第 3 章　3 种程序控制结构

第 1 章中已经介绍过，在结构化程序设计中所有程序都可以由 3 种基本结构组成，并给出了 3 种基本结构对应的流程图（图 1.1、图 1.2、图 1.3）。本章介绍 3 种基本程序控制结构的 C 语言实现。

3.1　顺 序 结 构

顺序结构是最简单、最基本的结构。顺序结构是按照 C 语句出现的次序顺序执行每条语句，直至程序结束。

【例 3.1】 顺序结构程序举例（求表达式的值）。

程序如下。

```
#include<stdio.h>
int main()
{
    int a,b,result;
    printf("please input two numbers:\n");
    scanf("%d,%d",&a,&b);
    result=a-2*b+3;
    printf("the result is:%d\n",result);
    return 0;
}
```

程序的运行结果如图 3.1 所示。

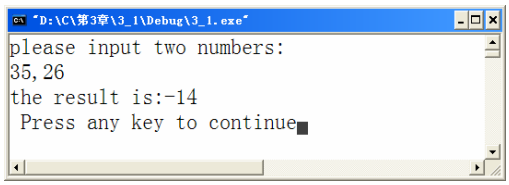


图 3.1　例 3.1 的运行结果

【例 3.2】 求方程 $ax^2+bx+c=0$ 的根，其中 a、b、c 由键盘输入，设 $b^2-4ac>0$。

程序如下。

```
#include<stdio.h>
#include<math.h>  //包含数学计算库函数的头文件
```

```
int main()
{
    int a,b,c;   //三个系数
    double x1,x2,disc,p,q;
    printf("请输入方程三个系数a,b,c的值：");
    scanf("%d%d%d",&a,&b,&c);
    disc=b*b-4*a*c;
    p=-b/(2.0*a);    //“2.0”是为了保障不同类型的转换int->double
    q=sqrt(disc)/(2.0*a);
    x1=p+q;
    x2=p-q;
    printf("方程的根为：%8.2lf%8.2lf\n",x1,x2);
    return 0;
}
```

程序的运行结果如图 3.2 所示。

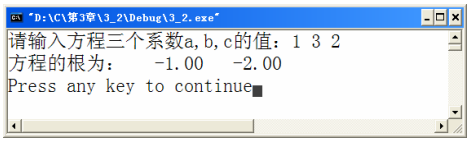


图 3.2　例 3.2 的运行结果

3.2　选 择 结 构

选择结构也称为分支结构，它的作用是根据是否满足给定条件，选择一组操作执行。C 语言提供了两种选择语句用于实现选择结构，分别是 if（条件语句）和 switch（开关语句）。

3.2.1　if 语句实现选择结构

1．if 语句的两种基本形式

基本形式一如下。

```
if (表达式)
    语句
```

基本形式二如下。

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

说明：

（1）“表达式”通常为关系表达式或逻辑表达式，也可以是其他表达式。**基本形式一**的执行过程是，首先计算表达式的值，若表达式的值为 true（非 0），则执行语句；否则，

跳过语句，整个 if 语句终止执行，然后执行 if 语句的后继语句。**基本形式二的**执行过程是，首先计算表达式的值，若表达式的值为 true（非 0），则执行语句 1；否则，执行语句 2。

（2）“语句”称作 if 语句的子语句，它可以是单语句、块语句（由花括号{ }括起来的多条语句组成，也称为复合语句）或空语句（只有一个分号的语句）。

（3）对于**基本形式二**，每次运行程序时，语句 1 和语句 2 不会都被执行，只能执行其一。

【例 3.3】 根据用户的输入决定向用户显示什么样的信息。

程序如下。

```
#include<stdio.h>
int main()
{
   char response;
   printf("Do you like programming?(Y or N):");
   scanf("%C", &response);
   if(response=='Y'||response=='y')
      printf("Just wait until you see what you can do!\n");
   if(response=='N'||response=='n')
   {
      printf("Keep trying! I bet you'll change your mind.\n");
      printf("Programming can be blast!\n");
   }
   return   0;
}
```

程序的运行结果如图 3.3 所示。

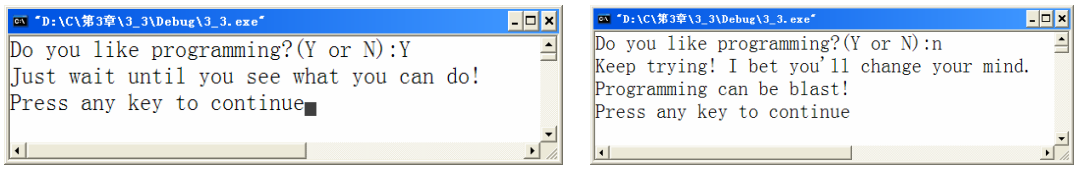


图 3.3　例 3.3 的运行结果

程序解析：

✓ 在这个程序中，response 是一个字符变量，用来存储用户通过键盘输入的任何字符。用户的响应通过语句

```
scanf(%C, &response);
```

输入到变量 response 中。

✓ if 语句用来判断用户输入的字符。询问“变量 response 的值等于字母 Y 或 y 吗？”这个问题，用 C 语言表达式表达成：

```
response=='Y'||response=='y'
```

注意：Y 或 y 不是程序设计中的一个变量，它用于比较变量 response 中保存的值，所

以使用单引号括起来，是字符常量。

✓ 由于第二个 if 语句在条件为真时要执行的语句多于一条，因此，需要使用花括号将这两条语句括起来，构成块语句。

【例 3.4】 使用 if-else 结构形式改写上例。

程序如下。

```
#include<stdio.h>
int main()
{
   char response;
   printf("Do you like programming?(Y or N):");
   scanf("%C", &response);
   if(response=='Y'||response=='y')
      printf("Just wait until you see what you can do!\n");
   else
   {
      printf("Keep trying! I bet you'll change your mind.\n");
      printf("Programming can be blast!\n");
   }
   return  0;
}
```

程序解析：

在这个程序中，原程序中的第二条 if 语句用 else 取代，使程序的逻辑结构更加清晰，代码的可读性更好。

注意：

（1）一般情况下，if 语句中的表达式常常是一个判断两个数据是否相等的关系表达式，要当心不要将==写成=。这是初学者最容易犯的错误！

（2）第一条 if 语句后面只有一条语句，可以省略不用加外边的{ }。但是，对于初学者来说，建议任何情况下都不要省略程序段（块语句或单一语句）外边的{ }。因为，如果在必须加{ }的某个程序段外边忘记加了，程序的逻辑就会出现异常。

下面关于块语句再做几点进一步的讨论。

1）关于变量的定义位置

在标准 C 中规定变量定义必须放在所有的执行语句之前！一旦在运行语句之后再有变量定义，则系统会报错误！

【例 3.5】 标准 C 中规定变量定义位置举例。

程序如下。

```
#include<stdio.h>
int main()
{
    char char1='A';
```

```
    printf("大写字符%c的ASCII码: %d\n",char1,char1);
    char char2=char1+32;
    printf("小写字符%c的ASCII码: %d\n",char2,char2);
    return 0;
}
```

将上述源程序命名为 3_5.c，在 VC++ 6.0 中调用 C 编译器编译该程序，将出现编译错误，如图 3.4 所示。

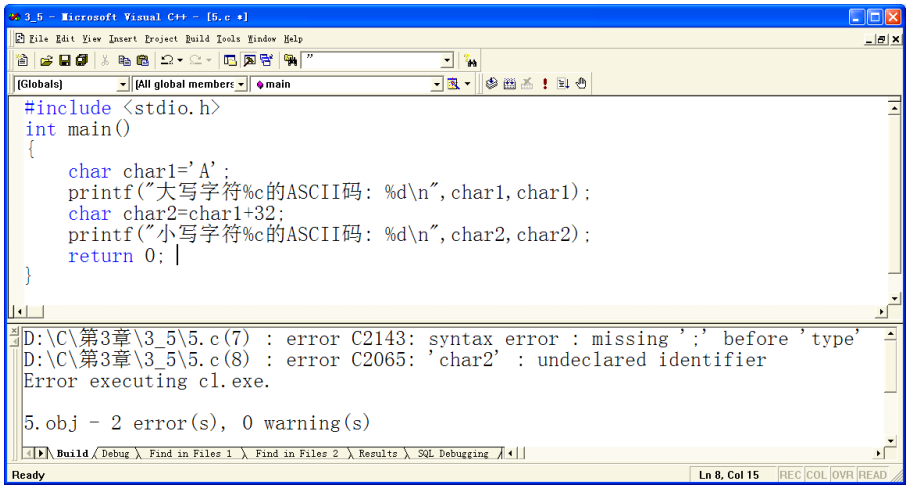


图 3.4　变量定义位置错误时的编译错误示例

程序解析：

✓ 这是因为在上述程序中关于变量 char2 的定义语句：

```
char char2=char1+32;
```

放在了执行语句：

```
printf("大写字符%c的ASCII码: %d\n",char1,char1);
```

的后面而导致的错误。

✓ 如果将上述源程序命名为 3_5.cpp，在 VC 6.0 中则会调用 C++编译器，编译就不会出错。这是因为编译器认为上述代码是 C++语法格式的，也就是说在 C++中没有严格要求变量定义一定要放在执行语句之前。

✓ 这就是 C 标准的问题，C89 规定：需要在任何执行语句之前，声明所有局部变量。但是，在 C99 以及 C++中则没有这个限制，即在首次使用之前，可在程序的任何位置声明变量。

✓ 如果交换上述程序中的第 5、6 行语句的位置，就不会出现编译错误，程序运行结果如图 3.5 所示。

2）在块语句中定义变量

块语句就是以花括号{ }包围起来的代码段，也称为 block。block 是允许出现在程序任

意位置的，故在一个 block 前面很可能有多条可执行语句。但是只要在 block 开始的地方定义变量，即在 block 内部的可执行语句前面定义变量就不会出现编译错误。该变量的作用域和生存期只在该 block 里，且该变量可以屏蔽 block 外面的同名变量。

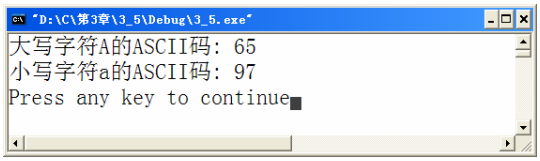


图 3.5　改正例 3.5 变量定义位置后的程序运行结果

如果将例 3.5 改写成以下形式，将第 6、7 行放入一对花括号{ }内，构成一个 block，编译时就不会出现错误提示，程序运行结果与图 3.5 相同。

```
#include<stdio.h>
int main()
{
    char char1='A';
    printf("大写字符%c的ASCII码: %d\n",char1,char1);
    {
        char char2=char1+32;
        printf("小写字符%c的ASCII码: %d\n",char2,char2);
    }
    return 0;
}
```

3）在块中定义变量的作用域与生命期

关于“作用域”与“生命期”的概念将在第 5 章中详细介绍，这里只针对在块中定义变量的问题做简单讨论。

在 block 中定义的变量的作用域和生存期只在该 block 里，且该变量可以屏蔽 block 外面的同名变量。为了说明这个问题，我们将对例 3.5 做如例 3.6 的变化，请读者通过程序中的注释及程序运行结果来加深问题的理解。

【例 3.6】 块中定义变量的作用域与生命期举例。

程序如下。

```
#include<stdio.h>
int main()
{
    char char1='A';
    char char2='B';                    //第一次（在块外）定义变量char2,并赋初值
    printf("大写字符%c的ASCII码: %d\n",char1,char1);
    printf("大写字符%c的ASCII码: %d\n",char2,char2);
    {
        char char2=char1+32;           //第二次（在块内）定义变量char2,并赋值,
                                       //该变量将屏蔽掉块外面定义的同名变量,
```

```
                                    //该变量的作用域从定义处起始至块尾标志的“}”处结束
        printf("小写字符%c的ASCII码: %d\n",char2,char2);
                                    //此时输出的是块内定义的char2的值
    } //从块中退出后，块内定义的变量char2生命期结束，不复存在
    printf("测试变量char2的ASCII码值: %d\n",char2);
                                    //此时输出的是块外定义的char2的ASCII码值
    return 0;
}
```

程序的运行结果如图 3.6 所示。

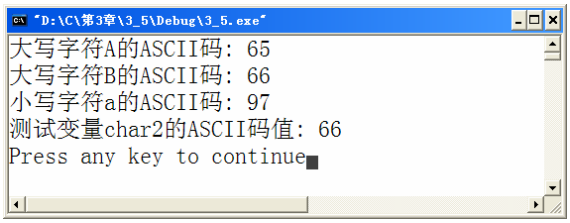


图 3.6　例 3.6 程序的运行结果

2．if 语句的嵌套

一个 if 语句可以用作另一个 if 语句的子语句，这个 if 语句称作嵌套的 if 语句。嵌套的 if 语句在使用时一定要谨慎，必须记住，当出现 if 语句嵌套时，不管书写格式如何，else 都将与它前面最靠近的未曾配对的 if 语句相配对，构成一条完整的 if 语句。

例如：

```
if (a==b)
    if (a==c)
        printf("a==b==c\n");
else
        printf("a!=b\n");
```

从书写格式上看，程序员想使 else 与“if（a == b）”相匹配，但实际情况是 else 与第二条 if 语句即“if（a==c）”相匹配的。如果想使 else 与“if（a == b）”相匹配，这段程序应写成：

```
if (a==b)
{
    if(a==c)
        printf("a==b==c\n");
}
else
        printf("a!=b\n");
```

说明：

当嵌套层次很多时，使用块语句，不仅可以避免错误，而且可以增加程序的可读性和

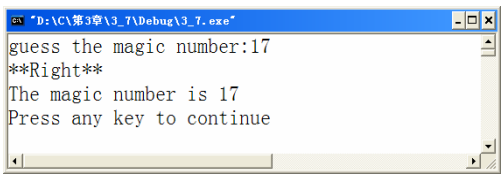
清晰性。

【例 3.7】 下面是一个猜数程序，当用户猜对正确的数时，打印“**Right**”；否则打印“** Wrong **”，并告诉用户所猜的数比正确的数是大还是小。

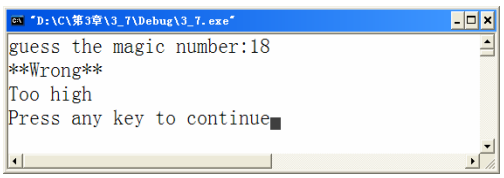
程序如下。

```
#include<stdio.h>
int  main()
{
    int magic=17;
    int guess;
    printf("guess the magic number:");
    scanf("%d", &guess);
    if(guess==magic)
    {
        printf("**Right**\n");
        printf("The magic number is %d\n", magic);
    }
    else
    {
        printf("**Wrong**\n");
        if(guess>magic)
            printf("Too high\n");
        else
            printf("Too low\n");
    }
    return   0;
}
```

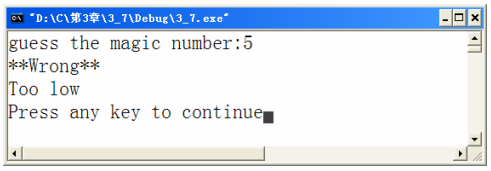
程序的运行结果如图 3.7 所示。



（a）



（b）



（c）

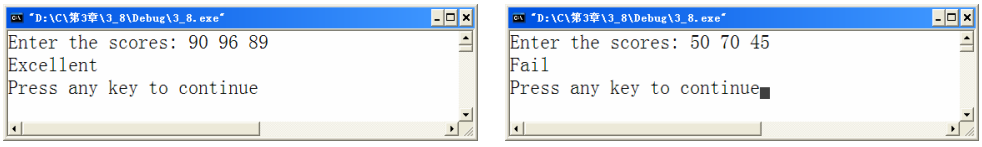
图 3.7　例 3.7 的运行结果

【例 3.8】 下面这个程序输入某个学生三门课的成绩，计算其平均值，然后根据其值打印出评语。

程序如下。

```
#include<stdio.h>
int  main()
{
    int  math,chem,phy;
    double ave; //平均值
    printf("Enter the scores:");
    scanf("%d%d%d",&math, &chem, &phy);
    ave=(math+chem+phy)/3.0+0.5;   //对平均值进行四舍五入
    if(ave>=90)
       printf("Excellent\n");
    else
    {
       if(ave>=80 && ave<90)
          printf("Good\n");
       else
       {
          if(ave>=70 && ave<80)
             printf("Average\n");
          else
          {
             if(ave>=60 && ave<70)
                printf("Pass\n");
             else
                printf("Fail\n");
          }
       }
    }
    return    0;
}
```

程序的运行结果如图 3.8 所示。



（a）　　（b）

图 3.8　例 3.8 的运行结果

程序解析：

✓ 程序中使用了嵌套的 if 语句自上而下对各种可能条件进行测试，当某一条件为真时

就执行与此条件有关的语句，并且跳过嵌套的 if 语句中的其余部分，如果没有一个条件为真，就执行最后一个 else，它表示成绩不及格者。

✓ 虽然在程序中加入了{}，使程序的选择结构清晰可见，但是当 if-else 嵌套层次较多时就不宜再使用这种形式，不但书写时容易出错，而且可读性不好。

上面例子的嵌套形式可用如下嵌套格式表示。

```
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
```

如图 3.9 所示，在这种嵌套形式中，若表达式 1 的值为 true，则执行语句 1；若为 false，则判断表达式 2，其结果为 true，执行语句 2；若为 false，则判断下一个 if 语句。若所有表达式的值都为 false，则执行语句 *n*+1，即 if 语句中的第 *n* 个 else 部分。

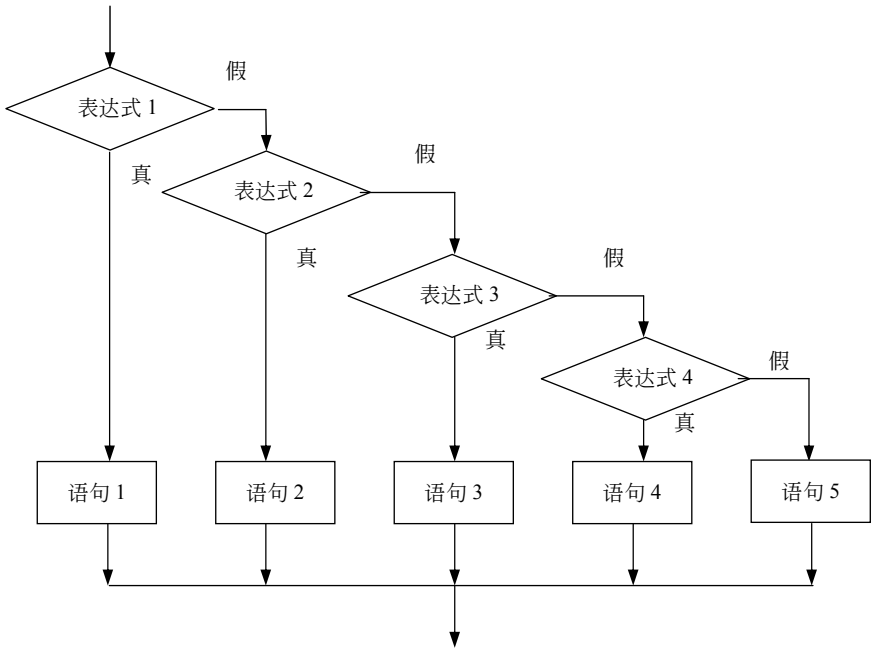


图 3.9　if 语句嵌套形式

【例 3.9】 使用 if 语句嵌套形式改写例 3.8。

程序如下。

```
#include<stdio.h>
int  main()
{
   int  math,chem,phy;
   double ave; //平均值
   printf("Enter the scores: ");
```

```
    scanf("%d%d%d", &math, &chem, &phy);
    ave=(math+chem+phy)/3.0+0.5;   //对平均值进行四舍五入
    if(ave>=90)
       printf("Excellent\n");
    else if(ave>=80)
       printf("Good\n");
    else if(ave>=70)
       printf("Average\n");
    else if(ave>=60)
       printf("Pass\n");
    else
       printf("Fail\n");
    return   0;
}
```

程序解析：

✓ 例 3.9 程序的代码明显优于例 3.8，结构清晰且提高了可读性。

✓ 还应注意例 3.9 程序中 if 语句或 else if 语句中进行条件进行测试的表达式的变化，由例 3.8 中的逻辑表达式变化为条件表达式，简化了条件测试的表达，请读者注意理解这种变化。

3.2.2 switch 语句实现选择结构

switch 语句是**多分支选择语句**，又称**开关语句**，是选择结构的另一种形式。它将一个表达式的值与某些常量进行连续测试，如果某一常量与该表达式的值匹配，则与之相应的语句便被执行。

switch 语句的一般形式如下。

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

其中语句组称为 switch 语句的子语句。在 switch 语句中，表达式值的类型和常量的类型必

须一致，且只能是字符型、整型或枚举类型（枚举类型将在第 8 章中介绍）。当表达式的值与 case 中指定的常量相等时，就执行相应 case 后的语句，直到遇到 break 语句或者到达 switch 语句的末尾。如果出现相等匹配都不成功的情况，则执行 default 后的语句。default 语句是可选项，如果没有 default 语句，那么当所有匹配都失败的情况下，不执行任何操作，程序直接执行 switch 语句之后的语句。

图 3.10 描述了 switch 语句的流程。

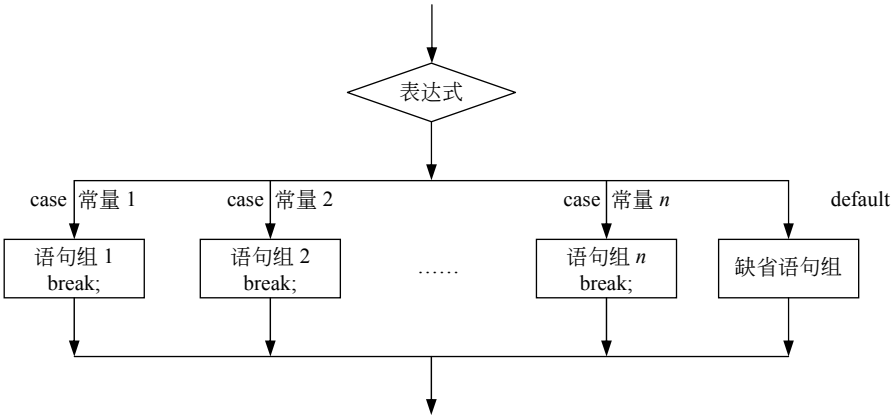


图 3.10　switch 语句流程图

注意：

（1）switch 下面的一对花括号不能省略，它的作用是将多分支结构视为一个不可分割的整体。每一个 case 后面的语句块不需要用大括号括起来，程序流程会自动顺序地执行该 case 后面的所有可执行语句。

（2）每个 case 中的 break 语句使 switch 语句只执行一个 case 中的语句，执行到 break 语句即从 switch 语句中跳出。若没有 break 语句，将继续执行该 case 下面各 case 部分的执行语句。

（3）所有常量表达式的值必须互不相同，否则编译时会发生错误。case 部分与 default 部分的顺序可以自由书写。如果 default 部分位于程序最后，default 部分的 break 语句便可以省略；否则 break 语句必不可少。

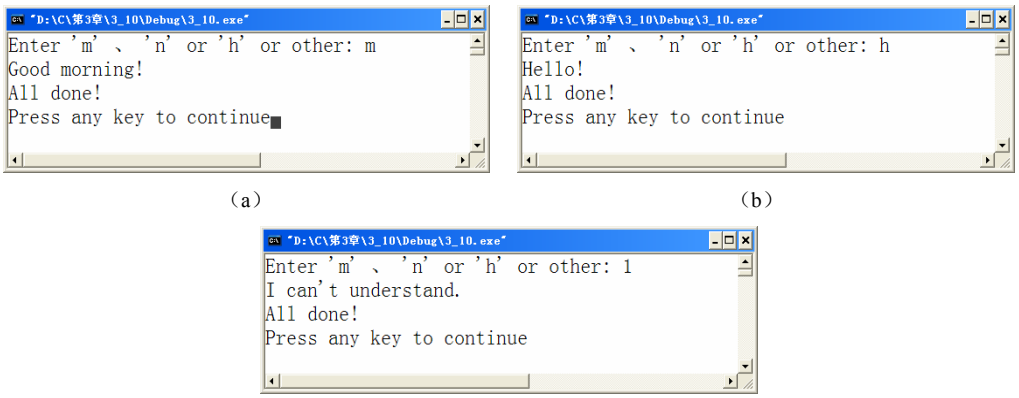
【例 3.10】 switch 语句使用举例。

程序如下。

```
#include<stdio.h>
int main()
{
    char ch;
    printf("Enter 'm' 、 'n' or 'h' or other: ");
    scanf("%c",&ch);
    switch(ch)
     {
         case 'm': printf("Good morning!\n");
```

```
                    break;
            case 'h': printf("Hello!\n");
                    break;
            case 'n': printf("Good night!\n");
                    break;
            default:  printf("I can't understand.\n");
        }
        printf("All done!\n");
        return    0;
}
```

程序的运行结果如图 3.11 所示。



（a）　　（b）

（c）

图 3.11　例 3.10 程序的运行结果

当若干个 case 所执行的内容可用一条语句（可以是块语句）表示时，允许这些 case 共用一条语句。这种情况下的 switch 结构变为如下形式。

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

```
        语句组 n+1;
}
```

这种结构的 switch 语句的流程是：当表达式的值与常量表达式 1 的值或常量表达式 2 的值，……，或常量表达式 *m* 的值之一匹配时，都执行语句组 *m*；当表达式的值为其他值时执行情况不变。

【例 3.11】 改写例 3.10，允许用户输入字母时不区分大小写。

程序如下。

```
#include<stdio.h>
int main()
{
    char ch;
    printf("Enter 'm' 、 'n' or 'h' or other: ");
    scanf("%c",&ch);
    switch(ch)
    {
        case 'm':
        case 'M': printf("Good morning!\n");
                  break;
        case 'H':
        case 'h': printf("Hello!\n");
                  break;
        case 'N':
        case 'n': printf("Good night!\n");
                  break;
        default: printf("I can't understand.\n");
    }
    printf("All done!\n");
    return   0;
}
```

程序解析：

✓ 程序运行时允许用户输入大小写字母，即输入 m 或 M 时程序输出结果一样。同理，输入 h 或 H、输入 n 或 N，程序输出结果也是一样的。

3.3 循 环 结 构

循环结构是指在一定条件成立的情况下，重复执行某个程序段的结构。一般来说，一个循环结构由 4 个主要部分构成。

（1）循环的初始部分。它保证循环结构能够开始执行的语句，往往编写在程序的开头部分，逻辑上先从这一部分开始执行。

（2）循环的工作部分，即循环体，完成循环程序的主要工作。

（3）循环的修改部分。它保证循环体在循环过程中，有关的变量能按一定的规律变化。

（4）循环的控制部分。它保证循环程序按规定的循环条件控制循环正确进行。

对于一个具体的程序，上述几个部分有时很明显就能分开，有时却很难分开。相互位置可前可后，或相互包容，但循环的初始部分一般应在循环的前面。

C 语言中构成循环结构的语句有 while 语句、do…while 语句和 for 语句。

3.3.1 while 语句

while 语句的一般形式如下。

```
while(表达式)
    语句      //循环体
```

while 语句首先计算表达式的值，在其值为真 true（非 0）时，执行循环体中的语句，而在其值为假为 false（0）时，终止循环的执行，程序接着执行循环体后的语句。

注意：

（1）while 后面的括号里是表达式而不是语句，表达式是没有分号的。

（2）循环体语句可以是简单语句、空语句或块语句。

（3）第一次计算表达式的值，如果为 false(0)，则循环体一次也不执行。

【例 3.12】 编写程序计算 1+2+3+…+99+100。

程序如下。

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

程序的运行结果如图 3.12 所示。

程序解析：

✓ 循环体如果包含多条语句，一定要用花括号括起来，以复合语句的形式出现。如果不用花括号，则循环体只有一条语句。例如，本例中 while 循环体语句中如果无花括号，则循环体只有一条语句“sum+=n;”。

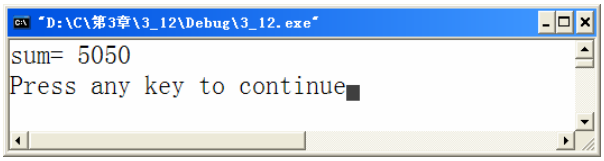


图 3.12　例 3.12 的运行结果

✓ 在循环体中应有使循环趋向于结束的语句。例如，在本例中循环结束的条件是 n>100，因此在循环体中应该有使 n 增值以最终能使 n>100 的语句，例中使用“n++;”语句来达到此目的。如果无此语句，则 n 的值始终不改变，循环永不结束，构成死循环，程序不能正常结束。

✓ 做累加时，存放结果的变量一般称为累加器，初始化为 0；而做累乘时（如求阶乘），存放结果的变量一般称为累乘器，初始化为 1。

课堂思考与练习：

（1）计算 100 以内偶数之和应如何修改程序？

（2）计算 10!。

【例 3.13】 分别统计出输入的所有正整数中小于 60 和大于等于 60 的数据的个数，然后显示。

程序如下。

```
#include<stdio.h>
int main()
{
    int x;
    int count1=0, count2=0; //定义两个计数器
    printf("请输入正整数，当输入0或负数时结束输入：");
    scanf("%d",&x);          //读入“导入值”
    while(x>0)
    {
        if(x<60)    count1++;
        else    count2++;
        scanf("%d",&x);      //循环读入数据
    }
    printf("小于60的正整数个数：%d\n",count1);
    printf("大于等于60的正整数个数：%d\n",count2);
    return 0;
}
```

程序的运行结果如图 3.13 所示。

程序解析：

✓ 程序中，进入 while 循环之前有下列一条数据输入语句：

```
scanf("%d",&x);
```

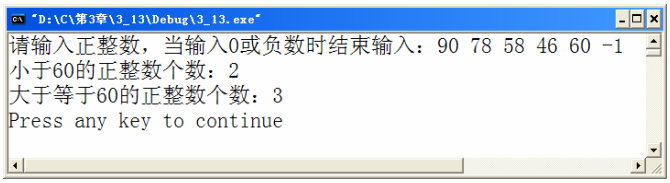


图 3.13　例 3.13 的运行结果

该语句为 while 循环读入了一个**导入值**，这是构成循环必不可少的。

✓ while 条件判断中用输入零或负数作为循环的结束标志，“零或负数”也常常被称为**哨兵**，是编写这类循环结构的典型用法。因为 while 循环常用于事先无法确切地知道循环次数的情况，必须要有一个承担循环结束标志的“哨兵”。

✓ 循环体中又有一条数据输入语句，“scanf("%d",&x);”看似重复，但是不能省略，否则，*x* 永远为第一次输入的值，可能会构成死循环。

3.3.2　do…while 语句

do…while 语句的一般形式如下。

```
do
{
    语句        //循环体
}while(表达式);
```

do…while 语句和 while 语句类似，只是对循环条件的检查在循环尾部进行。其执行过程是，先执行一次循环体语句，然后计算表达式的值，当表达式的值为 true（非 0）时，返回重新执行循环体语句，如此反复，直到表达式的值为 false（0）为止，此时循环结束。

注意：

（1）do…while 循环的循环体至少执行一次。

（2）while（表达式）后面的分号不可省略。

【例 3.14】 使用 do…while 语句重写例 3.12。

程序如下。

```
#include<stdio.h>
int main()
{
   int sum=0;
   int n=1;
   do
   {
       sum+=n;
       n++;
   }while(n<=100);
```

```
    printf("sum= %d\n",sum);
    return 0;
}
```

程序的运行结果与图 3.12 相同。

程序解析：

对于一个循环问题，既可以用 while 循环，也可以用 do…while 循环来解决问题。while 语句和 do…while 语句的区别在于：while 语句条件测试在先，如果这个测试为假，则循环体一次也不执行。do…while 语句的条件测试在后，其循环体至少执行一次。

课堂思考与练习：

（1）用 do…while 循环计算 100 以内偶数之和。

（2）用 do…while 循环计算 10!。

3.3.3 for 语句

for 语句的一般形式如下。

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

圆括号内由分号隔成三个部分，每部分是一个表达式。其中表达式 2 用于控制循环执行，当其值为真时，重复执行循环体，而在其值为假时，终止循环语句的执行，程序转去执行该语句之后的语句。for 循环的执行过程可以用 while 循环来说明。

```
表达式1;
while(表达式2)
{
    语句
    表达式3;
}
```

即在进入 for 循环时，表达式 1 求值一次，然后对表达式 2 进行求值，如果表达式 2 的值为真，执行 for 循环的循环体中的语句，随后对表达式 3 求值，紧接着又对表达式 2 求值，由表达式 2 的值决定是否进行下次循环。

for 语句是 C 语言中使用最灵活方便的一种循环语句，它不仅用于循环次数已知的情况，还能用于循环次数预先不能确定而只给出循环结束条件的情况。

注意：

（1）for 语句中的 3 个表达式必须用分号隔开，其中表达式 1 一般用来初始化循环控制变量。

（2）表达式 2 为表示循环条件的表达式，用作循环条件控制，其作用与前两类循环语句中的表达式完全一样，用法也基本相同。

（3）表达式 3 用来修改循环控制变量，用以表示循环控制变量的增量或减量，常用自

增或自减运算符。

（4）for 语句中的 3 个表达式允许全部省略或部分省略，以充分体现其灵活性，但用作分隔符的两个分号绝不能省略。但是建议初学者尽量地按照标准格式来使用，培养自己编写出专业、规范的代码。另外，当 for 语句的循环体中只有 1 条语句时，循环体外面的{}可以省略，甚至可以将这 1 条语句与 for 语句写在同一行，但是要知道这不是一个好的编程习惯。

【例 3.15】 用 for 循环重写例 3.12。

程序如下。

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

程序的运行结果与图 3.12 相同。

程序解析：

✓ 语句“for(n = 1; n <= 100; n++)　sum += n;”相当于以下语句：

```
n=1;
while(n<=100)
{
   sum+=n;
   n++;
}
```

显然，用 for 语句更加简单方便。

✓ 需要说明的是，for 语句中的表达式 1 和表达式 3 一般为简单表达式，但也可以使用逗号表达式，这是 for 语句的一个很有用的特性，当使用逗号表达式时，可一次完成对多个变量赋初值和修改多个变量的功能，例如：

```
for(i=0, j=100; i<j; i++, j--)   k=i+j;
```

其中，表达式 1 和表达式 3 都使用了逗号表达式，即同时为两个变量赋初值，同时使两个变量的值发生改变。

课堂思考与练习：

（1）用 for 循环计算 100 以内偶数之和。

（2）用 for 循环计算 10!。

【例 3.16】 摄氏温度和华氏温度对照表。

程序显示一个摄氏温度和华氏温度对照表，摄氏温度和华氏温度的数学式如下。

C=5/9*(F-32)

该程序从华氏零度到 300 度每隔 5 度显示一个表项。

程序如下。

```
#include<stdio.h>
int main()
{
    double degCel;       //摄氏
    int   degFahr;       //华氏
    for (degFahr=0; degFahr<=300; degFahr+=5)
    {
       degCel=(5.0/9.0)*(degFahr-32.0);
       printf("Fahrenheit:%d\t\tCelsius:%6.2lf\n", degFahr, degCel);
    }
     return   0;
}
```

程序的运行结果如图 3.14 所示。

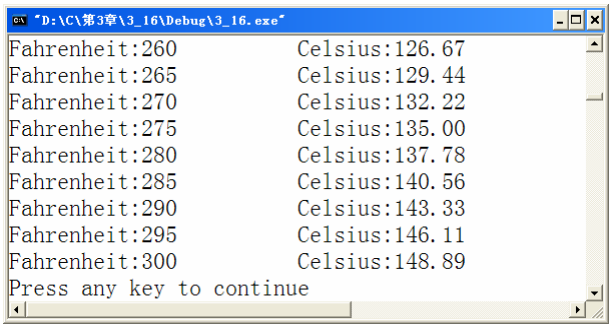


图 3.14　例 3.16 程序运行部分结果的截图

3.3.4　break 语句和 continue 语句

break 语句有两个用途，一种是用于 switch 语句中，前面已经作了介绍。另一种用途是用在循环语句的循环体中。当在循环体中遇到 break 语句时，循环立即终止，程序跳过 break 语句后的其余语句，从循环语句后的第一条语句继续执行。

continue 语句只用于循环语句的循环体中，当在循环体中遇到 continue 语句时，程序跳过 continue 语句后的其余语句，开始下一次循环。

注意：

（1）在实际应用中，break 语句和 continue 一般都是与 if 条件语句配合使用的。当正常的循环终止条件尚未满足，而满足了其他给定的条件（由 if 条件语句给出的）时，使程序流程在循环的中途强行退出本次循环。

（2）break 语句也可以用于嵌套的循环结构中，在这种情况下，如执行 break，则仅仅退出包含该 break 语句的那层循环，即 break 语句不能使程序控制退出一层以上的循环。

下面给出的这个例子较好地说明了 break 语句和 continue 语句的用法。

【例 3.17】 输出 100～200 之间**不能**被 3 整除的数，只输出前 20 个即可。

程序如下。

```
#include<stdio.h>
int  main()
{
    int i;
    int count=0; //计数器，两个作用：1.统计已输出数据的个数 2.用于控制输出格式
    for(i=100; i<=200; i++)
    {
        if(i%3==0)  continue;    //当i能被3整除时，结束本次循环，进入下一次循环
        count++;
        if(count>20)    break;   //当已输出20个数据后强行退出整个循环
        printf("%-6d",i);
        if(count%5==0)  printf("\n");//一行输出5个数据
    }
    return 0;
}
```

程序的运行结果如图 3.15 所示。

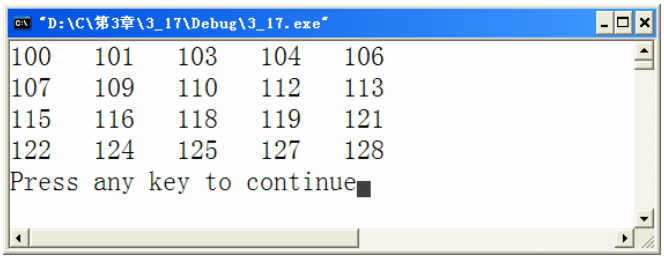


图 3.15　例 3.17 的运行结果

程序解析：

✓ 当 *i* 能被 3 整除时，执行 continue 语句，结束本次循环（即跳过 if…continue 之后的 4 条语句），进行下一次循环，继续寻找满足条件的数据。只有 *n* 不能被 3 整除时才执行 if…continue 后面的这 4 条语句。

✓ 100～200 之间不能被 3 整除的数共有 68 个，但是题目要求只输出前 20 个数据。那么循环多少次就恰好输出前 20 个呢？程序员事先是不知道的。所以只能使用了 break 语句，与 if 条件语句配合，找到前 20 个满足条件的数据后就不必让循环继续了，强行中断整个循环。

注意：continue 和 break 的作用有相似之处，却也有着根本区别，continue 语句只是结束本次循环，还要进行下一次循环，而不是要结束整个循环；break 语句则是结束整个循环，转而执行循环体外的其他语句。

3.3.5 循环嵌套

一个循环体内包含另一个完整的循环结构称为**循环的嵌套**，内嵌的循环中还可以嵌套循环，这就是**多重循环**。

【例 3.18】 输出 100 以内的所有素数。

算法分析：

（1）判断一个数 m 是否是素数，让 m 被 2 到 $\sqrt{m}$ 之中的整数去除，如果 m 能被这些数之中的任何一个整除，则说明 m 不是素数；否则 m 是素数。

（2）利用穷举法，对 2～100 之间（1 不是素数，2 是最小的素数）的每一个自然数进行素数的判断，若是，则输出之。

（3）上述步骤（1）用一个内循环实现，步骤（2）用一个外循环实现。

程序如下。

```
#include<stdio.h>
#include<math.h>    //标准库函数,包含数学函数，本例中用到了求平方根函数sqrt
#define NUMBER 100  //定义符号常量，作为区间上限
int main()
{
    int m,n;                    //m外循环变量(2～100)，n内循环变量(2～k)
    int k;                      //为减少循环次数，令k=sqrt(m)
    int flag=1;                 //设置素数判断标志，1为素数，0为非素数
    int count=0;                //计数器
    for(m=2; m<=NUMBER; m++)    //m外循环变量(2～100)
    {
        k=(int)sqrt(m+1);       //加1是为避免可能出现的误差
        flag=1;                 //注意每次进入内循环之前都要重新赋值flag=1
        for(n=2; n<=k; n++)     //内循环：判断一个数m是否是素数
        {
            if(m%n==0)
             {
                flag=0;         //标志非素数
                break;          //跳出当前循环（内循环）
             }
        }//内循环结束
        if(flag==0) continue;   //跳过当前循环结构（外循环）中后面的语句，
                                //进入下一次循环（外循环）
        printf("%-5d",m);       //输出素数
        count++;
        if(count%10==0) printf("%\n");
    }//外循环结束
    printf("\n***%d以内共有%d个素数***\n",NUMBER,count);
    return 0;
}
```

程序的运行结果如图 3.16 所示。

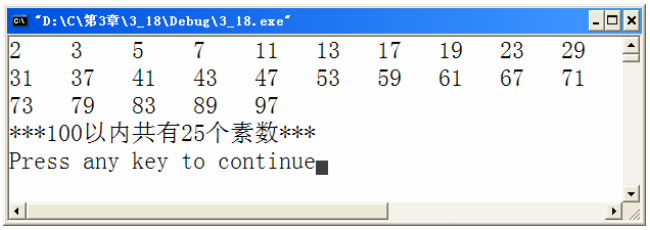


图 3.16　例 3.18 的运行结果

程序解析：

✓ 该程序是由两层循环组成的循环嵌套结构。外循环和内循环各自功能不同，相互配合完成了题目要求。

✓ 本程序中用到了如下的一个定义“标志”的语句：

```
int flag=1;    //设置素数判断标志，1为素数，0为非素数
```

设置一个“标志（flag）”是编写程序代码时常用的一种方法，用于控制程序的流程。一般情况下，flag 只有两个值 1 或 0，标志两种状态。由于 C 语言中没有布尔类型的数据，可将 flag 定义为 int 类型。

进入循环前将 flag 值设置为 1，当我们希望退出循环时，将 flag 值设置为 0。请读者结合本例中 flag 的用法，学习并掌握这种编程技巧。

注意：

（1）3 种循环可以互相嵌套，但在循环的嵌套中要注意，内层循环应完全在外层循环里面，也就是不允许出现交叉。

（2）如果循环的控制条件永远成立，循环体将永无休止地反复执行，程序就将陷入“死循环”，这显然是应当防止的。

在嵌套的循环结构中，如用缺口矩形表示每层循环结构时，则图 3.17 中（a）、（b）是正确的多层循环结构，而图 3.17（c）是错误的多层循环结构，因为它出现了循环结构的交叉。

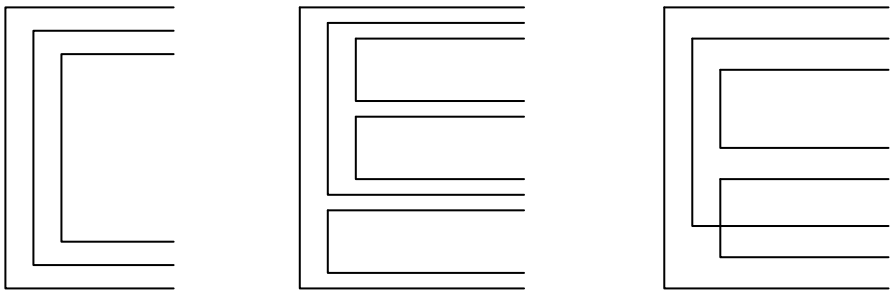


图 3.17　多层循环结构示意图

3.4　程序结构综合举例

按照结构化程序设计的思想，所有程序都可以由 3 种基本结构组成。而且 3 种基本结构可以相互多层嵌套，这样就能表示复杂的算法。本章介绍了 C 语言中用于构造 3 种基本结构的 if 语句、switch 语句、while 语句、do…while 语句和 for 语句，这些语句之间可以任意嵌套，即 if、switch 语句中可以有 while、do…while 或 for 语句，而 while、do…while 或 for 语句中也可以有 if、switch 语句甚至它本身的语句，从而形成复杂的控制结构。

下面介绍的这些例子，演示了综合使用控制结构的编程思路及一些经典的算法。另外，还需注意这些例子的实现方法不是唯一的，比如实现循环的例子中既可以用 for 语句也可以用 while 语句，请读者用心体会这些例子并多做一些语句变化（将 for 语句改为 while 语句等）方面的练习。

【例 3.19】 输出 9×9 乘法表。

算法分析：分行与列考虑，共 9 行 9 列，*i* 控制行，*j* 控制列。

程序如下。

```
#include<stdio.h>
int main()
{
    int result;              //计算结果
    int i,j;
    for(i=1;i<=9;i++)        //外循环控制行：i表示行(1～9)
    {
        for(j=1;j<=i;j++)    //内循环环控制列：j表示列（1～i）
        {
            result=i*j;
            printf("%d*%d=%-2d  ",i,j,result);
        }
        printf("\n");        //注意换行格式的控制
    }
    return 0;
}
```

程序的运行结果如图 3.18 所示。

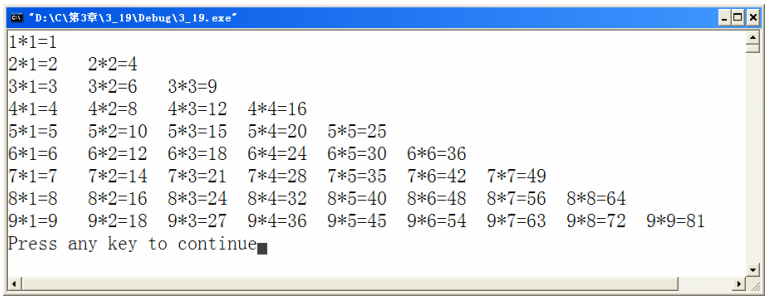


图 3.18　例 3.19 的运行结果

【例 3.20】 求 1!+2!+…+10!

算法分析：

（1）循环结构中，最常用的算法就是累加和累乘。一般累加和累乘是通过循环结构和循环体内的一句表示累加性或累乘性语句来实现的，如前面多个例子中已运用过的累加器和累乘器。要注意，累加器的初值赋为 0，而累乘器的初值赋为 1。另外，赋初值的操作一定要在循环结构外进行。

（2）本例要用一个双循环来实现，外循环用来求累加，内循环用来求累乘。

程序如下。

```
#include<stdio.h>
int main()
{
    int sum=0,fact=1;    //说明累加器和累乘器
    int i=1,j=1;         //说明两个循环变量
    while(i<=10)         //外循环求和
    {
        fact=1;          //注意在内循环外令fact=1
        j=1;
        while(j<=i)      //内循环求阶乘
        {
            fact=fact*j;
            j++;
        }
        sum=sum+fact;
        i++;
    }
    printf("1!+2!+…+10!=%d\n",sum);
    return 0;
}
```

程序的运行结果如图 3.19 所示。

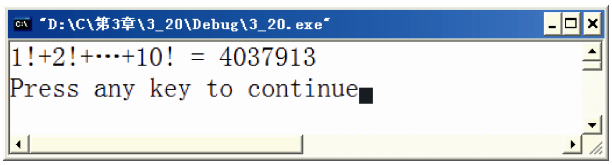


图 3.19　例 3.20 的运行结果

【例 3.21】 求自然对数 e 的近似值，要求使其误差小于 10^{-6}，近似公式如下。

$$e=1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\cdots+\frac{1}{i!}+\cdots=\sum_{i=0}^{\infty}\frac{1}{i!}\approx 1+\sum_{i=1}^{m}\frac{1}{i!}$$

算法分析：

（1）依据近似公式，应该先求第 i 项的阶乘，再将各项阶乘的倒数进行累加。因此程

序中设计两个变量 sum 和 fact 分别作为累加器和累乘器。另外，要考虑数据的精度问题。为防止项数过大时阶乘溢出，定义 fact 为 double 类型；sum 也定义为 double 类型,是为满足题目的求值精度要求。

（2）如何判断循环结束？由于循环次数不确定，所以用 while 循环好一些。

程序如下。

```
#include<stdio.h>
int main()
{
    double sum=1,fact=1;
    int i=1;                        //表示第几项
    while(1/fact>0.0000001)         //根据题目要求给出计算精度
    {
        fact=fact*i;                //求第i项的阶乘
        i++;
        sum=sum+1/fact;             //累加
    }
    printf("e≈%lf\n",sum);
    return 0;
}
```

程序的运行结果如图 3.20 所示。

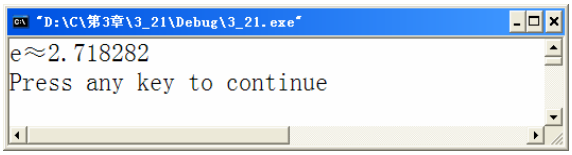


图 3.20　例 3.21 的运行结果

【例 3.22】 百钱买百鸡。即假定小鸡每只 5 角，公鸡每只 2 元，母鸡每只 3 元，问：若用 100 元钱买 100 只鸡，小鸡、公鸡、母鸡各买多少只？

算法分析：

（1）这是一个用“穷举法”解题的典型问题。“穷举法”也称为“枚举法”或“试凑法”，即将可能出现的各种情况一一测试，判断是否满足条件，一般采用循环来实现。

（2）列出所有可能的购鸡方案：

设母鸡、公鸡、小鸡各为 x、y、z 只，根据题目要求，列出方程为：

$$x + y + z = 100$$

$$3x + 2y + 0.5z = 100$$

3 个未知数，两个方程，此题有若干个解。

（3）最简单直接的编程思路是利用三重循环来实现求解过程。

程序如下。

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

```
{
    int x,y,z;      //x母鸡，y公鸡，z小鸡
    printf("母鸡个数   公鸡个数   小鸡个数\n");
    for(x=0;x<=100;x++)
    {
      for(y=0;y<=100;y++)
       {
           for(z=0;z<=100;z++)
           {
              if((x+y+z==100)&&(3*x+2*y+z*0.5==100))
                   printf("%-10d%-10d%-10d\n",x,y,z);
           }
       }
    }
    return 0;
}
```

程序的运行结果如图 3.21 所示。

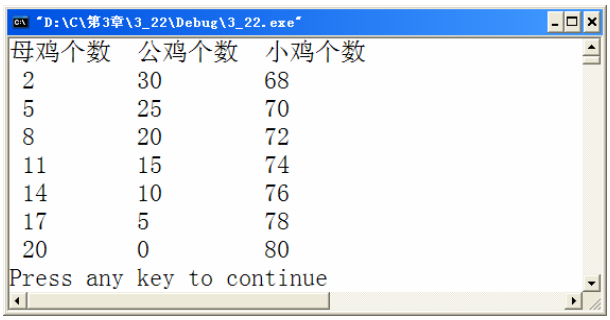


图 3.21　例 3.22 的运行结果

程序解析：

分析 3 个未知数的关系（见下面程序代码中的注释），本例利用两重循环即可实现，提高了程序的效率。程序改写如下。

```
#include<stdio.h>
int main()
{
    int x,y,z;
    printf("母鸡个数   公鸡个数   小鸡个数\n");
    for(x=0;x<=33;x++)       //100钱都用来买母鸡最多能买33只
    {
      for(y=0;y<=50;y++)     //100钱都用来买公鸡最多能买50只
       {
           z=100-x-y;
           if(3*x+2*y+z*0.5==100)  //3个未知数的关系
```

```
                printf("%-10d%-10d%-10d\n",x,y,z);
            }
        }
        return 0;
    }
```

程序运行结果与图 3.21 相同。

在程序代码中加入时间函数，对方法 1（三重循环）和方法 2（双重循环）进行运行时间的比较，从而了解提高程序的运行效率的意义。为了让两种方法的程序运行时间上的差距大一些而便于比较，将原题目的“百钱买百鸡”改为“千钱买千鸡”。方法 1 程序改写如下，方法 2 程序改写类似，请读者自己完成。

程序如下。

```
#include<stdio.h>
#include<time.h>     //时间和日期函数需要的头文件
#define N 1000       //定义一个符号常量，方便代码调试修改
int main()
{
    int x,y,z;
    time_t  t1, t2; //time_t是<time>中定义的与时间有关的类型
    double t2_t1;   //表示两个时间之间的间隔时间
    t1=clock();     //clock()是库函数中的计时函数,t1表示循环开始时间
    for (x=0;x<=N;x++)
        for (y=0;y<=N;y++)
           for(z=0;z<=N;z++)
               if ((x+y+z==N) && (3*x+2*y+0.5*z==N))
                  printf("%5d%5d%5d\n",x,y,z);
    t2=clock();      //t2表示循环结束时间
    t2_t1=(double)(t2-t1)/CLOCKS_PER_SEC;
      //CLOCKS_PER_SE是在time.h文件中定义的常量，表示一秒钟会有多少个时钟计时单元
    printf("此方法用去%f秒的时间\n",t2_t1);
    return 0;
}
```

方法 1 程序改写之后的运行结果如图 3.22 所示，方法 2 程序改写之后的运行结果如图 3.23 所示。需要说明的一点是：本例在不同的机器上运行所显示的时间可能是不同的。

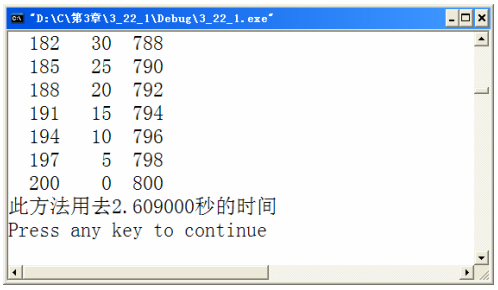


图 3.22　方法 1 加入时间函数的运行结果

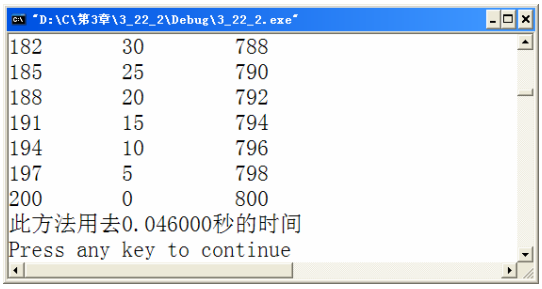


图 3.23　方法 2 加入时间函数的运行结果

【例 3.23】 求 Fibonacci 数列：1，1，2，3，5，8，…的前 40 个数。

该数列的生成方法为：F_1=1 (n=1)，F_2=1 (n=2)，F_n=F_{n-1}+F_{n-2} (n≥3)，即从第 3 个数开始，每个数等于前两个数之和。

算法分析：

（1）这是一个用“迭代法”解题的典型问题。迭代法也称辗转法，是一种不断地利用变量的旧值递推新值的过程。利用迭代算法解决问题，需要做好以下 3 个方面的工作:

① 确定迭代变量。在可以用迭代算法解决的问题中，至少存在一个直接或间接地不断由旧值递推出新值的变量，这个变量就是迭代变量。

② 建立迭代关系式。所谓迭代关系式，指如何从变量的前一个值推出其下一个值的公式（或关系）。迭代关系式的建立是解决迭代问题的关键，通常可以使用递推或倒推的方法来完成。

③ 对迭代过程进行控制。在什么时候结束迭代过程？这是编写迭代程序必须考虑的问题。不能让迭代过程无休止地重复执行下去。

（2）本例中，根据 Fibonacci 数列的特点，至少需要定义两个变量 f1 和 f2 分别用来保存 F_{n-1} 和 F_{n-2}。可以直接将 f1+f2 赋给 f1，这时 f1 就是新数列值，这时再把 f1+f2 赋给 f2,f2 则是下一个新的数列值。也就是说，如果循环体这样设计:

f1=f1+f2;

f2=f1+f2;

则一次可求出两个数列项的值。

程序如下。

```
#include<stdio.h>
int main()
{
    int f1=1, f2=1;     //定义并初始化数列的头两个数
    int i;
    for(i=1; i<=20; i++)    //每次循环控制输出两个数，40个数需20次循环
    {
        printf("%12ld%12ld",f1,f2);      //输出当前的两个数
        if(i%2==0)  printf("\n");        //输出4个数后控制换行
        f1=f1+f2;                        //计算新的f1
        f2=f2+f1;                        //计算新的f2
    }
    return 0;
}
```

程序的运行结果如图 3.24 所示。

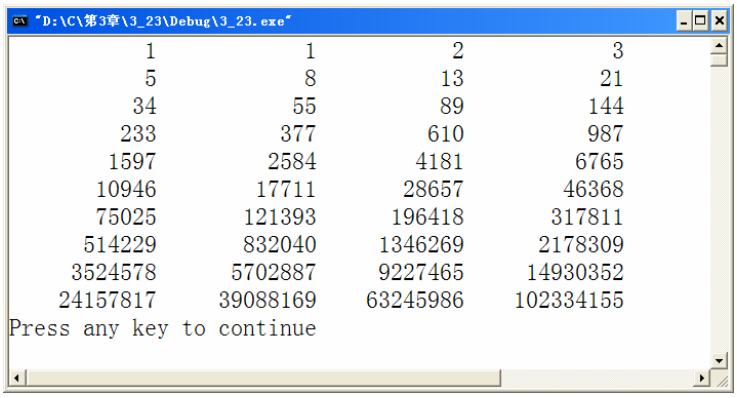


图 3.24　例 3.23 的运行结果

习　　题

1．任意输入 3 个整数，按由小到大的顺序输出。

2．求解在 100～200 之间，“既能被 5 整除又能被 6 整除”的所有的数。

3．输入一行字符，分别统计出其中英文字母、空格、数字和其他字符的个数。

4．求圆周率 π 的近似值，要求最后一项的绝对值小于 10^{-8}，近似公式如下。

$$\frac{\pi}{4}\approx 1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots$$

5．对下列数学式求值，要求至少求出前 20 项（$n\geqslant 20$）或最后一项的值小于 10^{-3}。

$$\frac{1}{1\times 2}+\frac{1}{2\times 3}+\frac{1}{3\times 4}+\cdots+\frac{1}{n(n+1)}$$

6．求出所有的“水仙花数”。

“水仙花数”是指一个三位数，其各位数字的立方和恰好等于该数本身。例如，153=1*1*1+5*5*5+3*3*3，所以 153 是“水仙花数”。依题意，从 100~999 循环查找“水仙花数”即可。

7．编一程序，将任意自然数 n 的立方表示为 n 个连续的奇数之和。例如：

$1^3=1$

$2^3=5+3=8$

$3^3=11+9+7=27$

$4^3=19+17+15+13=64$

8．输入年份和月份，打印出该年份该月份的天数。

根据历法，凡是 1、3、5、7、8、10、12 月，每月 31 天，凡是 4、6、9、11 月，每月 30 天，2 月份闰年 29 天，平年 28 天。据此，题中采用多个 case 可共同使用一个语句块的 switch 语句，完成不同天数的选择。

另外，判断某年是否为闰年的规则是：如果此年份能被 400 整除，则是闰年；否则，如果此年份能被 4 整除，而不能被 100 整除，则是闰年；否则不是闰年。

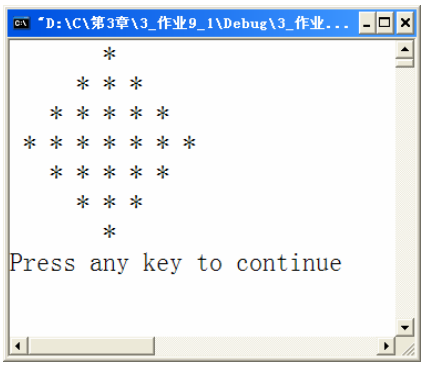
9．编程输出以下菱形图案。

（1）基本菱形图案，固定图形行数，如图 3.25（a）所示。

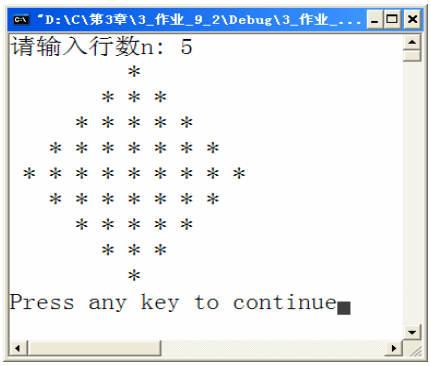
（2）可变行数的菱形图案，从键盘输入行数 n，如图 3.25（b）所示。

（3）增加编程难度，将“*”改变为字母，菱形图案行数固定，如图 3.25（c）所示。

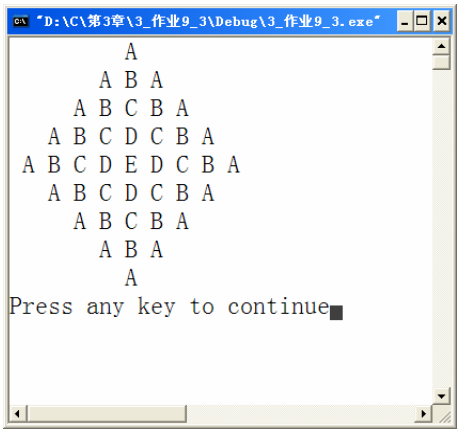
（4）在（3）的基础上，使菱形图案的行数可变，从键盘输入图案中心字母，如图 3.25（d）所示。



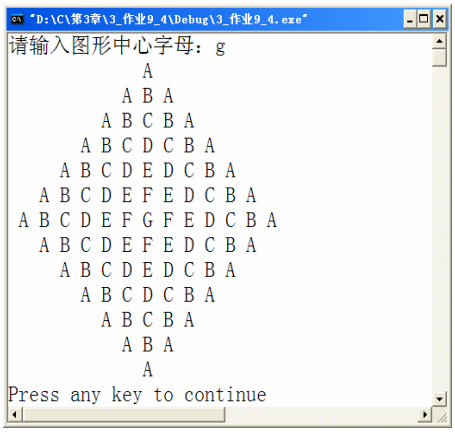
（a）



（b）



（c）



（d）

图 3.25　习题 9 的运行结果