



# 第6章 函数

## ——函数的定义、调用和参数传递



哈尔滨工业大学

苏小红

sxh@hit.edu.cn

# 本节要讨论的主要问题

---

- 如何定义一个函数？
- 如何调用一个函数？
- 函数调用时，参数是如何传递的？



# 大话三国

使者：丞相夙兴夜寐，罚二十以上皆亲览焉。所啖之食，日不过数升。

懿：孔明食少事烦，其能久乎？

懿：孔明寝食及事之烦简若何？



# 分而治之

- 如果将`main()`函数比作诸葛亮，.....?
- 一个函数中适合放多少行代码？
  - \* 1986年IBM的研究结果：多数有错误的函数大于500行
  - \* 1991年对148,000行代码的研究表明：小于143行的函数更易于维护
- 分而治之(Divide and Conquer, Wirth, 1971 )
  - \* 把一个复杂的问题分解为若干个简单的问题，提炼出公共任务，把不同的功能分解到不同的模块中
  - \* 复杂问题求解的基本方法，模块化编程的基本思想



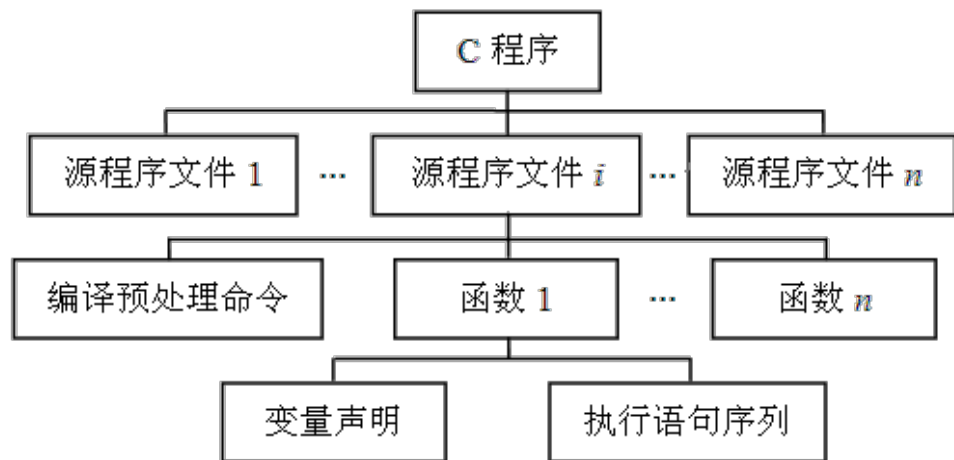
# 模块化编程（Modular Programming）

## ■ 函数（Function）

- \* 构成C语言程序的基本模块，模块化编程的最小单位

## ■ C程序的逻辑结构

- \* 一个C程序由一个或多个源程序文件组成
- \* 一个源程序文件由一个或多个函数组成
- \* 可把每个函数看作一个模块（Module）



# 函数的分类

---

## ■ 标准库函数

- \* ANSI/ISO C定义的标准库函数

- ◆ 使用时，必须在程序开头把定义该函数的头文件包含进来

- \* 第三方库函数

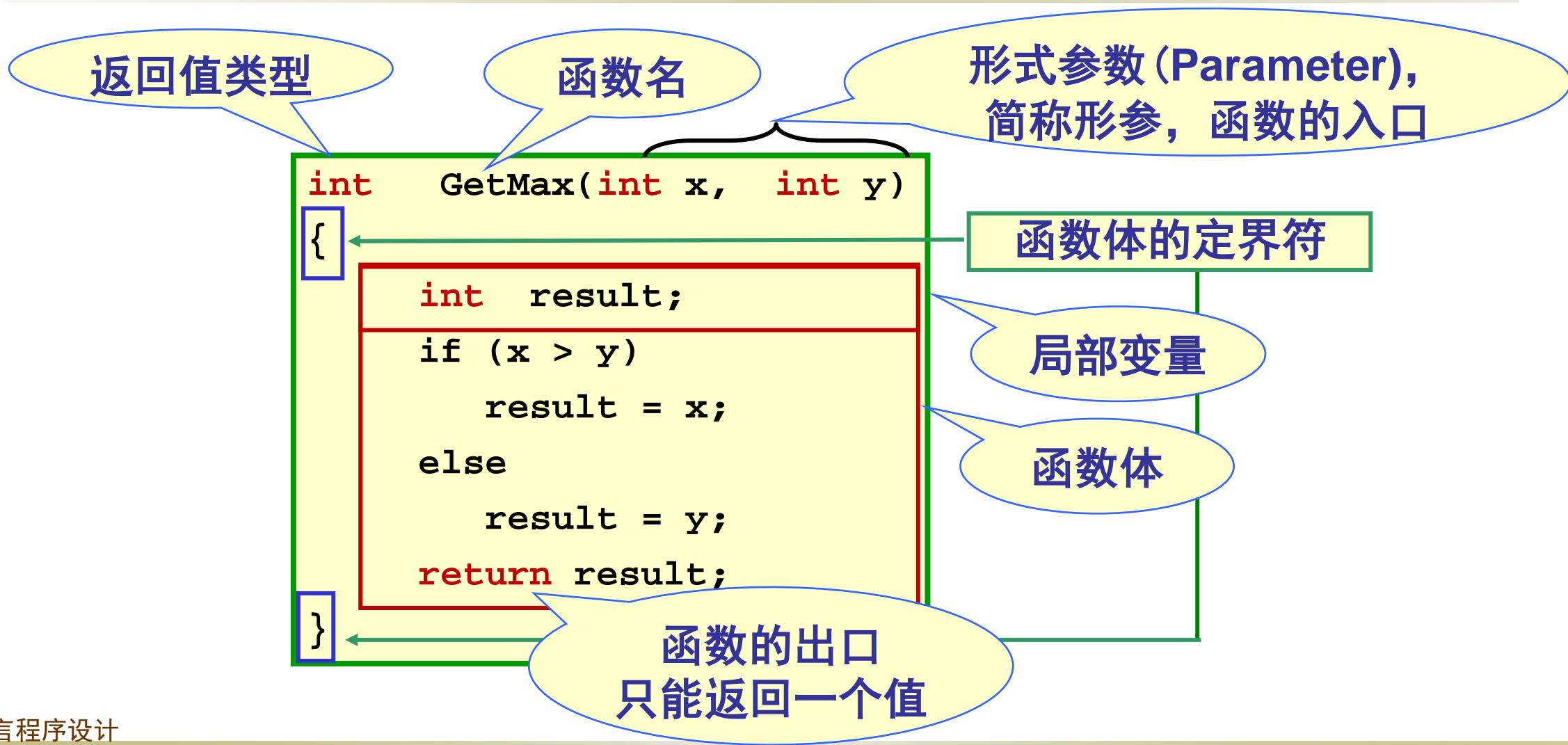
- ◆ 不在标准范围内，能扩充C语言的功能，由其他厂商自行开发的C语言函数库

## ■ 自定义函数

- \* 用户自己定义的函数

- ◆ 包装后，也可成为函数库，供别人使用

# 函数的定义



# 函数的定义

```
int    GetMax(int x,  int y)
{
    int result;
    if (x > y)
        result = x;
    else
        result = y;
    return result;
}
```



```
int    GetMax(int x,  int y)
{
    if (x > y)
        return x;
    else
        return y;
}
```



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



# 函数的定义

表示无返回值  
省略默认为int

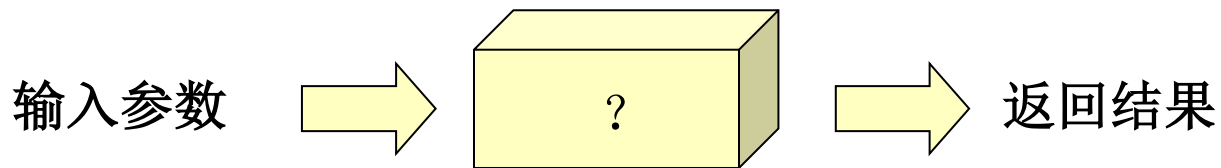
表示无参数，  
可省略

```
void GetMax( void )  
{  
    int x, y, result;  
    scanf("%d, %d", &x, &y);  
    if (x > y)  
        result = x;  
    else  
        result = y;  
    printf("max = %d\n", result);  
    return ;  
}
```

return后无任何表达式  
或没有return

# 使用函数编程的好处

- 信息隐藏（Information Hiding, Parnas, 1972）
  - \* 对于函数的使用者，无需知道函数内部如何运作
  - \* 只了解其与外界的接口（Interface）即可
  - \* 把函数内的具体实现细节对外界隐藏起来，只要对外提供的接口不变，就不影响函数的使用
  - \* 便于实现函数的复用和模块化编程



# 函数调用的基本方式

- 主调函数通过**函数名**调用被调函数
- 函数无返回值时，单独作为一个函数调用语句

```
int main()  
{  
    ...  
    DisplayMenu();  
    ...  
    return 0;  
}
```

```
void DisplayMenu (void)  
{  
    printf("1. input");  
    printf("2. output");  
    printf("0. exit");  
    ...  
    return ;  
}
```

# 函数调用的基本方式

- 调用者通过**函数名**调用函数
- 有返回值时，可放到一个**赋值表达式语句**中

```
int main()  
{  
    int a=12, b=24, ave;  
    ...  
    ave = Average(a, b);  
    ...  
    return 0;  
}
```

```
int Average(int x, int y)  
{  
    int result;  
  
    result = (x + y) / 2;  
  
    return result;  
}
```

# 函数调用的基本方式

- 调用者通过**函数名**调用函数
- 有返回值时，可放到一个**赋值表达式语句**中
- 还可放到一个**函数调用语句**中，作为另一个函数的参数

```
int main()  
{  
    int a=12, b=24;  
    ...  
    printf("%d\n", Average(a, b));  
    ...  
    return 0;  
}
```

```
int Average(int x, int y)  
{  
    int result;  
  
    result = (x + y) / 2;  
  
    return result;  
}
```

# 函数调用时的数据传递

- 函数定义时的参数，形式参数（Parameter），简称**形参**
- 函数调用时的参数，实际参数（Argument），简称**实参**

```
int main()  
{  
    int a=12, b=24;  
    ...  
    printf("%d\n", Average(a, b));  
    ...  
    return 0;  
}
```

```
int Average(int x, int y)  
{  
    int result;  
  
    result = (x + y) / 2;  
  
    return result;  
}
```

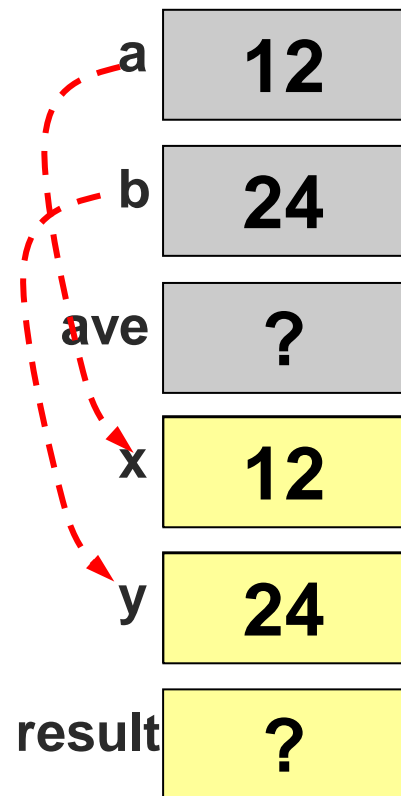
# 函数调用的过程

## ■ 每次执行函数调用时

- 现场保护并为函数内的局部变量（包括形参）分配内存
- 把实参值复制一份给形参，**单向传值**（实参→形参）
- 实参与形参的数目、类型和顺序要一致

```
int main()  
{  
    int a=12, b=24, ave;  
    ...  
    ave = Average(a, b);  
    ...  
    return 0;  
}
```

```
int Average(int x, int y)  
{  
    int result;  
    result = (x + y) / 2;  
    return result;  
}
```



# 函数调用的过程

- 程序控制权交给被调函数，执行函数内的语句
- 当执行到`return`语句或`}`时，从函数退出

```
int main()  
{  
    int a=12, b=24, ave;  
    ...  
    ave = Average(a, b);  
    ...  
    return 0;  
}
```

```
int Average(int x, int y)  
{  
    int result;  
    ②  
    result = (x + y) / 2;  
    return result;  
}
```

a	12
b	24
ave	?
x	12
y	24
result	18



# 函数调用的过程

- 从函数退出时
  - 根据函数调用栈中保存的返回地址，返回到本次函数调用的地方
  - 把函数值返回给主调函数，同时把控制权还给调用者
  - 收回分配给函数内所有变量（包括形参）的内存

```
int main()  
{  
    int a=12, b=24, ave;  
    ...  
    ave = Average(a, b);  
    ...  
    return 0;  
}
```

```
int Average(int x, int y)  
{  
    int result;  
    result = (x + y) / 2;  
    return result;  
}
```

a	12
b	24
ave	18
x	12
y	24
result	18

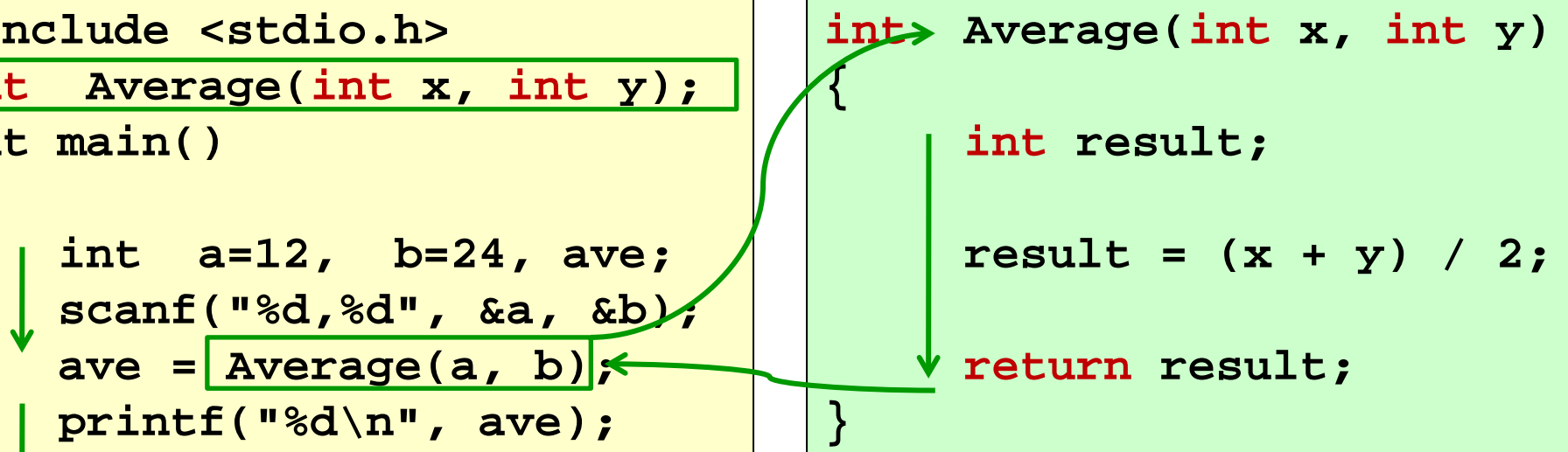
# main函数的特殊性

```
#include <stdio.h>
int Average(int x, int y);
int main()
{
    int a=12, b=24, ave;
    scanf("%d,%d", &a, &b);
    ave = Average(a, b);
    printf("%d\n", ave);
    return 0;
}
```

```
int Average(int x, int y)
{
    int result;

    result = (x + y) / 2;

    return result;
}
```



The diagram illustrates the execution flow between the `main` function and the `Average` function. A green arrow originates from the `Average(a, b)` call in the `main` function, points to the `int Average` function definition, and then returns to the assignment `ave =`. Another green arrow points from the `return result;` statement in the `Average` function back to the `return 0;` statement in the `main` function, indicating the return of control to the caller.

# 讨论

- 如何理解分而治之和信息隐藏？这样做的好处是什么？



