

函数

Wang Houfeng

CCES, PKU

wanghf@pku.edu.cn

内容

➤ 函数基本概念与函数定义

- 函数调用、执行与函数定义的位置关系
- 函数嵌套

什么是函数

数学函数
一般形式

$$y = f(x)$$

- 已知一个数，求其平方根

```
r = sqrt(100.0);
```

- 已知底数 x ，幂指数 y ，求 x^y

```
k = pow(x, y);
```

- 求一个字符串的长度

```
i = strlen (str);
```

- 比较两个字符串的大小

```
v = strcmp(str1, str2)
```

C 语言中的典型函数

初识C/C++的函数设计与使用

```
#include <iostream>
using namespace std;
```

```
int absolute(int n)
{
```

```
    if (n < 0)
```

```
        return (-n);
```

```
    else
```

```
        return n;
```

```
}
```

```
int main()
{
```

```
    int m = -123, result = 0;
```

```
    result = absolute(m);
```

```
    cout << result;
```

```
    return 0;
```

```
}
```

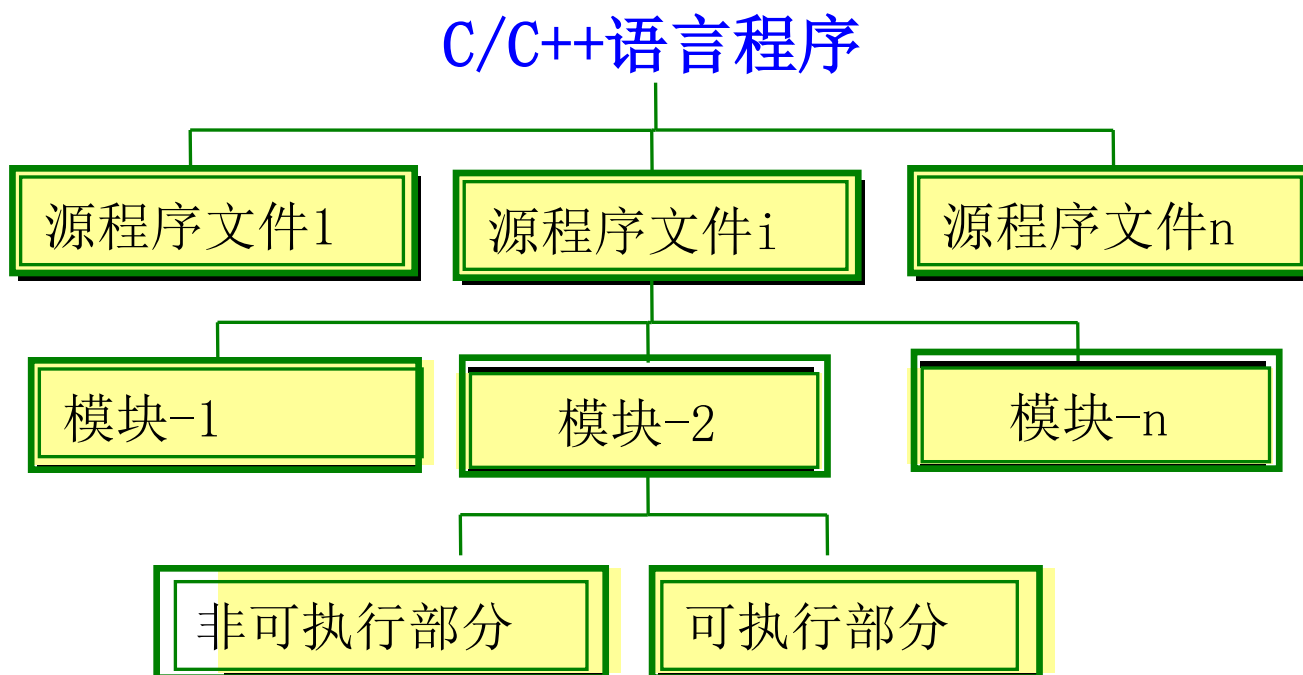
定义函数中的自变量
(形参)

调用实际是的参数
(实参)

另一个例子：延时函数

```
#include <iostream>
using namespace std;
void delay(int n)
{
    for (int i = 0; i < n * 10000; i++);
    return;
}
int main()
{
    for (int j = 0; j < 100; j++)
    {
        cout << j << endl;
        delay(1000);
    }
    return 0;
}
```






C/C++语言程序的模块化设计



为什么要模块化

- **基本思想：** 将一个大的问题，逐步划分成较小的子问题，直至每个子问题容易求解。相应地，一个大的程序，也分割成小模块。
- **模块分割的要求：**
 - 模块间相对独立、功能单一、接口简单
- **模块分割的优点：**
 - 降低程序设计的复杂性
 - 提高构件的可靠性
 - 缩短开发周期
 - 避免程序开发的重复劳动（如：系统函数）
 - 易于维护和功能扩充

C/C++ 语言程序结构的特点

-  一个程序由多个函数（模块）组成（**模块化结构**）
-  有且只能有一个名为main的主函数
-  程序的执行总是从main函数开始，在main中结束
-  所有函数之间都是并列的平行结构
-  函数**不能嵌套定义**

什么时候设计函数：功能独立

- 例题：从键盘输入一个正整数a，编一个程序判断a是否为素数？
- 思路：
 - 素数判断的功能非常明确（独立）
 - 计划设计一个函数 `bool checkprime(int a)`
 - 函数结果为真假值（判断性）：
 - 如果是，该函数返回true，
 - 否则，返回false。

函数被使用

```
#include <iostream>
#include <cmath>
using namespace std;
bool checkPrime(int);
int main()
{
    int a;
    cout << "请输入一个整数" << endl;
    cin >> a;
    if( checkPrime(a) )
        cout << "是质数" << endl;
    else
        cout << "不是质数" << endl;
    return 0;
}
```

函数定义

```
bool checkPrime(int number)
{
    int i, k;
    k = sqrt(number);
    for (i = 2; i <= k; i++)
    {
        if (number % i == 0)    //只要有一个数被出除尽
            return 0;          //则不是素数。
    }
    return 1;                  //走到这一步，说明没能被除尽
}
```

什么时候设计函数：功能复用

- 当某一功能要被反复使用时（减少重复代码）
- 当某一功能相对独立时（有利于结构清晰）

例子：计算组合数

$$\binom{10}{2} = \frac{10!}{2!(10-2)!}$$

– 阶乘的功能反复使用，可以提炼函数：
 ❖ 计算 $n!$ ；

C/C++ 语言程序设计特点：模块化设计

– C/C++的函数分类

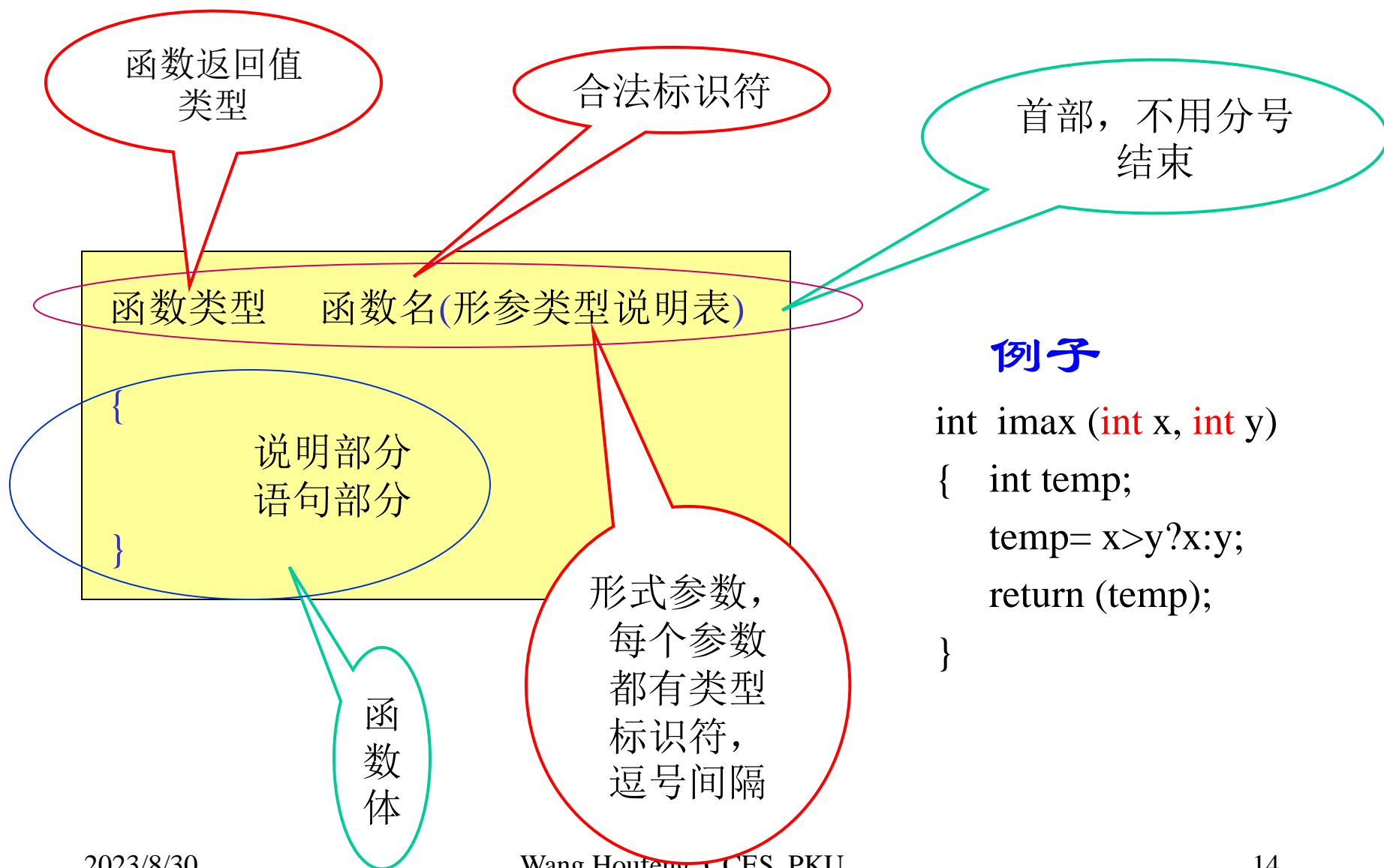
- 从用户角度

- 标准函数（库函数）：由系统提供，include 包含
- 用户根据需要，自己定义新的函数

- 从函数形式

- 无参函数（函数不包含参数，main 函数通常不含参数）
- 有参函数

如何设计函数——一般格式



例子

```
int imax (int x, int y)
{
    int temp;
    temp= x>y?x:y;
    return (temp);
}
```

说明

```
int imax (int x, int y)
{
    int temp;
    temp= x>y?x:y;
    return (temp);
}
```

函数首部（三个部分）

- 类型标识符：表明函数的结果类型。函数体内必须对应有 `return` 语句返回一致类型的表达式值。
- 函数名部分：满足标识符的命名规则
- 参数表部分：表示形式参数，指明自变量的类型和名称，特别注意，每个自变量前必须有类型标识符（不能写成： `int imax (int x, y)` ）。参数表内可以没有参数，但**括号不能省略**。函数的参数只在函数体内起作用！

函数的参数与返回类型

void

空

无参函数

```
void printstar()  
{  
    cout<<"*****"<<endl;  
}
```

void 可以不返回值

有参函数)

```
int max(int x, y)
```

int x, int y

```
{ int z;  
  z=x>y?x:y;  
  return(z);  
}
```

(X)

函数中的空类型（void）

- **void** 是一种特殊的类型（对比于 **int**, **char** 等）
- **void** 可以表示函数的类型：表示没有返回值
- **void** 可以表示函数的参数：表示没有参数

无参函数

```
void printstar()  
{  
    cout<<"*****"<<endl;  
}
```

无参函数

```
void printstar(void )  
{  
    cout<<"*****"<<endl;  
}
```

```
printstar()  
{  
    cout<<"*****"<<endl;  
    return;  
}
```

C可以这样写

再看不同形式的函数

```
//求两个整数中较大的整数。
int imax(int a, int b){
    if(a>b)
        return a;
    else
        return b;
}
//标准输出两个整数的和。
void printsum(int a, int b){
    cout<<a<<" + "<<b<<" = "<<a+b<<endl;
}
//获得圆周率的值。
double pi(void){
    return 3.1415926;
}
//标准输出一行星号。
void printline(void){
    cout<<"*****"<<endl;
}
```

void类型
无返回值

void
表示无参数

内容

- 函数基本概念与函数定义
- 函数调用、执行与函数定义的位置关系
 - 函数嵌套

对函数的调用

- 函数调用也称函数的使用：使用所定义的函数
- 基本格式：
函数名(参列表)
- 出现的位置：
 - 以语句形式出现；
 - 以表达式形式出现（包括出现在其它函数参数中）；
- 对库函数的调用，需要用 `#include` 包含库的名字；
- 调用函数时的参数，称为实在参数（实参）

函数调用的三种情况

- 以语句形式使用：

例

```
printf("Hello,World!\n"); //不需要返回值
```

- 作为表达式的一部分（带返回值的函数）：

例 `m=max(a,b)*2;`

- 作为参数使用（表达式的特例）：

例 `printf("%d",max(a,b));`

```
m=max(a,max(b,c)); // 三个数的最大值
```

一个实例

- 问题:

- 构造一个函数，按给定的符号和长度值输出一个图形条。
- 函数输入为：给定的符号（字符型）和符号的个数（int型）
- 函数输出（按符号输出）：

XXXXXXXXXXXXXXXX

#####

\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$

一个实例

```
void drawBar(int n, char ch)
{
    int i;
    for(i = 1; i <= n; i++)
        cout << ch;
    cout << endl;
}
```

函数定义：按字符画图

```
const char DOLLAR='$';
int main()
{
    int income;
    char symbol;
    income = 20; symbol = '#';
    drawBar (12, 'X');
    drawBar (3*5, symbol);
    drawBar (income, DOLLAR);
    return 0;
}
```

函数调用（参数对应）

课堂思考

- 下面给出的矩阵代表三年来三种产品的产量,画出条形图

	T1	T2	T3
1990	4	6	8
1991	3	10	5
1992	4	7	9

1990

T1 XXXX

T2 #####

T3 \$\$\$\$\$\$\$\$

1991

T1 XXX

T2 #####

T3 \$\$\$\$\$

1992

T1 XXXX

T2 #####

T3 \$\$\$\$\$\$\$\$

另一个例子

- 随机函数的使用

- 函数 `short int rand();` //在`stdlib.h` 或 `cstdlib`
- 返回一个0到32767之间的值。

```
for (int i=0;i<10;i++)
```

```
    cout<<rand()<<" ";
```

函数调用：以表达式形式出现

```
41 18467 6334 26500 19169 15724
11478 29358 26962 24464
```

一个问题： 如何限定随机值范围

- 如何限定你需要的范围？

- 1-6之间？

- ??

```
rand()%6+1;
```

- 例如 10-50之间？

- ??

```
rand()%41+10;
```

- num1 到 num2 之间 ($\text{num2} > \text{num1}$)

- ??

```
rand()%(num2-num1+1)+num1;
```

rand()函数的几点说明

- rand () 是C/C++库函数提供的一个伪随机函数，重复执行它，得到重复的一组随机函数。
- 可以用另一个函数为rand()提供一个随机选择的种子，种子不同，rand()可以产生不同的随机数
 - unsigned short srand(unsigned short);
 - srand() 用来初始化随机种子数，因为rand的内部实现是用线性同余法做的，他不是真的随机数，只不过是因为其周期特别长，所以有一定的范围里可看成是随机的

种子相同时，产生的伪随机数相同，可用时钟初始化

一个例子

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main()
{
    int a;
    srand((unsigned)time(NULL));
    a = rand() % 51 + 13;
    printf("%d\n",a);
    return 0;
}
```

按时钟值生成种子，再
生成随机数

随机数的范围？

函数调用的执行

- 一个函数调用的执行过程可以分为3 个阶段：
 - ① **保留现场**：首先把实参值传入被调用函数形参的对应单元中，中断主调函数当前的执行，并且保存返回地点(称为断点)。
 - ② **转入被调用的函数执行**：直到return 语句返回。若被调用函数中没有return 语句，则直到其全部语句执行完毕后自动返回到位于主调函数中的断点处。
 - ③ **恢复现场继续执行**：从保存的断点处，主调函数继续执行剩余的语句。

函数调用的执行

```
#include <iostream>
using namespace std;
float max(float a, float b)
{
```

- (1) 初始化max();
- (2) 传递参数;
- (3) 保存当前现场;

```
    return b;
```

```
}
```

```
int main()
```

- (1) 接收函数的返回值;
- (2) 恢复现场, 从断点处继续执行;

```
    return 0;
```

```
}
```

main()

```
int main()
{
    int m = 3, n = 4;

    float result = 0;

    result = max(m, n);

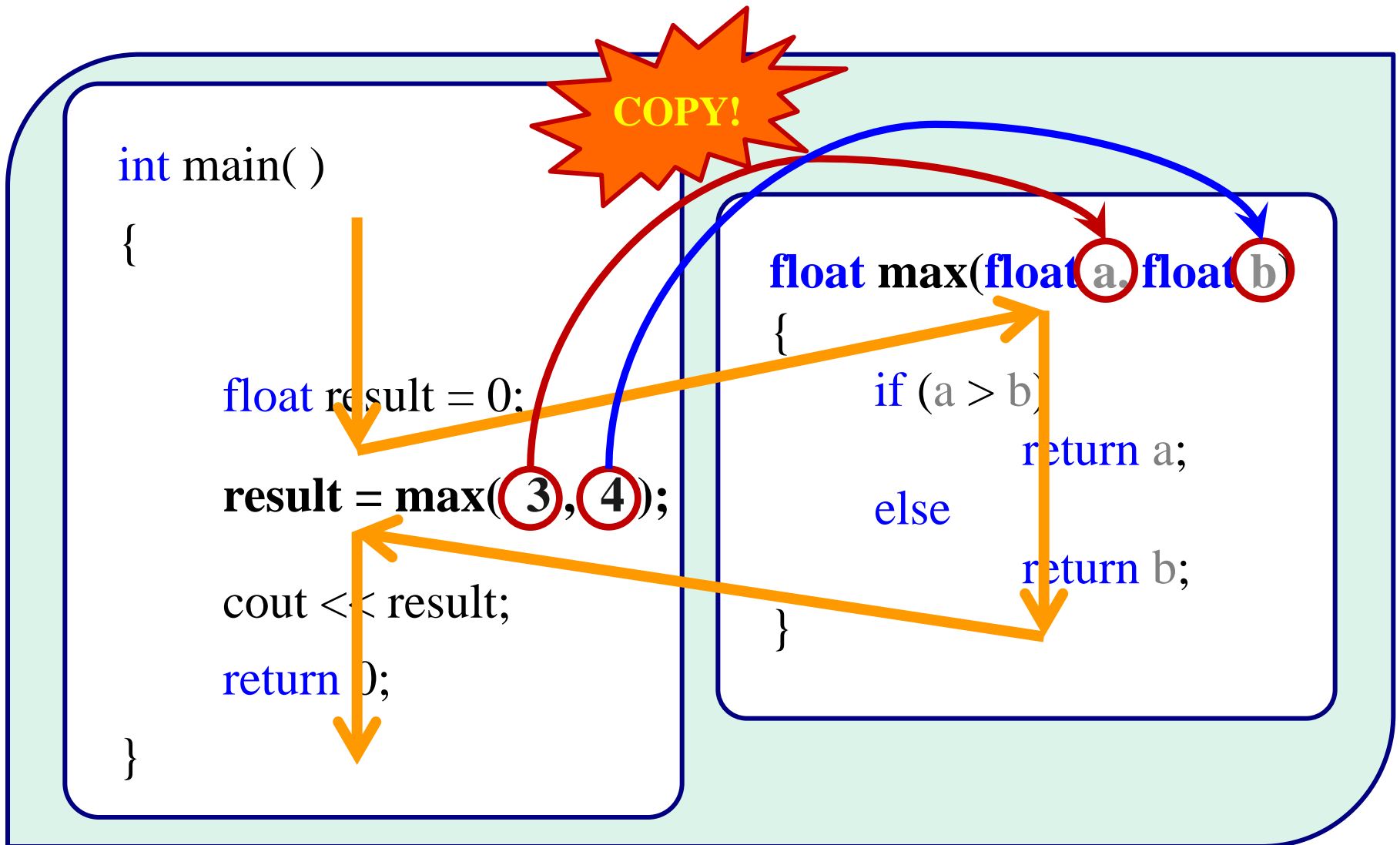
    cout << result;

    return 0;
}
```

max()

```
float max(float a, float b)
{
    if (a > b)
        return a;
    else
        return b;
}
```

函数调用详解



定义的函数与调用函数位置关系

- 定义的函数必须在调用**之前**；
- 如果要在定义前使用，必须将定义的函数的首部作为**函数原型**，提前在调用函数之前声明

再看例子

函数的原型 =
返回值类型 + 函数名 + 参数类型

```
#include <iostream>
using namespace std;
float max(float a, float b)
{
    if (a > b)
        return a;
    else
        return b;
}

int main()
{
    cout<< max(3, 4);
    return 0;
}
```

```
#include <iostream>
using namespace std;
```

float max(float, float);

```
int main()
{
    cout<< max(3, 4);
    return 0;
}

float max(float a, float b)
{
    if (a > b)
        return a;
    else
        return b;
}
```

函数原型声明

- 原型声明 (**不要忘记分号;**) 有两种方式:
 - 直接声明 (见前面例子,) , 格式:
函数类型 函数名 (类型1, 类型2, ...,);
函数类型 函数名 (类型1, 参数1, 类型2, 参数2, ...,);
 - 间接声明 (用 #include)
- 如被调用函数先定义后调用, 可以不声明原型
- 函数原型的引入是为了遵循**先定义, 后使用的规范!** (变量的定义可以先使用, 后定义吗?)

再看原型声明

```
unsigned int max(unsigned int, unsigned int);  
unsigned int max(unsigned int x, unsigned int y);  
void wait_1_minute() ;
```

注意：

- 原型声明与后面定义的函数在类型上，函数名字上，对应的参数个数和参数类型上必须一致。

内容

- 函数基本概念与函数定义
- 函数调用、执行与函数定义的位置关系
- 函数嵌套

C/C++ 函数不能嵌套定义

- C/C++ 函数之间都是并列的结构，不能嵌套定义。

```
int fun1(void)
{
    printf("in fun1!\n");
}
```

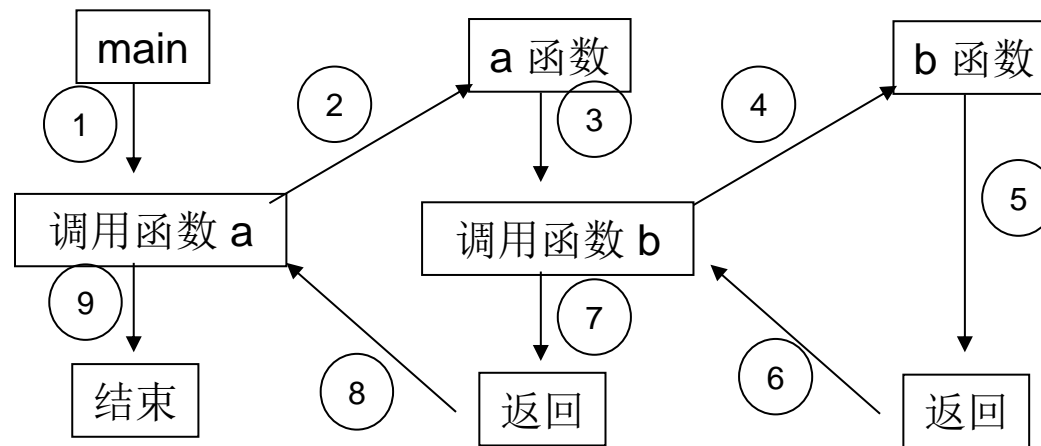
```
void fun2(void)
{
    printf("in fun2!\n");
}
```

```
fun2();
return 0;
}
```

error C2601: "fun2": 本地函数定义是非法的

C/C++ 函数可以嵌套调用

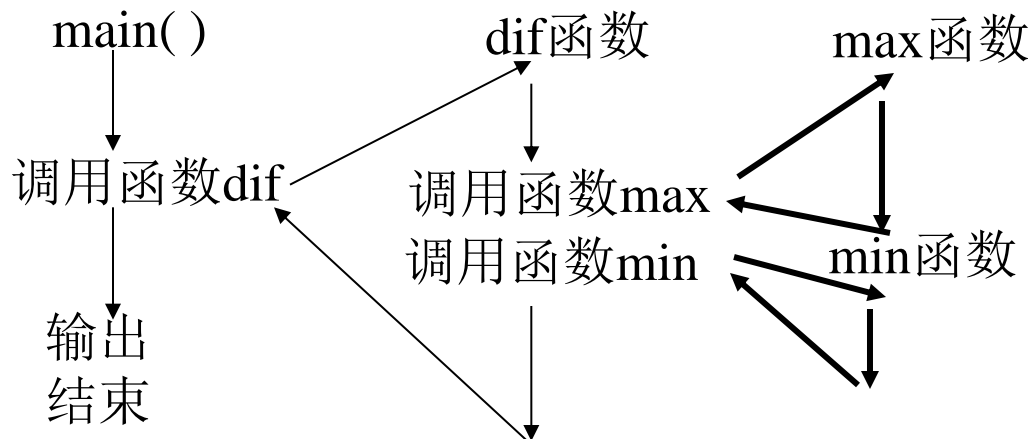
- 调用的嵌套：被调用的函数又调用其他函数



- 从调用上体现嵌套：
`imax(imax(x1,x2),imax(x3,x4))`

三个数中最大数和最小数的差

```
#include <stdio.h>
int dif(int x,int y,int z);
int max(int x,int y,int z);
int min(int x,int y,int z);
void main()
{ int a,b,c,d;
  scanf("%d%d%d",&a,&b,&c);
  d=dif(a,b,c);
  printf("Max-Min=%d\n",d);
}
```

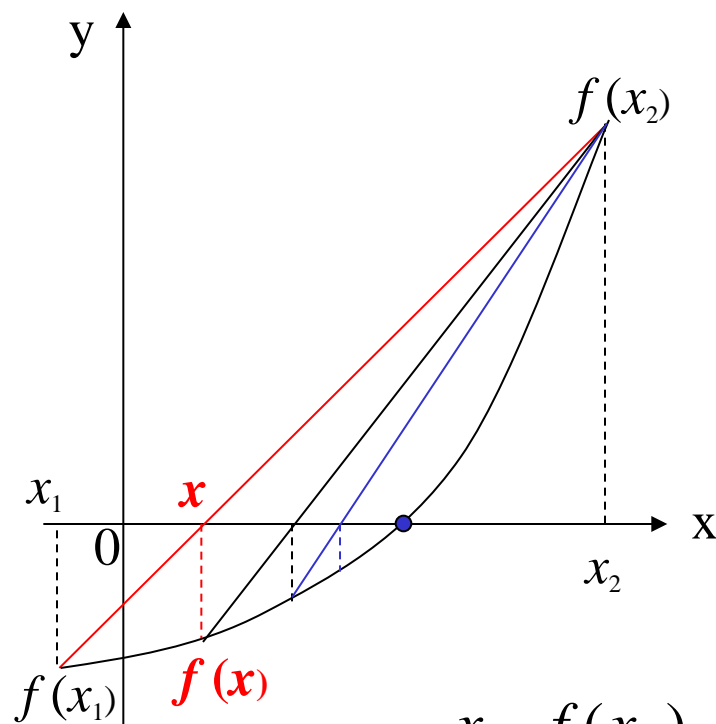


```
int dif(int x,int y,int z)
{ return max(x,y,z)-min(x,y,z); }
```

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

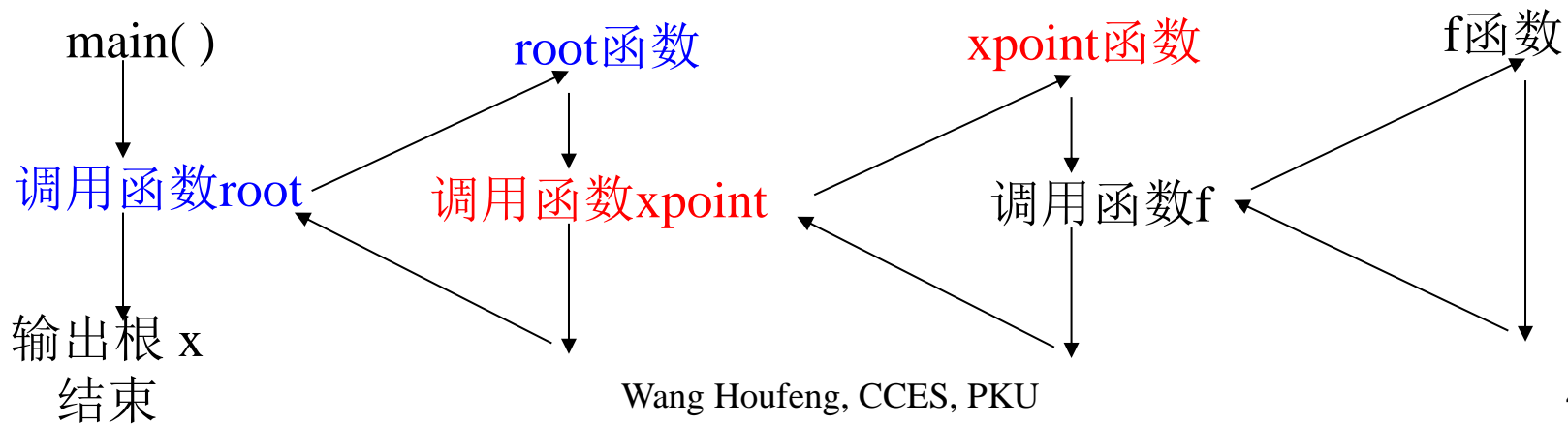
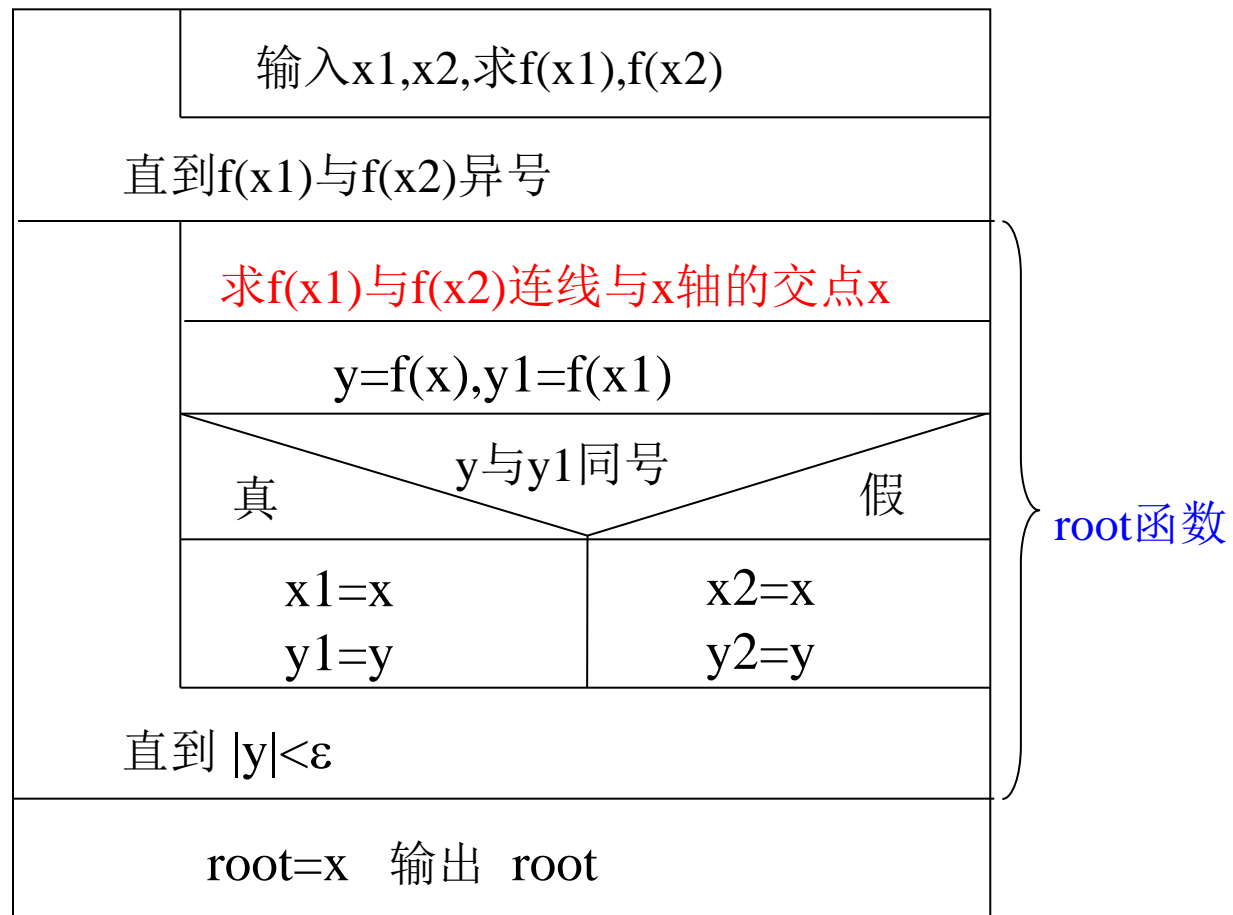
```
int min(int x,int y,int z)
{ int r;
  r=x<y?x:y;
  return(r<z?r:z);
}
```

例 用弦截法求方程根



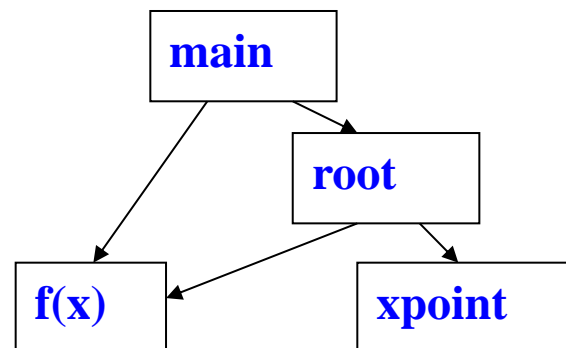
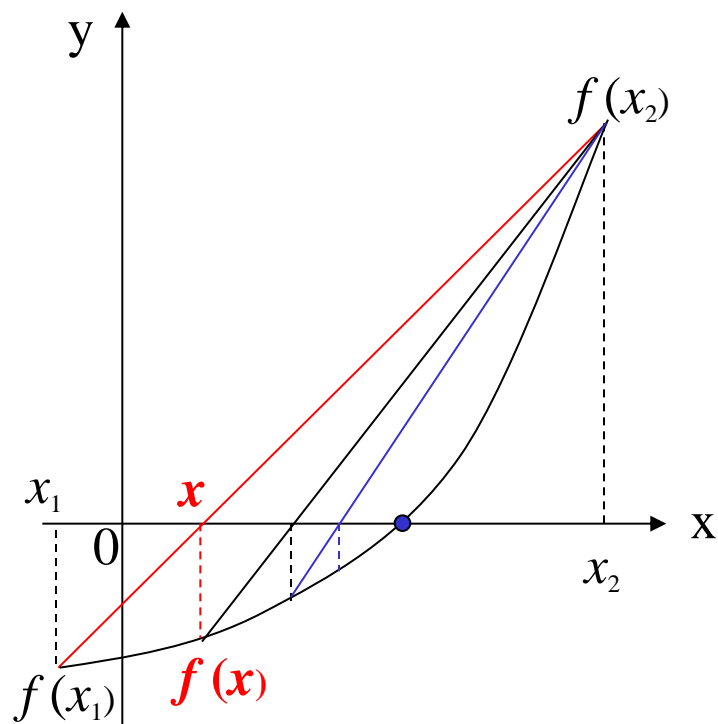
$$f(x) = x^3 - 5x^2 + 16x - 80 = 0$$

$$x = \frac{x_1 \cdot f(x_2) - x_2 \cdot f(x_1)}{f(x_2) - f(x_1)}$$



函数的嵌套组织

$$f(x) = x^3 - 5x^2 + 16x - 80 = 0$$



组织出的函数

- `root(x1, x2)`:
 - 输入 `x1, x2` 输出根;
- 求 `f(x)`:
 - 输入 `x`, 计算 `f(x)` 的值;
- `xpoint(x1, x2)`:
 - 输入 `x1, x2` 输出弦与 `x` 轴的交点;

```

float root(float, float);
float xpoint(float, float);
float f(float x)
int main( ){
    float x1, x2, f1, f2, x;
    do {
        cin>>x1>>x2;
        f1=f(x1); f2=f(x2);
    }while(f1*f2 >= 0)
    x=root(x1,x2);
    cout<<"root is "<<x;
    return 0;
}

```

```

float f(float x)
{
    float y;
    y=((x-5.0)*x+16.0)*x-80.0;
    return(y);
}

```

```

float root(float x1, float x2)
{
    int i; float x, y, y1;
    y1 = f(x1);
    do {
        x = xpoint(x1, x2);
        y = f(x) ;
        if (y*y1>0) { y1= y; x1= x;}
        else x2 = x;
    } while (fabs(y)>=0.0001);
    return(x);
}

```

```

float xpoint(float x1, float x2)
{
    float x;
    x=(x1*f(x2)-x2*f(x1))/(f(x2)-f(x1));
    return(x);
}

```

函数嵌套：一个例子，求组合数

- 书架上有10本不同的书，从中任取2本，有多少种取法？

$$\binom{10}{2} = \frac{10!}{2!(10-2)!}$$

发现公共功能

```
long combinations(int n, int k)
```

```
{
```

```
    return ( fact(n)/(fact(k)*fact(n-k)) );
```

```
}
```

```
long fact(int n)
{
    long t = 1, i = 1;
    for(i = 1; i <= n; i++)
        t = t * i;
    return ( t );
}
```

嵌套：存在多层调用关系

执行

- 对于求组合数的程序，执行函数main()，在main()函数中调用combinations()，执行函数combinations()时又调用了fact()。

main()

```
{ int m = 10,n = 2; long c;  
    c = combinations(m,n);  
}
```

```
long combinations(int n, int k)  
{  
    return ( fact(n)/(fact(k)*fact(n-k)) );  
}
```

long fact(int n)

```
{ long t = 1, i = 1;  
    for(i = 1; i <= n; i++)  
        t = t * i;  
    return ( t );  
}
```

嵌套调用的执行

```
long fact(int n)
```

```
{ long t = 1, i = 1;
```

```
  for(i = 1; i <= n; i++)
```

```
    t = t * i;
```

```
  return ( t );
```

```
}
```

```
long combinations(int n, int k)
```

```
{
```

```
  return ( fact(n)/fact(k)*fact(n-k) );
```

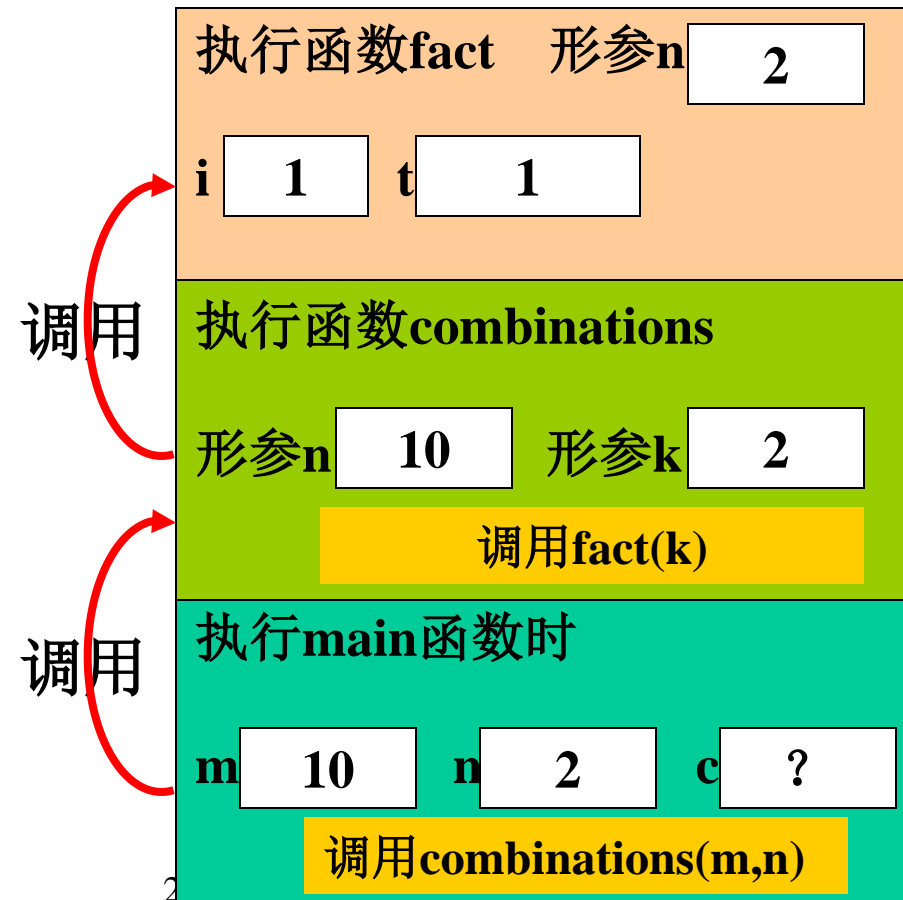
```
}
```

```
main()
```

```
{ int m = 10, n = 2; long c;
```

```
  c = combinations(m,n);
```

```
}
```



嵌套调用的执行

```
long fact(int n)
```

```
{ long t = 1, i = 1;
```

```
  for(i = 1; i <= n; i++)
```

```
    t = t * i;
```

```
  return ( t );
```

```
}
```

```
long combinations(int n, int k)
```

```
{
```

```
  return ( fact(n)/(fact(k))*fact(n-k) );
```

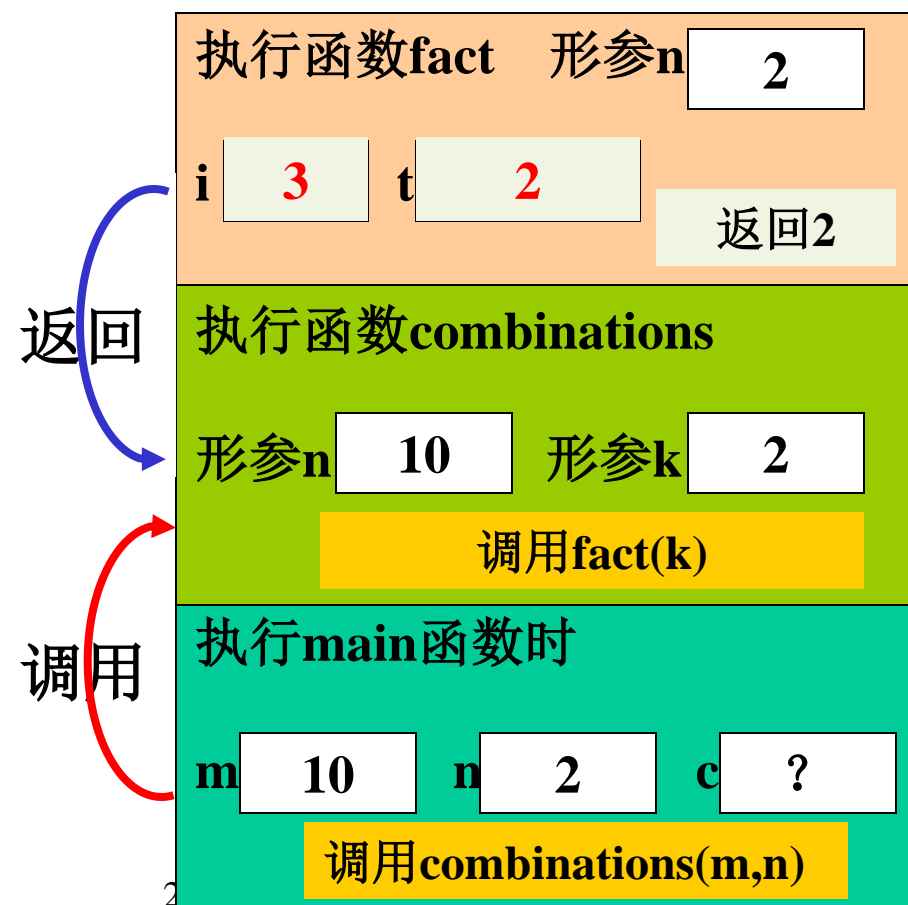
```
}
```

```
main()
```

```
{ int m = 10,n = 2; long c;
```

```
  c = combinations(m,n);
```

```
}
```



嵌套调用的执行

```
long fact(int n)
```

```
{ long t = 1, i = 1;
```

```
  for(i = 1; i <= n; i++)
```

```
    t = t * i;
```

```
  return ( t );
```

```
}
```

```
long combinations(int n, int k)
```

```
{
```

```
  return ( fact(n)/( 2 * fact(n-k) );
```

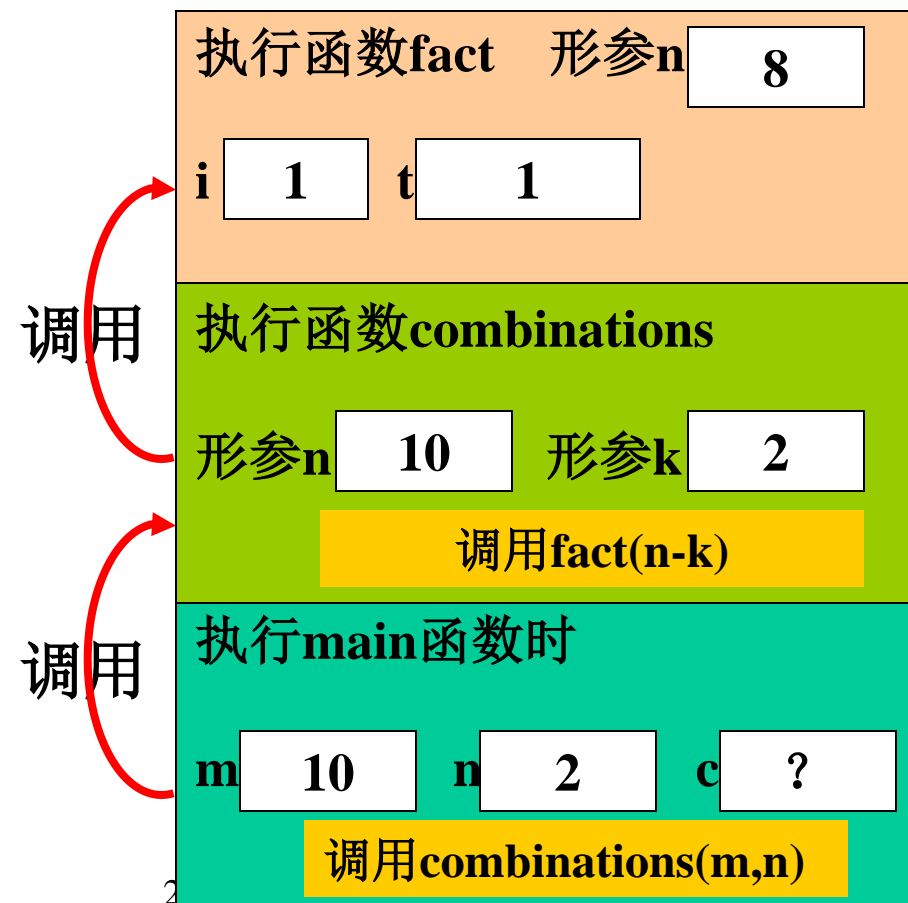
```
}
```

```
main()
```

```
{ int m = 10, n = 2; long c;
```

```
  c = combinations(m,n);
```

```
}
```



嵌套调用的执行

```
long fact(int n)
```

```
{ long t = 1, i = 1;
```

```
    for(i = 1; i <= n; i++)
```

```
        t = t * i;
```

```
    return ( t );
```

```
}
```

```
long combinations(int n, int k)
```

```
{
```

```
    return ( fact(n)/( 2 * fact(n-k) );
```

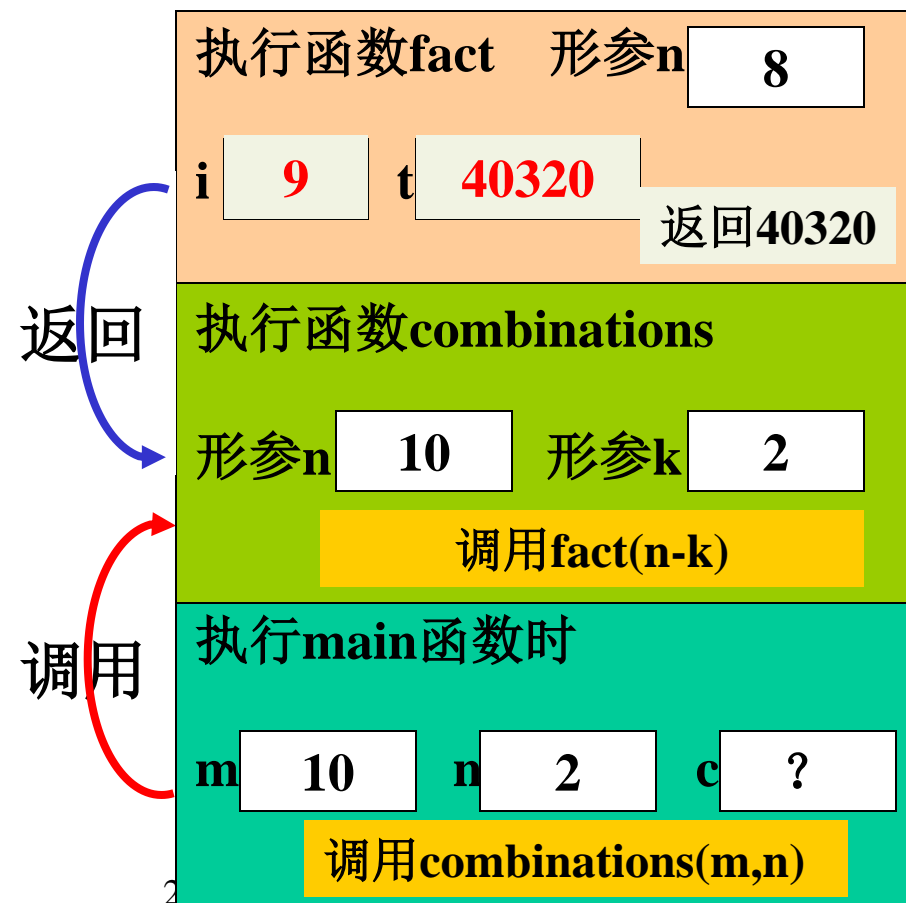
```
}
```

```
main()
```

```
{ int m = 10, n = 2; long c;
```

```
    c = combinations(m,n);
```

```
}
```



嵌套调用的执行

```
long fact(int n)
```

```
{ long t = 1, i = 1;
```

```
  for(i = 1; i <= n; i++)
```

```
    t = t * i;
```

```
  return ( t );
```

```
}
```

```
long combinations(int n, int k)
```

```
{
```

```
  return ( fact(n)/( 2 * 40320 );
```

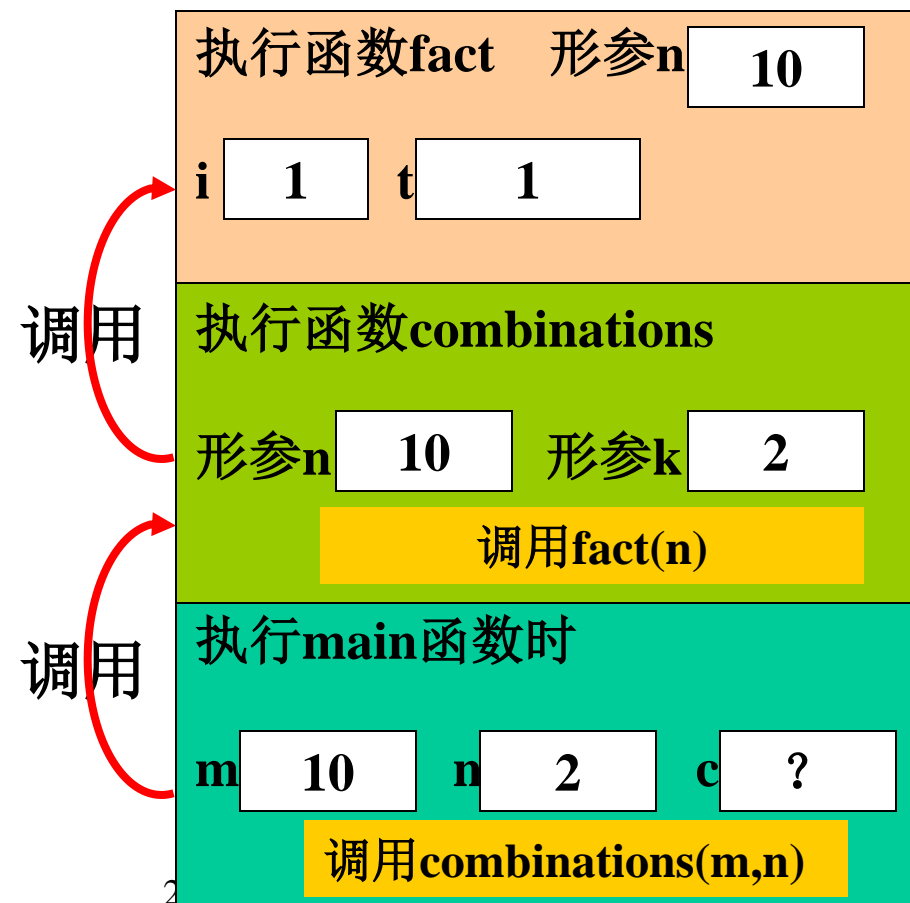
```
}
```

```
main()
```

```
{ int m = 10,n = 2; long c;
```

```
  c = combinations(m,n);
```

```
}
```



嵌套调用的执行

```
long fact(int n)
```

```
{ long t = 1, i = 1;
```

```
  for(i = 1; i <= n; i++)
```

```
    t = t * i;
```

```
  return ( t );
```

```
}
```

```
long combinations(int n, int k)
```

```
{
```

```
  return ( fact(n)/( 2 * 40320 );
```

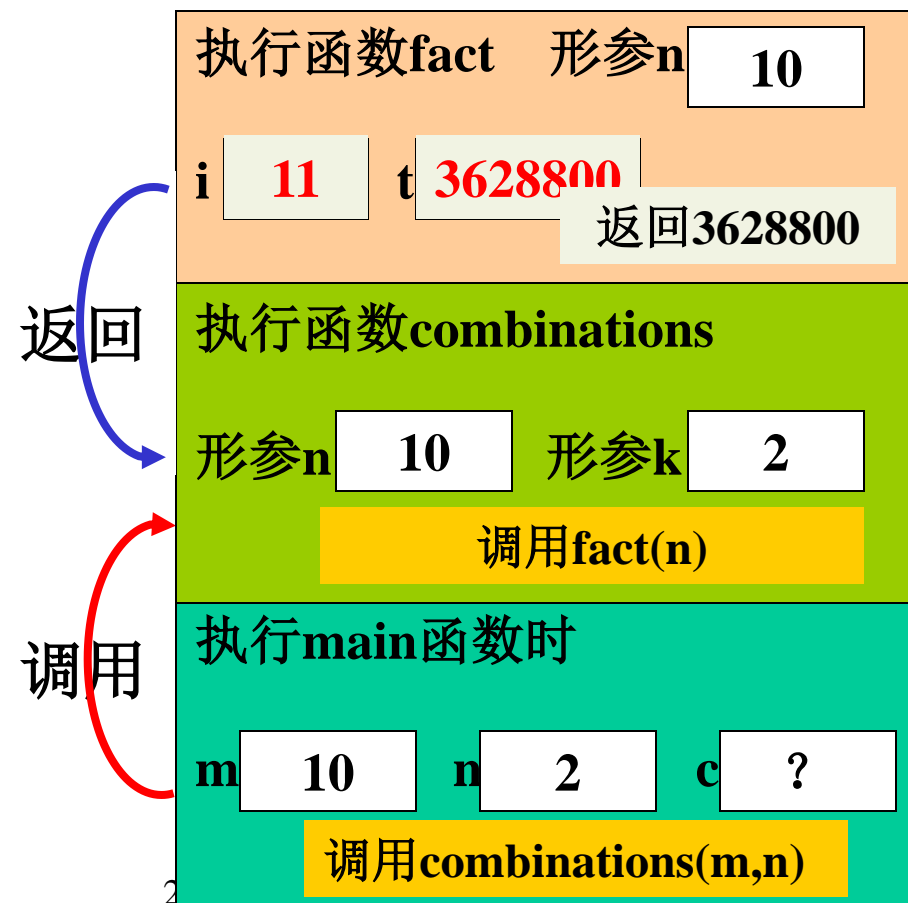
```
}
```

```
main()
```

```
{ int m = 10,n = 2; long c;
```

```
  c = combinations(m,n);
```

```
}
```



嵌套调用的执行

```
long fact(int n)
```

```
{ long t = 1, i = 1;
```

```
    for(i = 1; i <= n; i++)
```

```
        t = t * i;
```

```
    return ( t );
```

```
}
```

```
long combinations(int n, int k)
```

```
{
```

```
    return 3628800 / ( 2 * 40320 );
```

```
}
```

```
main()
```

```
{ int m = 10, n = 2; long c;
```

```
    c = combinations(m,n);
```

```
}
```

执行函数combinations

形参n 10

形参k 2

返回45

执行main函数时

m 10

n 2

c ?

调用combinations(m,n)

2

eng, }

返回

嵌套调用的执行

```
long fact(int n)
```

```
{ long t = 1, i = 1;
```

```
    for(i = 1; i <= n; i++)
```

```
        t = t * i;
```

```
    return ( t );
```

```
}
```

```
long combinations(int n, int k)
```

```
{
```

```
    return ( fact(n) / (fact(k)*fact(n-k)) );
```

```
}
```

```
main()
```

```
{ int m = 10, n = 2; long c;
```

```
    c = 45 ;
```

```
}
```

执行main函数时

m	10	n	2	c	45
---	----	---	---	---	----