。

```
#include <stdio.h>
int main(){
    int i,j;
    double item, sum;
    sum=0;
    for(i=1;i<=100;i++){
        item=1;
        for(j=1;j<=i;j++)
            item=item*j;
        sum=sum+item;
    }
    printf("1!+2!+3!+…+100!=%e\n",sum);
    return 0;
}
```

运行结果为：

```
1!+2!+3!+…+100!=9.426900e+157
```

在累加求和的外层 for 语句循环体中，每次计算 i 的阶乘之前，都需要重置 item 的初值为 1，以保证每次计算阶乘都从 1 开始连乘。

但是，如果把程序中的嵌套循环写成如下形式：

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

由于将 item=1 放在外层循环之前，除计算 1!时 item 从 1 开始外，计算其他阶乘都是用原 item 值乘以新的阶乘值。这样，最后求出的累加和是：$1! + 1!\times 2! + 1!\times 2!\times 3! + \cdots + 1!\times 2!\times\cdots\times 100!$。出错的原因是循环初始化语句被放错了位置，混淆了外层循环变量和内层循环变量的初始化。

在实际编程时，使用多重循环结构注意事项如下。

（1）内层循环必须完整地嵌套在外层循环内，两者不允许相互交叉。例如：

```
for(i=0; i<10; i++)
    for(j=0; j<5; j++)
        printf("i=%d,j=%d \n", i, j);
```

是正确的。而下面代码则是错误的：

```
i=0;
while(i<10)
{
    j=0;
    while(j<5)
    {
        printf("i=%d,j=%d \n", i, j);
        i++;  //这里错误，应为 j++
    }
    j++;   //这里错误,应为 i++
}
```

（2）并列的循环变量可以同名，但嵌套循环变量不允许同名。例如：

```
for(i=0;i<10;i++)
{
    for(j=0;j<5;j++)
    //与下面的 for 语句里的循环变量同名但与上一层循环变量不同名
        printf("i=%d,j=%d\n", i, j);
    for(j=10; j<15; j++)
        printf("i=%d,j=%d\n", i, j);
}
```

（3）三种循环语句可以相互嵌套，但不允许交叉。例如：

```
for (i=0;i<10;i++)   //for 语句和 while 语句相互嵌套
{
    j=0;
    while(j<5)
    {
        printf("i=%d,j=%d\n", i, j);
        j++;
    }
}
```

（4）选择结构和循环结构彼此之间可以相互嵌套。例如：

```
for(i=1;i<=5;i++)
{
    switch (i)
```

```
    {
        case  1: printf("*"); break;
        case  2:
        case  3: printf("***"); break;
        case  4:
        default: printf("*****")
    }
}
```

（5）可以用 break 语句从内层循环跳转到外层循环，但不允许从外层循环跳转到内层循环。例如：

```
for(i=1; i<=10; ++)
{
    for(j=1, s=0; j<=100; j++)
    {
        s=s+j+i;
        if(s>200)   break;  //从内层循环跳转到外层循环
    }
    printf("i=%d,j=%d,s=%d\n", i, j, s);
}
```

（6）continue 语句处于多重循环中时，仅仅影响包含它的循环语句。例如：

```
for(i=1; i<10; i++)
{
    for(j=1, s=0; j<100; j++)
    {
        if(j%2==0)  continue;   //只影响 j 所在的循环语句
        s=s+j+i;
    }
    printf("i=%d,j=%d,s=%d\n", i, j, s);
}
```

3.3.6 典型例题

【例 3-12】 水仙花数。输入两个正整数 m 和 n（$m \geqslant 1$，$n \leqslant 1000$），输出 m～n 之间的所有水仙花数。水仙花数是指各位上数字的立方和等于其自身的数。例如，153 的各位上数字的立方和是 $1^3+5^3+3^3=153$。

```
#include <stdio.h>
int main()
{
    int i,t,s,m,n,digit;
    printf("Input m: ");
    scanf("%d",&m);
```

```
    printf("Input n: ");
    scanf("%d",&n);
    for (i=m;i<=n;i++)
    {
        t=i;
        s=0;
        while(t!=0)
        {
            digit=t%10;
            s+=digit*digit*digit;
            t=t/10;
        }
        if (s==i)
            printf("%d\n",i);
    }
    return 0;
}
```

运行结果为：

```
Input m: 1
Input n: 999
1
153
370
371
407
```


【例 3-13】 验证哥德巴赫猜想：任何一个大于 2 的偶数均可表示为两个素数之和。例如，4=2+2，6=3+3，8=3+5，…。要求：将 6～100 之间的偶数都表示为两个素数之和，输出时一行 5 组。若有多组结果满足条件，则输出第一个被加素数最小的情况，例如，14=3+11 和 14=7+7，输出前一种情况。

```
#include <stdio.h>
#include <math.h>
int main()
{
    int i,k,m,n,flagm,flagn,count;
    count=0;
    for (i=3;i<=50;i++)
    {
        m=1;
        do
        {
            m=m+1;
            n=2*i-m;
```

```
            flagm=1;
            flagn=1;
            for(k=2;k<=(int)(sqrt(m));k++)
            {
                if(m%k==0)
                {
                    flagm=0;
                    break;
                }
            }
            for(k=2;k<=(int)(sqrt(n));k++)
            {
                if(n%k==0)
                {
                    flagn=0;
                    break;
                }
            }
        }while(flagm*flagn==0);
        count++;
        printf("%4d=%2d+%2d",2*i,m,n);
        if(count%5==0)
            printf("\n");
    }
    printf("\n");
    return 0;
}
```

运行结果为：

```
  6= 3+ 3   8= 3+ 5  10= 3+ 7  12= 5+ 7  14= 3+11
 16= 3+13  18= 5+13  20= 3+17  22= 3+19  24= 5+19
 26= 3+23  28= 5+23  30= 7+23  32= 3+29  34= 3+31
 36= 5+31  38= 7+31  40= 3+37  42= 5+37  44= 3+41
 46= 3+43  48= 5+43  50= 3+47  52= 5+47  54= 7+47
 56= 3+53  58= 5+53  60= 7+53  62= 3+59  64= 3+61
 66= 5+61  68= 7+61  70= 3+67  72= 5+67  74= 3+71
 76= 3+73  78= 5+73  80= 7+73  82= 3+79  84= 5+79
 86= 3+83  88= 5+83  90= 7+83  92= 3+89  94= 5+89
 96= 7+89  98=19+79 100= 3+97
```

【例 3-14】 自守数，也称同构数，是指一个数的平方的尾数等于该数自身的自然数。例如，5、25 和 76 是自守数，因为 5×5=25，25×25=625，76×76=5776，其平方后的尾数等于自身。任意输入一个自然数，判断是否为自守数并输出 Yes 或 No。

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

```
{
    int i,n;
    printf("输入一个整数:");
    scanf("%d",&n);
    i=1;
    while(i<=n) i*=10;
    if(n*n%i==n)
        printf("Yes\n");
    else
        printf("No\n");
    return 0;
}
```

运行结果为：

```
输入一个整数:76
Yes
```

【例 3-15】 编写程序判断整数 *m* 是否为素数。

分析：素数也就是质数，它的特点是，除 1 和它本身外，不能被其他任何数整除。在本例中，如果 *m* 是素数，那么在[2,*m*−1]区间中的所有整数都不能被 *m* 整除。只要有一个数能被它整除，其他的数不必再验证，就可以确定 *m* 一定不是素数。

```
#include <stdio.h>
int main()
{
    int  m, i;
    scanf("%d",&m);
    for (i=2;i<m;i++)
        if(m%i==0) break;     //i 能否整除 m
    if(i==m)                  //表明上面的循环已结束，没有一个数能被 m 整除
        printf("%d is a prime number.\n", m);
    else
        printf("%d is not a prime number.\n", m);
    return 0;
}
```

运行结果为：

```
1999
1999 is a prime number.
```

【例 3-16】 最大公约数与最小公倍数。输入两个数 *n* 和 *m*，计算 *n* 和 *m* 的最大公约数与最小公倍数。

分析：该算法可采用数学中的“辗转相除法”。例如，求 224 和 35 的最大公约数，把 224 作为被除数，35 作为除数，得余数 14，把上次的除数 35 作为本次的被除数，上次的余数 14 作为本次的除数，得余数 7，重复上述步骤，被除数为 14，除数为 7，余数为 0。当余数为 0

时，结束计算，最后一次的除数 7 便是要求的最大公约数。求最小公倍数的数学公式如下：

最小公倍数=两数乘积÷最大公约数

程序如下：

```
#include <stdio.h>
int main()
{
    int  a, b, num1, num2, temp;
    printf("Please input two integer numbers:\n");
    scanf("%d%d", &num1, &num2);
    while (num1<1||num2<1)
    {
        printf("Input isn't correct. Please input again:\n");
        scanf("%d%d", &num1, &num2);
    }
    a=num1; b=num2;
    while (b!=0)
    {
        temp=a%b;
        a=b;
        b=temp;
    }
    printf("The largest common divisor:%d\n", a);
    printf("The least common multiple:%d\n", num1*num2/a);
    return 0;
}
```

运行结果为：

```
Please input two integer numbers: 10 28↓
The largest common divisor:2
The least common multiple:140
```

求解这个问题还有一种更简单的方法（穷举法）：假设这两个整数用 *a* 和 *b* 表示，*m* 表示 *a*、*b* 中较小的数，那么 *a* 和 *b* 的最大公约数一定是 1～*m* 之间的某个整数，从 *m* 开始以步长为 1 递减，看 *m* 能否同时被 *a* 和 *b* 整除。若能整除，则 *m* 一定是它们的最大公约数。程序如下：

```
#include <stdio.h>
int main()
{
    int  a, b, m;
    scanf("%d%d", &a, &b);
    while (a<1||b<1)
    {
        printf("Input isn't correct. Please input again:\n");
```

```
        scanf("%d%d", &a, &b);
    }
    for (m=(a<b?a:b);  ; m--)
        if (a%m==0&&b%m==0)   break;
    printf("Largest common divisor:%d\n", m);
    printf("Least common multiple:%d\n", a*b/m);
    return 0;
}
```

【例 3-17】 输出以下图形（输入行数为 4）：

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

程序如下：

```
//图形打印
#include <stdio.h>
int main(){
    int i,j;
    int n;
    scanf("%d",&n);    //输入行数 n
    for(i=1;i<=n;i++){
        for (j=1;j<=n-i;j++)
            printf(" ");
        for (j=1;j<=2*i-1;j++)
            printf("*");
        printf("\n");
    }
    for(i=1;i<n;i++){
        for (j=1;j<=i;j++)
            printf(" ");
        for (j=1;j<=2*(n-i)-1;j++)
            printf("*");
        printf("\n");
    }
    return 0;
}
```

【例 3-18】 按下述形式输出九九乘法表。

```
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
```

分析：该九九表为一个 9 行 9 列呈阶梯状的图表。按行观察，每行的两个乘数中的第二个数字均相同；按列观察，每列的两个乘数中的第二个数字均相同。并且，每行最末一列两个数字相同，也就是说，每行的数学等式的个数与行数是相等的，即第 1 行有 1 个式子，第 2 行有 2 个式子，……，第 9 行有 9 个式子。由于计算机通常是按行输出的，因此，可以利用双重循环来写程序：外层循环控制行数，共 9 行；内层循环控制列数，由当前的行数决定列数的变化。

程序如下：

```
#include <stdio.h>
int main()
{
    int  i, j;
    for (i=1; i<=9; i++)
    {
        for (j=1; j<=i; j++)
            printf("%d*%d=%-4d", i, j, i*j);
        printf("\n");
    }
    return 0;
}
```

3.4 其他常用解题方法

3.4.1 顺推法

【例 3-19】 累加。输入 n 个整数，计算平均值。

```
#include <stdio.h>
int main()
{
    int n,i,number;
    double avg,sum=0;
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
```

```
        scanf("%d",&number);
        sum+=number;
    }
    avg=sum/n;
    printf("%lf",avg);
    return 0;
}
```

运行结果为：

```
5
12 42 36 31 8
25.800000
```

【例 3-20】 累乘。计算 $x=1+\frac{1}{1!\times 2!}+\frac{1}{1!\times 2!\times 3!}+\cdots+\frac{1}{1!\times 2!\times 3!\times\cdots\times 100!}$ 的值。

```
#include <stdio.h>
int main(){
    int i,j;
    double item, sum;
    sum=0;
    item=1;
    for(i=1;i<=100;i++){
        for(j=1;j<=i;j++)
            item=item*j;
        sum=sum+1/item;
    }
    printf("sum=%lf\n",sum);
    return 0;
}
```

运行结果为：

```
sum=1.586835
```

3.4.2 逆推法

【例 3-21】 猴子吃桃。猴子第一天摘下若干个桃子，当即吃了一半，还不过瘾，又多吃了一个。第二天早上将剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃掉前一天剩下的一半再加一个。到第 n 天时，只剩下一个桃子了。问第一天共摘了多少桃子？

```
#include <stdio.h>
int main()
{
    int i, peach,n;
```

```
    peach =1;
    scanf("%d",&n);//输入天数
    for(i=1;i<n;i++)
    {
        peach=(peach+1)*2;
    }
    printf("%d\n",peach);
    return 0;
}
```

运行结果为：

```
6
94
```

3.4.3 迭代法

【例 3-22】 上楼梯的走法。楼梯有 n 个台阶，上楼可以 1 步上 1 个台阶，也可以 1 步上 2 个台阶。编程计算共有多少种不同的走法。

分析：设 n 个台阶的走法数为 $f(n)$种。如果只有 1 个台阶，则走法有 1 种（1 步上 1 个台阶），即 $f(1)=1$。如果有 2 个台阶，则走法有 2 种（一种是上 1 个台阶，再上 1 个台阶，另一种是 1 步上 2 个台阶），即 $f(2)=2$。当有 n 个台阶（$n>3$）时，我们缩小问题规模，可以这样想：最后是一步上 1 个台阶的话，之前上了 $n-1$ 个台阶，走法为 $f(n-1)$种，而最后是一步上 2 个台阶的话，之前上了 $n-2$ 个台阶，走法为 $f(n-2)$种，故 $f(n)=f(n-1)+f(n-2)$，属迭代法。

程序如下：

```
#include <stdio.h>
int main()
{
    int i,n,n1,n2,n3;//其中 n1 和 n2 分别代表 f(n-1)种走法和 f(n-2)种走法
    n1=1;
    n2=2;
    printf("Input n=");
    scanf("%d",&n);
    if(n==1)
    {
        printf("%d plans",n1);
        return 0;
    }
    else if(n==2)
    {
        printf("%d plans",n2);
        return 0;
    }
```

```
        else
        {
            for(i=3; i<=n; i++)
            {
               n3=n1+n2;
               n1=n2;
               n2=n3;
            }
            printf("%d plans",n3);
        }
        return 0;
    }
```

运行结果为：

```
Input n=15
987 plans
```

【例 3-23】 用牛顿迭代法求方程 $2x^3-4x^2+3x-6=0$ 在 1.5 附近的根。

```
#include<stdio.h>
#include<math.h>
int main()
{
    float x1,x,f1,f2;
    static int count=0;
    x1=1.5//定义初值
    do
    {
        x=x1;
        f1=x*(2*x*x-4*x+3)-6;
        f2=6*x*x-8*x+3;//对函数 f1 求导
        x1=x-f1/f2;
        count++;
    }while(fabs(x1-x)>1e-5);
    printf("%8.7f\n",x1);
    printf("%d\n",count);
    return 0;
}
```

运行结果为：

```
2.0000000
5
```

表示根为 2，迭代 5 次。

【例 3-24】 用二分法求方程 $x^3+4x^2-10=0$ 的根。

分析：观察方程，设两个变量 x_1 和 x_2，使代数式 $f(x_1)$与 $f(x_2)$的值符号相反（本题中可取 1 和 4），则方程 $f(x)=0$ 在$[x_1,x_2]$区间内肯定有根；若 $f(x)$在$[x_1,x_2]$区间内单调，则至少有一个实根；重新设一个变量 x，取 $x=(x_1+x_2)/2$，并在 x_1 和 x_2 中舍去与 $f(x)$同号者，那么解就在 x 和另外那个没有舍去的点组成的区间内；如此反复取舍，直到 x_n 与 x_{n-1} 非常接近时，那么 x 便是方程 $f(x)$的近似根。效果如图 3-10 所示。

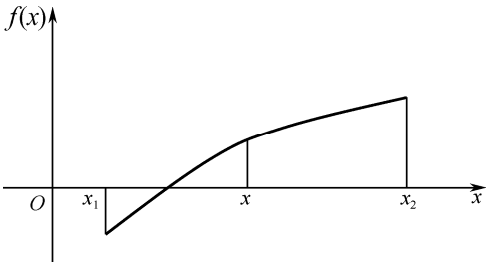

图 3-10　用二分法求方程的根

算法描述如下：

```
while (误差>给定误差)
    if (f(x) == 0)
        x 就是根，不再迭代;
    else if (f(x)*f(x1)<0)
        x2=x;
    else
        x1=x;
```

程序如下：

```
#include <stdio.h>
#include <math.h>
int main()
{
    float  x1=1.0, x2=4.0, x, f, f1;
    f1=x1*x1*x1+4*x1*x1-10;
    while(fabs(x2-x1)>1e-6)
    {
        x=(x1+x2)/2;
        f=x*x*x+4*x*x-10;
        if (f1*f<0)
            x2=x;
        else
            x1=x;
    }
```

```
    printf("The root is: %f\n",x);
    return 0;
}
```

运行结果为：

```
The root is: 1.365230
```

【例 3-25】 Fibonacci 数列。有一对小兔子，从出生后第 3 个月起每个月都生一对小兔子。小兔子长到第 3 个月后每个月又生一对小兔子。按此规律，假设没有兔子死亡，第 1 个月有一对刚出生的小兔子，问第 n 个月有多少对兔子？

分析：第 1 个月和第 2 个月小兔子没有繁殖能力，所以还是一对。第 3 个月后，生下一对小兔子，共有两对。三个月以后，老兔子又生下一对，因为小兔子还没有繁殖能力，所以一共是三对。可以看出来，这是一个 Fibonacci 数列：1, 1, 2, 3, 5, 8…。Fibonacci 数列是数学上的一个“递推”的例子，分析参考例 1-5。

本例设 Fibonacci 数列由 $x_1, x_2, \cdots, x_n$ 组成，由它的定义可知：

$$x_1=1；\ x_2=1；\ x_3=x_1+x_2；\ x_4=x_2+x_3；\ \cdots；\ x_n=x_{n-2}+x_{n-1}$$

为此，我们定义三个整型变量：x1,x2,x3。令第 1 个数 x1=1，第 2 个数 x2=1，则第 3 个数 x3=x1+x2；之后，不再需要第 1 个数，令 x1=x2，x2=x3，求出第 4 个数 x3=x1+x2……如此循环，即可求出 Fibonacci 数列各项的值。

程序如下：

```
#include <stdio.h>
int main()
{
    long int  x1, x2, x3;
    int  i;
    x1=x2=1;
    printf("%6ld %6ld", x1, x2);
    for (i=3; i<=20; i++)
    {
        x3=x1+x2;
        printf("%6ld", x3);
        if (i%5==0)  printf("\n");
        x1=x2;
        x2=x3;
    }
    return 0;
}
```

运行结果为：

```
    1       1       2       3       5
    8      13      21      34      55
   89     144     233     377     610
  987    1597    2584    4181    6765
```

3.4.4 穷举法

【例 3-26】 求解例 1-4 百钱买百鸡问题。

分析：具体分析见例 1-4。

程序如下：

```
#include <stdio.h>
int main()
{
    int  x, y, z;
    for (x=0;x<=100;x++)
        for (y=0;y<=100;y++)
        {
            z=100-x-y;
            if (15*x+9*y+z==300)
                printf("x=%-5d y=%-5d z=%-5d\n", x, y, z);
        }
    return 0;
}
```

运行结果为：

```
x=0     y=25    z=75
x=4     y=18    z=78
x=8     y=11    z=81
x=12    y=4     z=84
```

本例中，内层循环体执行次数为 101×101=10201。但是，由于公鸡 5 钱一只，百钱买百鸡，因此 x 不可能超过 20，因为若用 100 钱买了 20 只公鸡后就不能买别的鸡了，鸡的总数不对；同理，y 不可能超过 33。所以本例中的两个 for 循环条件：x<=100 可改成 x<20，y<=100 可改成 y<=33。这样内层循环体执行次数为 20×34=680，程序效率大大提高。所以在设计程序时，对所用的参数值要仔细考虑，以提高程序效率。另外，本例 if 语句中的条件表达式改进了方程 $5x+3y+z/3=100$，避免了鸡的只数为小数的情况。

【例 3-27】 渔夫分鱼。5 个渔夫（A,B,C,D,E）夜间合伙捕鱼，凌晨都疲惫不堪，各自在草丛中熟睡。第二天清晨：

A 先醒来，他把鱼均分为 5 份，把多余的一条扔回湖中，便拿了自己的一份回家；

B 醒来后，他不知道 A 已经分完鱼了，就又把鱼均分为 5 份，把多余的一条扔回湖中，便拿了自己的一份回家；

C,D,E 也按同样方法分鱼。问 5 人至少捕到多少条鱼？

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i,j,sum;
    for(i=6;;i++)
    {
        sum=i;
        for(j=0;j<5;j++)
        {
            if(sum%5==1&&sum>1)
                sum=(sum-(sum/5))-1;
            else
                break;
        }
        if(sum%4==0&&j==5)
        {
            printf("至少要捕%d\n",i);
            break;
        }
    }
    return 0;
}
```

运行结果为：

```
至少要捕3121
```

【例 3-28】 比赛分组（穷举法）。两个乒乓球队进行比赛，各出三人。甲队为 a, b, c 三人，乙队为 x, y, z 三人。已抽签决定比赛名单。有人向队员打听比赛的名单。a 说他不和 x 比，c 说他不和 x, z 比，请编写程序找出比赛名单。

```
#include <stdio.h>
int main()
{
   char i,j,k;//i 是 a 的对手，j 是 b 的对手，k 是 c 的对手*/
   for(i='x';i<='z';i++)
      for(j='x';j<='z';j++){
         if(i!=j)
            for(k='x';k<='z';k++){
               if(i!=k&&j!=k){
                  if(i!='x'&&k!='x'&&k!='z')
                     printf("order is a--%c\tb--%c\tc--%c\n",i,j,k);
               }
```

```
            }
        }
    return 0;
}
```

运行结果为：

```
order is a--z   b--x    c--y
```

3.5 小结

1）三种程序结构：顺序、分支、循环的不同语法。

2）使用条件语句来实现选择，它们根据条件判断的结果选择所要执行的程序分支，其中条件可以用表达式来描述，如关系表达式和逻辑表达式。

3）要构成一个有效的循环，应当指定两项内容：需要重复执行的操作和循环结束条件。

4）注意 while 和 do-while 语句是不同的。如果循环体中有多于一条语句，应当把循环体中的多条语句用{}括起来，形成复合语句，否则系统认为循环体中只有一条简单的语句。

5）合理使用 break 和 continue 语句处理多循环条件。break 语句将结束整个循环过程，不再判断执行循环的条件是否成立。而 continue 语句只结束本次循环，而不是终止整个循环的执行。

6）循环可以嵌套。在一个循环体中可以包含另外一个完整的循环结构。三种循环语句可以相互嵌套，即任一条循环语句可以成为任一个循环结构中循环体的一部分。

综合练习题

1．输出下一秒。

【问题描述】编写一个程序，输出当前时间的下一秒。

【输入形式】用户在第一行按照“小时:分钟:秒”的格式输入一个时间。

【输出形式】程序在下一行输出这个时间的下一秒。

【样例输入】

```
23:59:59
```

【样例输出】

```
00:00:00
```

【样例说明】用户按照格式要求输入时间，程序输出此时间的下一秒，输出时每个数字占两位，高位补 0。

2．时钟指针角度。

【问题描述】

普通时钟都有时针和分针。在任意时刻，时针和分针都有一个夹角，并且假设时针和分针都是连续移动的。现已知当前的时刻，试求出在该时刻时针和分针的夹角 A（$0° \leqslant A \leqslant 180°$）。

注意：当分针处于 0 分和 59 分之间时，时针相对于该小时的起始位置也有一个偏移角度。

【输入形式】

从标准输入读取一行，是一个 24 小时制的时间。格式是以冒号（:）分隔的两个整数 m（$0 \leqslant m \leqslant 23$）和 n（$0 \leqslant n \leqslant 59$），其中，$m$ 表示时，n 表示分。

【输出形式】

向标准输出打印结果。输出一个浮点数，是时针和分针夹角的角度值。该浮点数保留三位小数。

【样例输入】

```
8:10
```

【样例输出】

```
175.000
```

【样例说明】8:10 这个时刻时针与秒针的夹角是 175.000° 。

3．球的反弹高度。

【问题描述】

已知一球从高空落下时，每次落地后反弹至原高度的四分之一再落下。编写程序，从键盘输入整数 *n* 和 *m*，求该球从 *n* 米的高空落下后，第 *m* 次落地时经过的全部路程以及第 *m* 次落地后反弹的高度，并输出结果。

【输入形式】

从键盘输入整数 *n* 和 *m*，以空格隔开。

【输出形式】

输出两行：

第一行输出总路程，结果保留两位小数。

第二行输出第 *m* 次落地后反弹的高度，结果保留两位小数。

【样例输入】

```
40 3
```

【样例输出】

```
65.00
0.63
```

4．统计字符个数。

【问题描述】

输入 10 个字符，统计其中英文字母、空格或回车符、数字字符和其他字符的个数。

【输入形式】

从键盘输入 10 个字符。

【输出形式】

各字符个数。

【样例输入】（下画线部分表示输入）

```
Input 10 characters: Shuer 123?
```

【样例输出】

```
letter=5,blank=1,digit=3,other=1
```

【样例说明】

输入提示符后要加一个空格。例如，“Input 10 characters: ”中的“:”后要加一个且只能有一个空格。

输出语句的“=”两边无空格。

英文字母区分大小写。必须严格按样例输出格式打印。

5．最大公约数和最小公倍数。

【问题描述】

输入两个正整数 a 和 b（$0 \leqslant a,b \leqslant 1000000$），求其最大公约数和最小公倍数并输出。

【输入形式】从标准输入读取一行，是两个整数 a 和 b，以空格分隔

【输出形式】向标准输出打印以空格分隔的两个整数，分别是 a、b 的最大公约数和最小公倍数。在输出末尾要有一个回车符。

【样例输入】

```
12 18
```

【样例输出】

```
6 36
```

【样例说明】12 和 18 的最大公约数是 6，最小公倍数是 36。

6．换钱的交易。

【问题描述】

一个百万富翁碰到一个陌生人，陌生人找他谈了一个换钱的计划。该计划如下：我每天给你 10 万元，而你第一天给我 1 分钱，第二天我仍给你 10 万元，你给我 2 分钱，第三天我仍给你 10 万元，你给我 4 分钱。你每天给我的钱是前一天的两倍，直到满 n（$0 \leqslant n \leqslant 30$）天。百万富翁非常高兴，欣然接受了这个契约。编写一个程序，计算这 n 天中，陌生人给了富翁多少钱，富翁给了陌生人多少钱。

【输入形式】输入天数 n（$0 \leqslant n \leqslant 30$）。

【输出形式】控制台输出。分行给出这 n 天中，陌生人所付出的钱和富翁所付出的钱。输出舍弃小数部分，取整。

【样例输入】

```
30
```

【样例输出】

```
3000000
1073741823
```

【样例说明】两人交易了 30 天，陌生人给了富翁 3000000 元，富翁给了陌生人 1073741823 元。

7．兑换钱币。

【问题描述】

对于给定的人民币金额 n（单位为分），问有多少种方案将其兑换成 1 分、2 分、5 分的组合。

【输入形式】

输入数据有若干行。每行中有一个正整数表示以分为单位的人民币金额 n，对应一种情形。

【输出形式】

对于每种情形，先输出“Case #:”（#为序号，从 1 起），然后依次输出 n，逗号，结果，最后换行。

【样例输入】

```
10
100
150
```

【样例输出】

```
Case 1: 10, 10
Case 2: 100, 541
Case 3: 150, 1186
```

8．sin*x* 计算公式。

【问题描述】

已知 sin*x* 的近似计算公式如下：

$$\sin x = x - x^3/3! + x^5/5! - x^7/7! + \cdots + (-1)^{n-1}x^{2n-1}/(2n-1)!$$

式中，*x* 为弧度，*n* 为正整数。编写程序根据用户输入的 *x* 和 *n*，利用上述公式计算 sin*x* 的近似值。结果保留 8 位小数。

【输入形式】

从控制台输入小数 *x*（0≤*x*≤20）和整数 *n*（1≤*n*≤5000），两数中间用空格分隔。

【输出形式】

从控制台输出计算结果，保留 8 位小数。

【样例输入 1】

```
0.5236 4
```

【样例输出 1】

```
0.50000105
```

【样例输入 2】

```
0.5236 50
```

【样例输出 2】

```
0.50000106
```

【样例说明】

输入 *x* 为 0.5236，*n* 为 4，求得 sin*x* 近似值为 0.50000105；同样，输入 *x* 为 0.5236，*n* 为 50，求得 sin*x* 近似值为 0.50000106。

注意：为保证数据的准确性和一致性，应使用 double 型数据保存计算结果。

9．求同构数

【问题描述】

设 *b* 是 *a* 的平方，若 *a* 与 *b* 的尾部相同，则称 *a* 是同构数。例如，5 的平方是 25，所以 5 是同构数，25 也是同构数。

编写程序满足如下要求：

输入两个整数 *m* 和 *n*，找出 *m*、*n* 之间全部的同构数（包括 *m* 和 *n* 本身）。

【输入形式】

从控制台输入数据范围的下限 *m* 和上限 *n*，要求 *m* 和 *n* 都为整数，*m* 和 *n* 之间用一个空格分隔。

【输出形式】

在屏幕上按照由小到大的顺序输出所有同构数，每个整数占一行。若在该范围内没有同构数，则输出字符串 No Answer。

【样例输入 1】

```
0 30
```

【样例输出 1】

```
0
1
5
6
25
```

【样例说明 1】

在 0～30 之间的同构数有 0, 1, 5, 6, 25。

【样例输入 2】

```
100 200
```

【样例输出 2】

```
No Answer
```

【样例说明 2】

在 100～200 之间，因为没有同构数，所以输出 No Answer。