第5章 chapter 5

函　数

5.1 本章目标

- 掌握使用函数对程序进行模块化编程的方法。
- 理解函数调用的参数传递、局部变量与全局变量的作用范围。
- 使用 Visual Studio 调试函数编程，发现程序执行上的逻辑错误。

5.2 函数的使用

使用函数在编程中有很多优点，包括但不限于以下几点。

（1）代码简洁与可重用性：通过将复杂的逻辑或任务分解为独立的函数，可以使代码更加简洁和易于理解。此外，一旦定义了函数，就可以在程序的其他部分重复使用，提高了代码的可重用性。

（2）模块化编程：函数是模块化编程的基础。通过将代码划分为不同的函数，可以更容易地管理和维护代码。每个函数都执行特定的任务，使得代码更加模块化和可组织。

（3）提高可读性：良好的函数命名和注释可以使代码更容易阅读和理解。其他开发人员可以更容易地理解你的代码，这也有助于代码的交接和维护。

（4）抽象和封装：函数提供了一种抽象和封装机制，可以隐藏不必要的细节，只暴露必要的接口。这使得代码更加清晰，也更容易进行错误排查和调试。

（5）易于测试和维护：由于每个函数都执行特定的任务，因此可以更容易地编写单元测试来验证函数的正确性。这有助于在修改或扩展代码时确保功能的正确性。

（6）支持并行和并发编程：在一些编程语言中，函数可以作为并发或并行执行的单元，使得程序能够更有效地利用多核处理器或分布式计算资源。

总的来说，使用函数可以使代码更加简洁、可重用、可维护和可扩展。这也是函数式编程受到广泛欢迎的原因之一。

5.2.1 使用函数提高复用性

使用函数求三个数的最大值。

以下程序只有一个 main 函数，功能是：用户输入三个整数，程序输出其中的最大值。程序为：

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int main()
{
    int x,y,z,max;
    printf("input 3 numbers:\n");
    scanf("%d%d%d",&x,&y,&z);
    max =x>y?x:y;
    if (z>max)
        max =z;
    printf("maxmum=%d",max);
    return 0;
}
```

求三个整数的最大值的代码段可能被经常使用，或者为了使程序的模块化更加好，从而使编程思路更加清晰，可将这部分代码转移到一个函数中，在 main 中调用该函数即可。

为了将以上程序改为使用自定义函数，需要首先清楚函数的功能，才能决定该把哪几行代码放到自定义函数中，哪些代码保留在 main 中。

要使用自定义函数，需要以下三个步骤。

(1) 定义函数。

自定义函数如下：

```
int getMax(int x,int y, int z)
{
    int max;
    max =x>y?x:y;
    if (z>max)
        max =z;
    return max;
}
```

以上函数的作用为：接收三个整数作为参数，经过运算后，返回三者中的最大值。

(2) 调用函数。

如果需要计算 a,b,c 三者的最大值，那么，可将 a,b,c 三个变量作为实参，传递给 x,y,z 三个形参，以此调用 getMax 函数，代码如下：

```
int a,b,c;
scanf("%d%d%d",&a,&b,&c);
getMax(a,b,c);
```

(3) 使用返回值。

部分函数不返回值，函数类型为 void；大部分函数都有返回值，函数类型可为 int、double、char 等。

以上 getMax 函数有一个返回值，类型为 int，我们需要得到这个返回值，并输出。可以使用一个变量 max 获得返回值，如下：

```
int max;
max=getMax(a,b,c);
```

这样，整个程序包含两个函数，首先定义 getMax 函数，在 main 函数中，只需要接收用户输入、调用 getMax 函数，再输出结果即可。

为了显示改写的过程，下面将原程序和使用函数的程序并列放到一起进行对比，如图 5-1 所示。

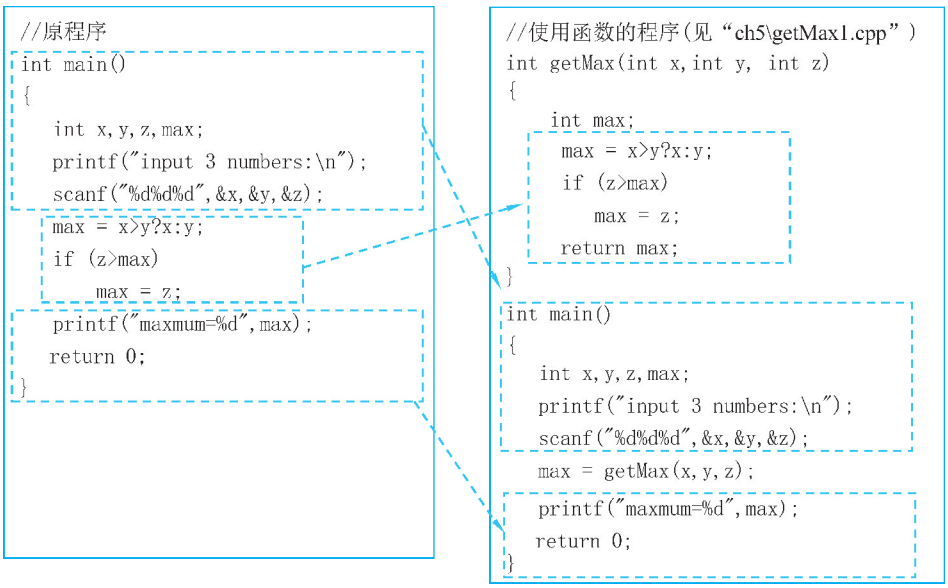

图 5-1　提炼出函数的示意图

图 5-1 中，main 函数的变量名仍然使用了 x、y、z(没有修改成 a、b、c，主要是为了让程序的改动最小)，与 getMax 函数中的变量名相同，它们不会产生冲突。

模块化编程的思想是：每个程序员负责自己的独立模块，模块与模块之间不会产生干扰，即使使用了同名的局部变量，也不会干扰。如果使用了全局变量，可能相互干扰，基于此，建议在程序中尽量少使用全局变量。

以上的例子中，函数将返回一个值，所以函数的调用可以作为一个表达式，和其他普通表达式一样参与运算。比如以下代码：

```
if (getMax(x,y,z)>80)
    printf("high score");
else
    printf("need more work");
```

在以上例子中，getMax 的返回值，用于关系表达式进行大小的比较。因为 getMax 函数的返回值是 int 类型的值，它可以像一个普通的 int 型常量一样参与各种运算，包括作为函数的实参传递给形参，如以下程序(见“ch5\getMax2.cpp”)：

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int getMax(int x, int y)
```

```
{
    return x >y ? x : y;
}
int main()
{
    int x, y, z, max;
    printf("input 3 numbers:\n");
    scanf("%d%d%d", &x, &y, &z);
    max =getMax(getMax(x, y), z);
    printf("maxmum=%d", max);
    return 0;
}
```

以上代码中，定义的 getMax 为求两个数的较大值，为了求三个数的最大值，可将 getMax 嵌套调用，首先调用 getMax 求出 x 和 y 的较大值，再将它的返回值和 z 一起调用 getMax 求得最后的较大值。这里，getMax 的返回值又作为实参调用 getMax 函数。

类似以上改写过程，我们可以将一些常用的功能重写为函数，供自己或他人调用。C、C++ 语言的库函数即完成此功能，比如求绝对值的函数等一些数学函数，或字符串处理函数等。

前面章节学习的一些程序，比如判断素数、判断闰年等，都可以改写成函数，在将来可以直接调用，以此达到代码复用的目的。

除了代码复用，为了代码的清晰，我们也需要用到函数。在编程时，一般一个函数的代码长度不要超过一屏所能看到的范围，最长不要超过一屏半的长度。如果一个函数过长，将使得程序员难以看到一个函数的全貌，从而产生一些不该有的错误。因此，为了将一个长函数变短，需要将其中的一些相对独立的功能抽出来，定义为用户函数。

此外，使用函数进行模块化编程，也可使编程的思路更加清晰。

5.2.2 模块化编程

为理解函数的作用，掌握函数在编程中的用法，下面继续使用实例进行讲解。

(1) 使用函数判断素数。

4.2 节中求 100～200 的全部素数的程序中，使用了二重循环，其中外循环从 100 到 200 对所有整数循环，内循环判断整数 m 是否是素数。算法如下：

```
int m;
for(m=100;m<=200;m++)
{
      //判断 m 是否是素数
      //如果是素数，则输出到屏幕
}
```

以上算法中，“判断 m 是否是素数”是一个较复杂的逻辑，若将这个逻辑直接嵌在循环中，则使得循环的逻辑也很复杂。如果将它独立成一个函数，循环就变得很简洁，代码如下：

```
int m, result;
```

```
for(m=100;m<=200;m++)
{
      result=isPrime(m);
      if (result) printf("%d ", m);
}
```

只需要在一个自定义函数中,写好 isPrime 判断素数的算法即可,这样程序结构显得很清晰。

图 5-2 是改写的过程(源代码见"ch5\primeFunc.cpp"):

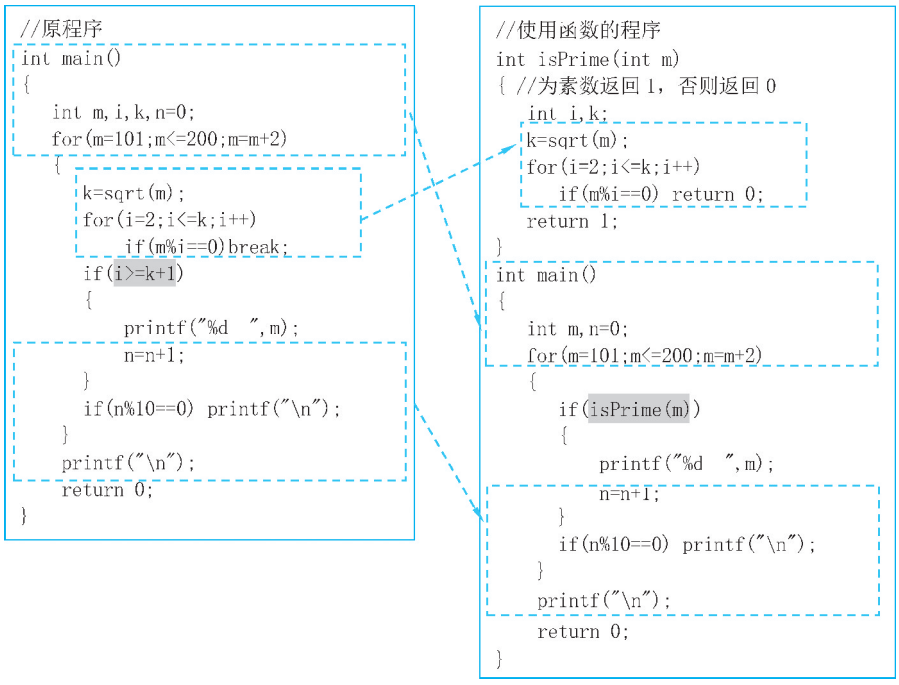

图 5-2 提炼出函数的示意图

(2) 由用户输入年和月,程序输出它的下一个月的最大天数。比如输入 2019 年 7 月,则它下一个月的最大天数为 31。

为了完成这个程序,我们先思考一个大概的程序流程,流程图如图 5-3 所示。

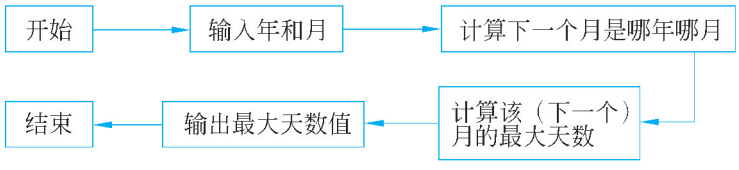

图 5-3 求下一个月的最大天数的流程图

根据以上流程图,其中的输入输出只需要一个语句即可实现,而两个计算操作需要进一步细化。下面我们将每一个操作都定义成一个用户函数,代码如下:

```
int getNextMonth(int inYear, int inMonth)
{   //注意,由于函数只能返回一个值,返回的值中包含了年和月,比如返回 202401
```

```
    int outYear, outMonth;
    if (inMonth ==12)
      {
        outYear =inYear +1;
        outMonth =1;
      }
    else
      {
        outYear =inYear;
        outMonth =inMonth +1;
      }
   return outYear * 100 +outMonth;  //将年和月组装成一个整数返回
}
int getDays(int inYear, int inMonth)
{  int days;
   switch(inMonth)
   {
      case 1:case 3:case 5:case 7:case 8:case 10:case 12:
          days=31;break;
      case 4:case 6:case 9:case 11:
          days=30;break;
      case 2:
          if (isLeap(inYear))  //这里需要判断是否为闰年,再调用一个函数
              days =29;
          else days =28;
   }
   return days;
}
```

在 getDays 函数中,调用了另一个函数 isLeap 判断是否为闰年。需要注意,isLeap 函数应该位于 getDays 函数之前。isLeap 函数的内容如下:

```
int isLeap(int year)
{
    int leap;
    if ((year%4 ==0 && year%100 !=0) || (year%400 ==0))  leap=1;
    else  leap=0;
    return leap;
}
```

定义了以上函数,main 函数变得非常简洁,如下(完整的程序见 ch5\getNextMonthDays.cpp):

```
int main(){
    int year, month, nextMonth, days;
    printf("please input year and month:");
    scanf("%d%d", &year, &month);
    nextMonth =getNextMonth(year, month);
    //由于 nextMonth 为 202401 的形式,所以下面调用函数时要拆分成年和月
    days =getDays(nextMonth/100, nextMonth %100);
    printf("next month has %d days \n", days);
    return 0;
}
```

从以上程序编写过程可知,使用函数符合人们的思维习惯,编程思路更清晰。

5.2.3 变量作用范围

本节考虑两种变量：局部变量和全局变量。在函数内部定义的变量(包括形参)是局部变量,它的作用范围局限在函数内部,一旦程序执行流程出了函数,局部变量不再存在(未考虑静态局部变量的情况)。而全局变量定义在函数外,在变量定义的位置之后的所有函数都能引用该变量。一般地,全局变量在文件的最前面(所有函数之前)定义。

下面通过一个具体的例子理解局部变量和全局变量的区别。

```
int x=5,y=10;
void exchange(int x, int y)
{  int t;
   t=x; x=y; y=t;
}
void swap()
{  int t;
   t=x; x=y; y=t;
}
void print()
{
    printf("x=%d, y=%d\n", x, y);
}
int main()
{
    int x =1, y =2;
    exchange(x, y);
    printf("local x=%d, local y=%d\n", x, y);
    swap();
    print();
    return 0;
}
```

以上程序中,三个地方都定义了 x 和 y,需要区分清楚,定义在 main 函数内部的变量,以及 exchange 函数的形参都是局部变量,而在程序最前面定义的 x 和 y 是全局变量,在 swap 和 print 函数中引用的 x 和 y 是该全局变量。

以上程序运行结果为：

```
local x=1, local y=2
x=10, y=5
```

注意：在 main 函数中,调用 exchange 函数交换 x 和 y 的值,但该交换只影响了 exchange 函数内部的局部变量的值,对于 main 函数的 x 和 y 变量的值没有影响。

以下用一个例子演示如何使用全局变量传递数据。

在 5.2.2 节的程序 getNextMonthDays.cpp 中,getNextMonth 需要返回两个值,包括年和月,但是一个函数只能返回一个值,所以我们将年和月组装在一起,作为一个整数返回。这种返回值的方式很不方便,以下改成使用全局变量的方式从函数中返回值。如下：

```
int monthNext, yearNext;
void getNextMonth(int inYear, int inMonth)
{
```

```
    if (inMonth ==12)
    {
        yearNext =inYear +1;
        monthNext =1;
    }
    else
    {
        yearNext =inYear;
        monthNext =inMonth +1;
    }
}
int main()
{
    int year, month;
    printf("please input year and month:");
    scanf("%d%d", &year, &month);
    getNextMonth(year, month);
    printf("next month:%d-%d\n", yearNext, monthNext);
    return 0;
}
```

以上程序使用了全局变量，从而 getNextMonth 函数不需要返回值。全局变量增加了函数之间的沟通渠道，这一点看起来非常好，编程很方便。但是切记，全局变量增加了沟通渠道，同时也使得一个变量被意外改变的可能大大增加。如果一个程序由多个函数组成，并且使用了很多全局变量，那么，某个变量的值出现错误时，将很难查出究竟是在哪个函数将该值修改成了错误的值。

因此建议，编程时：**尽量少用全局变量！函数之间尽量只用参数传递和函数返回值的方式进行联系。**

5.3 调试程序

5.3.1 单步执行跟踪进入函数

前面章节已经学习了程序的单步调试、跟踪，程序中包含自定义函数后，跟踪程序执行流程更加困难。以下通过一个程序讲解如何单步跟踪带函数的程序。

源代码(见“ch5\ getMax3.cpp”)：

```
1:    #define _CRT_SECURE_NO_WARNINGS
2:    #include <stdio.h>
3:    int getMax(int x, int y, int z)
4:    {
5:        int max3;
6:        max3 =xx>yy? xx:yy;
7:        if (zz>max3)
8:          max3 =zz;
9:        return max3;
10:   }
11:   int main()
```

```
12:     {
13:         int x,y,z,max;
14:         printf("input 3 numbers:\n");
15:         scanf("%d%d%d", &x, &y, &z);
16:         max =getMax(x,y,z);
17:         printf("maxmum=%d", max);
18:         return 0;
19:     }
```

首先在第 16 行设一个断点,再按 F5 键启动调试。为程序输入三个数“5 8 2”(即变量的值 x=5,y=8,z=2),在断点处停下来后,为跟踪进入 getMax 函数,需要单击 按钮(快捷键 F11)。

单击 进入函数后,再按一次 F10 或 F11 键,执行到第 6 行,界面如图 5-4 所示。

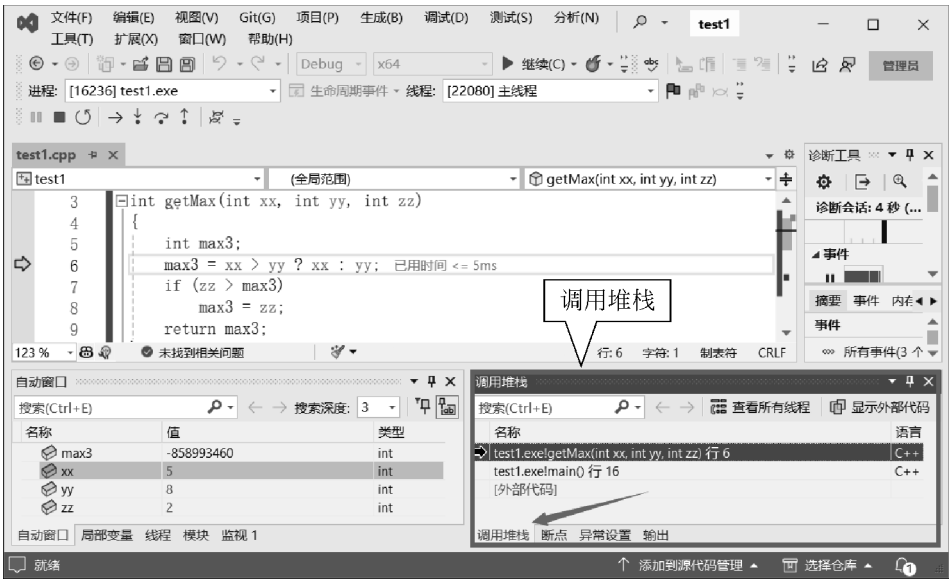

图 5-4　跟踪进入函数后的界面

图中左下角的“自动窗口”中显示了变量 xx,yy,zz 的值。单击“监视 1”转到监视变量窗口,输入变量 x,y,z(图 5-5),发现变量 x,y,z 都显示为“未定义标识符”,因为它们是 main 函数中的局部变量,而我们当前的位置在 getMax 函数中,所以不能访问 x,y,z。

图 5-5　自己输入监视变量

图 5-4 中,可单击右下角箭头所指部位,切换到“调用堆栈”。如果界面上没有“调用

堆栈”，则可单击菜单“调试”→“窗口”→“调用堆栈”，VS 将显示函数的调用栈。从图中可知：当前位于 getMax 函数中的第 6 行代码，调用 getMax 的语句是 main 函数中的第 16 行代码。

由于图 5-4 中不能查看 main 函数局部变量 x、y、z 的值，我们可在“调用堆栈”的 main 函数那一行（即调用堆栈的第 2 行）双击，此时，代码窗口中多了一个绿色箭头指向 main 函数的第 16 行，“监视 1”窗口中也显示出 main 函数局部变量的值（图 5-6）。

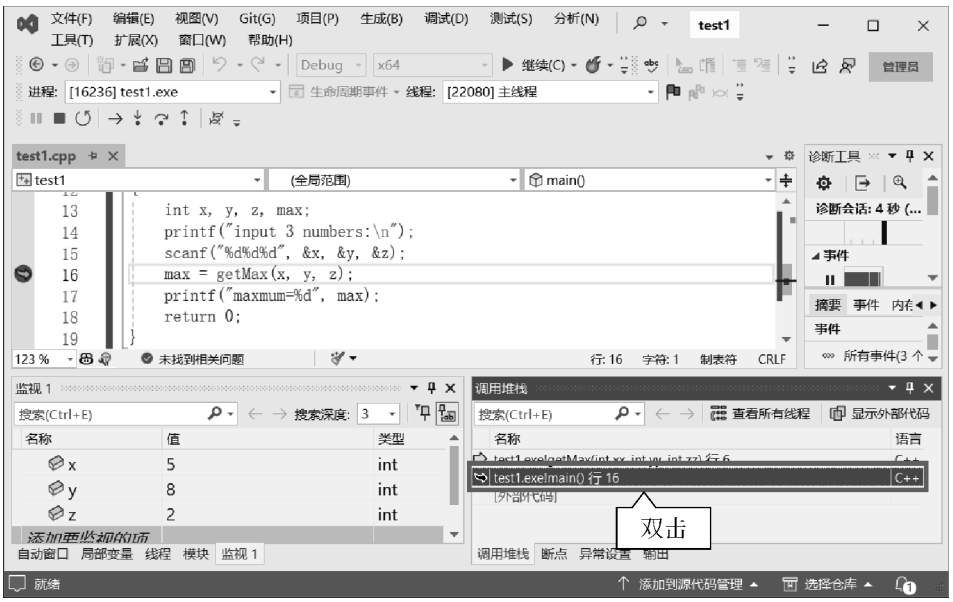

图 5-6 转换到 main 函数，查看其局部变量

图 5-4 中，在 getMax 中单步执行程序（按 F10 键或 F11 键），执行到第 10 行时（函数的结尾花括号处），再单步执行，程序的执行流程返回 main 函数。

如果程序当前执行第 6 行，我们不希望逐行执行程序，可以单击 按钮（快捷键为 Shift＋F11）直接将本函数执行完，跳出函数，返回到调用本函数的地方（第 16 行）。

【知识点】

按钮：功能为跟踪进入函数、执行程序，快捷键为 F11。

按钮：功能为逐行执行程序，快捷键为 F10。

以上两个按钮的区别如下。

单步执行程序时，如果黄色箭头指向一个函数调用语句，单击 按钮将跟踪到函数内部，而单击 按钮，则将该行代码作为一个普通语句，一步执行过去，不会进入函数中。如果黄色箭头指向一个普通语句（非函数调用语句），两个按钮的功能没区别。

按钮：功能为将函数执行完毕直到跳出函数，快捷键为 Shift＋F11。

5.3.2 调试排错

以下程序的功能是计算1!＋2!＋…＋n!，请改正其中的错误。