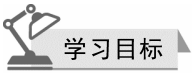
第 3 章
用函数实现模块化程序设计

模块化程序设计是指将整个程序划分为若干功能模块，每个模块完成一个特定的功能，并在这些模块之间建立必要的联系，通过模块的互相协作完成整个功能的程序设计。第 1 章介绍的主函数（main）就是主模块，标准输入函数（scanf）和输出函数（printf）就是系统提供的功能模块。C 程序可由一个主函数和若干其他函数构成。主函数调用其他函数；其他函数可以互相调用，但不能调用主函数。用函数实现模块化的程序设计，程序的逻辑关系简单而清晰，降低了程序的出错率，使编写、调试和维护程序更加方便。



- 了解 C 语言的程序构成和运行步骤。
- 理解函数、形参、实参、全局变量、局部变量、静态变量、自动变量等概念。
- 掌握函数的定义、调用，以及迭代、递推、递归的程序设计方法；掌握模块化程序设计方法。

3.1 函数

“函数”一词从英文 function 翻译而来。function 在英文中既有函数的意思，也有功能的意思，所以函数就是功能。函数名是具有某特定功能的一个程序单元的名字，因此函数名应反映这个函数所具有的功能。通常，采用功能对应的英文单词或缩写来作为函数的名字，如 main 作为主函数的名字，sin 作为正弦函数的名字。

3.1.1 函数的定义与调用

在 C 语言中，所有的函数都必须“先定义，后调用”。主函数是由操作系统（如 Windows）来调用的，其他函数都是直接或间接地由主函数调用的。通过调用函数来使用函数，发挥函数的功能。

1．函数定义的要素

（1）指定函数的名字，以便按函数名来调用该函数。

（2）指定函数的类型，即函数返回值的类型。如果没有返回值，则类型为“void”，即返回值为“空”。

（3）指定函数参数的名字和类型，以便在调用函数时通过参数向函数传递数据。对无参函数不需要这项。函数的参数又称形参，即函数的形式参数。

（4）指定函数应当完成什么操作，即函数的功能。函数的功能在函数体中实现，函数体通常有一条或多条语句。

2．函数定义的格式

函数由函数首部和函数体组成。函数首部包括函数的类型名、函数名和形参列表。

（1）无参函数的定义格式：

```
类型名   函数名( )
{
    函数体
}
```

或

```
类型名   函数名(void)
{
    函数体
}
```

（2）有参函数的定义格式：

```
类型名   函数名(形式参数列表)
{
    函数体
}
```

3．函数调用的形式

（1）把函数调用单独作为一个语句，如 printf("a=%d\n",10);。

（2）函数调用出现在另一个表达式中，如 a=sum( );。将自定义函数 sum 的返回值赋给变量 a。

（3）函数调用作为另一个函数调用时的参数，如 printf("a= %d\n", sum());。

【例 3-1】无参函数的定义和调用，计算并输出正整数 1 到 100 之和。

解题思路：

定义无参函数 sum，sum 函数的功能是计算正整数 1 到 100 之和。计算方法采用循环结构和迭代累加的方式，即从 1 到 100 共循环 100 次，执行 100 次 n = n + i，依次求出前 i 项的正整数之和 n，n 初值为 0。

程序的流程图如图 3-1 所示。

计算过程如下：

i=1，求出 n = 0 + 1 = 1；

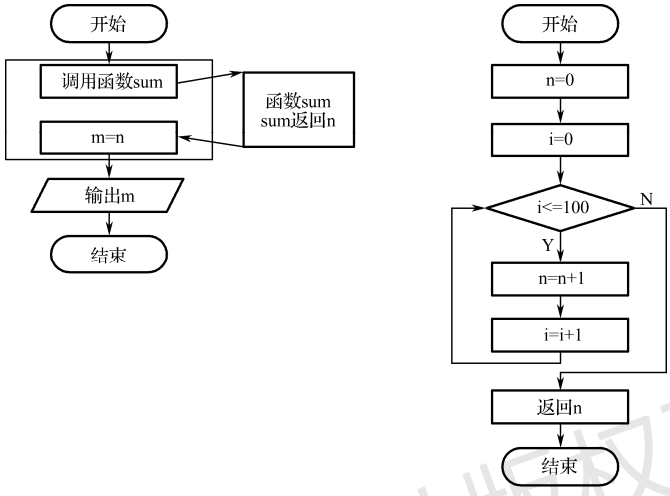
i=2，求出 n = 1 + 2 = 3；

i=3，求出 n = 3 + 3 = 6；

…

i=100，求出 n = 100 + 4950 = 5050。

通过主函数调用 sum 函数，将 sum 函数的返回值赋给变量 a，最后输出变量 a 的值。



（a）主函数流程图　　（b）sum函数流程图

图 3-1

程序代码：

```
#include <stdio.h>
int sum( )                          //定义无参函数，函数名为 sum，返回值为 int 型
{                                   //函数功能：计算正整数 1 到 100 之和
    int i, n=0;
    for(i=1; i<=100; i++)
    {
        n += i;                     //等效于 n=n+i;
    }
    return n;                       //函数返回值：正整数 1 到 100 之和 n
}

int main()
{
    int m;
    m = sum();                      //调用 sum 函数，将函数的返回值赋给变量 m
    printf("正整数 1 到 100 之和：%d\n",m);
    return 0;
}
```

运行结果：

```
正整数1到100之和：5050
Press any key to continue
```

程序分析：

（1）本例中采用了函数调用的前两种形式。第一种是将 printf 函数单独作为一个语

句来调用，第二种是将 sum 函数作为赋值运算等号右端的表达式来调用。

因为 sum 函数的类型（即返回值的数据类型）是 int 型，所以赋值运算等号左边的变量 a 也必须是 int 型。

（2）如果采用函数调用的第三种形式，把函数的返回值作为调用另一个函数的参数，只需将本例中的主函数改为

```
int main()
{
    printf("正整数 1 到 100 之和：%d\n",sum());
    return 0;
}
```

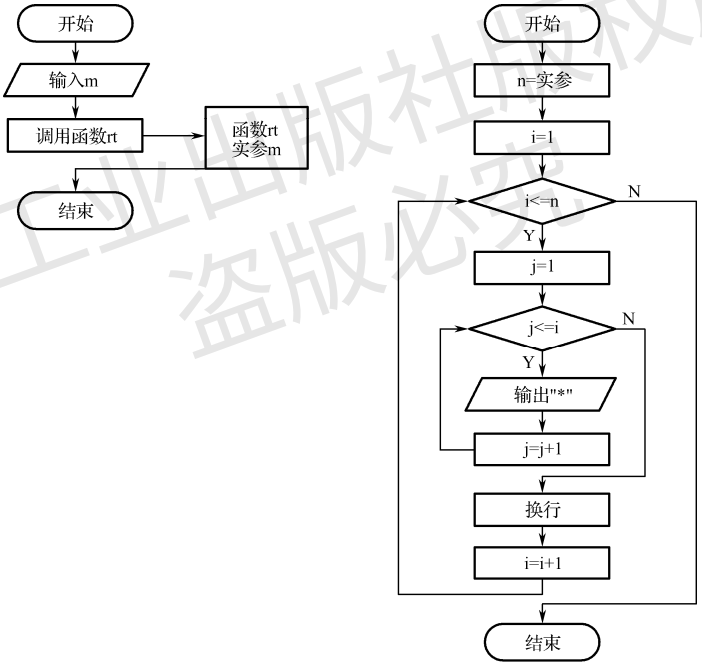
程序的运行结果与之前的运行结果完全相同。

【例 3-2】有参函数的定义和调用，输出直角三角形（right triangle）图案。

解题思路：

定义有参函数 rt。rt 函数的功能是输出直角三角形图案。在主函数中输入直角边的边长 m，并调用 rt 函数，通过 rt 函数输出图案。

程序的流程图如图 3-2 所示。



（a）主函数流程图　　（b）rt函数流程图

图 3-2

程序代码：

```
#include <stdio.h>
void rt(int n);                                    //声明 rt 函数
int main( )
{
    int m;                                         //定义三角形直角边的边长 m
    printf("输入边长（大于等于 2 的正整数）：");    //输出提示信息
```

```
    scanf("%d",&m);
    rt(m);                                          //调用函数 rt，实参为 m
    return 0;
}

void rt(int n)                      //函数的首行（或称为函数的首部），形参为 n
{
    int i, j;                       //循环变量 i 控制行输出，j 控制列输出
    for(i=1;i<=n;i++)               //输出 n 行星号，n 的值即主函数中 m 的值
    {
        for(j=1;j<=i;j++)           //第 i 行，循环 i 次，输出 i 个星号
            printf("*");            //输出 1 个星号
        printf("\n");
    }
}
```

运行结果：

```
输入边长（大于等于2的正整数）：5
*
**
***
****
*****
Press any key to continue
```

程序分析：

（1）函数必须“先定义，后调用”，或者“先声明，后调用”。

在例 3-1 的程序中，sum 函数的定义位于 main 函数之前，属于“先定义，后调用”。

在本例的程序中，rt 函数的定义位于 main 函数之后，不符合“先定义，后调用”的原则。C 语言规定，函数的定义可以位于调用它的函数（如本例中的 main 函数）之后，但在调用该函数之前需用函数原型进行声明（如本例中的“void rt(int n);”语句）。

（2）函数原型即该函数的首行。用函数原型声明函数时，应以分号结束；而定义函数时，函数的首行“void rt(int n)”后面没有分号。

（3）定义函数时，函数名后面括号中的变量为“形式参数”（简称“形参”）或“虚拟参数”。本例中，函数首部“void rt(int n)”中的“n”就是形参。

（4）调用函数时，函数名后面括号中的参数称为“实际参数”（简称“实参”）。本例中，语句“rt(m);”中的“m”就是实参。程序执行到“rt(m);”语句时，实参 m 的值被赋给了形参 n，即当 m 等于 5 时，函数的调用实现了实参与形参之间数据的传递，5 被赋给了 n。

上面两个例题分别介绍了有返回值无参函数 sum 的定义和调用，以及无返回值有参函数 rt 的定义和调用。读者可参考这两个例题，以此类推，编写程序实现无返回值无参函数的定义和调用，以及有返回值有参函数的定义和调用。

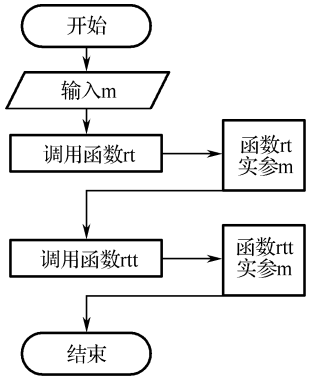
【例 3-3】运用函数的嵌套调用，输出上下连接的两个直角三角形（right triangle）图案，下面的三角形边长等于上面的三角形边长加 2。

解题思路：

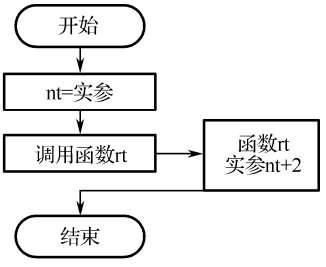
定义有参函数 rt 和 rtt。rt 函数的功能是输出直角三角形图案，如例 3-2 所示。rtt

函数的功能是接受主函数的调用，再调用 rt 函数输出一个较大的直角三角形图案。在主函数中输入直角边的边长 m，以 m 为参数调用 rt 函数，以 m+2 为参数调用 rtt 函数，输出一小一大两个直角三角形。

流程图如图 3-3 所示（rt 函数流程图参见图 3-2（b））。



（a）主函数流程图



（b）rtt函数流程图

图 3-3

程序代码：

```
#include <stdio.h>
void rt(int n);                                    //声明 rt 函数
void rtt(int n);                                   //声明 rtt 函数
int main()
{
    int m;                                         //三角形直角边的边长 m
    printf("输入边长（大于等于 2 的正整数）：");    //输出提示信息
    scanf("%d",&m);
    rt(m);                                         //调用函数 rt，参数为 m
    rtt(m);                                        //调用函数 rtt，参数为 m
    return 0;
}

void rt(int n)                          //函数名取英文 right triangle 的首字母
{
    int i, j;                           //循环变量 i 控制行输出，j 控制列输出
    for(i=1;i<=n;i++)                   //输出 n 行星号，n 的值即主函数中 m 的值
    {
        for(j=1;j<=i;j++)               //第 i 行，循环 i 次，输出 i 个星号
            printf("*");                //输出 1 个星号
        printf("\n");
    }
}
```

```
void rtt(int nt)                    //函数名取英文 right triangle transfer 的首字母
{
    rt(nt+2);                       //调用函数 rt，参数为 nt+2
}
```

运行结果：

程序分析：

（1）当主函数调用 rt 函数时，实参是变量 m，本例中 m 等于 3；

（2）当主函数调用 rtt 函数时，实参是变量 m，本例中 m 等于 3；

（3）当 rtt 函数调用 rt 函数时，实参是表达式 nt+2，因为 nt 的值等于 3，所以 nt+2 等于 5，即实参为 5。

（4）实参可以是常量、变量或表达式，但要求它们有确定的值，如“rt(5);”就是用常量作为实参调用函数 rt。

3.1.2　函数的参数与变量的作用域

函数的参数分为形式参数（形参）和实际参数（实参）。形参出现在被调函数中，如前面的 rt 函数的 n 和 rtt 函数的 nt；而实参出现在主调函数中，如 main 函数的 m 和 rtt 函数的 nt+2。

当调用函数时，主调函数把实参的值赋给被调函数的形参，从而实现函数间的数据传递。

1．形参

（1）定义函数时，根据函数要实现的功能来设定形参。

（2）形参的数据类型可以是整型（int）、浮点型（float）、字符型（char）等。

（3）形参的命名规则与变量相同。

（4）当被调函数开始执行时，系统才给形参分配存储空间，即形参才开始有效。当被调函数执行完成，即调用结束时，系统将分配给形参的内存单元收回，或者说这些内存单元被释放，因此调用结束后，不能再使用这些形参。

（5）函数的形参实际就是该函数的局部变量，它的作用域是该函数的函数体。C 语言中，每一个变量都有一个作用域，即变量在内存中的生存期，或者说变量在那一段程序范围内有效。

2．实参

（1）实参与形参的类型应相同或赋值兼容。形参定义为变量时，实参可以是常量、变量或表达式。形参是 char 型时，实参为 100，就是赋值兼容的例子。

（2）变量作为实参，如例 3-3 中 main 函数调用 rt 函数时的参数 m。

（3）表达式作为实参，如例 3-3 中 rtt 函数调用 rt 函数时的参数 nt+2。

（4）常量作为实参，如果例 3-2 中的 m 等于 3，将 main 函数中的“rt(m);”改为“rt(3);”，即用常量 3 替代变量 m 来调用 rt 函数，程序运行结果不变。

（5）赋值兼容是指实参的类型与形参的类型不同，但两者之间可以正常转换。如果将例 3-2 中的“rt(m);”改为“rt('a');”，其效果相当于将“rt(m);”改为“rt(97);”。因为字符常量可以自动转换成相对应的整数，所以“rt('a');”与“rt(97);”等价。

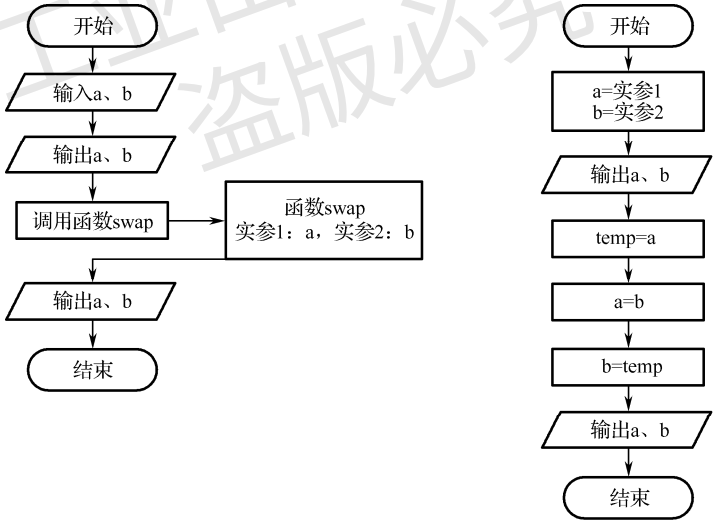
读者可以用“rt(5);”“rt('a');”“rt('a'-92);”“rt('a'-m);”等语句替换例 3-2 中的语句“rt(m);”，以比较用常量、变量或表达式作为实参调用 rt 函数的效果。

【例 3-4】局部变量与函数之间数据的传递。在 main 函数中输入两个整数，在自定义函数 swap 中交换这两个整数，并输出交换的结果。

解题思路：

为了清晰显示出变量在 main 函数和 swap 函数中的值，在调用 swap 函数之前和之后各输出一次 a、b 的值；在 swap 函数中，交换 a、b 的前后也各输出一次 a、b 的值。

流程图如图 3-4 所示。



（a）主函数流程图　　（b）swap函数流程图

图 3-4

程序代码：

```
#include <stdio.h>
void swap(int a, int b);                    //声明 swap 函数
int main(void)
```

```
{
    int a, b;
    printf("输入  a, b:");
    scanf("%d,%d", &a, &b);
    printf("在 main 函数中，调用 swap 前: a = %d, b = %d\n", a, b);
    swap(a, b);                          //以 a 为实参 1，b 为实参 2，调用 swap 函数
    printf("在 main 函数中，调用 swap 后: a = %d, b = %d\n", a, b);
    return 0;
}
void swap(int a, int b)                  //定义 swap 函数，形参 a、b 与 main 函数中的变量同名
{
    int temp;
    printf("在 swap 函数中，ab 交换之前: a = %d, b = %d\n", a, b);
    temp = a;                            //将 a 的值赋给中间变量 temp
    a = b;                               //将 b 的值赋给变量 a
    b = temp;                            //将中间变量 temp 的值赋给变量 b
    printf("在 swap 函数中，ab 交换之后: a = %d, b = %d\n", a, b);
}
```

运行结果：

```
输入 a, b:10,26
在main函数中, 调用swap前: a = 10, b = 26
在swap函数中, ab交换之前: a = 10, b = 26
在swap函数中, ab交换之后: a = 26, b = 10
在main函数中, 调用swap后: a = 10, b = 26
Press any key to continue
```

程序分析：

（1）在一个函数内部定义的变量只在该函数范围内有效，也就是说只有在该函数内才能引用它们，在该函数以外是不能使用这些变量的，称这样的变量为局部变量或内部变量。变量的有效范围也称为变量的作用域。本例中 swap 函数的形参 a、b 与 main 函数调用 swap 的实参 a、b 同名。前者是 swap 函数中的局部变量，其作用域是 swap 函数内；后者是 main 函数中的局部变量，其作用域是 main 函数内。

（2）main 函数调用 swap 函数时，将 main 函数中变量 a、b 的值赋给了 swap 函数中的变量 a、b。如果 main 函数中变量 a 等于 10，b 等于 26，通过函数调用来实现实参与形参之间的数据传递，结果是 swap 函数中的变量 a 等于 10，b 等于 26。

（3）在 swap 函数内，将 a、b 的值交换之后，a 等于 26，b 等于 10。

（4）在 main 函数内，a、b 的值并没有进行交换，所以 a 仍然等于 10，b 仍然等于 26。

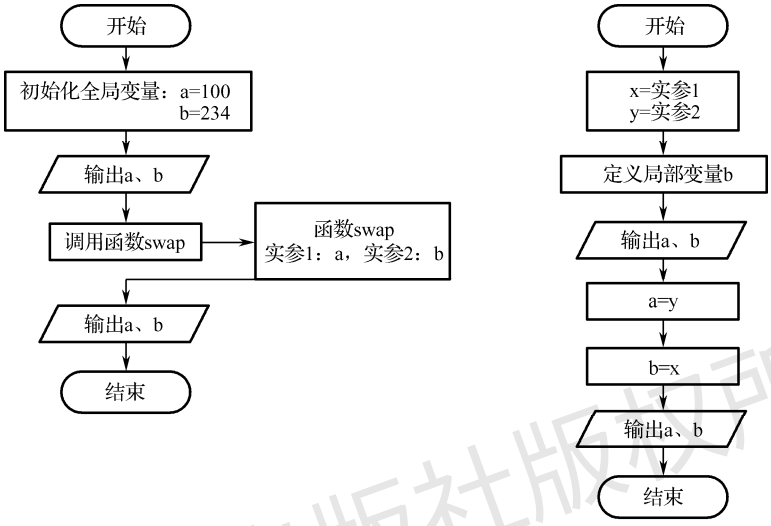
这类似于 1 班有一个李明同学，2 班也有一个李明同学。虽然名字相同，实际上是两个人。1 班考试的时候，2 班的李明不能参加；2 班考试的时候，1 班的李明不能参加。可视为两个李明的作用域不同。所不同的是，程序中 main 函数以 a 为实参对应 swap 函数中的形参 a，可以将数据从 main 函数中的 a 传递给 swap 函数中的 a。而 1 班的李明不可能将考分传递给 2 班的李明。此处比喻只是帮助读者理解局部变量的概念，不能将两者完全等同。

【例 3-5】全局变量与局部变量的关系。在 main 函数中给全局变量 a、b 赋值，在自定义函数 swap 中交换 a、b 的值，在调用函数 swap 前后分别输出 a、b 的值。

解题思路：

为了清晰显示出全局变量与局部变量的关系，在 main 函数中给全局变量 a、b 赋值，并在自定义函数 swap 中重新定义变量 b，然后交换变量 a、b 的值。在调用 swap 函数前后各输出一次 a、b 的值，在 swap 函数中交换 a、b 前后也各输出一次 a、b 的值。

流程图如图 3-5 所示。



（a）主函数流程图　　（b）swap函数流程图

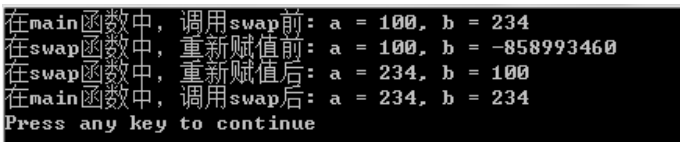
图 3-5

程序代码：

```
#include <stdio.h>
void swap(int x, int y);
int a=100,b=234;                    //初始化全局变量 a、b
int main(void)
{
    printf("在 main 函数中，调用 swap 前: a = %d, b = %d\n", a, b);
    swap(a, b);                     //以 a 为实参 1，b 为实参 2，调用 swap 函数
    printf("在 main 函数中，调用 swap 后: a = %d, b = %d\n", a, b);
    return 0;
}

void swap(int x, int y)             //定义 swap 函数，形参为 x、y
{
    int b;                          //定义局部变量 a、b
    printf("在 swap 函数中，重新赋值后: a = %d, b = %d\n", a, b);
    a=y;                            //将 y 值赋给 a
    b=x;                            //将 x 值赋给 b
    printf("在 swap 函数中，重新赋值后: a = %d, b = %d\n", a, b);
}
```

运行结果：



程序分析：

（1）初始化全局变量 a、b，即在定义变量 a、b 的同时，给变量 a、b 赋值。在函数外定义的变量是全局变量，也称为外部变量。全局变量的生存期是从定义变量的位置开始，到本源文件结束。

（2）在 main 函数中输出 a、b。

（3）在 swap 函数中定义变量 b，此变量与全局变量 b 同名，但占用不同的内存空间。

（4）在 swap 函数中，重新赋值前输出 a、b。因为存在两个变量 b 时，起作用的变量是在 swap 函数中定义的局部变量，而局部变量 b 没有赋值，所以输出的是无意义的数。在编译时，系统将显示警告信息：“local variable 'b' used without having been initialized”（使用局部变量'b'之前没有进行初始化）。

（5）将 y 值赋给 a，将 x 值赋给 b，然后输出 a、b。赋值运算执行后，在 swap 函数中 a、b 的值与 main 函数中 a、b 的值刚好相反。

（6）返回到 main 函数后，输出 a、b 的值。变量 a 的值被交换为变量 b 原来的值，而变量 b 的值不变。因为全局变量 a 在程序的整个执行过程中都有效，而全局变量 b 在程序执行 swap 函数时尽管仍然占用内存空间，但不参与相关的操作，参与操作的是 swap 函数内定义的局部变量 b，因此全局变量 b 的值没有改变。

通常，全局变量的有效范围（作用域）与生存期是一致的，当且仅当全局变量与局部变量同名时，在局部变量的有效范围内，全局变量被局部变量屏蔽。

进行模块化程序设计的目的是使程序具有较好的移植性和较强的可读性。为了实现这一目的，模块的功能要尽量单一，模块之间的相互影响要尽量少，尽可能少使用或不使用全局变量，而是通过“实参-形参”的方式来实现数据的传递和功能的调用。

*3.1.3　函数的递归调用

在调用一个函数的过程中又出现直接或间接地调用该函数本身，称为函数的递归调用。

【例 3-6】使用函数的递归调用，计算并输出正整数 n 的阶乘。

解题思路：

求 $n!$，即求 $1\times2\times3\times\cdots\times(n-1)\times n$，也可以表示为

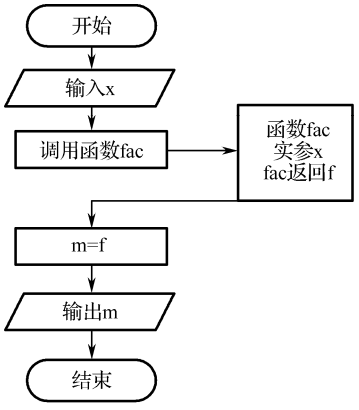
$$n!=\begin{cases}1 & (n=0,1)\\ n\times(n-1)! & (n>1)\end{cases}$$

从上式可知，当 $n>1$ 时，要求出 $n!$，需先求出$(n-1)!$；要求出$(n-1)!$，需先求出$(n-2)!$；要求出$(n-2)!$，需先求出$(n-3)!$；以此类推，直至 $n=1$，而 1！为已知数 1。上述过程称

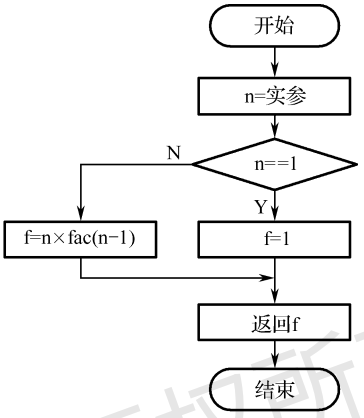
为回溯，即从 *n*!开始回溯到源头 1!。到了源头后，再从源头的已知数开始进行递推，一步步求出 2!=2×1!，3!=3×2!，…，(*n*−1)!=*n*×(*n*−2)!，直至求出 *n*!=*n*×(*n*−1)!。

定义函数 fac，fac 函数的功能是通过递归调用计算正整数 *n* 的阶乘。主函数调用 fac 函数，实参为 n；将 fac 函数的返回值赋给变量 m，然后输出变量 m 的值。

流程图如图 3-6 所示。



（a）主函数流程图



（b）fac函数流程图

图 3-6

在图 3-6（b）中，程序执行“f = n×fac(n−1)”后，并不直接“返回 f”，而是调用 fac 函数，调用函数的实参为 n−1。当再次执行 fac 函数时，函数的形参 n=n−1。

程序代码：

```
#include <stdio.h>
int fac(int n);                          //声明函数 fac
int main()
{
    int x,m;
    printf("输入一个正整数：");
    scanf("%d",&x);
    m = fac(x);                          //调用 fac 函数，实参为 x，返回值赋给 m
    printf("正整数 %d 的阶乘：%d\n",x,m);
    return 0;
}

int fac(int n)                           //定义函数 fac，形参 n 和返回值均为 int 型
{                                        //函数功能：计算 n!
    int f;
    if(n==1)
        f=1;
    else
        f = n*fac(n-1);                  //调用 fac 函数，实参为 n-1，返回值乘 n 后赋给 f
    return f;                            //函数返回 f
}
```

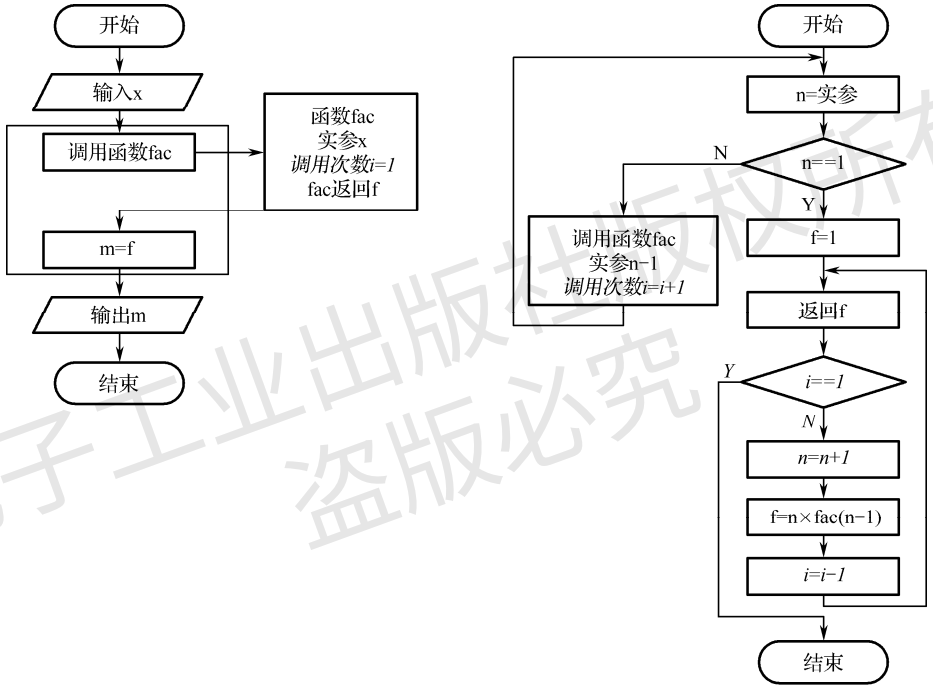
运行结果：

```
输入一个正整数：5
正整数 5 的阶乘：120
Press any key to continue
```

程序分析：

（1）输入 5，表示要调用 5 次 fac 函数，调用函数的实参分别是 5、4、3、2、1。前 4 次调用 fac 函数都没有将 fac 函数中的流程走完整，因为执行到“f = n*fac(n−1);”语句时，又重新调用 fac 函数，直至第 5 次调用 fac 函数，调用函数的实参为 1，这一次调用 fac 函数的返回值为 1。

（2）程序执行的实际流程图如图 3-7 所示。流程图中的斜体字部分在程序源代码中并没有相应的语句，但编译系统会进行相应的操作，以保证程序能够被正确地执行。



（a）主函数流程图　　（b）fac函数流程图

图 3-7

第 5 次调用 fac 函数，函数返回 1 后，重新执行前面的 4 次调用没有完成的语句，将函数的返回值依次替代“f = n*fac(n-1);”中的“fac(n-1)”：

将 fac(1)的返回值 1 替代 fac(2-1)，求出 fac(2)的返回值 2；

将 fac(2)的返回值 2 替代 fac(3-1)，求出 fac(3)的返回值 6；

将 fac(3)的返回值 3 替代 fac(4-1)，求出 fac(4)的返回值 24；

将 fac(4)的返回值 24 替代 fac(5-1)，求出 fac(5)的返回值 120。

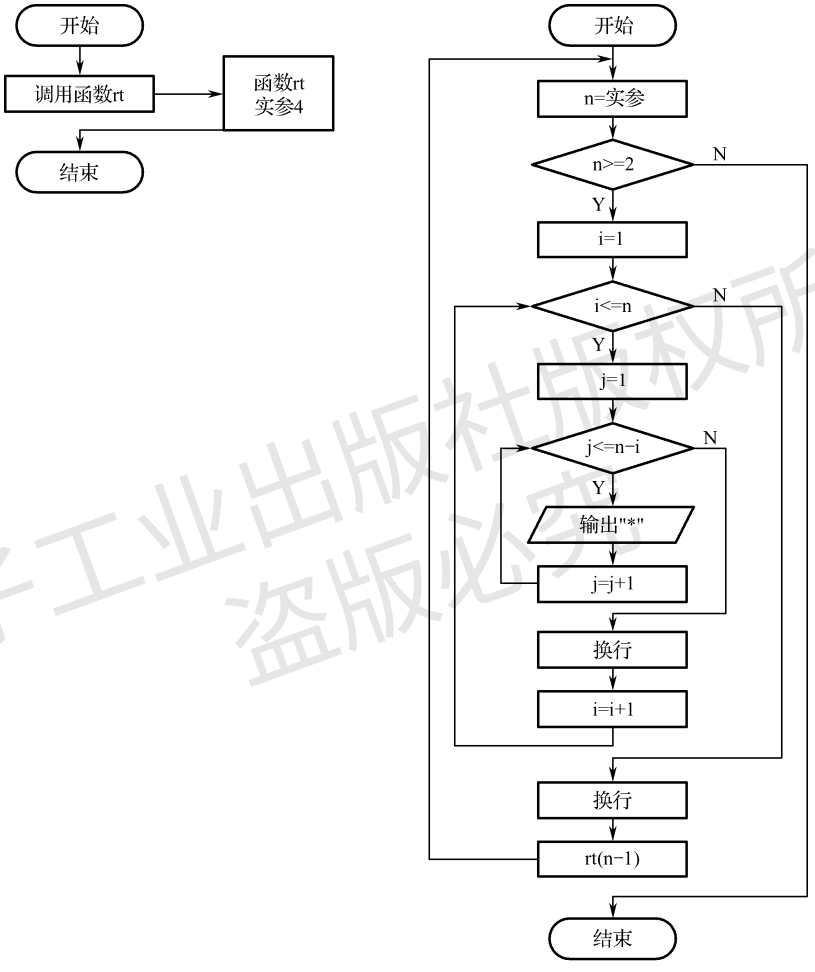
（3）递推过程中这种依次替代的程序设计方法称为迭代法。本章例 3-1 就是采用迭代法求正整数 1 到 100 之和。采用函数递归调用的程序中一定会用到迭代，而采用迭代法设计的程序不一定会用到函数的递归调用。多数情况下，这两种方法可以互相替代，

例 3-1 也可以用函数递归的方法求解，而例 3-6 也可以采用迭代法求解。

【例 3-7】采用递归调用函数方法输出 3 个直角三角形图案，第 1 个三角形的直角边边长为 4，另两个三角形的边长依次为 3 和 2。

解题思路：

可采用例 3-2 中介绍的 rt 函数绘制直角三角形图案，把 rt 函数中的“for(j=1; j<=i;j++)”改为“for(j=1;j<=n-i;j++)”，将生成的一个倒立的三角形。当 n>=2 时才绘制三角形图案，绘制完图案后再以 n-1 为实参调用 rt 函数。程序的流程图如图 3-8 所示。



（a）主函数流程图　　（b）rt函数流程图

图 3-8

程序代码：

```
#include <stdio.h>
void rt(int n);
void main()
{
    rt(4);                                  //调用 rt 函数，实参为 4
}
```

```
void rt(int n)
{
    int   i,j;
    if ( n >= 2)                          //保证三角形边长大于等于 2
    {
        for(i = 0; i < n; ++i)
        {
            for(j = 0; j < n - i; ++j)    //在例 3-2 中，“j<n-i”为“j<i”
                printf(" *");             //输出一个带空格的星号
            printf("\n");
        }
        printf("\n");
        rt(n-1);                          //调用 rt 函数，实参为 n-1
    }
}
```

运行结果：

程序分析：

（1）主函数调用 rt 函数时，可用变量 m 作为实参，则生成 m-1 个三角形图案。

（2）本例中生成的三角形为倒立的三角形。如果将 rt 函数中的“j<n−i”改为“j<i”，则生成正立的三角形。

（3）本例在例 3-2 的基础上增加了函数的递归调用，在 rt 函数内，用实参 n-1 调用 rt 函数本身。

（4）如果将“rt(n-1);”语句调整到循环语句前，则绘制三角形的顺序为由小到大。思考为什么会产生这样的结果？

3.2　C 程序的构成与运行

本节的大部分内容在前面介绍过，由于分散在不同的章节，难于形成一个整体的印象，所以有必要集中起来进行系统的说明。

3.2.1　C 程序的构成

从整体上，C 程序由一个或多个源程序文件构成，如图 3-9 所示。一个规模较小的

程序通常只包含一个源程序文件，前面介绍的例题都只包含一个源程序文件。规模比较大的程序往往包含多个源程序文件。分为多个源程序文件有两个好处，一是将大程序划分成较小的程序后降低了编写和查错的难度，提高了效率；二是便于团队合作开发，每一个团队成员编写和编译一部分程序。

一个源程序文件可以包含 3 个部分：预处理命令、函数和数据的声明、函数。函数由函数首部和函数体组成，函数体通常包含数据声明和执行语句。

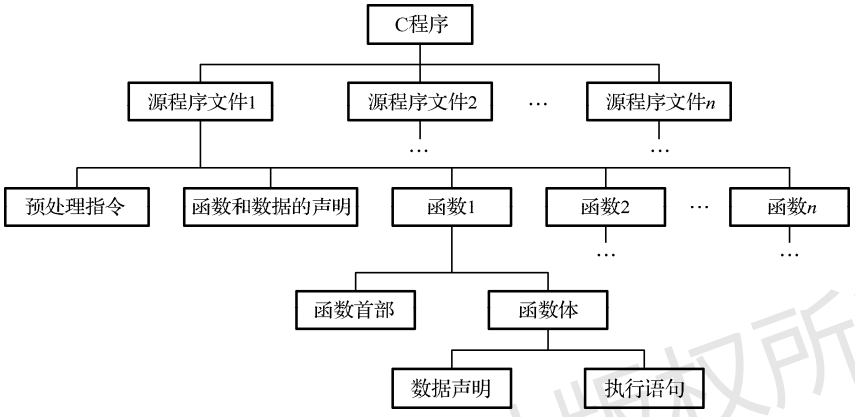


图 3-9

3.2.2 多个源程序文件的 C 程序

在前面的例题中，所有的程序都只有一个源程序文件，下面通过实例介绍多个源程序文件的设计方法。

【例 3-8】多个源程序文件的 C 程序示例。

解题思路：

用一个简单的加、减运算的程序来说明多个源程序文件的程序结构。源程序文件 c3-08.c 包含主函数 main 和自定义函数 out，函数 out 的功能是输出计算结果。源程序文件 arith.c 包含 sum 和 differ 两个自定义函数，函数 sum 的功能是求两个数之和，函数 differ 的功能是求两个数之差。

程序代码：

（1）源程序文件 1：c3-08.c。

```
#include <stdio.h>                          //本程序调用了标准库函数 scanf 和 printf
#include <math.h>                           //本程序调用了标准库函数 abs 求一个数的绝对值
#include "arith.c"                          //本程序调用了 arith.c 文件中的函数 sum 和 differ
void out(int);                              //声明 out 函数
int main()                                  //主函数首部
{                                           //此花括号到下一个花括号之间为 main 函数的函数体
    int a,b;                                //数据声明
    printf("输入两个整型数 a,b: ");          //执行语句
    scanf("%d,%d",&a,&b);
    printf("a 加 b 等于 %d\n", sum(a, b));      //调用 sum 函数计算 a、b 之和
```

```
    out(differ(a,b));                    //把 differ 函数的返回值作为调用 out 函数的实参
    return 0;
}

void out(int c)                          //out 函数首部
{
    if (c>=0)
        printf("a 比 b 大 %d\n", c);
    else
    {
        c=abs(c);          //调用 abs 函数，将 c 的绝对值赋给 c
        printf("a 比 b 小 %d\n", c);
    }
}
```

（2）源程序文件 2：arith.c。

```
int sum(int a, int b)          //sum 函数首部
 {
     return a + b;             //返回 a 与 b 之和
 }
int differ(int a, int b)       //differ 函数首部
 {
     return a - b;             //返回 a 减 b 的值
 }
```

运行结果：

```
输入两个整型数 a,b: 35,60
a 加 b 等于 95
a 比 b 小 25
Press any key to continue
```

程序分析：

（1）源程序文件。

本程序由两个源程序文件 c3-08.c 和 arith.c 构成。

（2）预处理命令#include。

#include 的作用是把""或<>内指定的文件包含到该程序中，使之为该程序的一部分。被包含的文件可以是系统提供的头文件，如 stdio.h 和 math.h 文件；也可以是自己编写的头文件或源程序文件，如 arith.c。

① 如果使用绝对路径，则使用""和<>的效果相同，例如：

```
#include<f:/file/arith.c>
```

和

```
#include "f:/file/arith.c"
```

都是到“f:/file/”文件夹中搜索“arith.c”文件。

② 如果不使用绝对路径，则两者的搜索顺序不同。

#include "arith.c"的主要搜索顺序是：在发出 include 指令的程序所在的文件夹内搜索；如果找不到，再在编译器设置的 include 路径内搜索；如果找不到，再在系统的

INCLUDE 环境变量内搜索。

#include <arith.c>的主要搜索顺序是：在编译器设置的 include 路径内搜索；如果找不到，再在系统的 INCLUDE 环境变量内搜索。

（3）函数和数据的声明。

如果一个函数的定义位于调用它的函数或语句（如本例中的 main 函数）后，则需要在调用该函数前用函数原型进行函数声明，如本例中的“void out(int);”语句。

在函数外进行数据声明，即定义全局变量，可参见例 3-5。

（4）函数。

C 程序的执行是从 main 函数开始的，如果在 main 函数中调用了其他函数，在调用后流程应返回到 main 函数，并且在 main 函数中结束整个程序的运行。

所有函数都是平行的，即在定义函数时是分别进行的，是互相独立的。一个函数并不从属于另一个函数，即函数不能嵌套定义。函数间可以互相调用，但不能调用 main 函数。main 函数是被操作系统调用的。

通常函数分为库函数和自定义函数。库函数由系统提供，如本程序中用到的 printf、scanf 和 abs 等函数。不同的 C 语言编译系统提供的库函数的数量和功能略有差异。自定义函数是程序员根据需要自己编写的函数，以解决特定的问题，如本程序中的 sum、differ 和 out 函数。

（5）函数首部。

函数首部包括函数的类型名、函数名和形参列表。

形参列表可以为空，即主调函数不向被调函数传递数据。

（6）函数体。

函数的功能在函数体中实现，函数体通常包含数据声明和语句。

函数由函数首部和函数体组成。

（7）数据声明。

在函数体内进行数据声明，即定义局部变量，如本程序中的“int a,b;”。

（8）语句。

C 程序中的语句分为：控制语句、函数调用语句、表达式语句、空语句、复合语句等 5 类。

在每个数据声明和语句的最后必须有一个分号。

（9）注释。

虽然“注释”没有出现在图 3-9 所示的 C 程序的构成图中，但并不表示它不重要。在程序中进行注释可以提高代码的可读性，有助于程序员之间的沟通交流。

常用的注释方式有以下两种：

① 单行注释。格式是：

```
//
```

② 多行注释。格式是：

```
/*
…
*/
```

多行注释不能嵌套使用，“/*”总是和离它最近的“*/”配对，将两者之间的所有符号作为注释。

需要注意的是字符串中的“//”和“/*　*/”不是注释符号。

3.2.3 C 程序中的语句

C 程序中的语句如图 3-10 所示。

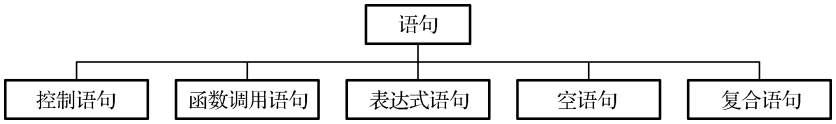


图 3-10

1．控制语句

① if()…else…（条件语句）。
② for()…（循环语句）。
③ while()…（循环语句）。
④ do…while ()（循环语句）。
⑤ continue（跳过本次循环语句）。
⑥ break（中止执行 switch 或循环语句）。
⑦ switch（多分支选择语句）。
⑧ return（返回语句）。
⑨ goto（转向语句，在结构化程序中基本不用 goto 语句）。

2．函数调用语句

函数调用语句由一个函数调用加一个分号构成。如果没有分号，则仅仅是一个函数调用，如：

```
printf("输入两个整型数  a,b: ")
```

给这个函数调用加一个分号，就成为一个函数调用语句，如：

```
printf("输入两个整型数  a,b: ");
```

3．表达式语句

表达式语句由一个表达式和一个分号构成。赋值语句是最典型的表达式语句。

如果没有分号，则仅仅是一个赋值表达式，如：

```
c=abs(c)
```

给这个赋值表达式加一个分号，就成为一个赋值表达式语句，即赋值语句，如：

```
c=abs(c);
```

4．空语句

只有一个分号的语句是空语句，如：

```
;
```

空语句可用来作为循环语句中的循环体（循环体是空语句，表示循环体什么也不做）。

5．复合语句

可以用{ }把一些语句和声明括起来使其成为复合语句（又称语句块）。

```
{
    c=abs(c);                          //调用 abs 函数，将 c 的绝对值赋给 c
    printf("a 比 b 小 %d\n", c);
}
```

复合语句常用在 if 语句或循环中，此时程序需要连续执行一组语句。

3.3 综合实例——控制台程序设计

本节介绍两个控制台程序设计的综合实例。通过这两个实例，综合运用前面介绍的知识，了解和学习编写程序的基本方法。

3.3.1 通过菜单选择生成几何图案

本实例以前面例题中介绍的生成图案的程序为基础，扩展到生成更多的几何图案，并创建一个菜单，通过菜单选择生成哪一种几何图案。

【例 3-9】编写程序，实现从菜单中选择生成不同的几何图案，如三角形图案、平行四边形图案或菱形图案等，并且几何图案的大小可变。

解题思路：

本程序包括 4 个模块，分别是主函数模块、三角形图案模块、菱形图案模块和平行四边形图案模块。模块结构如图 3-11 所示。

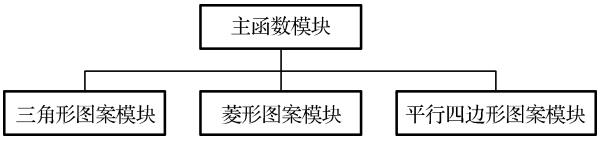


图 3-11

每个模块都有一个单独的源程序文件。

文件 c3_09.c 包含主函数模块，在主函数中实现菜单功能。

文件 c3_09_1.c 包含三角形图案模块，功能是生成三角形。

文件 c3_09_2.c 包含菱形图案模块，功能是生成菱形。

文件 c3_09_3.c 包含平行四边形图案模块，功能是生成平行四边形。

程序代码：

（1）源程序文件 1：c3_09.c。

```
#include <stdio.h>
#include <stdlib.h>                              //引用 stdlib.h 文件
#include <conio.h>                               //引用 conio.h 文件
#include "e:/创意编程/第 3 章/c3_09_1.c"      //引用 c3_09_1.c 文件
#include "e:/创意编程/第 3 章/c3_09_2.c"      //引用 c3_09_2.c 文件
#include "e:/创意编程/第 3 章/c3_09_3.c"      //引用 c3_09_3.c 文件
int main(void)
{
    char choose='\0';
    do
    {
        system("cls");
        printf("\n |~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|\n");
        printf(" |       请输入选项编号（0～3）     |\n");
        printf(" |~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|\n");
        printf(" |              1:三角形               |\n");
        printf(" |              2:菱形                 |\n");
        printf(" |              3:平行四边形           |\n");
        printf(" |              0:退出                 |\n");
        printf(" |~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|\n");
        choose=getch();
        switch(choose)
        {
            case '1': printf("  您选择了三角形\n");
                t1();        break;          //本行有两条语句
            case '2': printf("  您选择了菱形\n");
                t2();        break;
            case '3': printf("  您选择了平行四边形\n");
                t3();        break;
            case '0': printf("  退出程序\n");
                exit(0);                          //循环的唯一出口
            default : printf(" %c 为非法选项！ ",choose);
        }
        printf(" \n  按任何键返回\n");
        getch();
    }while(1);                                    //循环条件永远成立
    return 0;
}
```

（2）源程序文件 2：c3_09_1.c。

```
int t1( )                                         //生成三角形图案
{
    int i,k,n;
    printf("  请输入三角形的行数:");
    scanf("%d",&n);
    for(i=0; i<=n;i++)
    {
        printf("     ");                          //每行前面都留 5 个空格
```

```
            for(k=0; k<i;k++)
                printf("*");
            printf("\n");
        }
        return 0;
}
```

（3）源程序文件 3：c3_09_2.c

```
int t2( )                                          //生成菱形图案
{
    int i, j, k, m, n, size;
    printf ("  请输入菱形的行数（奇数）: ");          //提示信息
    scanf ("%d", &size);                           //输入菱形的大小（行数）
    if (size <= 0 || size % 2 == 0)                //判断输入是否合理
        printf ("\n  菱形的行数必须是正奇数!\n ");
    else
    {
        printf("\n");
        for (i = 1; i <= size; i++)                //控制行数
        {
            n = (i <= (size+1)/2) ? i : size-i+1;  //每行中"*"的个数
            n = 2 * n - 1;
            m = (size - n) / 2 + 5;                //每行打印"*"前应打印的空格数
            for (k = 1; k <= m; k++)               //打印每行前面的空格
                printf (" ");
            for (j = 1; j <= n; j++)               //打印每行的"*"
                printf ("*");
            printf ("\n");                         //打印完 1 行后，换行
        }
    }
    return 0;
}
```

（4）源程序文件 4：c3_09_3.c。

```
int t3 ( )                                         //生成平行四边形图案
{
    int i,j,k,n;
    printf("  请输入平行四边形的行数:");
    scanf("%d",&n);
    printf("\n");
    for(i=0; i<=n;i++)
    {
        printf("          ");
        for(j=0; j<=i;j++)
            printf(" ");
        for(k=0; k<=n;k++)
            printf("*");
        printf("\n");
    }
    return 0;
}
```

运行结果如下。

（1）程序的主菜单如图 3-12 所示。

（2）输入 1，然后输入 5，效果如图 3-13 所示。

图 3-12

图 3-13

（3）输入 2，然后输入 7，效果如图 3-14 所示。

（4）输入 3，然后输入 6，效果如图 3-15 所示。

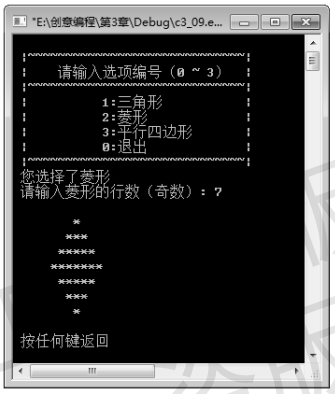


图 3-14

图 3-15

（5）输入 2，然后输入 6，效果如图 3-16 所示。

（6）输入 a，效果如图 3-17 所示。

（7）输入 0，效果如图 3-18 所示。

图 3-16

图 3-17

图 3-18

程序分析：

（1）预处理命令#include<stdlib.h>引用头文件 stdlib.h。

程序中使用函数 system("cls")来调用系统命令清屏，即将屏幕上显示的内容全部清除。该函数包含在头文件 stdlib.h 中。

（2）预处理命令#include<conio.h>引用头文件 conio.h。

程序中使用函数 getch()来读取用户输入的字符。getch()函数是一个不回显函数，当用户按下某个键时，函数自动读取该键所对应的字符，无须回车。不回显函数是指用户输入的这个字符不会显示在屏幕上。该函数包含在头文件 conio.h 中。

（3）本程序除了包含主函数的源程序文件 c3_09.c，还有 3 个源程序文件，因此在 c3_09.c 的主函数前需要用预处理命令#include 引用其他 3 个源程序文件。

采用多个源程序文件，充分利用了模块化程序设计的优点，每个模块都比较简单，便于编程、调试。

（4）主函数模块中采用了 do…while 语句与 switch、case 语句嵌套的框架结构。这种框架结构不仅可以实现本程序中的菜单功能，还可以用于游戏软件、管理系统软件等。

3.3.2 简单的射击游戏——飞弹和爆炸效果

【例 3-10】编写程序，在屏幕的第 1 行显示一个靶子，当从下方射出的飞弹击中靶子时，产生爆炸效果。

解题思路：

（1）在第 1 行显示 5 个星号表示靶子。

（2）从第 15 行开始，显示 1 个由星号组成的三角形表示飞弹。

（3）输入选项 1、2、3，设置飞弹移动的速度。1 为慢速，2 为中速，3 为快速。

（4）用 sleep()函数控制飞弹移动的速度。

（5）用循环结构控制飞弹向上移动，每循环 1 次，飞弹向上移动 1 行。

（6）当飞弹移动到第 1 行时，表示击中靶子。显示散开的星号表示爆炸的效果。

程序代码：

```
#include <stdio.h>
#include <stdlib.h>                                    //引用 stdlib.h 文件
#include <windows.h>                                   //引用 windows.h 文件
int main()
{
    int i, j, k;                                       //循环变量
    int v, n, m;                                       //控制速度、行数和空格的变量
    int size=5,nn=20;
    do
    {
        printf(" 输入移动速度  1.低速  2.中速  3.高速:");
        scanf("%d",&v);
        if (v==1 || v==2 || v==3)
            break;                                     //输入了正确的值后，跳出循环
        else
            system("cls");
            printf (" 移动速度的取值范围（1—3）\n\n");
    }while(1);
    v=150/v;                                           //飞弹移动的速度为每 150/v 毫秒 1 行
    for(i=15; i>1; i--)
```

```
    {
        system("cls");
        for (j = 1; j <= nn; j++)                    //打印第 1 行靶子前面的空格
            printf (" ");
        for (j = 1; j <= size; j++)                  //打印第 1 行的靶子
            printf ("*");
        for (j = 1; j < i; j++)                      //控制飞弹上面的行数
            printf ("\n");
        for (j = 1; j <= size; j++)                  //打印飞弹
        {
            n = 2 * j - 1;                           //每行中"*"的个数
            m = (size - n) / 2 + nn;                 //每行打印"*"前应打印的空格数
            for (k = 1; k <= m; k++)                 //打印每行前面的空格
                printf (" ");
            for (k = 1; k <= n; k++)                 //打印每行的"*"
                printf ("*");
            printf ("\n");                           //打印 1 行后，回车换行
        }
        sleep(v);
    }
    system("cls");
    for(i=0; i<size; i++)                            //用 size 控制散开弹片的行数
    {
        for (j = 1; j <= nn-(i*3+1); j++)            //打印每行前面的空格
            printf (" ");
        for (j = 1; j < size+(i*3+1); j++)           //打印每行散开的弹片
            printf ("* ");                           //每个"*"后带一个空格
          printf ("\n");
        Sleep(v);
    }
    printf("\n");
    return 0;
}
```

运行结果：

（1）程序运行后显示提示，输入移动速度，如图 3-19 所示。

（2）输入 2，回车，飞弹向上移动，如图 3-20 所示。

图 3-19

图 3-20

（3）飞弹到靶子的效果，如图 3-21 所示。

（4）爆炸的效果，如图 3-22 所示。

图 3-21

图 3-22

程序分析：

（1）预处理命令#include<stdlib.h>引用头文件 stdlib.h。

使用函数 system("cls")来调用系统命令清屏。

（2）预处理命令#include<windows.h>引用头文件 windows.h。

使用函数 sleep()来控制飞弹移动的速度，该函数包含在头文件 windows.h 中。

（3）输入选项 1、2、3，设置飞弹移动的速度。

当 v==1 时为慢速，当 v==2 时为中速，当 v==3 时为快速。

重新给 v 赋值 v=150/v，用 v 作为参数调用 sleep()函数。因为 sleep()函数的参数以毫秒为单位，所以当输入选项为 3 时，飞弹速度为每 50 毫秒向上移动 1 行。

3.4 习题

1．什么是函数？什么是主函数?

2．编写一个程序，生成一个如图 3-23 所示的倒立等腰三角形，大小由用户确定，并画出程序的流程图。

3．编写一个程序，生成一个如图 3-24 所示的“回”字形，大小由用户确定，并画出程序的流程图。

图 3-23

图 3-24

4．画出程序 c3_09.c 的流程图。

5．画出程序 c3_09_1.c 的流程图。

6．画出程序 c3_09_2.c 的流程图。

7．画出程序 c3_09_3.c 的流程图。

8．画出例 3-10 的流程图。

9．编写一个程序，生成一个自己喜欢的图形。