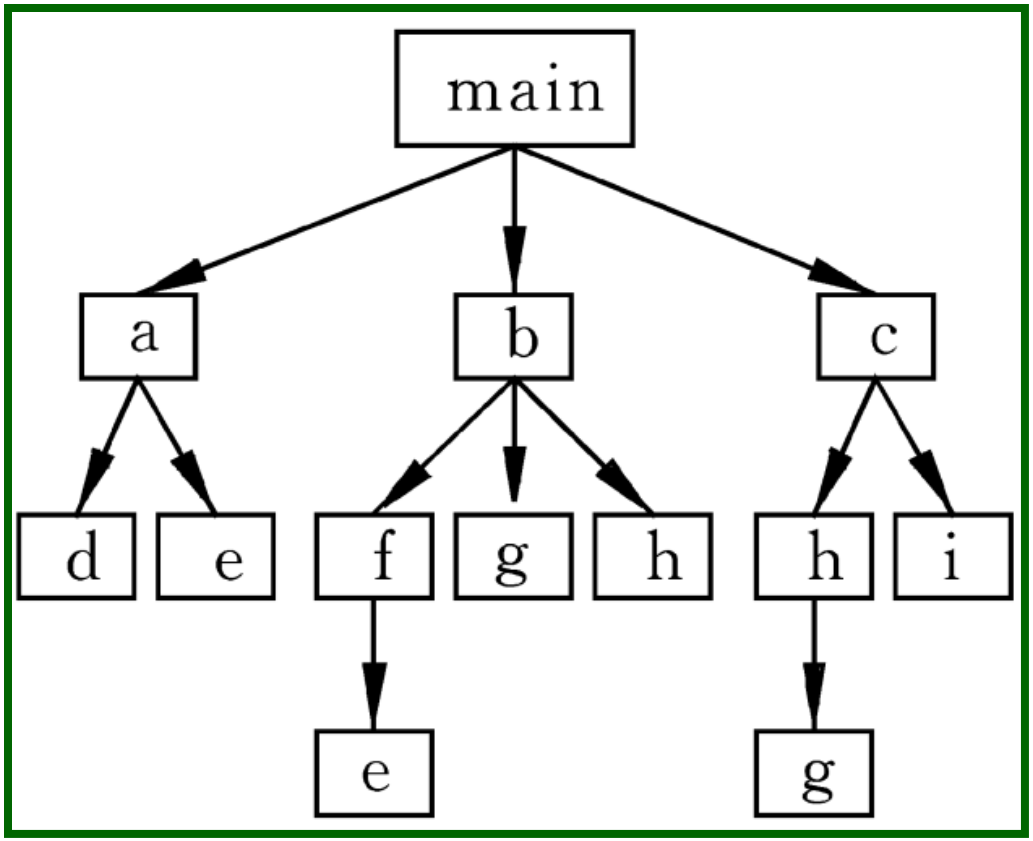
第八章

- 本章要点

  - 函数的概念
  - 函数的定义与调用
  - 函数的递归调用
  - 变量的作用域
  - 函数的作用域

- 主要内容
  - § 8.1 概述
  - § 8.2 函数定义的一般形式
  - § 8.3 函数参数和函数的值
  - § 8.4 函数的调用
  - § 8.5 函数的嵌套调用
  - § 8.6 函数的递归调用
  - § 8.7 数组作为函数参数
  - § 8.8 局部变量和全局变量
  - § 8.9 变量的存储类别
  - § 8.10 内部函数和外部函数

§8.1概述



一种较大的程序可分为若干个**程序模块**，每一种模块用来实现一种特定的功能。在高级语言中用**子程序**实现模块的功能。子程序由函数来完毕。一种C程序可由一种主函数和若干个其他函数构成。

函数间的调用关系

由主函数调用其他函数，其他函数也能够相互调用。同一种函数能够被一种或多种函数调用任意屡次。

例8.1先举一种函数调用的简朴例子

```
# include <stdio.h>
void main()
{
  void printstar();          /*对printstar函数申明*/
  void print_message();
                             /*对print_message函数申明*/
  printstar();               / *调用printstar函数* /
  print_message();           /*调用print_message函数*/
  printstar();               / *调用printstar函数*/
 }
```

```
void printstar()  / *定义printstar函数* /
{
 printf("* * * * * * * * * * * * * * * * * * * * *\n");
}

void print_message()  / *定义print_message函数* /
{
  printf("How do you do!\n");
 }
```

运营情况如下：

```
* * * * * * * * * * * * * * * * * * * * *
How do you do!
* * * * * * * * * * * * * * * * * * * * *
```

- 说明：
- （1）一个C程序由一个或多个程序模块组成，每一个程序模块作为一个源程序文件。对较大的程序，一般不希望把所有内容全放在一个文件中，而是将他们分别放在若干个源文件中，再由若干源程序文件组成一个C程序。这样便于分别编写、分别编译，

（2）一种源程序文件由一种或多种函数以及其他有关内容（如命令行、数据定义等）构成。一种源程序文件是一种编译单位，在程序编译时是以源程序文件为单位进行编译的，而不是以函数为单位进行编译的。

（3）C程序的执行是从main函数开始的，如是在main函数中调用其他函数，在调用后流程返回到main函数，在main函数中结束整个程序的运营。

（4） 全部函数都是平行的，即在定义函数时是分别进行的，是相互独立的。一种函数并不隶属于另一函数，即函数不能嵌套定义。函数间能够相互调用，但不能调用ｍａｉｎ函数。ｍａｉｎ函数是系统调用的。

（5）从顾客使用的角度看，函数有两种：

① 原则函数，即库函数。这是由系统提供的，顾客不必自己定义这些函数，能够直接使用它们。应该阐明，不同的C系统提供的库函数的数量和功能会有某些不同，当然许多基本的函数是共同的。

② 顾客自己定义的函数。用以处理顾客的专门需要。

（6） 从函数的形式看，函数分两类：

①无参函数。如例8.1中的printstar和print_message就是无参函数。在调用无参函数时，主调函数不向被调用函数传递数据。无参函数一般用来执行指定的一组操作。例如，例8．1程序中的printstar函数。

②有参函数。在调用函数时，主调函数在调用被调用函数时，经过参数向被调用函数传递数据，一般情况下，执行被调用函数时会得到一种函数值，供主调函数使用。

§8. 2 函数定义的一般形式

§8.2.1. 无参函数的定义一般形式

定义无参函数的一般形式为:

```
类型标识符    函数名（）
{
  申明部分
  语句部分
}
```

在定义函数时要用“类型标识符”指定函数值的类型，即函数带回来的值的类型。例8. 1 中的printstar和print_message函数为void类型，表达不需要带回函数值。

§8.2.2. 有参函数定义的一般形式

定义有参函数的一般形式为:

类型标识符　函数名（形式参数表列）

{

　申明部分

　语句部分

}

例如:

```
int max(int x, int y)
  {int z; /*函数体中的申明部分*/
        z=x>y? x:y;
        return(z);
  }
```

§8.2.3 空函数

定义空函数的一般形式为：

类型标识符　函数名（）

{ }

例如：

dummy（）

{ }

调用此函数时，什么工作也不做，没有任何实际作用。在主调函数中写上“dummy（）；”表白“这里要调用一种函数”，而目前这个函数没有起作用，等后来扩充函数功能时补充上。

§8.3 函数参数和函数的值

§8.3.1 形式参数和实际参数

大多数情况下，主调函数和被调用函数之间有数据传递的关系。

在不同的函数之间传递数据，能够使用的方法：

- 参数：经过形式参数和实际参数
- 返回值：用return语句返回计算成果
- 全局变量：外部变量

例8.2 调用函数时的数据传递

```
#include <stdio.h>
void main ()
 { int max(int x, int y);
   /* 对max函数的申明 */
  int a, b, c;
  scanf("%d, %d", &a, &b);
   c=max(a, b);
  printf("Max is %d", c);
 }
```

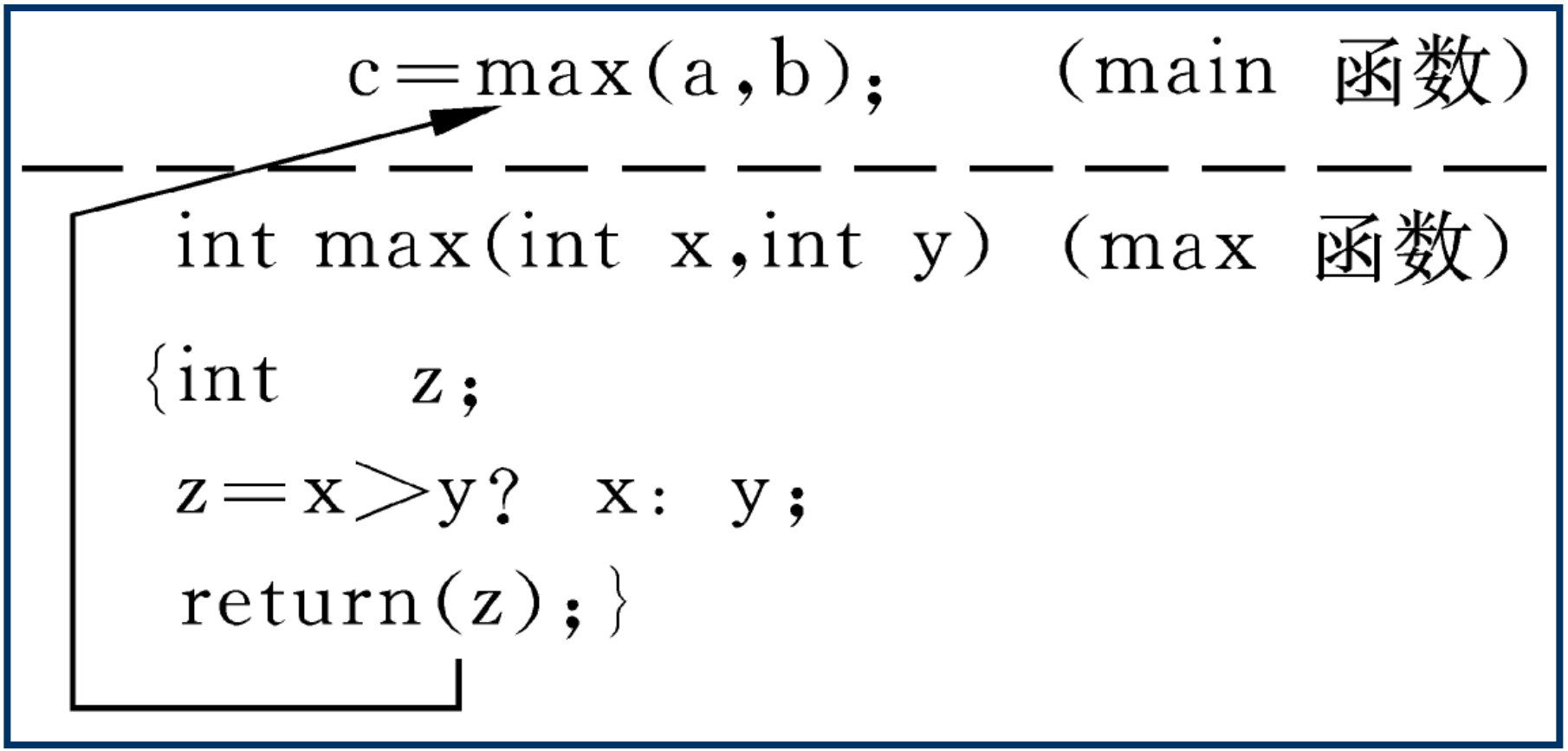
```
int max(int  x , int  y ) /*定义有参函数max */
   {
      int  z ;
       z = x > y ? x : y ;
      return ( z ) ;
   }
```

运营情况如下：

7，8↙

Max is 8

经过函数调用，使两个函数中的数据发生联络



有关形参加实参的阐明：

(1) 在定义函数中指定的形参，在未出现函数调用时，它们并不占内存中的存储单元。只有在发生函数调用时，函数ｍａｘ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被释放。

(2) 实参能够是常量、变量或体现式，如：

ｍａｘ（３，ａ＋ｂ）；

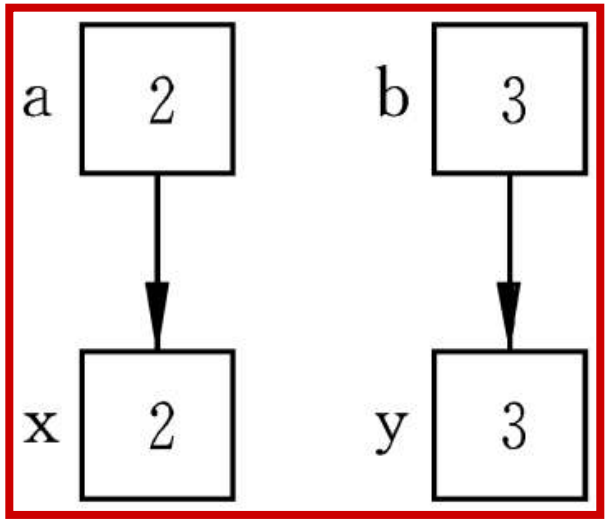
但要求它们有拟定的值。在调用时将实参的值赋给形参。

(3) 在被定义的函数中，必须指定形参的类型（见例8.2程序中的“c＝max（a，b）；”）。

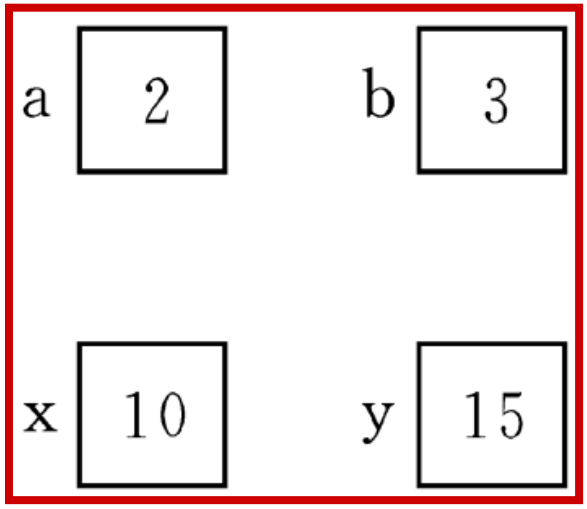
(4) 实参加形参的类型应相同或赋值兼容。例8．2中实参和形参都是整型。假如实参为整型而形参x为实型，或者相反，则按第3章简介的不同类型数值的赋值规则进行转换。

例如实参值a为3.5，而形参x为整型，则将实数3.5转换成整数3，然后送到形参b。字符型与整型能够相互通用。

(5) 在C语言中，实参向对形参的数据传递是“值传递”，单向传递，只由实参传给形参，而不能由形参传回来给实参。在内存中，实参单元与形参单元是不同的单元。



在调用函数时，给形参分配存储单元，并将实参相应的值传递给形参，调用结束后，形参单元被释放，实参单元仍保存并维持原值。所以，在执行一种被调用函数时，形参的值假如发生变化，并不会变化主调函数的实参的值。例如，若在执行函数过程中 x 和 y 的值变为１０和１５，而 a 和 b 仍为 2 和 3 。



§8.3.2 函数的返回值

一般，希望经过函数调用使主调函数能得到一种拟定的值，这就是函数的返回值。例如，例8.2中，max（2，3）的值是3，max（5，2）的值是5。赋值语句将这个函数值赋给变量c。

有关函数返回值的某些阐明：

(1) 函数的返回值是经过函数中的return语句取得的。

return语句将被调用函数中的一种拟定值带回主调函数中去。见图8.2中从return语句返回的箭头。

假如需要从被调用函数带回一种函数值供主调函数使用，被调用函数中必须包括return语句。假如不需要从被调用函数带回函数值能够不要return语句。

一种函数中能够有一种以上的return语句，执行到哪一种return语句，哪一种语句起作用。return语句背面的括弧也能够不要，

如：“return z;” 等价于 “return（z）;”

return背面的值能够是一种体现式。
例如，例8．2中的函数max能够改写成：

```
max（int x，int y）
 {
   return（x＞y？x：y）；
 }
```

(2) 函数的返回值应该属于某一种拟定的类型，在定义函数时指定函数返回值的类型。

例如:下面是3个函数的首行:

```
int   max（float  x ，  float  y ）      /* 函数值为整型 */
char   letter（char c1，  char c2）    /* 函数值为字符型 */
double  min（int  x ，  int  y ）      /* 函数值为双精度型
*/
```

在C语言中，凡不加类型阐明的函数，自动按整型处理。例8. 2 中的m a x 函数首行的函数类型int能够省写，用Turbo C 2.0编译程序时能经过，但用Turbo C ++ 3.0编译程序时不能经过，因为C++要求全部函数都必须指定函数类型。所以,提议在定义时对全部函数都指定函数类型。

（3）在定义函数时指定的函数类型一般应该和return语句中的体现式类型一致。

假如函数值的类型和return语句中体现式的值不一致，则以函数类型为准。对数值型数据，能够自动进行类型转换。即函数类型决定返回值的类型。

（4）对于不带回值的函数，应该用“void”定义函数为“无类型”（或称“空类型”）。这么，系统就确保不使函数带回任何值，即禁止在调用函数中使用被调用函数的返回值。此时在函数体中不得出现return语句。

例 8.3 返回值类型与函数

```
# include <stdio.h>
void main （）
  {  int m a x （float x， float y）;
     float a， b；
     int c；
     scanf（" % f， % f， "， & a， & b）；
     c ＝m a x （a， b）；
     printf（" M a x i s % d \ n "， c）；
  }
int max（float x， float y）
  {  float z；                    /* z为实型变量 */
     z ＝ x ＞ y ？ x ： y；
     return（z）；
  }
```

运营情况如下：

1．5，　2．5↙

Max is　2

§8.4 函数的调用

§8.4.1 函数调用的一般形式

函数调用的一般形式为：函数名（实参表列）

假如是调用无参函数，则“实参表列”能够没有，但括弧不能省略。

假如实参表列包括多种实参，则各参数间用逗号隔开。实参加形参的个数应相等，类型应匹配。实参加形参按顺序相应，一一传递数据。

例 8. 4 实参求值的顺序

```
#include <stdio.h>
void main()
  {
   int f(int a,int b);  /* 函数申明 */
   int i=2,p;
   p=f(i,++i);          /*  函数调用 */
   printf("%d\n",p);
  }
```

```
int f(int a,int b)                    /*  函数定义 */
{
  int c;
  if(a>b) c=1;
   else if(a==b) c=0;
          else c=-1;
   return(c);
}
```

对于函数调用

```
int i=2,p;
p=f(i,++i);
```

假如按自左至右顺序求实参的值，则函数调用相当于f（2，3）

假如按自左至右顺序求实参的值，则函数调用相当于f（3，3）

§8.4.2 函数调用的方式

按函数在程序中出现的位置来分，可以有以下三种函数调用方式：

1．函数语句

把函数调用作为一种语句。如例8.1中的printstar(),这时不要求函数带回值，只要求函数完毕一定的操作。

2．函数体现式

函数出目前一种体现式中，这种体现式称为函数体现式。这时要求函数带回一种拟定的值以参加体现式的运算。例如：c＝2*max（a，b）；

3．函数参数

函数调用作为一种函数的实参。例如:

```
m = max (a , max ( b , c ) ) ;
```

其中max (b , c)是一次函数调用，它的值作为max另一次调用的实参。m的值是a、b、c三者中的最大者。

又如: `printf ("%d", max (a,b));`也是把max (a , b)作为printf函数的一种参数。

函数调用作为函数的参数，实质上也是函数体现式形式调用的一种，因为函数的参数原来就要求是体现式形式。

§8.4.3 对被调用函数的申明和函数原型

在一个函数中调用另一函数（即被调用函数）需要具备哪些条件呢？

(1) 首先被调用的函数必须是已经存在的函数（是库函数或顾客自己定义的函数）。但光有这一条件还不够。

(2) 假如使用库函数，还应该在本文件开头用＃ｉｎｃｌｕｄｅ命令将调用有关库函数时所需用到的信息“包括”到本文件中来。

(3) 假如使用顾客自己定义的函数，而该函数的位置在调用它的函数（即主调函数）的背面（在同一种文件中），应该在主调函数中对被调用的函数作申明。

函数原型的一般形式为

(1) 函数类型 函数名(参数类型1，参数类型2……)；

(2) 函数类型 函数名(参数类型1，参数名1，参数类型2，参数名2……)；

“申明”一词的原文是declaration，过去在许多书中把它译为“阐明”。申明的作用是把函数名、函数参数的个数和参数类型等信息告知编译系统，以便在遇到函数调用时，编译系统能正确辨认函数并检验调用是否正当。（例如函数名是否正确，实参加形参的类型和个数是否一致）。

注意： 函数的“定义”和“申明”不是一回事。函数的定义是指对函数功能确实立，涉及指定函数名，函数值类型、形参及其类型、函数体等，它是一种完整的、独立的函数单位。而函数的申明的作用则是把函数的名字、函数类型以及形参的类型、个数和顺序告知编译系统，以便在调用该函数时系统按此进行对照检验。

例8．5 对被调用的函数作申明

```
# include <stdio.h>
void main（）
 {   float add（float x，  float y）；
                  /*对被调用函数add的申明*/
     float a，b，c;
     scanf（"%f，%f"，&a，&b）；
    c＝add（a，b）；
    printf（"sum is %f \n"，c）；
}
float add（float x，float y）        /*函数首部*/
  {  float z；                      /*  函数体 */
     z＝x＋y；
     return（z）；
   }
```

假如 被调用函数的定义出目前主调函数之前，能够不必加以申明。因为编译系统已经先懂得了已定义函数的有关情况，会根据函数首部提供的信息对函数的调用作正确性检验。

改写例 8.5

```
# include <stdio.h>
float add（float x，float y）        /*函数首部*/
  {  float z；                      /* 函数体 */
     z＝x＋y；
     return（z）；
  }
void main（）
 {
    float a，b，c;
    scanf（"%f，%f"，&a，&b）；
    c＝add（a，b）；
    printf（"sum is %f \n"，c）；
}
```

函数举例

例1 计算10！+9！+6！

例2 写一种函数使输入的字符串按反顺序存储，然后输出。

作业

① 输入10个数x，计算下列函数值。

$$y=\begin{cases}\dfrac{\cos x+\sin x}{2} & x\geq 0\\ \dfrac{\cos x-\sin x}{2} & x<0\end{cases}$$

② 编写程序计算
1+(1+2)+(1+2+3)+…+(1+…+n)

③ 编写函数计算m!/(m-n)!

函数回忆

函数定义的一般格式

```
函数类型 函数名（形参表）// 函数首部
{// 函数体
    函数实现过程

    return 体现式;
}
```

函数的参数

形式参数表

类型1 参数1，类型2 参数2，……，类型n 参数n

- 参数之间用逗号分隔，每个参数前面的类型都必须分别写明
- 函数定义时的参数被称为形式参数（简称形参）

  int max(int x，int y)
- 函数调用时的参数被称为实际参数（简称实参）

  C= max(a，b)

函数原型申明

只写函数定义中的第1行（函数首部），并以分号结束。

- 函数类型 函数名(参数表);

  int max(int x , int y);

- 函数申明：阐明函数的类型和参数的情况，以确保程序编译时能判断对该函数的调用是否正确。
- 函数必须先定义后调用，将主调函数放在被调函数的背面，就像变量先定义后使用一样。
- 假如自定义函数在主调函数的背面，就需要在函数调用前，加上函数原型申明。

参数传递

- 实参→形参
  - 在参数传递过程中，实参把值复制给形参。
  - 形参和实参一一相应：数量一致，类型一致，顺序一致
  - 形参：变量，用于接受实参传递过来的值
  - 实参：常量、变量或体现式

例 求100以内的全部素数，每行输出10个

```
#include <stdio.h>
#include <math.h>
int main(void)
{   int count, m;
    int prime (int m);
    count = 0;
    for(m = 2; m <= 100; m++){
       if ( prime(m) != 0   ){
          printf("%6d", m );
                 count++;

                 if (count %10 == 0)
            printf ("\n");
       }
    }
    printf ("\n");
}
```

```
int prime (int m)
{   int i, n;
    if ( m == 1 ) return 0;
    n = sqrt (m);
    for( i = 2; i <= n; i++)
       if (m % i == 0){
         return 0;
       }
    return 1;
}
```

§8.5 函数的嵌套调用

嵌套定义就在定义一种函数时，其函数体内又包括另一种函数的完整定义 。

C语言不能嵌套定义函数，但能够嵌套调用函数，也就是说，在调用一种函数的过程中，又调用另一种函数。