School of Artificial Intelligence
Jilin University

C++面向对象程序设计

人工智能 学院

田原

课程纲要

① **第三章 函数**

函数的定义与使用

内联函数

带默认形参的函数

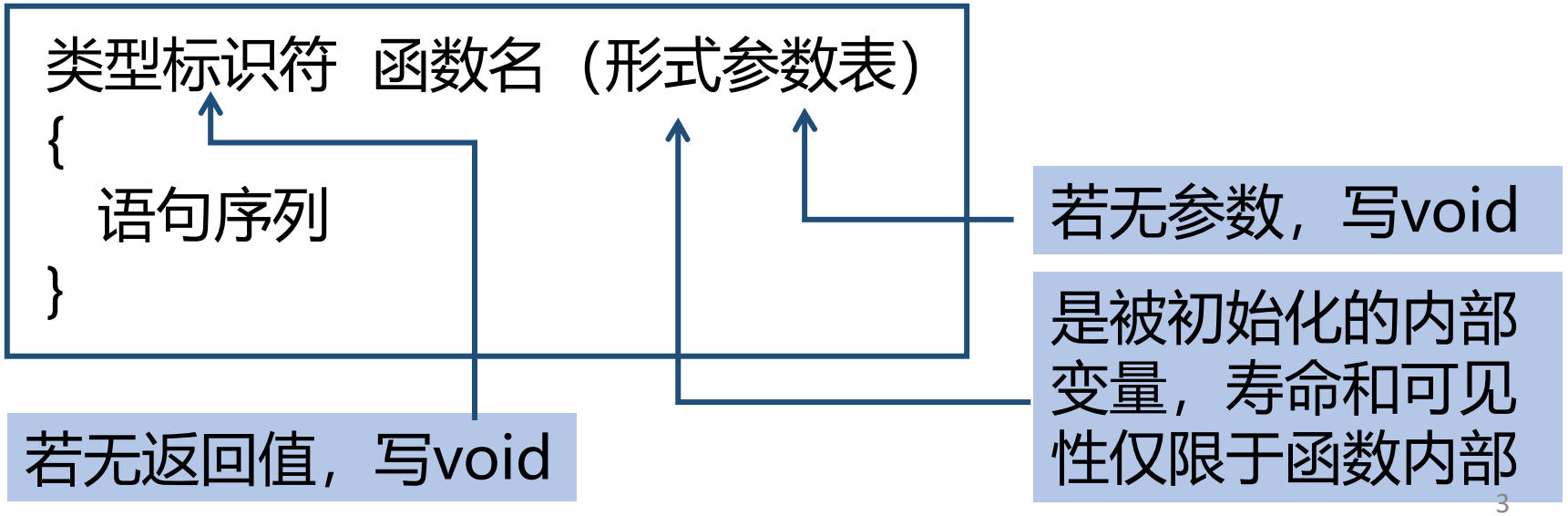
函数重载

使用C++系统函数

深度探索

函数的定义

- 函数的作用：
  - 在面向过程的程序设计中，函数是问题解决过程的抽象；
  - 在面向对象程序设计中，函数对类功能的抽象；
- 函数定义语法形式：



函数的定义

- 形式参数表
  - type1 name1, type2 name2, ..., typen namen
- 函数的返回值
  - 由 return 语句给出，例如：
    return 0;
  - 无返回值的函数（void类型），不必写return语句，或 return;

函数的声明

- 函数声明语法形式：

  类型标识符 函数名（形式参数表）

- 形式参数表
  - type1 name1, type2 name2, ..., typen namen
  - type1, type2, ..., typen

函数的调用

- 调用模式
  - 先定义，然后调用；
  - 先声明，然后调用，最后给出定义；
- 调用形式

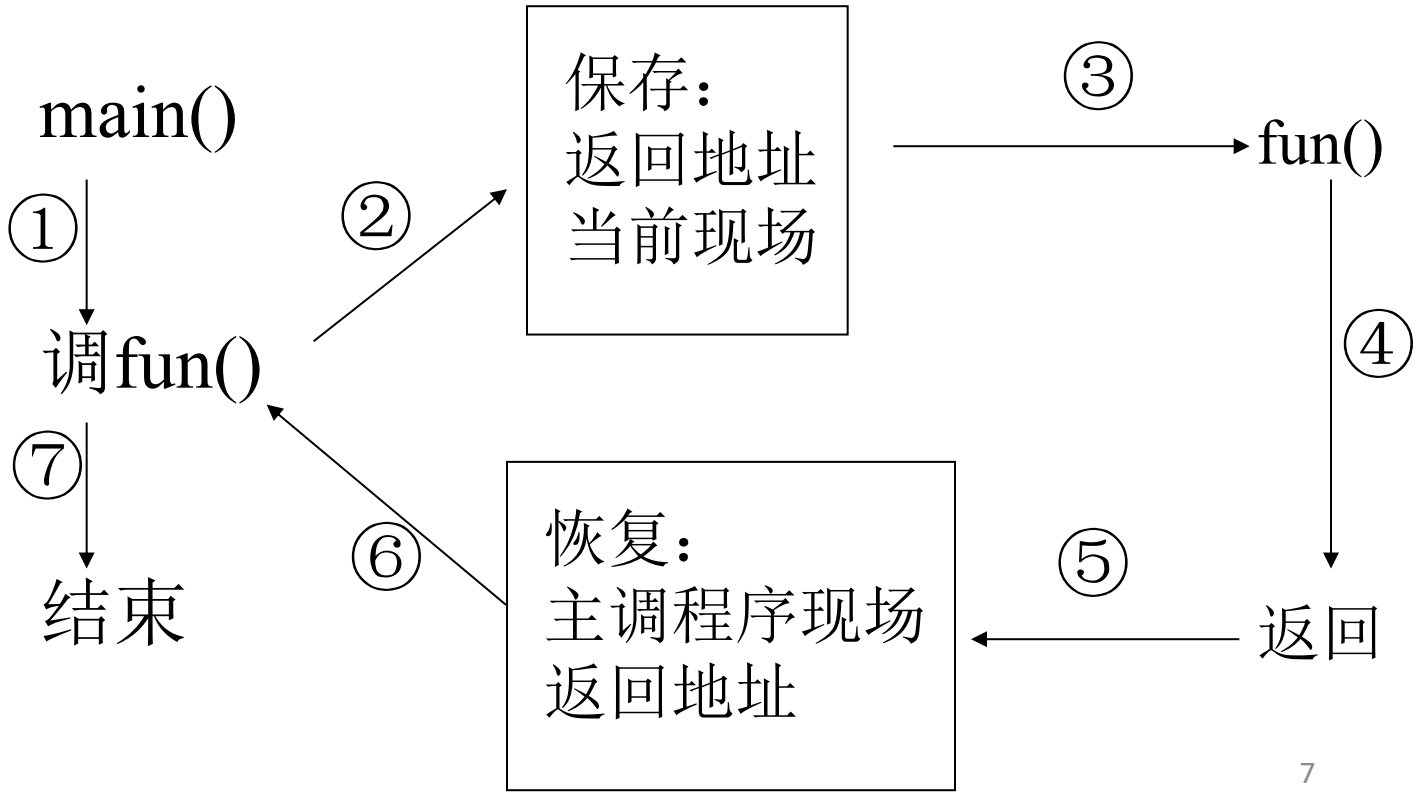
  函数名（实参列表）；
- 嵌套调用

  函数可以嵌套调用，但不允许嵌套定义。
- 递归调用

  函数直接或间接调用自身。

函数调用的执行过程

```
void main(void){
    int a = 1;
    double b = 2.0;
    fun();
    …
}
```

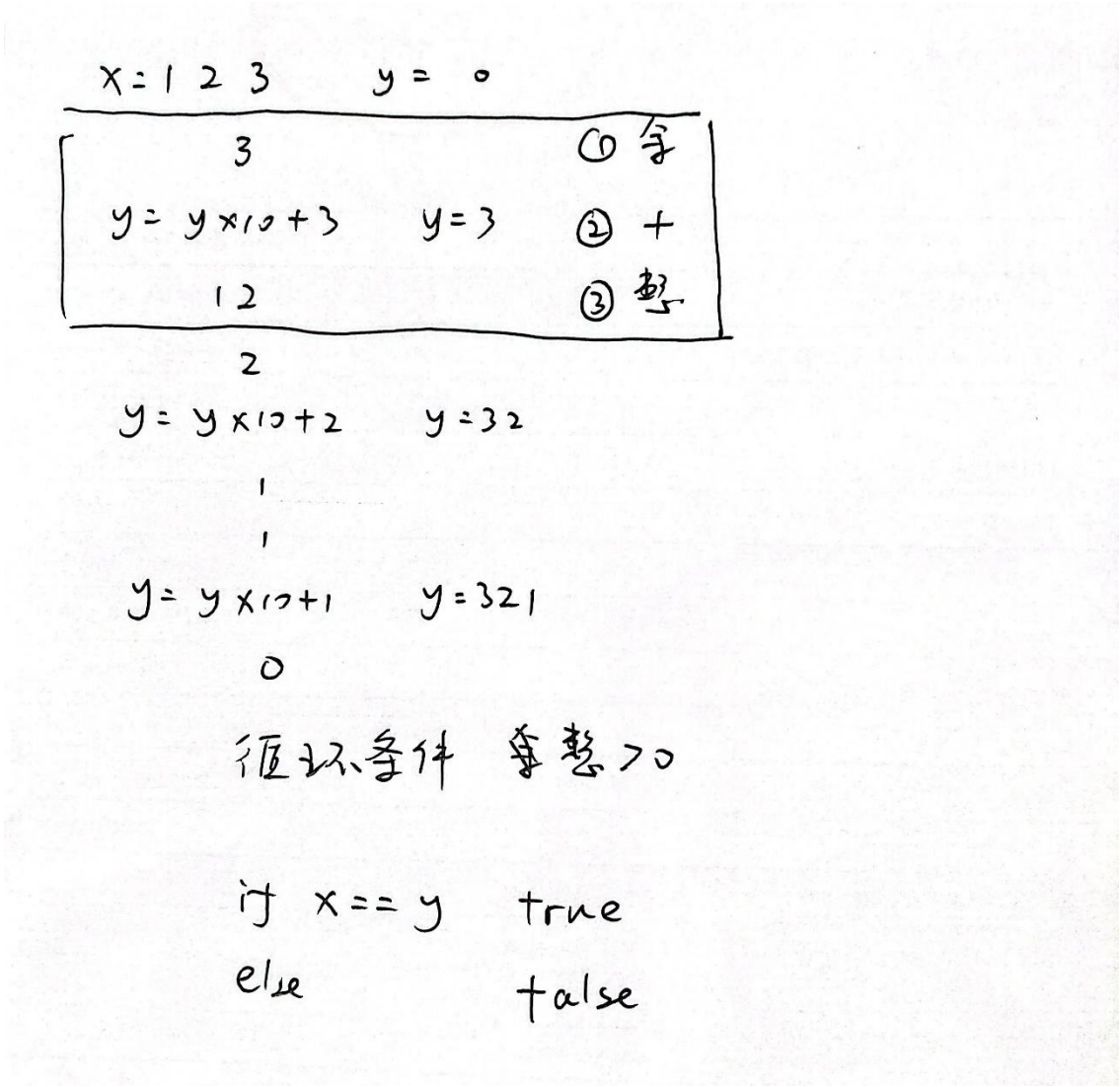


函数的调用

- 例3-4：寻找并输出11~999之间的数m，它满足m、m^2和m^3均为回文数。
  - 回文：各位数字左右对称的整数。
    例如：11满足上述条件
    11^2=121，113=1331。

- 思想：
  - 10取余的方法，从最低位开始，依次取出该数的各位数字，按反序重新构成新的数。比较与原数是否相等，若相等，则原数为回文。

函数的调用


x=123 y=0
3
y=yx10+3 y=3
12
①拿
②+
③整
2
y=yx10+2 y=32
1
y=yx10+1 y=321
0
循环条件 拿整>0
if x==y true
else false

```
#include <iostream>
using namespace std;
void main()
{
  bool symm(long n);
  long m;
  for(m=11; m<1000; m++)
  if (symm(m)&&symm(m*m)&&symm(m*m*m))
    cout<<"m="<<m<<"  m*m="<<m*m
          <<"  m*m*m="<<m*m*m<<endl;
}
```

```
bool symm(long n)
{
  long i, m;
  i=n ;  m=0 ;
  while(i)
  {
   m=m*10+i%10;
   i=i/10   ;
  }
  return ( m==n );
}
```

运行结果：

m=11 m*m=121 m*m*m=1331

m=101 m*m=10201 m*m*m=1030301

m=111 m*m=12321 m*m*m=1367631

函数的调用

- 例3-3编写程序求π的值

$$\pi = 16\arctan\left(\frac{1}{5}\right) - 4\arctan\left(\frac{1}{239}\right)$$

其中arctan用如下形式的级数计算：

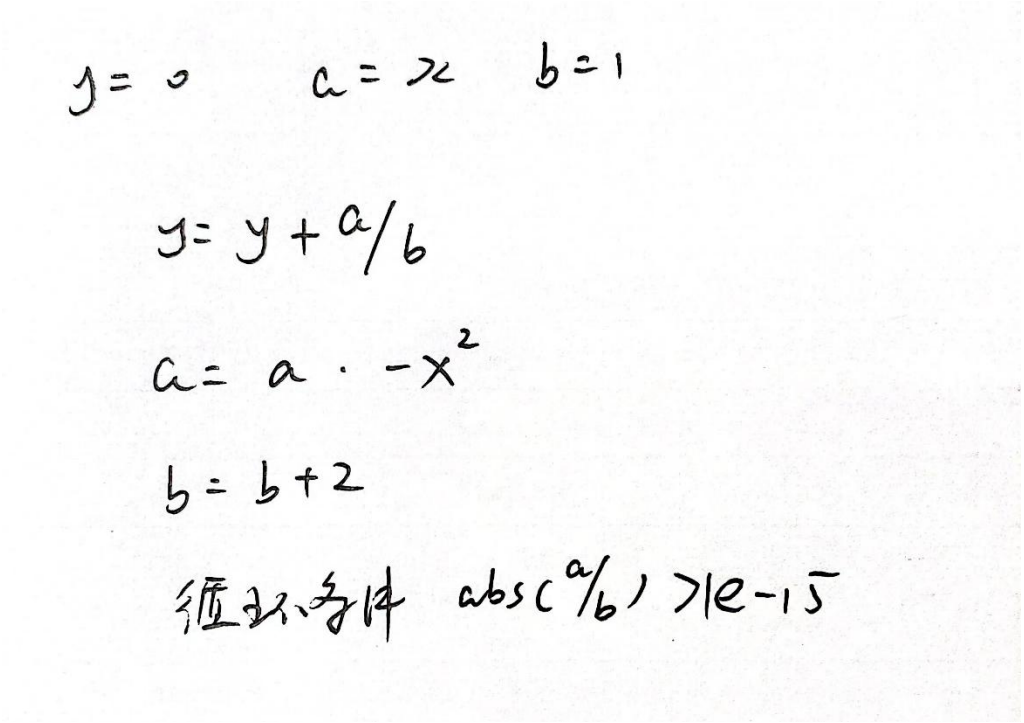
$$\arctan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots$$

直到级数某项绝对值不大于10^{-15}为止；π和x均为double型。

思想：arctan函数是一个累加，有明确结束条件的累加。

函数的调用

$$\arctan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots$$



```
#include<iostream>
using namespace std;
void main()
{
  double a,b;
  double arctan(double x) ; //函数原型声明
  a=16.0*arctan(1/5.0) ;
  b=4.0*arctan(1/239.0) ;
    //注意：因为整数相除结果取整，
    //如果参数写1/5，1/239，结果就都是0
  cout<<"PI="<<a-b<<endl;
}
```

```
double arctan(double x)
{  int i;
   double r,e,f,sqr;
   sqr=x*x;
   r=0;    e=x;    i=1;
   while(e/i>1e-15)
   {
     f=e/i;
     r=(i%4==1)? r+f : r-f   ;
     e=e*sqr;      i+=2;
   }
   return  r ;
}
```

运行结果：

PI = 3.14159

函数的调用

□例3-6投骰子的随机游戏

- 游戏规则是：每个骰子有六面，点数分别为1、2、3、4、5、6。游戏者在程序开始时输入一个无符号整数，作为产生随机数的种子。
- 每轮投两次骰子，第一轮如果和数为7或11则为胜，游戏结束；和数为2、3或12则为负，游戏结束；和数为其它值则将此值作为自己的点数，继续第二轮、第三轮...直到某轮的和数等于点数则取胜，若在此前出现和数为7则为负。
- 由rolldice函数负责模拟投骰子、计算和数并输出和数。

函数的调用

□思想：

- rolldice函数生成1-6的随机数两次，就和输出；
- 主函数：转态的转换，有明确结束条件的循环；

函数的调用

- rand

  函数原型：int rand(void);

  所需头文件：<cstdlib>

  功能和返回值：求出并返回一个大于等于0小于等于最大整型的伪随机数。

- srand

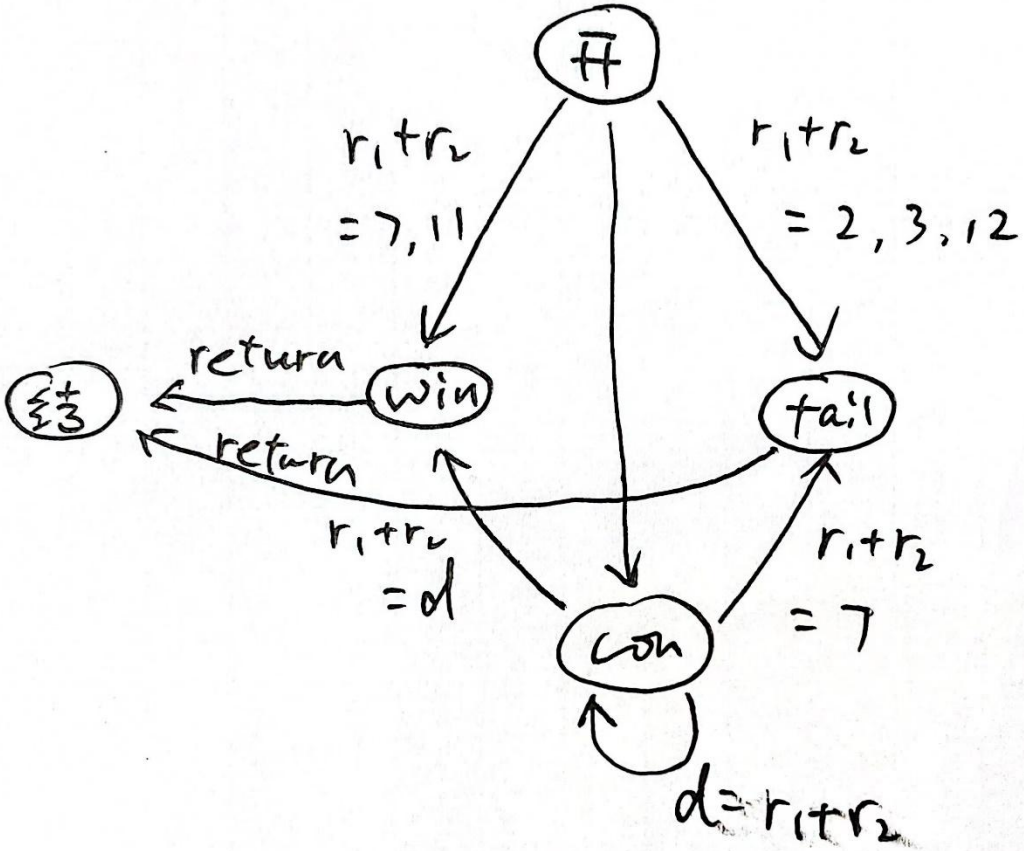
  函数原型：void srand(unsigned int seed);

  参数：seed产生随机数的种子。

  所需头文件：<cstdlib>

  功能：为使rand()产生一序列伪随机整数而设置起始点。使用1作为seed参数，可以重新初化rand()。

函数的调用


开
r1+r2
=7,11
r1+r2
=2,3,12
return
结
win
fail
return
r1+r2
=d
r1+r2
=7
con
d=r1+r2

```
int rolldice(void)
{ //投骰子、计算和数、输出和数
  int die1,die2,worksum;
  die1=1+rand()%6;
  die2=1+rand()%6;
  worksum=die1+die2;
  cout<<"player rolled
   "<<die1<<'+'<<die2<<'='<<worksum<<endl;
  return worksum;
}
```

```
#include <iostream>
#include <cstdlib>
using namespace std;
int rolldice(void);
void main()
{
  int gamestatus,sum,mypoint;
  unsigned seed;
  cout<<"Please enter an unsigned integer:";
  cin>>seed;          //输入随机数种子
  srand(seed);        //将种子传递给rand()
  sum=rolldice();   //第一轮投骰子、计算和数
```

```
switch(sum)
{
  case 7:  //如果和数为7或11则为胜,状态为1
  case 11:  gamestatus=1;
             break;
  case 2:  //和数为2、3或12则为负,状态为2
  case 3:
  case 12:  gamestatus=2;
            break;
  default:  //其它情况,游戏尚无结果,状态为0,记下点数,为下一轮做准备
       gamestatus=0;
       mypoint=sum  ;
       cout<<"point is "<<mypoint<<endl;
  break;
}
```

```
  while ( gamestatus==0 ) //只要状态仍为 0,就继续进行下一轮
  {
    sum=rolldice();
    if(sum==mypoint)    //某轮的和数等于点数则取胜,状态置为1
      gamestatus=1 ;
    else
      if ( sum==7 )    //出现和数为7则为负,状态置为2
        gamestatus=2;
  }
//当状态不为0时上面的循环结束,以下程序段输出游戏结果
  if( gamestatus==1 )
    cout<<"player wins\n";
  else
    cout<<"player loses\n";
}
```

运行结果2：

Please enter an unsigned integer:23

player rolled 6+3=9

point is 9

player rolled 5+4=9

player wins

函数的调用

例3-5：计算如下公式，并输出结果：

$$k=\begin{cases}\sqrt{SIN^2(r)+SIN^2(s)} & if\ r^2\le s^2\\ \frac{1}{2}SIN(r*s) & if\ r^2>s^2\end{cases}$$

其中r、s的值由键盘输入。SIN (x)的近似值按如下公式计算，计算精度为10^{-6}，当累加某一项的值小于精度时，然后停止计算（第一次除外）

$$SIN(x)=\frac{x}{1!}-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots=\sum_{n=1}^{\infty}(-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!}$$

思想：累加，有明确的终止条件，第一次除外。

```
#include <iostream>
#include<cmath>
using namespace std;
void main()
{
  double k,r,s;
  double tsin(double x);
  cout<<"r=";
  cin>>r;
  cout<<"s=";
  cin>>s;
  if (r*r<=s*s)
    k=sqrt(tsin(r)*tsin(r)+tsin(s)*tsin(s))  ;
  else
    k=tsin(r*s)/2;
  cout<<k<<endl;
}
```

```
double tsin(double x)
{
  const double p=0.000001,g=0,t=x;
  int n=1;
  do {
     g=g+t;
     n++;
     t=-t*x*x/(2*n-1)/(2*n-2);
  }while(fabs(t)>=p);
  return g;
}
```

运行结果：

r = 5

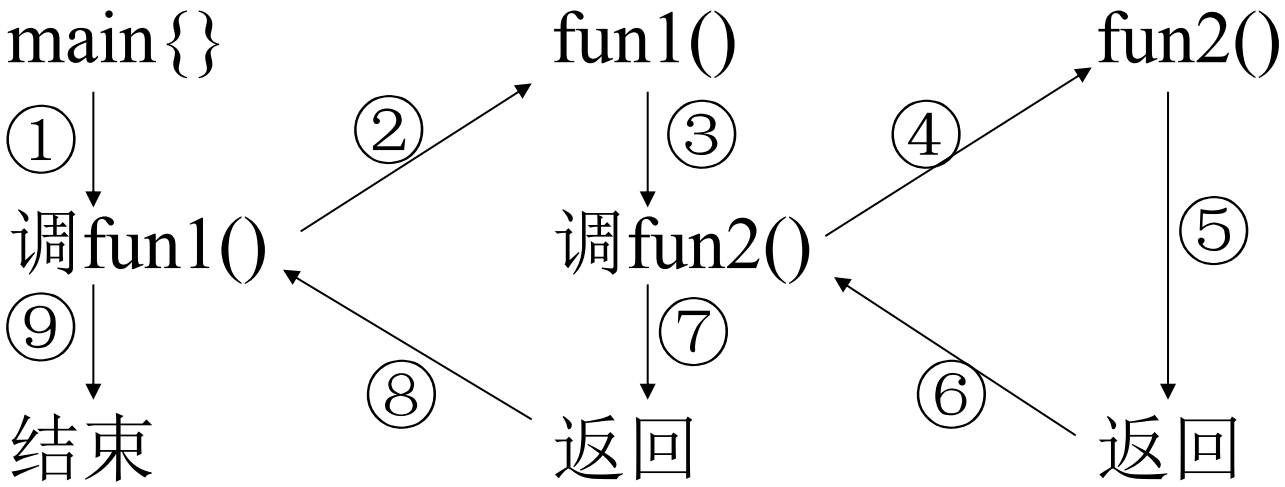
s = 8

1.37781

函数嵌套调用的执行过程

```
void fun2(void){
   …
}
int fun1(void){
   fun2();
   …
return 1;
}
```

```
void main(void){
   fun1();
}
```



函数嵌套调用举例

- 例3-7：计算平方和
- 思想：定义fun2计算平方；定义fun1计算平方和，调用fun2计算；主函数调用fun1，并加入输入，输出。

```
#include <iostream>
using namespace std;
void main(void)
{
   int a,b;
   int fun1(int x,int y);
   cin>>a>>b;
   cout<<"a、b的平方和：  "
            <<fun1(a,b)<<endl;
}
```

```
int fun1(int x,int y)
{
   int fun2(int m);
   return (fun2(x)+fun2(y));
}

int fun2(int m)
{
   return (m*m);
}
```

运行结果：

3 4

a、b的平方和：25

函数递归调用

- 定义：函数直接或间接地调用自身，称为递归调用。

```
int  test(int  x){
     int   y;
     y=test(x);
     return (2*y);
}
```

• 直接调用

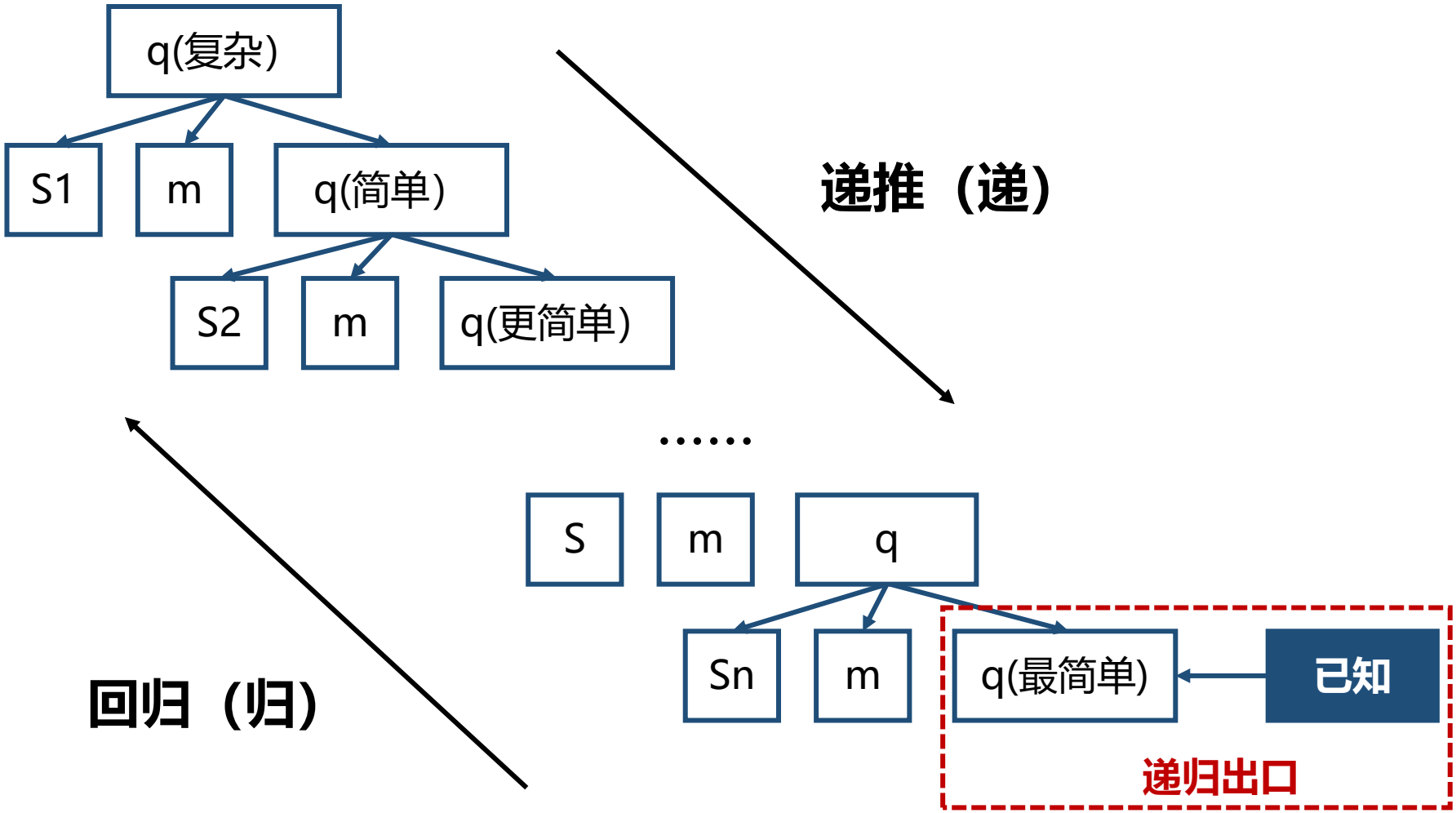
```
int first(int x){
      int b;
      b=second(x);
      return(2*b);
}
int second(int y){
      int a;
      a=first(y);
      return(2*a);
}
```

• 间接调用

函数递归调用

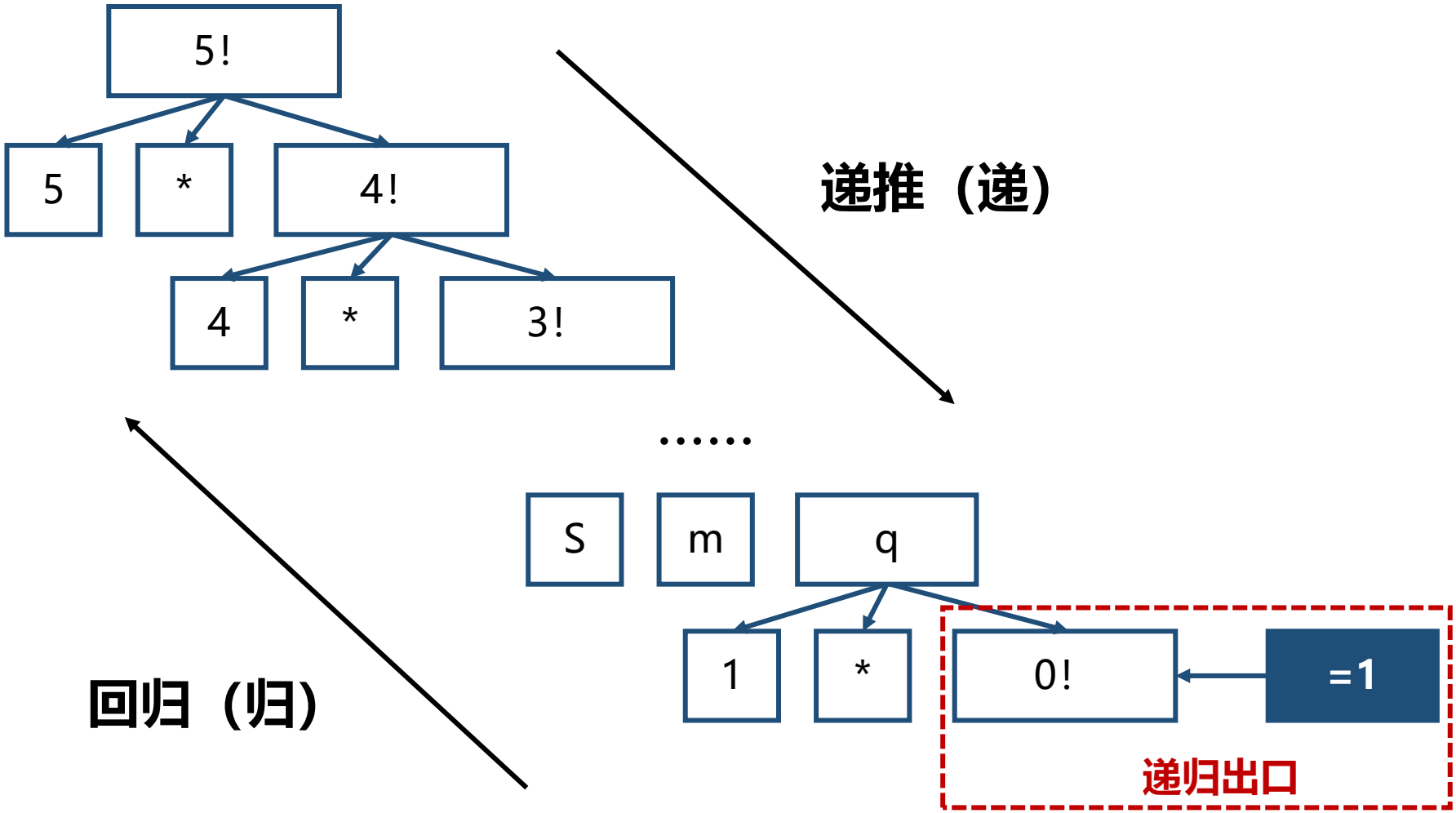
- 应用场景：一个问题q(复杂)可以分解为{已知元素S1，关系m，q(简单）}，q(简单)可以在分解为{已知元素S2，关系m，q(更简单）}，……，q可以在分解为{已知元素Sn，关系m，q(最简单）}，q(最简单）是已知的。

函数递归调用



函数递归调用

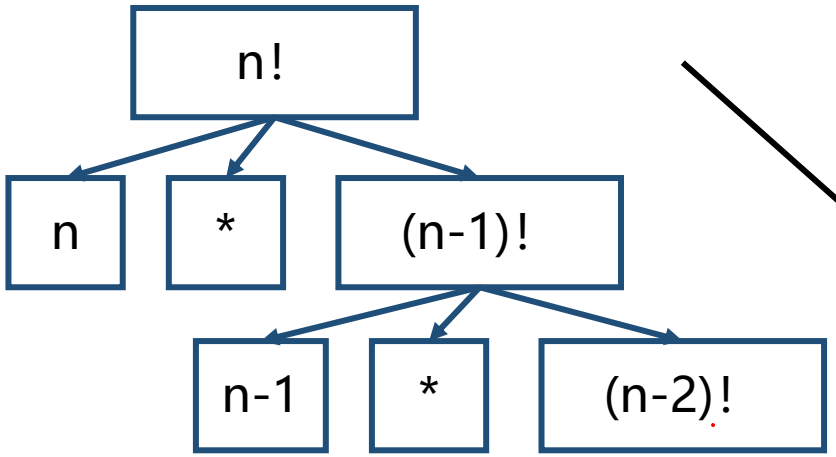
- 例如：5!



函数递归调用

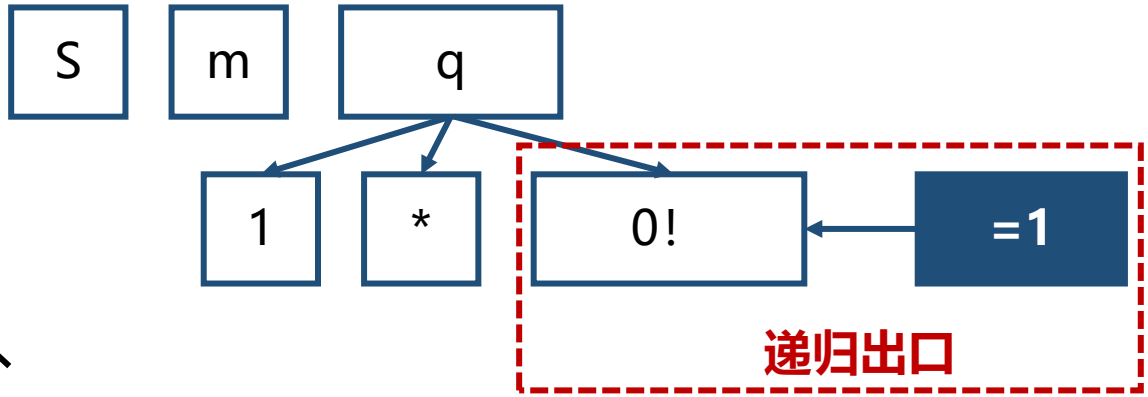
例3-8：n!

$$n! = \begin{cases} 1 & (n = 0) \\ n(n-1)! & (n > 0) \end{cases}$$



递推（递）

……



回归（归）

```
#include <iostream>
using namespace std;
long fac(int n)
{
  long f;
  if (n<0)
      cout<<"n<0,data error!"<<endl;
  else if (n==0) f=1;
  else f=fac(n-1)*n;
  return(f);
}
```

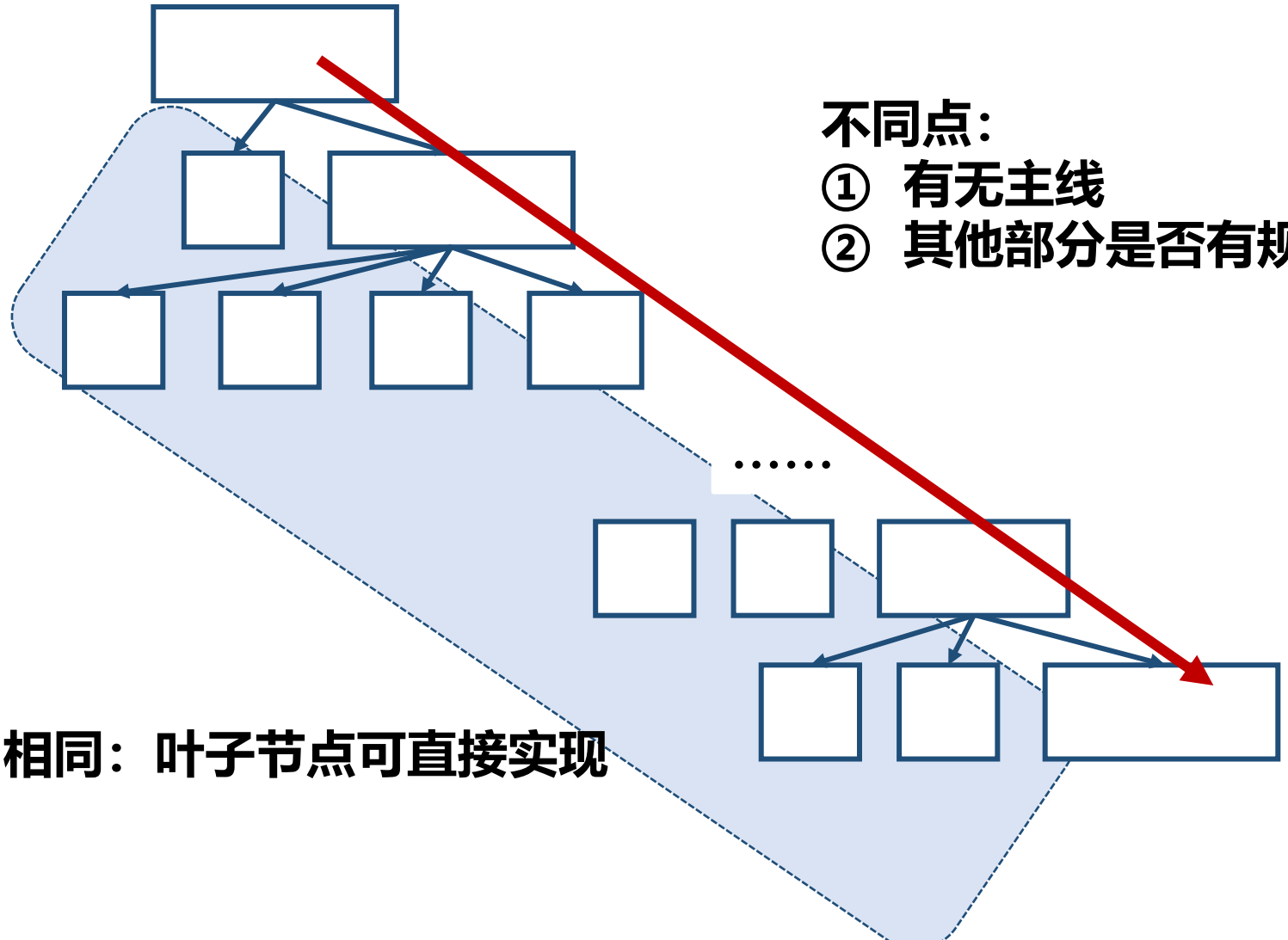
```
void main()
{
  long fac(int n);
  int n;
  long y;
  cout<<"Enter a positive integer:";
  cin>>n;
  y=fac(n);
  cout<<n<<"!="<<y<<endl;
}
```

运行结果：

Enter a positive integer:8

8!=40320

函数递归调用 VS 分治法（函数调用）



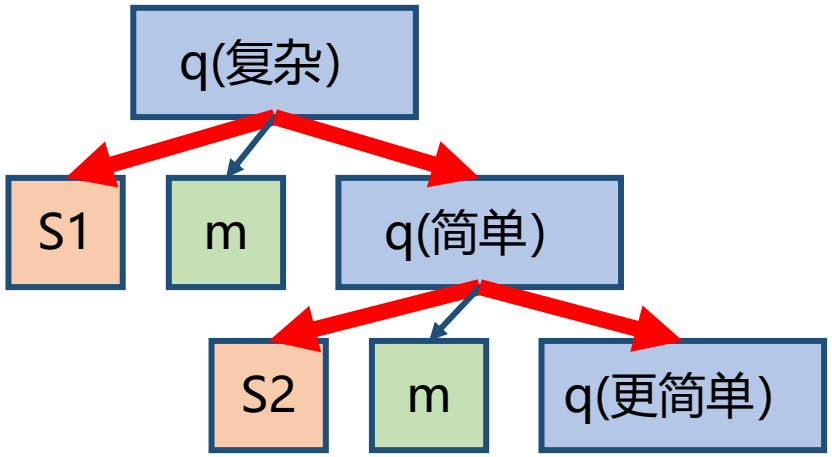
不同点：

① 有无主线

② 其他部分是否有规律

相同：叶子节点可直接实现

函数递归调用



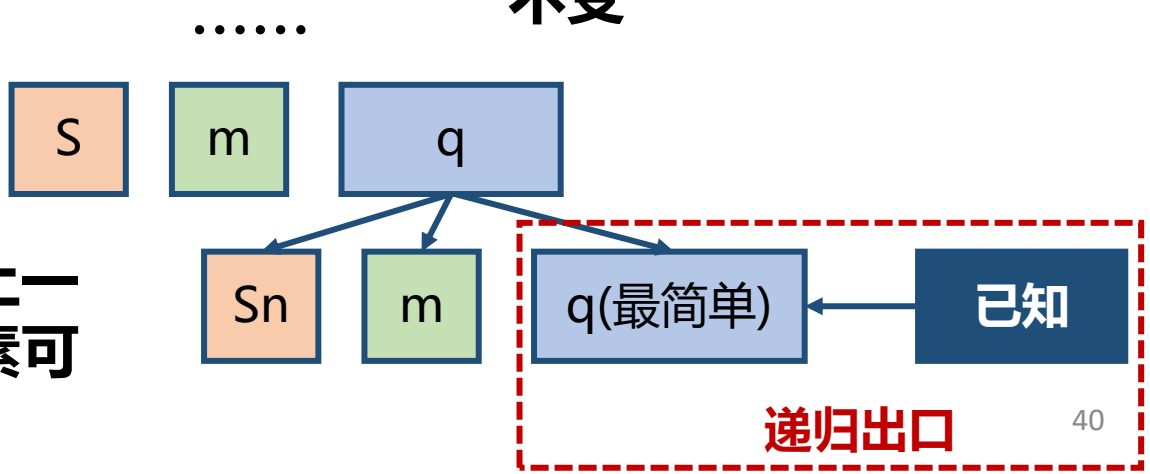
4不变：

① 不同层主元素的输入与输出关系不变

② 不同层元素间的关系不变

③ 不同层附元素的输入与输出关系不变

④ 上层主元素输入与下一层主元素（和附元素）输入关系不变



1注意：

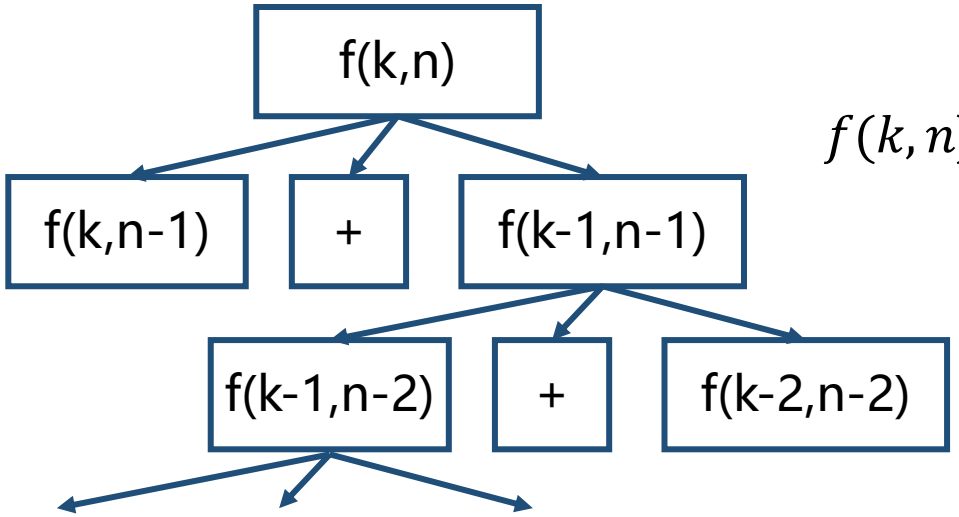
下一层主元素的输入比上一层简单，最终导致主元素可直接实现

函数递归调用

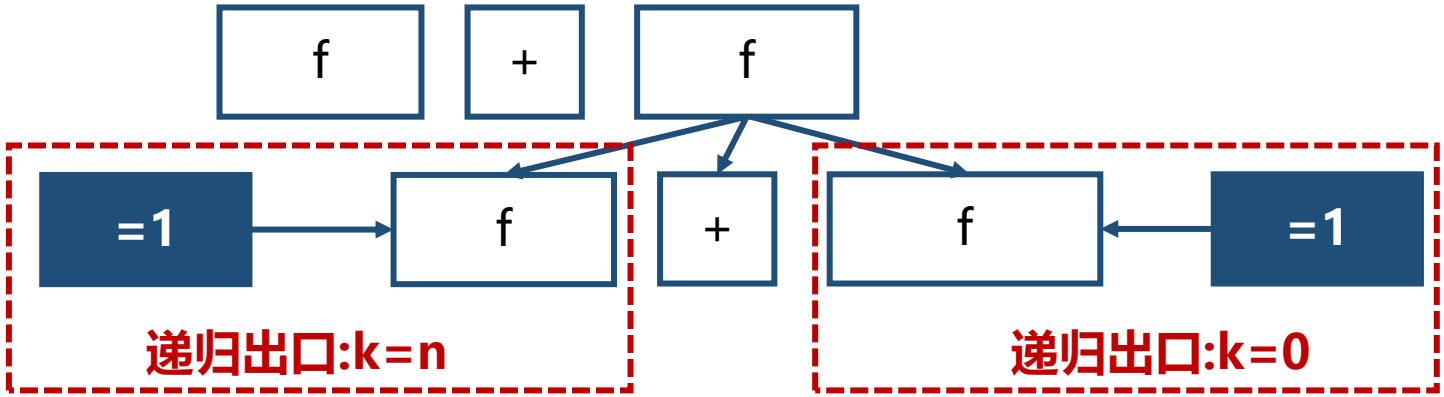
- 例3-9：用递归法计算从n个人中选择k个人组成一个委员会的不同组合数。
- 思想：
  - 由n个人里选k个人的组合数
    =由n-1个人里选k个人的组合数
    +由n-1个人里选k-1个人的组合数
  - 当n==k或k==0时，组合数为1

函数递归调用

例3-9



$$f(k,n)=\begin{cases}1 & k=0\\ 1 & k=n\\ f(k,n-1)+f(k-1,n-1)\end{cases}$$



```
#include<iostream>
using namespace std;
void main()
{   int n,k;
   int comm(int n, int k);
   cin>>n>>k;
   cout<<comm(n,k) <<endl;
}
int comm(int n, int k)
{   if (  k>n  )     return 0;
   else if(  n==k||k==0  )
     return 1;
   else
     return  comm(n-1,k)+comm(n-1,k-1)  ;
}
```

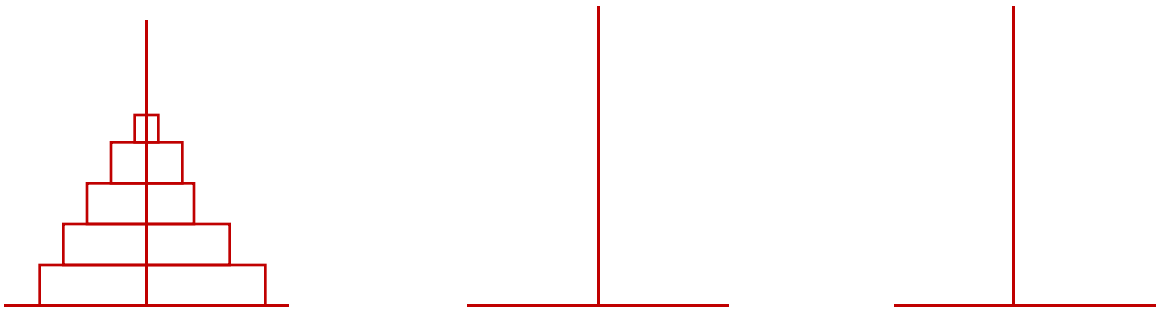
运行结果：

15 8

8568

函数递归调用

- 例3-10 汉诺塔：有三根针A、B、C。A针上有N个盘子，大的在下，小的在上，要求把这N个盘子从A针移到C针，在移动过程中可以借助B针，每次只允许移动一个盘，且在移动过程中在三根针上都保持大盘在下，小盘在上。



◘思想：

将n 个盘子从A针移到C针可以分解为下面三个步骤：

①将A 上n-1个盘子移到 B针上（借助C针）；

②把A针上剩下的一个盘子移到C针上;

③将n-1个盘子从B针移到C针上（借助A针）；

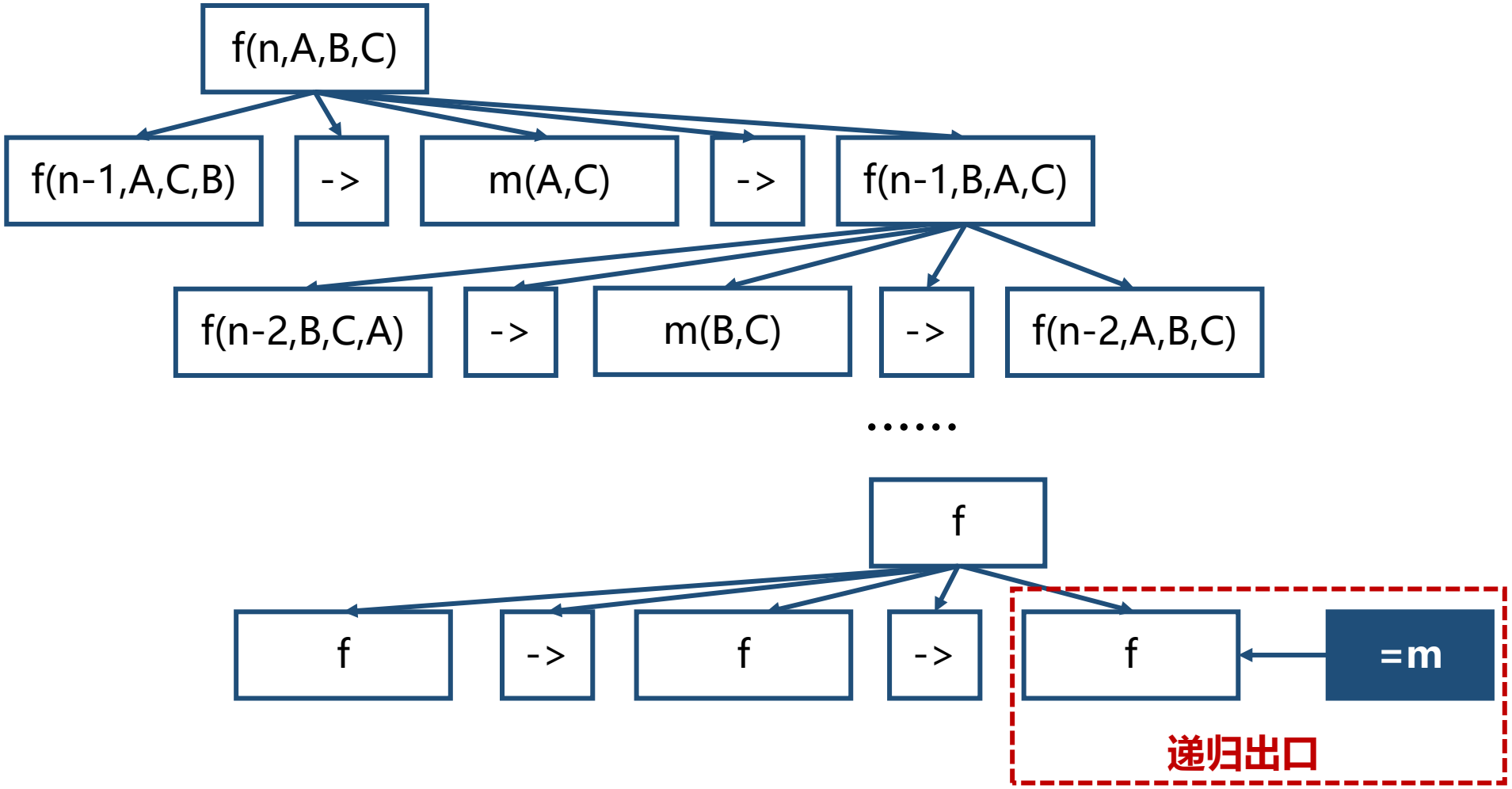
事实上，上面三个步骤包含两种操作：

①将多个盘子从一个针移到另一个针上，这是一个递归的过程。hanoi函数实现。

②将1个盘子从一个针上移到另一针上。
用move函数实现。

函数递归调用

例3-9



```
#include <iostream>
using namespace std;
void move(char getone,char putone)
{     cout<< getone <<"-->"<<putone<<endl; }
void hanoi(int n,char one,char two,char three)
{  void move(char getone,char putone);
   if (n==1) move (one,three);
   else
   {   hanoi (n-1,one,three,two);
       move(one,three);
       hanoi(n-1,two,one,three);
   }
}
```

```
void main()
{
    void hanoi(int n,char one,char two,char three);
    int m;
    cout<<"Enter the number of diskes:";
    cin>>m;
    cout<<"the steps to moving "<<m
    <<" diskes:"<<endl;
    hanoi(m,'A','B','C');
}
```

运行结果：

```
Enter the number of diskes:3
the steps to moving 3 diskes:
A-->C
A-->B
C-->B
A-->C
B-->A
B-->C
A-->C
```

函数的参数传递

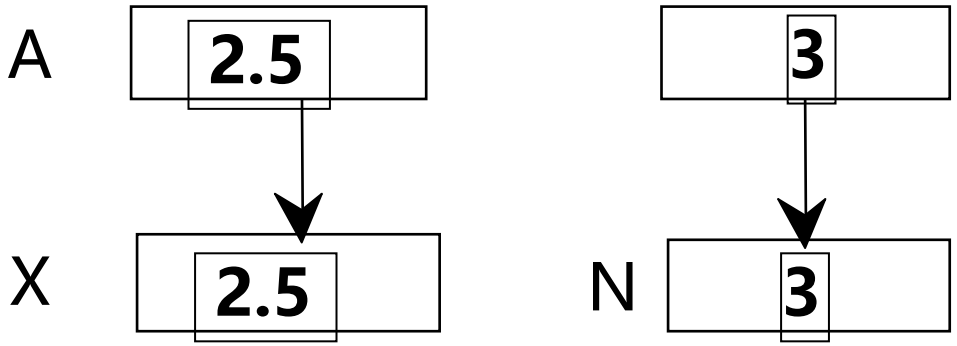
（1）值传递

- 定义：在函数被调用时才分配形参的存储单元，使用实参始使化（直接将实参的值传递给形参）。
- 原则：

- 实参可以是常量、变量或表达式。
- 实参类型必须与形参相符。
- 传递时是传递参数值，即单向传递。

函数的参数传递

主调函数： D = power(A,3)



被调函数：

double power(double X, int N)

```
void Swap(int a, int b)
{
  int t;
  t=a;
  a=b;
  b=t;
}
```

```
#include<iostream>
using namespace std;
//void Swap(int a, int b);
int main()
{
   int x(5), y(10);
   cout<<"x="<<x<<"    y="<<y<<endl;
   Swap(x,y);
   cout<<"x="<<x<<"    y="<<y<<endl;
   return 0;
}
```

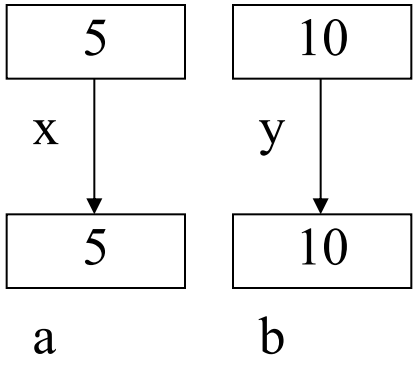
运行结果:

x=5 y=10

x=5 y=10

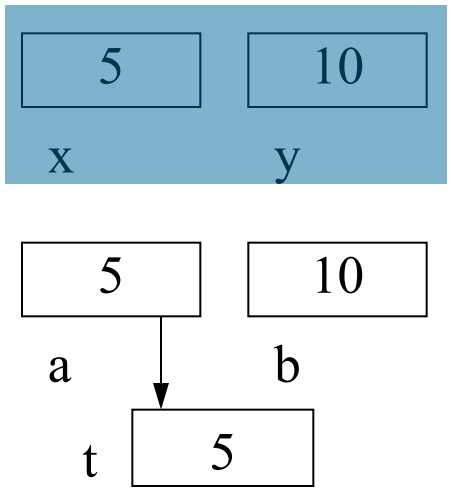
执行主函数中的函数调用

Swap(x,y);

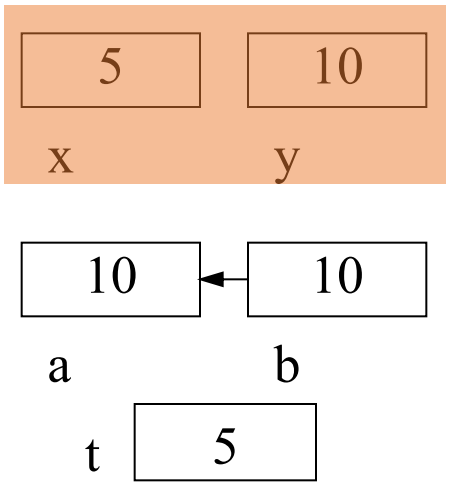


在Swap子函数中

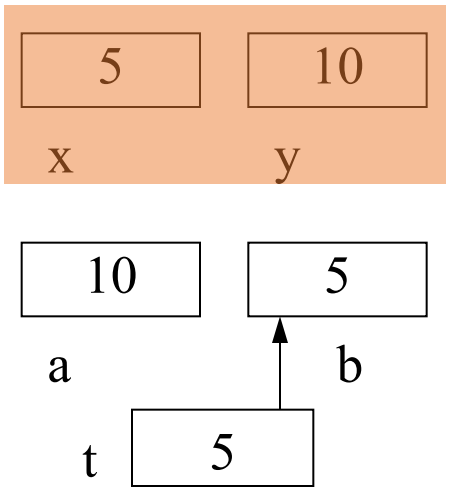
t=a;



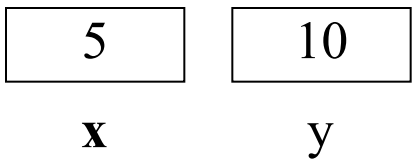
a=b;



b=t;



返回主函数以后



53

函数的参数传递

(2) 引用传递

- 引用的定义：一种特殊类型的变量，可以被认为是另一变量的别名。
- 引用的声明：变量类型 & 变量名；
- 例如

```
int i, j;
int & ri = i;
    //建立一个int型的引用ri,并将其
     //初始化为变量i的一个别名
j=10;
ri = j; //相当于 i=j=10;
```

函数的参数传递

(2) 引用传递

□引用的特点:

- 声明一个引用时，必须同时对它进行初始化，使它指向一个已存在的对象。
- 一旦一个引用被初始化后，就不能改为指向其它对象。
- 引用可以作为形参（引用传递）

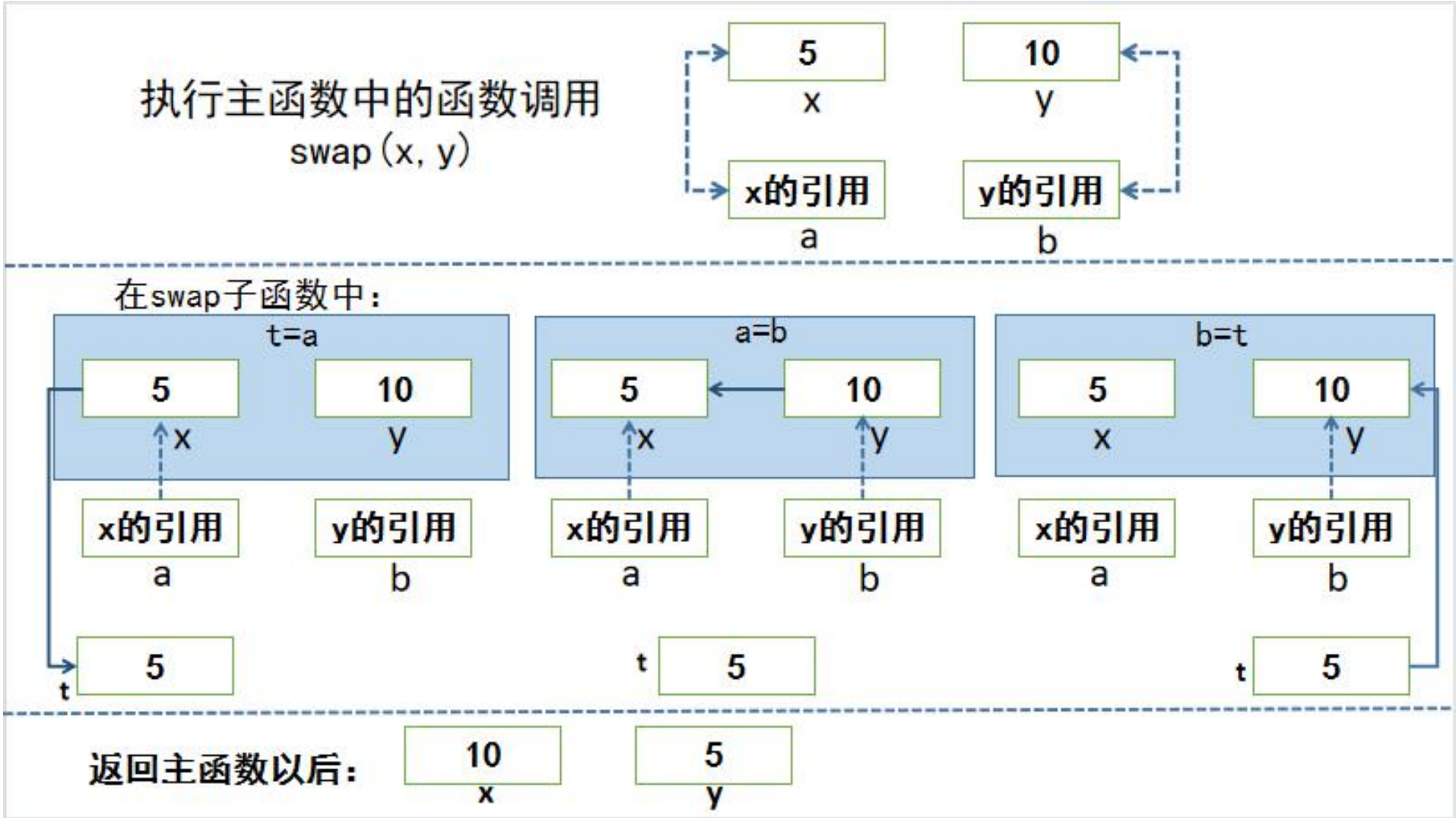
```
void Swap(int & a, int & b) {...}
```

```
#include<iostream>
using namespace std;
void Swap(int& a, int& b);
int main()
{  int x(5), y(10);
   cout<<"x="<<x<<"    y="<<y<<endl;
   Swap(x,y);
   cout<<"x="<<x<<"    y="<<y<<endl;
   return 0;
}
void Swap(int& a, int& b)
{  int t;
   t=a;
   a=b;
   b=t;
}
```

运行结果:

x=5　　y=10

x=10　　y=5


执行主函数中的函数调用
swap(x, y)
5
x
10
y
x的引用
a
y的引用
b
在swap子函数中：
t=a
5
x
10
y
x的引用
a
y的引用
b
5
t
a=b
5
x
10
y
x的引用
a
y的引用
b
t
5
b=t
5
x
10
y
x的引用
a
y的引用
b
t
5
返回主函数以后：
10
x
5
y

课程纲要

① **第三章**

函数的定义与使用

内联函数

带默认形式的函数

函数重载

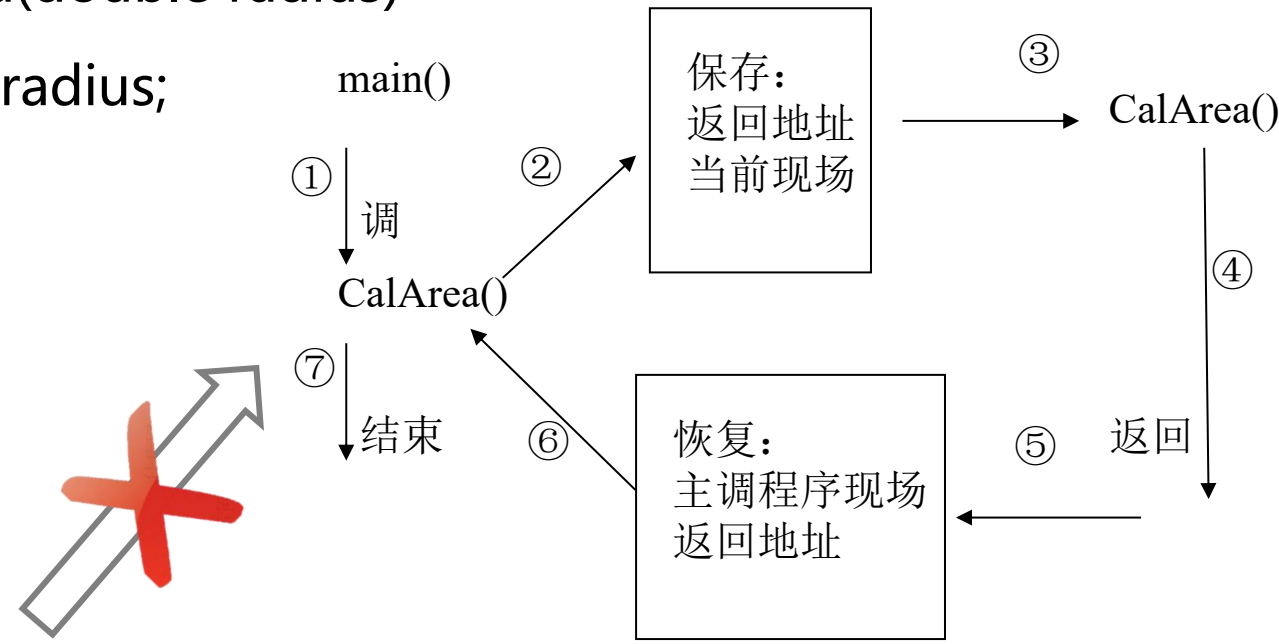
使用C++系统函数

深度探索

内联函数

- 声明时使用关键字 inline。
- 作用：内联函数被调用时不是发生控制转移，而是编译时在调用处用函数体进行替换，
- 好处：节省了参数传递、控制转移等开销。

```
#include<iostream>
using namespace std;
inline double CalArea(double radius)
{  return 3.14*radius*radius;
}
int main()
{
   double r(3.0);
   double area;
   area=CalArea(r);
   cout<<area<<endl;
   return 0;
}
```



area=3.14*r*r;

内联函数

注意：

- Inline关键字只是表示一种需求，编译器不承诺将inline修饰的函数作为内联函数，还要取决于函数体的复杂程度，例如直接递归函数无法作为内联函数处理。
- 即使不使用inline关键字，如果函数体足够简单，编译器也会直接把此函数当做是内联函数。

课程纲要

① **第三章**

函数的定义与使用

内联函数

带默认形式的函数

函数重载

使用C++系统函数

深度探索

带默认形参值的函数

- 函数在声明时可以预先给出默认的形参值，调用时如给出实参，则采用实参值，否则采用预先给出的默认形参值。
- 例如：

```
int add(int x=5,int y=6)
{    return  x+y;
}
```

```
void main(void)
{    add(10,20);
     add(10);
     add();
}
```

带默认形参值的函数

注意1：

- 默认形参值必须从右向左顺序声明，并且在默认形参值的右面不能有非默认形参值的参数。因为调用时实参取代形参是从左向右的顺序。
- 例：

```
int add(int x, int y=5,int z=6); //正确
int add(int x=1,int y=5, int z); //错误
int add(int x=1,int y, int z=6); //错误
```

带默认形参值的函数

注意2：

- 在相同的作用域内，不允许同一个函数的多个声明中对同一个参数重复设置默认值，默认值相同也不行。

```
int add(int x=5, int y=6);
void main(void)
{   add(); //调用在实现前
}
int add(int x, int y)
{   return x+y;   }
```

```
int add(int x=5, int y=6)
{   return  x+y;   }
void main(void)
{   add();   //调用在实现后
}
```

例3-15：

```
#include <iostream>
#include <iomanip>
using namespace std
int getVolume(int length, int width=2,int height=3);
int main(){
    const int X=10,Y=12,Z=15;
    cout<<”some box data is”;
    cout<<getVolume(X,Y,Z)<<endl;
    cout<<”some box data is”;
    cout<<getVolume(X,Y)<<endl;
    cout<<”some box data is”;
    cout<<getVolume(X)<<endl;
    return 0;
}
```

```
int getVolume(int length,int width/*=2 * /,int height/*=3 * /){
    cout<<setw(5)<<length<<setw(5)<<width<<setw(5)<<height<<'\t';
    return length*width*height;
}
```

运行结果：

```
some box data is  10  12  15 1800
some box data is  10  12  3   360
some box data is  10  2   3    60
```

课程纲要

① **第三章**

函数的定义与使用

内联函数

带默认形式的函数

函数重载

使用C++系统函数

深度探索

函数重载

- 两个以上函数具有**相同的函数名**，但是**形参个数或者类型不同**，编译器根据实参和形参的类型和个数最佳匹配，自动确定调用哪个函数。

- 例：

```
int add(int x, int y);
float add(float x, float y);
```

形参类型不同

```
int add(int x, int y);
int add(int x, int y, int z);
```

形参个数不同

函数重载

- 注意1：

不同形参名称，不同函数返回类型的同名函数，不是函数重载，编译器报错。

```
int add(int x,int y);
int add(int a,int b);
```

编译器不以形参名来区分

```
int add(int x,int y);
void add(int x,int y);
```

编译器不以返回值来区分

函数重载

- 注意2：

不要将不同功能的函数声明为重载函数，以免出现调用结果的误解、混淆。极其不推荐：

```
int add(int x,int y)
{   return x+y;   }
```

```
float add(float x,float y)
{   return x-y;   }
```

函数重载

- 注意3:

当使用具有默认形参值的函数重载时，需要防止二义性。

例如：

```
void fun(int length,int width=2,int height=3);
void fun(int length);
```

当以下面形式调用函数fun时，编译器无法确定应该调用哪一个函数

```
fun(1);
```

这样编译器就会指出语法错误。

□例3-16：

```
#include<iostream>
using namespace std;
int sumOfSquare(int a,int b){
    return a*a+b*b;
}
double sumOfSquare(double a,double b){
    return a*a+b*b;
}
int main(){
    int m,n;
    cout<<"Enter two integers:";
    cin>>m>>n;
    cout<<"Their sum of square:"<<sumOfSquare(m,n)<<endl;
```

```
double x,y;
    cout<<”Enter two real numbers:”;
    cin>>x>>y;
    cout<<”Their sum of square:”<<sumOfSquare(x,y)<<endl;
    return 0;
}
```

运行结果：

```
Enter two integers:3 5
Their sum of  square:34
Enter two real numbers:2.3 5.8
Their sum of  squaare :38.93
```

课程纲要

① **第三章**

函数的定义与使用

内联函数

带默认形式的函数

函数重载

使用C++系统函数

深度探索

C++语言的系统函数

- C++的系统库中提供了几百个函数可供程序员使用。
  例如：
  求平方根函数sprt( )、求绝对值函数abs( )等。

- 使用系统函数时要包含相应的头文件。
  例如：math.h (不推荐使用) 或 cmath

C++语言的系统函数

- 例3-17

  从键盘输入一个角度值，求出该角度的正弦值、余弦值和正切值。

- 思想：

  系统函数中提供了求正弦值、余弦值和正切值的函数：sin()、cos()、tan()，函数的说明在头文件cmath中。

  提示：角度值要先转换为弧度值。

$$y = x * \pi / 180$$

```
#include<iostream>
#include<cmath>
using namespace std;
const double pi(3.14159265);
void main()
{   double a,b;
    cin>>a;
    b=a*pi/180;
    cout<<"sin("<<a<<")="<<sin(b)<<endl;
    cout<<"cos("<<a<<")="<<cos(b)<<endl;
    cout<<"tan("<<a<<")="<<tan(b)<<endl;
}
```

运行结果：

30

sin(30)=0.5

cos(30)=0.866025

tan(30)=0.57735

C++语言的系统函数

- 一个网站http://www.cppreference.com

C++ 参考手册

C++98, C++03, C++11, C++14, C++17, C++20, C++23

编译器支持 (11, 14, 17, 20)
自立实现

语言
基本概念
关键词
预处理器
表达式
声明
初始化
函数
语句
类
重载
模板
异常

头文件

具名要求

功能特性测试宏 (C++20)

语言支持库
类型支持 - 特性 (C++11)
程序工具
关系运算符 (C++20)
numeric_limits - type_info
initializer_list (C++11)

概念库 (C++20)

诊断库

通用工具库
智能指针与分配器
unique_ptr (C++11) - shared_ptr (C++11)
日期和时间
函数对象 - hash (C++11)
字符串转换 (C++17)
工具函数
pair - tuple (C++11)
optional (C++17) - any (C++17)
variant (C++17) - format (C++20)

字符串库
basic_string
basic_string_view (C++17)
空终止字符串:
字节 - 多字节 - 宽

容器库
array (C++11) - vector
map - unordered_map (C++11)
priority_queue - span (C++20)
其他容器:
顺序 - 关联
无序关联 - 适配器

迭代器库

范围库 (C++20)

算法库

数值库
常用数学函数
数学特殊函数 (C++17)
数值算法
伪随机数生成
浮点环境 (C++11)
complex - valarray

输入/输出库
基于流的 I/O
同步输出 (C++20)
I/O 操纵符

文件系统库 (C++17)

本地化库

正则表达式库 (C++11)
basic_regex - 算法

原子操作库 (C++11)
atomic - atomic_flag
atomic_ref (C++20)

线程支持库 (C++11)
thread - mutex - condition_variable

技术规范

标准库扩展 (库基础 TS)
resource_adaptor — invocation_type

标准库扩展 v2 (库基础 TS v2)
propagate_const — ostream_joiner — randint
observer_ptr — 检测手法

标准库扩展 v3 (库基础 TS v3)
scope_exit — scope_fail — scope_success — unique_resource

并发库扩展 (并发 TS) — **事务性内存** (TM TS)

课程纲要

① **第三章**

函数的定义与使用

内联函数

带默认形式的函数

函数重载

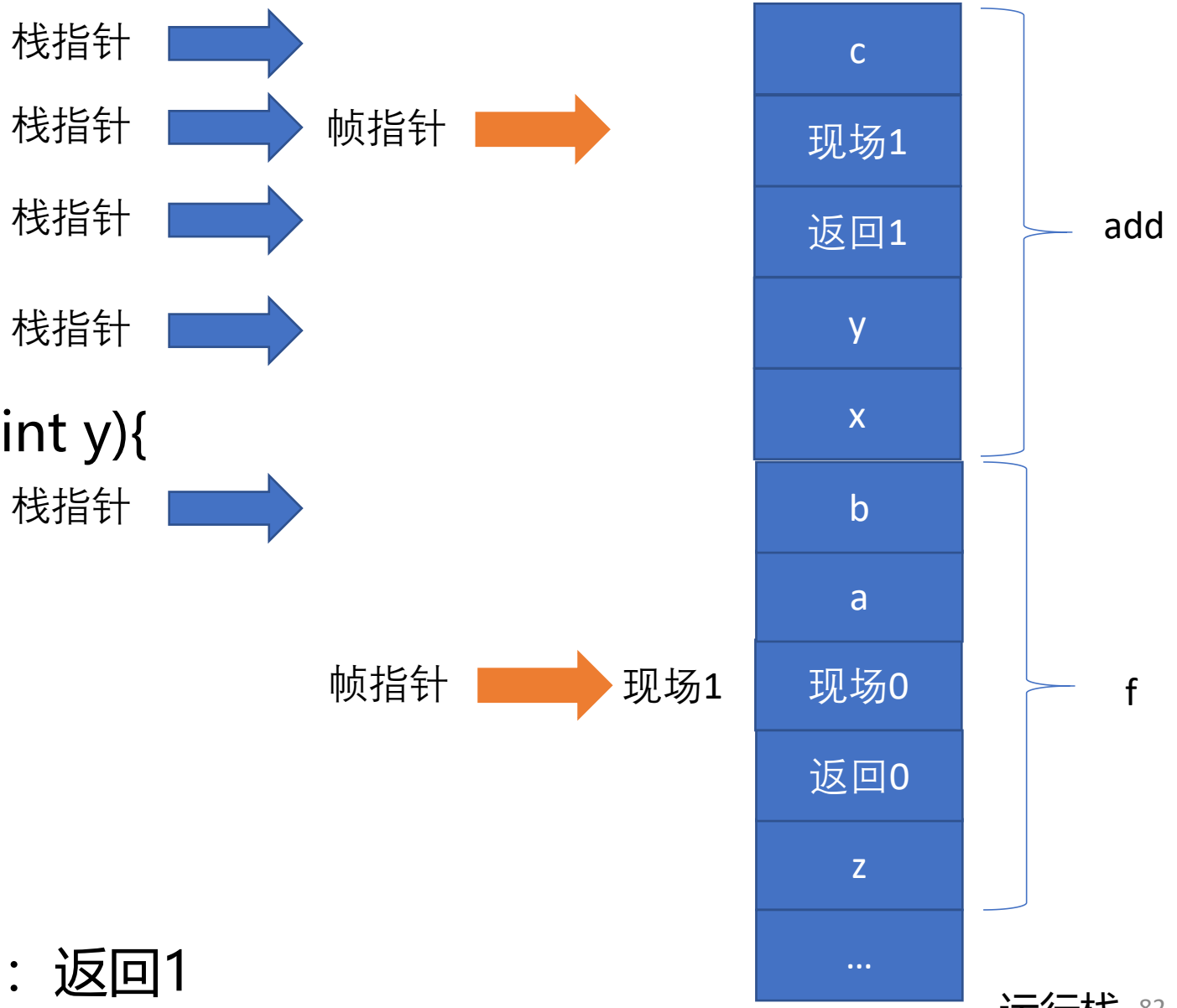
使用C++系统函数

深度探索

深度探索

- 运行栈：用于存放局部变量等函数调用信息的栈，先进后出。
- 栈指针：记录栈顶位置的。
- 帧指针：主调函数和被调函数之间的锚，当函数运行时，用于访问各个局部变量。当函数调用时，用于保存现场。

3.6 深度探索




```
int add(int x, int y){
      int c;
      x+ y;
}
void f(int z){
   int a;
  int b;
  add(a,b);
  XXXX //地址：返回1
}
```

课后作业

- 3-6,11,10,13,15
- 完成时间：下节课前