

第七章 函数

7.1 函数的概念

1、在程序设计过程中，为了实现某个功能需要编写多行代码，例如求一个二维数组中的最大值，如果该功能需要被多次使用，我们可以在每次使用时将原来的代码重复编写，但是这样未免有“凑代码”的嫌疑，而且编程效率也不高。为了避免重复劳动，我们可以将这部分代码**封装**到一个“**函数**”中，在需要使用该功能时**调用封装好的函数**即可

```

int array_max(int a[], int len)
{
    int i, max = a[0];
    for (i = 1; i < len; i++)
    {
        if (a[i] > max)
            max = a[i];
    }
    return max;
}

int main()
{
    int a[4] = {10, 5, 90, 7};
    int i, max = a[0];
    for (i = 1; i < 4; i++)
    {
        if (a[i] > max)
            max = a[i];
    }
    printf("The Max Number Is : %d\n", max);

    max = array_max(a, 4);
    return 0;
}

```

数组长度

函数名

需要操作的数组

2、函数就是具有特殊功能的指令的集合

3、C程序是由函数组成的，函数是 C 程序的基本模块，C程序从主函数main()开始执行。

4、从函数定义的角度出发，函数可以分为库函数和用户自定义函数两种。

- **库函数**：由标准C库提供，用户不需要自己实现，在使用时直接调用即可，例如printf、scanf函数等。
- **用户自定义函数**：需要用户自己根据实际需求自己实现，例如比较两个整数的大小，并且求出最大值

7.2 函数的定义

7.2.1 函数定义的语法规则

类型标识符 函数名(形参列表)

```
{  
  
    函数体  
  
}
```

说明：

- 1) 类型标识符：函数返回数据的类型，支持C语言所有的数据类型
- 2) 函数名：由用户定义的标识符
- 3) 形参列表：函数用来接收用户所传递数据的参数，参数可以是0个也可以是多个
- 4) 函数体：实现函数功能的代码块

7.2.2 无参函数的定义

1、函数不需要接收用户传递的数据

类型标识符 函数名()

```
{  
  
    函数体  
  
}
```

注意：形参列表为空

2、如果函数不需要返回值，“类型标识符”可以设计为**void**

```
void func()
{
    printf("hello world\n");
}
```

3、如果函数需要返回值，返回值是什么类型” 类型标识符 “就是什么类型，例如：设计一个函数返回1~100的和

```
int func()
{
    int i, sum = 0;
    for (i = 1; i <= 100; i++)
        sum += i;
    return sum;
}
```

7.2.3 有参函数的定义

1、可以根据实际需求设计函数的形参，**形参的作用就是接收用户传递的数据**

类型标识符 函数名(类型标识符 形参1, 类型标识符 形参2 ...)

```
{
    函数体
}
```

注意：形参的数据类型及个数由实际需求而定

2、设计一个函数计算两个整数的和

```
void sum(int a, int b)
{
    int c = a+b;
    printf("sum: %d\n", c);
}
```

7.3 函数的参数和返回值

7.3.1 函数的形参和实参

1、形参出现在函数的定义中，在整个函数体中都可以访问到，离开了该函数就不能被访问了

2、实参出现在函数调用时

3、函数调用时，将**实参的值的赋值**给形参

4、形参和实参有如下特点：

- 形参只有在函数被调用时才会被分配内存空间
- 实参可以是**常量、变量、表达式、函数名**

```

int max(int a, int b) ← 形参
{
    return a>b?a:b;
}
int main()
{
    int a = 10, b = 11;

    max(a, b); ← 实参为变量

    max(a, 100); ← 实参为常量

    max(10, 20); ← 实参为常量

    max(10+100, 100); ← 实参为表达式

    return 0;
}

```

7.3.2 函数的返回值

1、函数的返回值是函数将函数体中的**执行结果**返回(传递)给函数的调用者，通过**return**语句返回

```

return 表达式;
或者
return (表达式);

```

有没有()都是正确的，为了简明，一般也不写()。例如：

```
return max;  
return a+b;  
return (100+200);
```

2、函数执行完return语句后，函数执行完毕（函数退出）

3、函数的返回类型就是函数的类型

例如：

```
int func(){ 该函数的返回类型是int，那我们就说函数func是个整型函数
```

4、如果函数不需要返回值，函数的返回类型为**void**，在函数体中可以使用 "return ;" 语句退出函数，也可以不使用 "return ;"

```
void func()  
{  
    printf("hello\n");  
    return ; //这句话也可以不写  
}
```

5、如果函数返回的类型和return语句中表达式的值的数据类型不一致，则以函数返回类型为准，即函数返回类型决定返回值的类型。对数值型数据，可以自动进行类型转换

```
double max() // 函数的返回类型为double  
{  
    int a = 1;  
    return a; // 返回值a为int类型，会转为double类型再返回，最终返回 1.0  
}
```

7.4 函数的调用

1、函数在定义完以后，如果不被调用函数是不会被执行到的。

2、main函数是C程序的主函数，是会被自动执行到的，C程序中有且只有一个main函数

7.4.1 无参函数的调用

1、调用方法

```
函数名();  
或者：  
变量 = 函数名();
```

注意：当函数有返回值时，我们可以定义一个变量来接收函数的返回值，但是定义变量的数据类型必须和函数的返回类型一致，当然我们也可以不接收函数的返回值。

2、举例

```
int func()
{
    return 10;
}

int main()
{
    func();
    int x = func();
    return 0;
}
```

7.4.2 有参函数的调用

1、调用方法

```
函数名(实参列表)
或者
变量 = 函数名(实参列表)
```

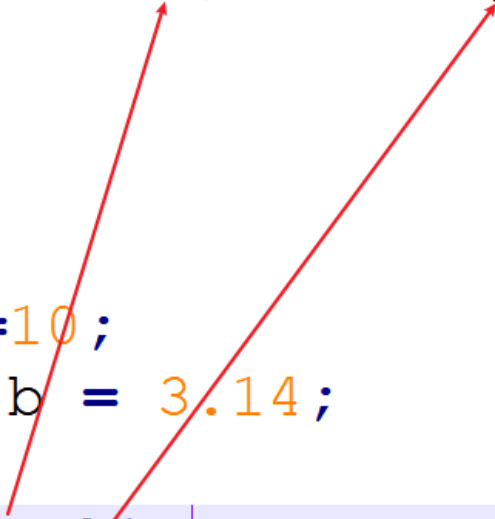
- **注意：**传递的实参是和形参——对应的


```

void func(int x, float y)
{
}
int main()
{
    int a=10;
    float b = 3.14;

    func(a, b);
    return 0;
}

```



- 注意：实参的个数和形参的个数必须一致！

7.5 函数的嵌套调用

1、函数的嵌套调用：某个函数的函数体中调用了另外一个函数

```

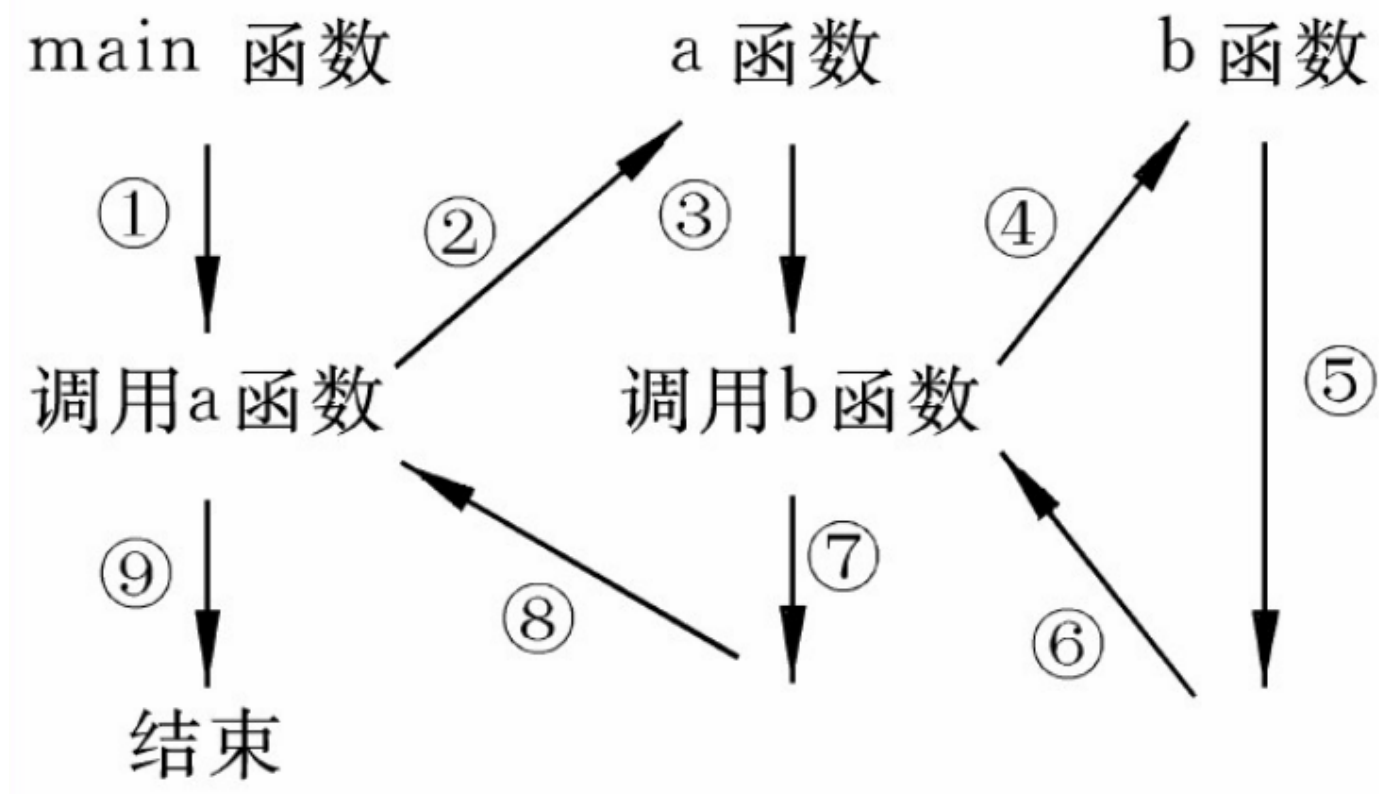
void b()
{
    printf("b\n");
}

void a()
{
    printf("a\n");
    b();
}

int main()
{
    a();
    return 0;
}

```

执行逻辑：



7.6 函数的声明

1、在上一个例子中，如果我们将函数a和b的位置互换，我们编译程序看看会发生什么情况

```
void a()
{
    printf("a\n");
    b();
}

void b()
{
    printf("b\n");
}

int main()
{
    a();
    return 0;
}
```

我们发现编译报 warning了，意思是函数b被隐式的声明了

```
warning: implicit declaration of function 'b'
```

当然，虽然报warning了，但是代码还是能够成功编译，并且能够正常执行。但是，如果是低版本的编译器，可能会编译报错：函数b是未声明的！

注意：编译程序时要将warning当error！

为什么程序编译会报warning甚至是error呢？因为编译器在编译代码时，如果代码中有调用函数的语句，会在该语句之前检索函数是否有被声明或者定义，如果没有则编译报warning或者error！

解决方法：

a) 将函数b的定义放在函数a之前

b) 在代码的前面部分对函数b进行声明

```
void b();
```

2、函数声明语法

函数类型 函数名(形参列表);

注意：

- 1、后面有封号；
- 2、函数声明时，函数类型（返回值类型）必须于函数实现时一致，函数名必须和函数实现时一致
- 3、形参列表中，可以不填写形参变量的名字，只需要填写形参的类型即可，例如：

```
int fun(int, float);
```

7.7 局部变量和全局变量

7.7.1 局部变量

1、定义在{}内的变量，称之为局部变量。

```
void func()
{
    int func_x = 100; //局部变量
}
int main()
{
    int i = 10; //局部变量
    {
        int i = 100; //局部变量

        int j = 100; //局部变量
    }
}
```

```
    return 0;
}
```

2、局部变量的作用域：所在的 {} 内

```
void func()
{
    int func_x = 100; //局部变量
}
int main()
{
    int i = 10; //局部变量
    {
        int i = 100; //局部变量
        printf("%d\n", i); // 100

        int j = 100; //局部变量
    }
    printf("%d\n", i); // 10

    return 0;
}
```

3、局部变量的生命周期：随 {} 的结束而结束

```
void func()
{
    int func_x = 100; //局部变量
}
int main()
{
    int i = 10; //局部变量
    {
        int i = 100; //局部变量
        printf("%d\n", i); // 100

        int j = 100; //局部变量
    }
    printf("%d\n", i); // 10
    printf("%d\n", j); //编译报错：error: 'j' undeclared
    printf("%d\n", func_x); //编译报错：error: 'func_x' undeclared

    return 0;
}
```

7.7.2 全局变量

1、定义在函数外部的变量称之为：全局变量

```
int cnt = 10; //全局变量
void func()
{
    int func_x = 100; //局部变量
    cnt++;
}
int main()
{
    func();
    printf("cnt: %d\n", cnt);
    return 0;
}
```

2、全局变量的初始化

全局变量在定义时如果不初始化，编译器会将值设置为默认值0

3、全局变量的作用域：整个程序

4、全局变量的生命周期：随程序的结束而结束

7.8 递归函数

7.8.1 递归函数的定义

1、一个函数在它的函数体内**调用自身**称为递归调用。这种函数称为递归函数

2、举例

```
void func()
{
    func();
}
```

7.8.2 递归函数的使用

1、思考：什么时候需要使用递归函数？

在实现某个函数的过程中，某个功能的实现与本函数一样，则使用函数的递归

2、思考：如果函数的递归调用是这种模式，会出现什么情况？

```
void func()  
{  
    func();  
}
```

注意：函数递归调用一定要有退出条件！！

3、实例：求n的阶乘

```
long long factorial(int n)  
{  
    if (n == 1)  
        return 1;  
    return n * factorial(n - 1);  
}
```

4、递归函数的调用流程

- 逐层调用过程

层次/层数	实参/形参	调用形式	需要计算的表达式	需要等待的结果
1	n=5	factorial(5)	factorial(4) * 5	factorial(4) 的结果
2	n=4	factorial(4)	factorial(3) * 4	factorial(3) 的结果
3	n=3	factorial(3)	factorial(2) * 3	factorial(2) 的结果
4	n=2	factorial(2)	factorial(1) * 2	factorial(1) 的结果
5	n=1	factorial(1)	1	无

- 逐层退出过程

层次/层数	调用形式	需要计算的表达式	从内层递归得到的结果 (内层函数的返回值)	表达式的值 (当次调用的结果)
5	factorial(1)	1	无	1
4	factorial(2)	factorial(1) * 2	factorial(1) 的返回值, 也就是 1	2
3	factorial(3)	factorial(2) * 3	factorial(2) 的返回值, 也就是 2	6
2	factorial(4)	factorial(3) * 4	factorial(3) 的返回值, 也就是 6	24
1	factorial(5)	factorial(4) * 5	factorial(4) 的返回值, 也就是 24	120

5、思考：如何使用递归实现求斐波那契数列的第n项的值

```
int Fibonacci(int n)
{
    if ((n == 1) || (n == 0))
        return n;
    return Fibonacci(n-2) + Fibonacci(n-1);
}
```

6、举一反三：求斐波那契数列的前n项的和

```
#include <stdio.h>

//求斐波那契数列的第n项
unsigned long long fac(int n)
{
```

```
    if (n == 1)
        return 1;
    return n*fac(n-1);
}

//求斐波那契数列的前n项的和
/*
 * 解题思路：前n项的和 = 第n项 + 前n-1项的和
 */
unsigned long long facSum(int n)
{
    if (n == 1 || n == 0)
        return n;
    return fac(n) + facSum(n-1);
}
int main()
{
    unsigned long long sum;
    int n;
    scanf("%d", &n);
    printf("%lld\n", facSum(n));
    return 0;
}
```

7、练习

- 有5个人坐在一起，问第5个人多少岁？他说比第4个人大2岁。问第4个人岁数，他说比第3个人大2岁。问第3个人，又说比第2个人大2岁。问第2个人，说比第1个人大2岁。最后问第1个人，他说是10岁。请问第5个人多大。

$$\text{age}(5) = \text{age}(4) + 2$$

$$\text{age}(4) = \text{age}(3) + 2$$

$$\text{age}(3) = \text{age}(2) + 2$$

$$\text{age}(2) = \text{age}(1) + 2$$

$$\text{age}(1) = 10$$

用数学公式表述如下：

$$\text{age}(n) = 10 \quad (n = 1)$$

$$\text{age}(n-1) + 2 \quad (n > 1)$$

```
• int age(int n)
{
    if (n == 1)
        return 10;
    return age(n-1) + 2;
}
```