第 6 章　函　　数

（视频讲解：0.5 小时）

一个 C 程序由一个主函数和若干其他函数构成。一个较大的程序可分为若干程序模块，每个模块实现一个特定的功能。在高级语言中用子程序实现模块的功能，而子程序由函数来实现。

学习目标：

- 函数的声明、定义与调用。
- 递归调用。
- 变量及函数的作用域。

6.1　函数的声明、定义与调用

6.1.1　函数的声明与定义

函数间的调用关系是，由主函数调用其他函数，其他函数也可以互相调用。同一个函数可以被一个或多个函数调用任意次，如图 6.1.1 所示。

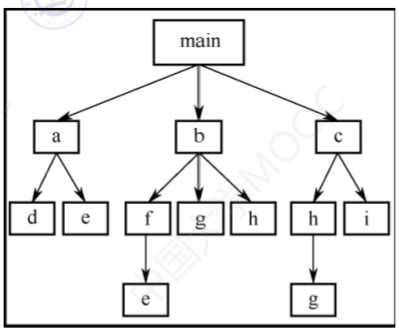

图 6.1.1　函数间的调用关系

下面来看例 6.1.1。例中有两个 c 文件，func.c 是子函数 printstar 和 print_message 的实现，也称定义；main.c 是 main 函数，func.h 中存放的是标准头文件的声明和 main 函数中调用的两个子函数的声明，如果不在头文件中对使用的函数进行声明，那么在编译时会出现警告。

【例 6.1.1】函数嵌套调用。

func.h

```
#include <stdio.h>
#include <stdlib.h>
```

```
int printstar(int i);  //函数声明
void print_message();
```

func.c

```
#include "func.h"

int printstar(int i)  //i 即为形式参数
{
    printf("**********\n");
    printf("printstar %d\n",i);
    return i+3;
}
void print_message()  //可以调用 printstar
{
    printf("how do you do\n");
    printstar(3);
}
```

main.c

```
#include "func.h"
int main()
{
    int a=10;
    a=printstar(a);
    print_message();
    printstar(a);
    return 0;
}
```

C 语言的编译和执行具有以下特点。

（1）一个 C 程序由一个或多个程序模块组成，每个程序模块作为一个源程序文件。对于较大的程序，通常将程序内容分别放在若干源文件中，再由若干源程序文件组成一个 C 程序。这样处理便于分别编写、分别编译，进而提高调试效率。一个源程序文件可以为多个 C 程序共用。

（2）一个源程序文件由一个或多个函数及其他有关内容（如命令行、数据定义等）组成。一个源程序文件是一个编译单位，在程序编译时是以源程序文件为单位而不是以函数为单位进行编译的。main.c 和 func.c 分别单独编译，在链接成为可执行文件时，main 中调用的函数 printstar 和 print_message 才会通过链接去找到函数定义的位置。

（3）C 程序的执行是从 main 函数开始的，如果在 main 函数中调用其他函数，那么在调用

后会返回到 main 函数中，在 main 函数中结束整个程序的运行。

（4）所有函数都是平行的，即在定义函数时是分别进行的，并且是互相独立的。一个函数并不从属于另一函数，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 函数是由系统调用的，例 6.1.1 的 main 函数中调用 print_message 函数，而 print_message 函数中又调用 printstar 函数，我们把这种调用称为嵌套调用。

函数的声明与定义的差异如下。

（1）函数的定义是指对函数功能的确立，包括指定函数名、函数值类型、形参及其类型、函数体等，它是一个完整的、独立的函数单位。

（2）函数的声明的作用是把函数的名字、函数类型及形参的类型、个数和顺序通知编译系统，以便在调用该函数时编译系统能正确识别函数并检查调用是否合法。

隐式声明：C 语言中有几种声明的类型名可以省略。例如，**函数如果不显式地声明返回值的类型，那么它默认返回整型**；使用旧风格声明函数的形式参数时，如果省略参数的类型，那么编译器默认它们为整型。然而，依赖隐式声明并不是好的习惯，因为隐式声明容易让代码的读者产生疑问：编写者是否是有意遗漏了类型名？还是不小心忘记了？显式声明能够清楚地表达意图！

6.1.2　函数的分类与调用

从用户角度来看，函数分为如下两种。

（1）标准函数：即库函数，这是由系统提供的，用户不必自己定义的函数，可以直接使用它们，如 printf 函数、scanf 函数。不同的 C 系统提供的库函数的数量和功能会有一些不同，但许多基本的函数是相同的。

（2）用户自己定义的函数：用以解决用户的专门需要。

从函数的形式看，函数分为如下两类。

（1）无参函数：一般用来执行指定的一组操作。在调用无参函数时，主调函数不向被调用函数传递数据。

无参函数的定义形式如下：

```
类型标识符　函数名()
{
  声明部分
  语句部分
}
```

在例 6.1.1 中，print_message 就是无参函数。

（2）有参函数：主调函数在调用被调用函数时，通过参数向被调用函数传递数据。

有参函数的定义形式如下：

```
类型标识符　函数名(形式参数表列)
{
  声明部分
  语句部分
}
```

在例 6.1.1 中，printstar 就是有参函数，int i 对应的 i 为形式参数，主调函数和被调用函数之间存在数据传递关系。

在不同的函数之间传递数据时，可以使用的方法如下：

（1）参数：通过形式参数和实际参数。

（2）返回值：用 return 语句返回计算结果。

（3）全局变量：外部变量。

下面来看一个全局变量的实例，如例 6.1.2 所示。

【例 6.1.2】全局变量的使用。

```
#include <stdio.h>
#include <stdlib.h>

int i=10;  //全局变量
void print(int a)
{
    printf("print i=%d\n",i);
}
int main()
{
    printf("main i=%d\n",i);
    i=5;
    print(i);
    system("pause");
    return 0;
}
```

全局变量存储在哪里？如图 6.1.2 所示，全局变量 i 存储在数据段，所以 main 函数和 print 函数都是可见的。全局变量不会因为某个函数执行结束而消失，在整个进程的执行过程中始终有效，因此工作中应尽量避免使用全局变量！在前几章中，我们在函数内定义的变量都称为局部变量，局部变量存储在自己的函数对应的栈空间内，函数执行结束后，函数内的局部变量所分配的空间将会得到释放。如果局部变量与全局变量重名，那么将采取就近原则，即实际获取和修改的值是局部变量的值。

图 6.1.2　全局变量的存储

思考题：如果把 print(int a)改为 print(int i)，那么 print 函数的打印结果会是多少？

关于形参与实参的一些说明如下。

（1）定义函数中指定的形参，如果没有函数调用，那么它们并不占用内存中的存储单元。只有在发生函数调用时，函数 print 中的形参才被分配内存单元。在调用结束后，形参所占的内存单元也会被释放。

（2）实参可以是常量、变量或表达式，但要求它们有确定的值，例如，print(i+3)在调用时将实参的值赋给形参。假如 print 函数有两个形参，如 print(int a,int b)，那么实际调用 print 函数时，使用 print(i,i++)是不合适的，因为 C 标准未规定函数调用是从左到右计算还是从右到左计算，因此不同的编译会有不同的标准，造成代码在移植过程中发生非预期错误。

（3）在被定义的函数中，必须指定形参的类型。如果实参列表中包含多个实参，那么各参数间用逗号隔开。实参与形参的个数应相等，类型应匹配，且实参与形参应按顺序对应，一一传递数据。

（4）实参与形参的类型应相同或赋值应兼容。

（5）实参向形参的数据传递是单向“值传递”，只能由实参传给形参，而不能由形参传回给实参。在调用函数时，给形参分配存储单元，并将实参对应的值传递给形参，调用结束后，形参单元被释放，实参单元仍保留并维持原值。

（6）形参相当于局部变量，因此不能再定义局部变量与形参同名，否则会造成编译不通。

6.2　递归调用

假设现在要求读者写一个程序来求数字 n 的阶乘。读者可能会觉得这很简单，写个 for 循环就可以实现。然而，使用递归来实现更好一些，因为使用递归在解决一些问题时，可以让问题变得简单，降低编程的难度。比如接下来的题目：假如有 n 个台阶，一次只能上 1 个台阶或 2 个台阶，请问走到第 n 个台阶有几种走法？为便于读者理解题意，这里举例说明如下：假如有 3 个台

阶，那么总计就有3种走法：第一种为每次上1个台阶，上3次；第二种为先上2个台阶，再上1个台阶；第三种为先上1个台阶，再上2个台阶。具体实现请看例6.2.1。

【例6.2.1】n的阶乘的递归调用实现。

```
#include <stdio.h>
#include <stdlib.h>
//求 n 的阶乘
int f(int n)
{
    if(1==n)
    {
        return 1;
    }
    return n*f(n-1);
}
//走楼梯
int step(int n)
{
    if(1==n)
    {
        return 1;
    }
    if(2==n)
    {
        return 2;
    }
    return step(n-1)+step(n-2);
}
int main()
{
    int n;
    int ret;
    scanf("%d",&n);  //请输入数字的大小
    ret=f(n);
    printf("%d\n",ret);
    scanf("%d",&n);  //请输入台阶数
    ret=step(n);
    printf("%d\n",ret);
```

```
    system("pause");
    return 0;
}
```

思考题：对于上面这个走台阶的问题，如果不用递归，那么应该怎样实现？使用递归方法比非递归方法有哪些优势？

6.3　变量及函数的作用域

6.3.1　局部变量与全局变量

1．内部变量

在一个函数内部定义的变量称为内部变量。它只在本函数范围内有效，即只有在本函数内才能使用这些变量，故也称局部变量。

关于局部变量需要注意如下几点：

（1）主函数中定义的变量只在主函数中有效，而不因为在主函数中定义而在整个文件或程序中有效。主函数也不能使用其他函数中定义的变量。

（2）不同函数中可以使用相同名字的变量，它们代表不同的对象，互不干扰。

（3）形式参数也是局部变量。

（4）在一个函数内部，可以在复合语句中定义变量，这些变量只在本复合语句中有效，这种复合语句也称“分程序”或“程序块”。例 6.3.1 中的 int j=5 就是如此，只在离自己最近的花括号内有效，若离开花括号，则在其下面使用该变量会造成编译不通。

2．外部变量

函数之外定义的变量称为外部变量。外部变量可以为本文件中的其他函数共用，它的有效范围是从定义变量的位置开始到本源文件结束，所以也称全程变量。

关于全局变量需要注意如下几点：

（1）全局变量在程序的全部执行过程中都占用存储单元，而不是仅在需要时才开辟单元。

（2）使用全局变量过多会降低程序的清晰性。在各个函数执行时都可能改变外部变量的值，程序容易出错，因此要有限制地使用全局变量。

（3）因为函数在执行时依赖于其所在的外部变量，如果将一个函数移到另一个文件中，那么还要将有关的外部变量及其值一起移过去。然而，如果该外部变量与其他文件的变量同名，那么就会出现问题，即会降低程序的可靠性和通用性。**C 语言一般要求把程序中的函数做成一个封闭体，除可以通过“实参→形参”的渠道与外界发生联系外，没有其他渠道。**

下面我们来看例 6.3.1，它由 main.c、func.c、func.h 组成。

【例 6.3.1】局部变量与全局变量。（针对多文件，大家知道可以，后期我们均使用单文件）

main.c

```
#include "func.h"


int k=10;
```

```
int main()
{
    int i=10;
    {
        int j=5;
    }  //局部变量的有效范围是离自己最近的花括号
    printf("i=%d,k=%d\n",i,k);
    print();
    print();
    return 0;
}
```

func.c

```
#include "func.h"

void print()
{
    int t=0;
    t++;
    printf("print execute %d\n",t);
    printf("print k=%d\n",k);
}
```

func.h

```
#include <stdio.h>
#include <stdlib.h>

void print();
```