第五章　函数与编译预处理

概述

函数是程序代码的一个自包含单元，用于完成某一特定的任务。

C++是由函数构成的，函数是C++的基本模块。

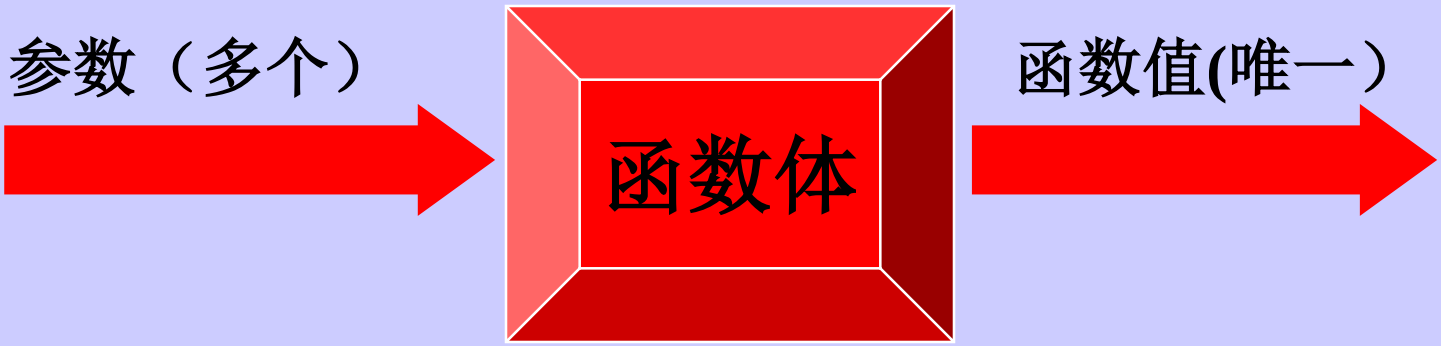
有的函数完成某一操作；有的函数计算出一个值。通常，一个函数即能完成某一特定操作，又能计算数值。

为什么要使用函数？

1、避免重复的编程。

2、使程序更加模块化，便于阅读、修改。

所编写的函数应尽量少与主调函数发生联系，这样便于移植。



说明：

1、一个源程序文件由一个或多个函数组成，编译程序以文件而不是以函数为单位进行编译的。

2、一个程序可以由多个源文件组成，可以分别编译，统一执行。

3、一个程序必须有且只有一个main( )函数，C++从main( )函数开始执行。

4、C++语言中，所有函数都是平行独立的，无主次、相互包含之分。函数可以嵌套调用，不可嵌套定义。

5、从使用角度来说，分标准函数和用户自定义函数；从形式来说，分无参函数和有参函数。

库函数是C++编译系统已预定义的函数，用户根据需要可以直接使用这类函数。库函数也称为标准函数。

为了方便用户进行程序设计，C++把一些常用数学计算函数（如sqrt()、exp()等）、字符串处理函数、标准输入输出函数等，都作为库函数提供给用户，用户可以直接使用系统提供的库函数。

库函数有很多个，当用户使用任一库函数时，在程序中必须包含相应的头文件。 如#include<iostream.h>等。

用户在设计程序时，可以将完成某一相对独立功能的程序定义为一个函数。用户在程序中，根据应用的需要，由用户自己定义函数，这类函数称为用户自定义的函数。

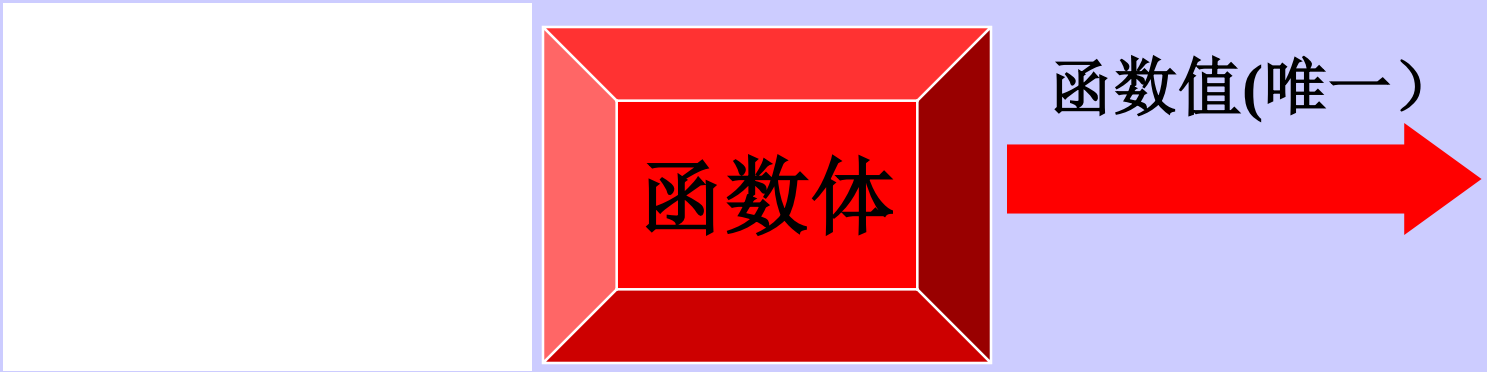
根据定义函数或调用时是否要给出参数，又可将函数分为：无参函数和有参函数。

函数定义的一般形式

一、无参函数

主调函数并不将数据传给被调函数。

无参函数主要用于完成某一操作。



类型说明　　函数名（void）

{　函数体　}

不传递参数

```
void main(void )
{   printstar ( );
    print_message ( );
    printstar( );
}
 void     printstar (void )
 {    cout<<"* * * * * * * * * * * *\n";
 }
void     print_message (void)
{    cout<<"  How do you do! \n";
}
```

调用函数

调用函数

调用函数

函数类型

函数名

函数体

两个被调函数主要用于完成打印操作。

输出：**********

How do you do!

****

二、有参函数

主调函数和被调函数之间有数据传递。主调函数可以将参数传递给被调函数，被调函数中的结果也可以带回主调函数。

```
类型说明    函数名（形式参数列表说明）
        {   函数体   }
```

函数类型

形参列表说明

```
int  max (int x,int y)
{   int  z;
    z=(x>y)? x : y ;
    return  z;
}
void main (void )
{   int   a,b,c;
    cin>>a>>b;
    c=max (a , b) ;
    cout<<“The max is”<< c<<endl;
}
```

函数名

函数体

函数值

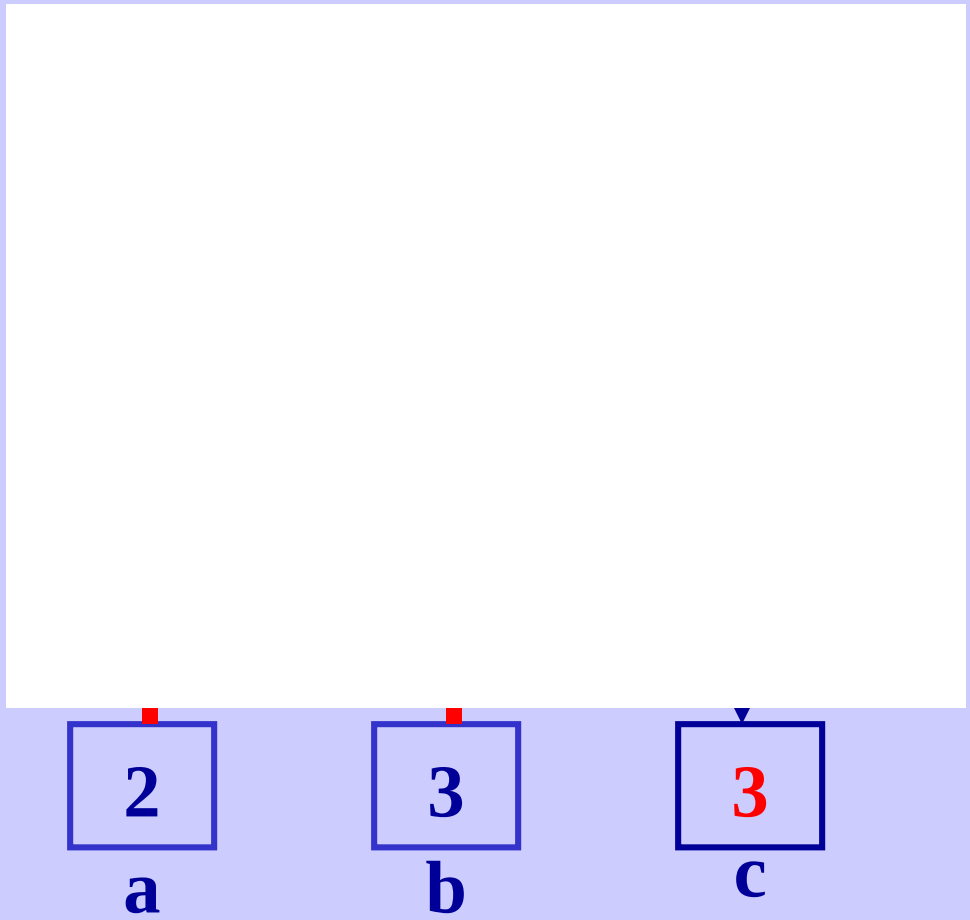
主调函数

调用函数

实际参数

将实际值a,b传给被调函数的参数x,y，计算后得到函数值z返回

```
int  max (int x,int y)
{   int  z;
    z=(x>y)? x : y ;
    return  z;
}
 void main (void )
 {    int   a,b,c;
      cin>>a>>b;
      c=max (a , b) ;
      cout<<“The max is”<< c<<endl;
 }
```



函数参数和函数的值

形参是被调函数中的变量；实参是主调函数赋给被调函数的特定值。实参可以是常量、变量或复杂的表达式，不管是哪种情况，在调用时实参必须是一个确定的值。

形参与实参类型相同，一一对应。

形参必须要定义类型，因为在定义被调函数时，不知道具体要操作什么数，而定义的是要操作什么类型的数。

```
int  max (int x,int y)
{   int  z;
    z=(x>y)? x : y ;
    return  z;
}
void main (void )
{    int   a,b,c;
     cin>>a>>b;
     c=max (a+b , a*b) ;
     cout<<“The max is”<<c<<endl;
}
```

若a为3，b为5，则实参为8, 15，分别送给形参x，y。

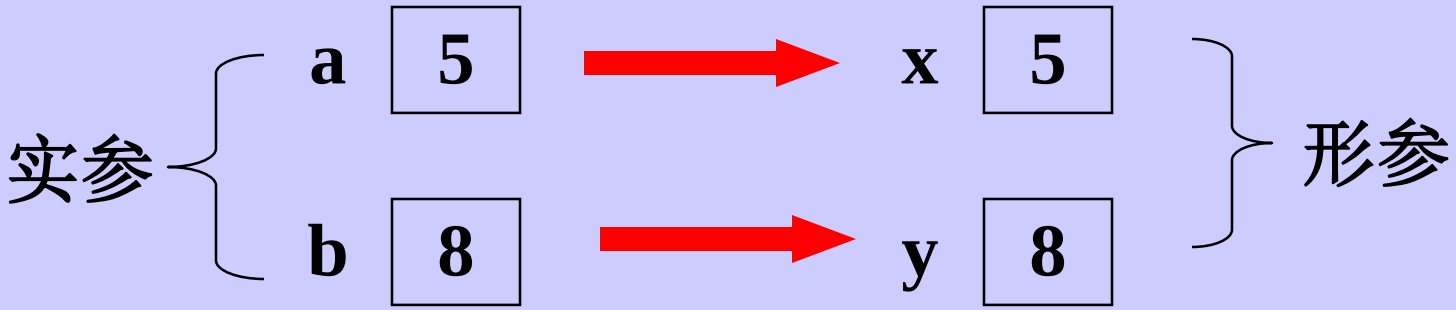
先计算，后赋值

说明：

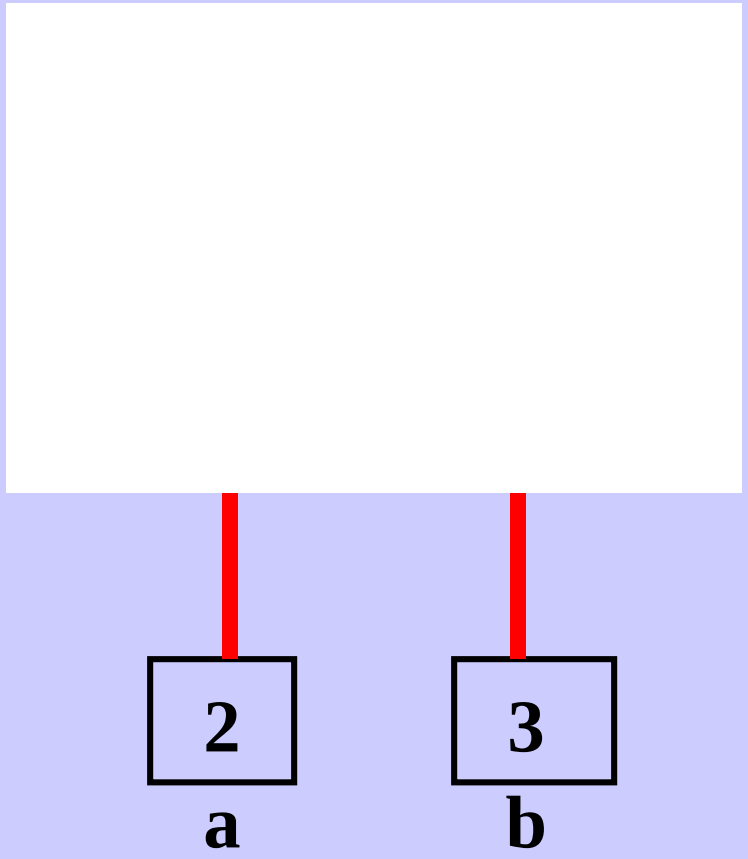
1、在未出现函数调用时，形参并不占内存的存储单元，只有在函数开始调用时，形参才被分配内存单元。调用结束后，形参所占用的内存单元被释放。

2、实参对形参变量的传递是“值传递”，即单向传递。在内存中实参、形参分占不同的单元。

3、形参只作用于被调函数，可以在别的函数中使用相同的变量名。

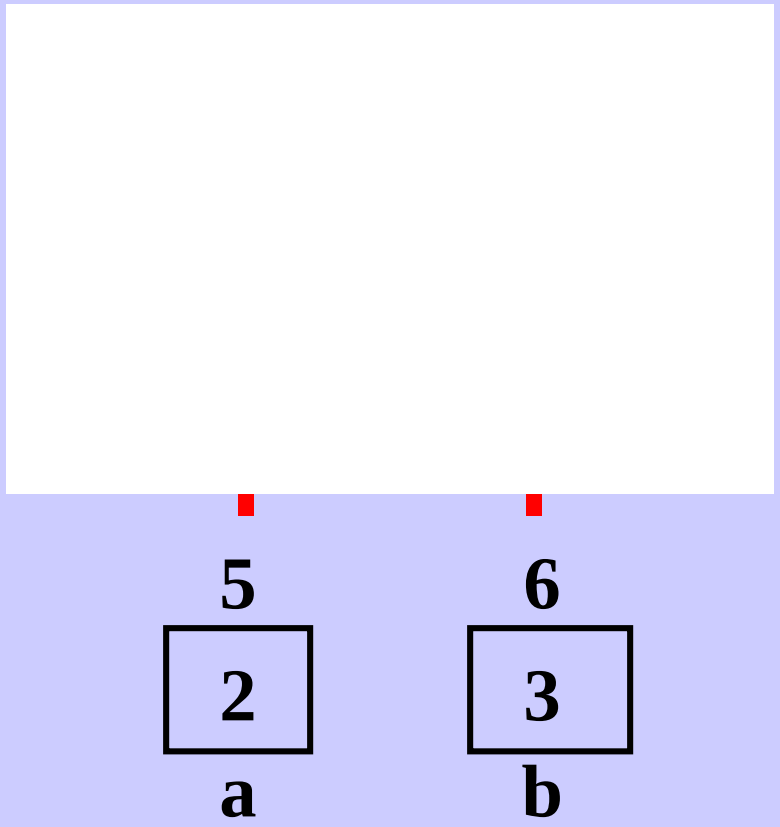


```
void fun(int a, int b)
{    a=a*10;
     b=b+a;
     cout<<a<<'\t'<<b<<endl;
}
void main(void)
{   int a=2, b=3;
    fun(a,b);
    cout<<a<<'\t'<<b<<endl;
}
```




```
20      23
2       3
```

```
void fun(int x, int y)
{    x=x*10;
     y=y+x;
     cout<<x<<'\t'<<y<<endl;
}
void main(void)
{   int a=2, b=3;
    fun(a+b,a*b);
    cout<<a<<'\t'<<b<<endl;
}
```




```
50          56
2           3
```

形参实参类型相等，一一对应

形参必须要定义类型，因为在定义被调函数时，不知道具体要操作什么数，而定义的是要操作什么类型的数。

形参是被调函数中的变量；实参是主调函数赋给被调函数的特定值。在函数调用语句中，实参不必定义数据类型，因为实参传递的是一个具体的值(常量)，程序可依据这个数值的表面形式来判断其类型，并将其赋值到对应的形参变量中。

函数的返回值

函数的返回值通过**return**语句获得。函数只能有唯一的返回值。

函数返回值的类型就是函数的类型。

return语句可以是一个表达式，函数先计算表达式后再返回值。

return语句还可以终止函数，并将控制返回到主调函数。

一个函数中可以有一个以上的**return**语句，执行到哪一个**return**语句，哪一个语句起作用。

```
int add ( int  a, int b)
{
    return (a+b);
}
int max ( int a, int b)
{
    if (x>y)
        return   x ;
    else  return  y;
}
```

先计算，后返回

若函数体内没有return语句，就一直执行到函数体的末尾，然后返回到主调函数的调用处。

可以有多个return语句

既然函数有返回值，这个值当然应属于某一个确定的类型，应当在定义函数时指定函数值的类型。

```
int   max (float a,  float  b) // 函数值为整型
```

函数返回值的类型，也是函数的类型

不带返回值的函数可说明为void型。

函数的类型与函数参数的类型没有关系。

```
double   blink ( int a,  int b)
```

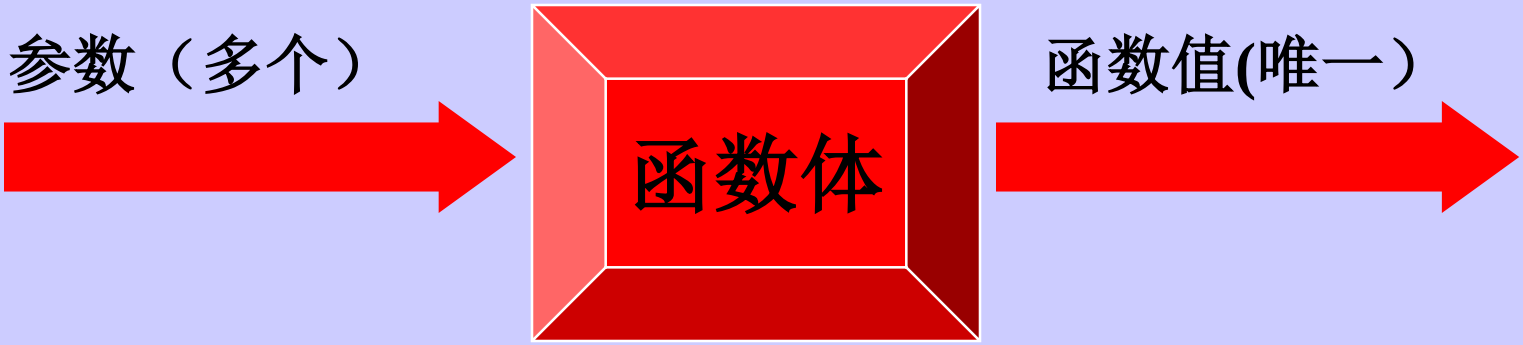
如果函数的类型和return表达式中的类型不一致，则以函数的类型为准。函数的类型决定返回值的类型。对数值型数据，可以自动进行类型转换。

如果有函数返回值, 函数的类型就是返回值的类型

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

如果有函数返回值,返回值就是函数值,必须惟一。

函数体的类型、形式参数的类型必须在函数的定义中体现出来。



函数的调用

函数调用的一般形式

函数名（实参列表）；

i=2; f (i, ++i);

实际调用： f (3, 3);

形参与实参类型相同，一一对应。

函数调用的方式

作为语句 **printstar( );**

作为表达式 **c=max (a,b);**

作为另一个函数的参数 **cout<<max (a,b);**

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) 被调用的函数必须是已存在的函数

2) 如果使用库函数，必须用 #include < math.h>

3) 函数调用遵循先定义、后调用的原则，即被调函数应出现在主调函数之前。

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

void main (void)
{  float   a,b, c;
   cin>>a>>b;
   c=max (a+b , a*b) ;
   cout<<“The max is”<<c<<endl;
}
```

形参必须说明参数类型

被调函数先定义

定义之后再调用

实参传递的是一个具体的值，不必说明参数类型

4) 如果使用用户自己定义的函数，而该函数与调用它的函数（即主调函数）在同一个程序单位中且位置在主调函数之后，则必须在调用此函数之前对被调用的函数作声明。

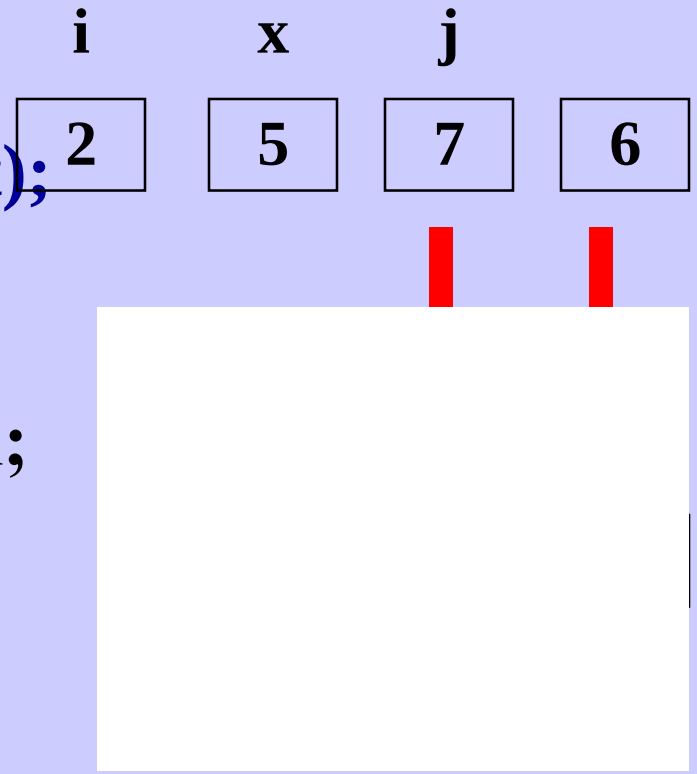
```
void main (void)
{    float   a,b, c;
     float   max (float,float);
     cin>>a>>b;
     c=max (a,b) ;
     cout<<"The max is"<<c<<endl;
}
float  max (float x, float y)
{   float  z;
     z=(x>y)? x : y ;
     return  z;
}
```

函数原型说明

函数定义

定义是一个完整的函数单位，而原型说明仅仅是说明函数的返回值及形参的类型。

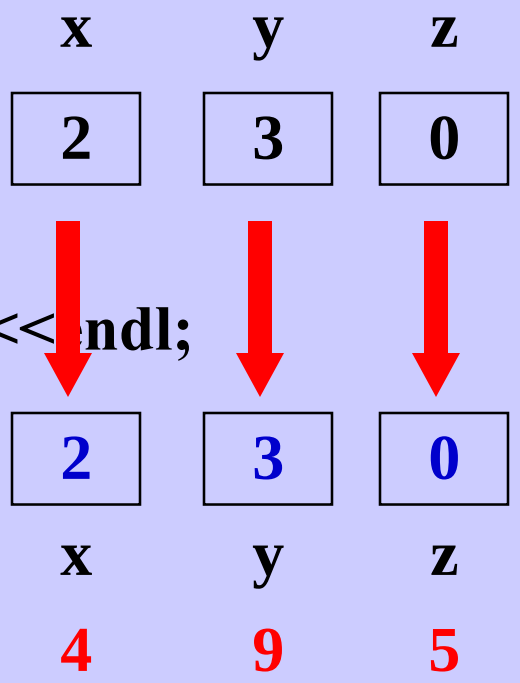
```
void main(void)
{  int  i=2, x=5, j=7;  void fun(int,int);
    fun ( j, 6);
  cout<<i<<‘\t’<< j<<‘\t’<< x<<endl;
}
void fun ( int i, int j)
{  int  x=7;
    cout<<i<<‘\t’<< j<<‘\t’<<x<<endl;
}
```



输出：　7　　6　　7

　　2　　7　　5

```
void main(void )
{   int  x=2,y=3, z=0;void add(int,int,int);
    cout<<“(1) x=“<<x<<“ y=“<<y<<“ z=“<<z<<endl;
    add (x, y, z);
    cout<< (“(3) x=“<<x<<“ y=“<<y<<“ z=“<<z<<endl;
}
void add ( int x, int y, int z)
{   z=x+y;  x=x*x;  y=y*y;
   cout<<(“(2) x=“<<x<<“ y=“<<y<<“ z=“<<z<<endl;
}
```



(1) x=2 y=3 z=0

(2) x=4 y=9 z=5

(3) x=2 y=3 z=0

编写程序，分别计算下列函数的值(x从键盘输入)

$$f_1(x)=3x^3+2x^2-1$$

$$f_2(x)=x^2+2x+1$$

```
 float  f1(float x)
{   float y;
     y=3*x*x*x+2*x*x-1;
     return y;
}
void main(void)
{
    float x, y;
    cin>>x;
    y=f1(x);
    cout<<“x=“<<x<<“ , y=“<<y<<endl;
}
```

编写程序，分别计算下列函数的值(x从键盘输入)

$$s=1+\frac{1}{x}+\frac{1}{x^2}+\frac{1}{x^3}+\cdots\left(x>1\right)$$

当最后一项小于0.00001时，累加结束。

```
float fun(float x)
{  float s=1, t=1;
    do
    { t=t/x;
       s+=t;
     }while (t>0.00001);
   return s;
}
```

```
void main(void)
{  float x;
     cin>>x;
     cout<<“s=“<<fun(x)<<endl;
}
```

计算100～200之间的素数，用函数prime( )判断一个数是否是素数，若是该函数返回1，否则返回0。

```
int prime(int x)
{       for(int i=2;i<x/2;i++)
                if(x%i==0)
                        return 0;
   return 1;
}
                void main(void)
                {  for(int i=100;i<=200; i++)
                     if(prime(i)==1)
                        cout<<i<<'\t';
                 }
```

计算输入两个数的最大公约数

```
int gys(int a, int b)
{    int r;
     if(a<b){r=a; a=b; b=r;}
     while(r=a%b)
      {  a=b;  b=r;}
     return b;
}
```

```
void main(void)
{  int x, y;
    cin>>x>>y;
   cout<<gys(x,y)<<endl;
}
```

计算输入三个数的最大公约数

```
int gys(int a, int b,int c)
{    int  r;
     if(a<b){r=a; a=b; b=r;}
     r=r>c?r:c;
     for(int i=r-1;i>=1;i--)
      { if(a%i==0&&b%i==0&&c%i==0)
                 break;
       }
      return i;
}
```

```
void main(void)
{  int x, y, z;
    cin>>x>>y>>z;
   cout<<gys(x,y,z)<<endl;
}
```

写一个函数验证哥德巴赫猜想：一个不小于6的偶数可以表示为两个素数之和，如6=3+3，8=3+5，10=3+7……。在主函数中输入一个不小于6的偶数n，函数中输出以下形式的结果：

34=3+31

函数的嵌套调用

C语言中，所有函数都是平行独立的，无主次、相互包含之分。函数可以嵌套调用，不可嵌套定义。

```
int max ( int a,  int b)
{    int  c;
     int min (  int  a,  int b)
      {  return  ( a<b? a: b);
      }
     c=min(a,b);
    return ( a>b? a : b);
}
```

嵌套定义

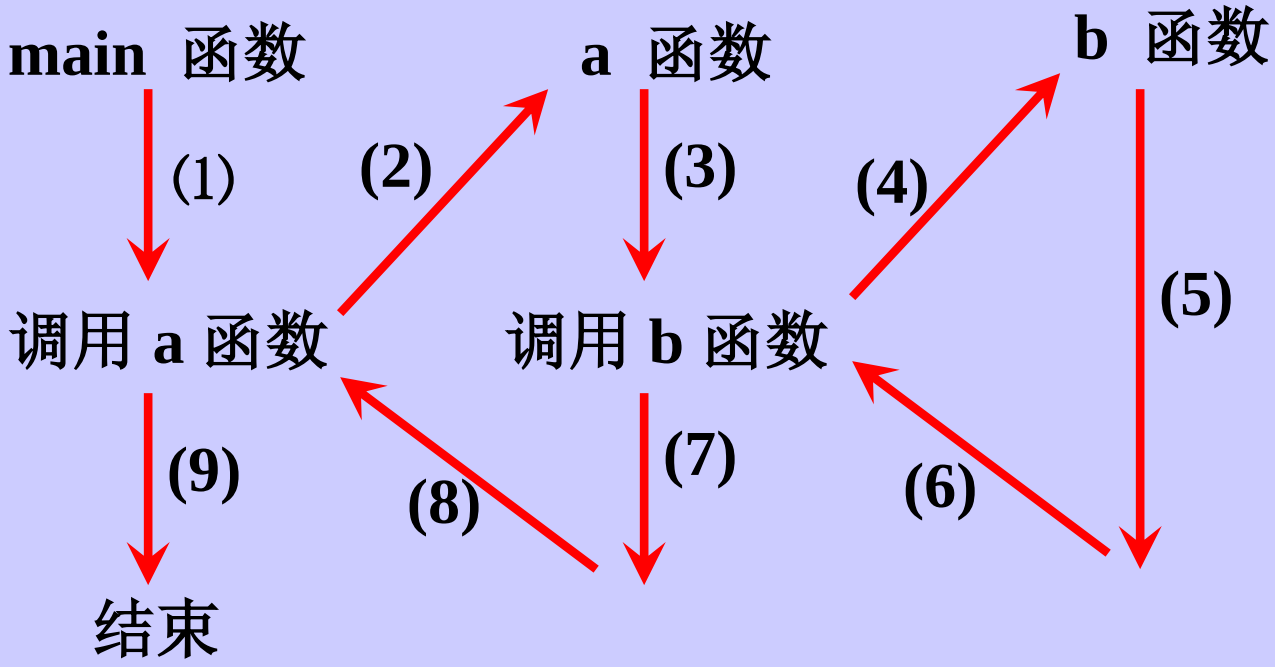
```
int min (  int  a,  int b)
{    return  ( a<b? a: b);
}
```

平行定义

```
int max ( int a,  int b)
{    int  c;
       c=min(a,b);
     return ( a>b? a : b);
}
```

嵌套调用

在main函数中调用a函数，在a函数中又调用b函数。



$$f(k,n)=1^k+2^k+3^k+....+n^k$$

```
int power(int m,int n)          //m^n
{           int i,product=m;
            for(i=1;i<n;i++)
                        product=product*m;
            return product;
}
int sum_of_power(int k,int n)          //n^k的累加和
{           int i,sum=0;
            for(i=1;i<=n;i++)
                        sum+=power(i,k);
            return sum;
}
 void main(void)
 {          int k,m;
            cin>>k>>m;
            cout<<"f("<<k<<","<<m<<")="<<sum_of_power(k,m)<<endl;
            //m^k的累加和
 }
```

平行定义

平行定义

嵌套调用

嵌套调用

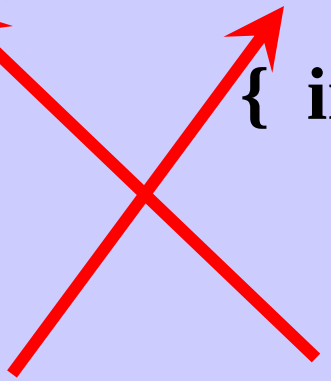
函数的递归调用

在调用一个函数的过程中直接或间接地调用函数本身，称为函数的递归调用。

```
int  f(int x)
{  int  y,z ;
    .....
    z=f(y);
    ....
   return  (2*z);
}
```

```
int  f1(int x)
{  int  y,z ;
    .....
    z=f2(y);
    ....
   return  (2*z);
}
```

```
int  f2(int t)
{  int  a, c ;
    .....
    c=f1(a);
    ....
   return  (3+c);
}
```



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个人多少岁，他说比第3个人大2岁。问第3个人多少岁，他说比第2个人大2岁。问第2个人多少岁，他说比第1个人大2岁。问第1个人多少岁，他说是10岁。请问第5个人多大？

age(5)=age(4)+2

age(4)=age(3)+2

age(3)=age(2)+2

age(2)=age(1)+2

age(1)=10

$$age(n)=\begin{cases}10 & n=1\\ age(n-1)+2 & n>1\end{cases}$$

必须有递归结束条件

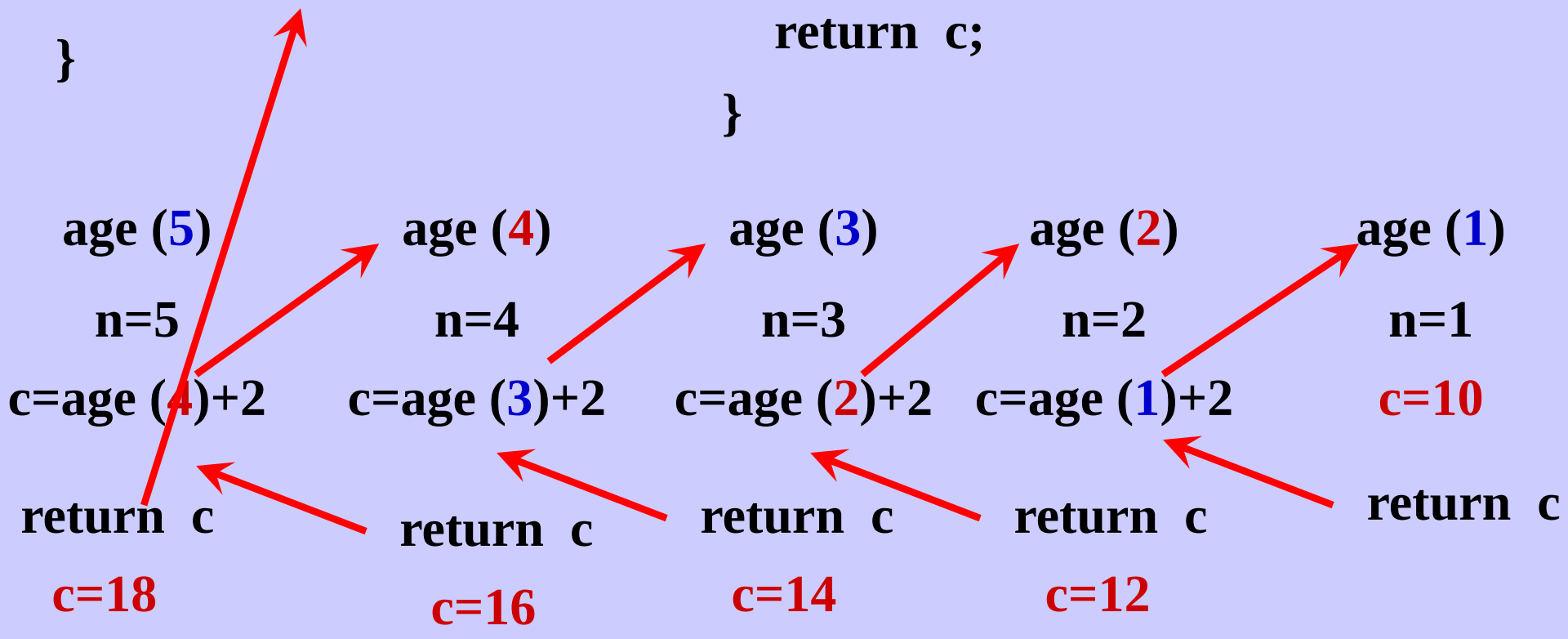
```
void main(void)
{    int age(int);
     cout<<age(5)<<endl;
}
```

```
int  age ( int  n )
{  int  c;
   c=age(n-1)+2;
   return  c;
}
```

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

```
void main(void)
{   int age(int);
    cout<<age(5)<<endl;
}
```

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



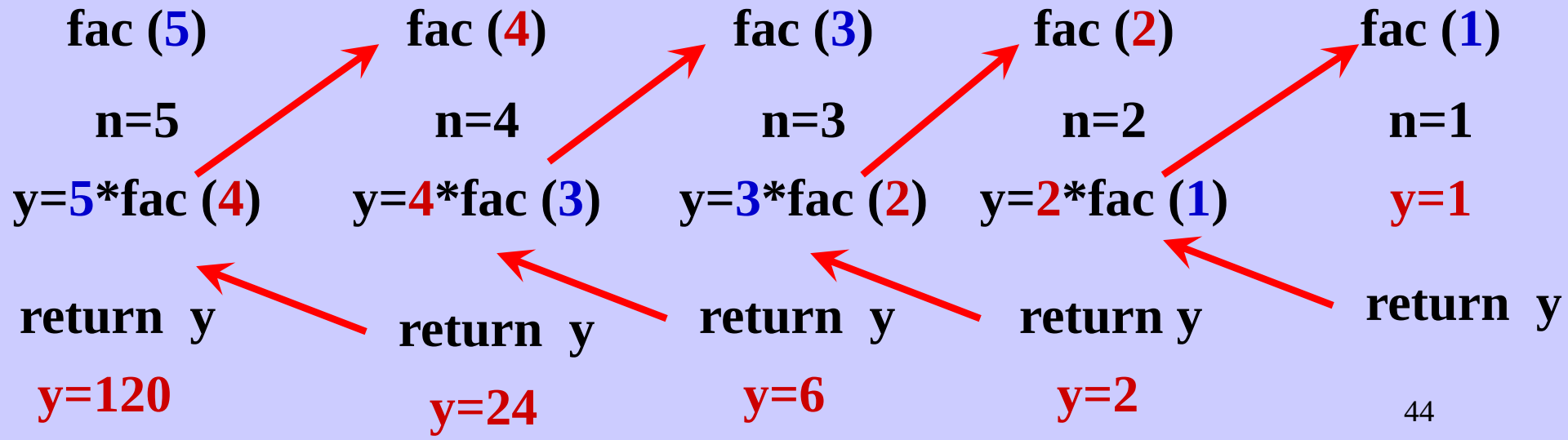
虽然算法一致，但n不同，c不同，在内存中每一层函数变量所在的内存单元均不相同。必须有递归终止条件。

用递归方法求n!

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

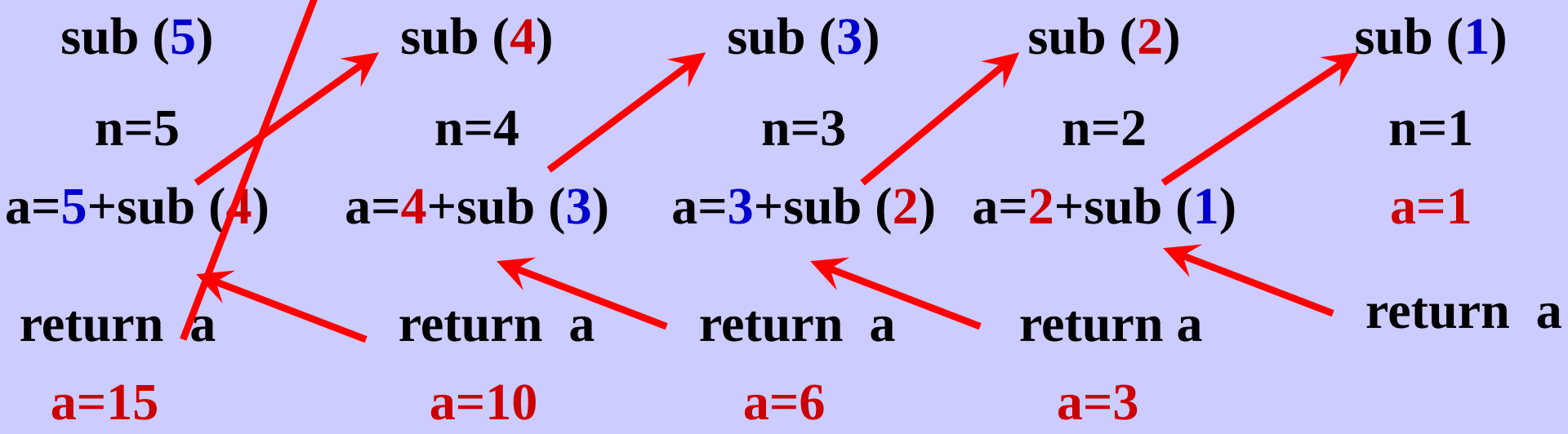
```
float  fac (int n)
{  float  y;
   if ((n= =0)|| (n= =1)   y=1;
   else  y=n*fac(n-1);
   return  y;
}
```

```
void main (void)
{  float  y;   int  n;
    cout<<“Input  n:\n”;
    cin>>n ;
    cout<<n<<“!=”<<fac(n)<<endl;
}
```



```
int sub(int);
void main (void)
{   int  i=5;
    cout<<sub(i)<<endl;
}
```

```
int sub (int  n )
{   int  a;
    if (n= =1)   return  1;
    a=n+sub(n-1);
    return   a;
}
```



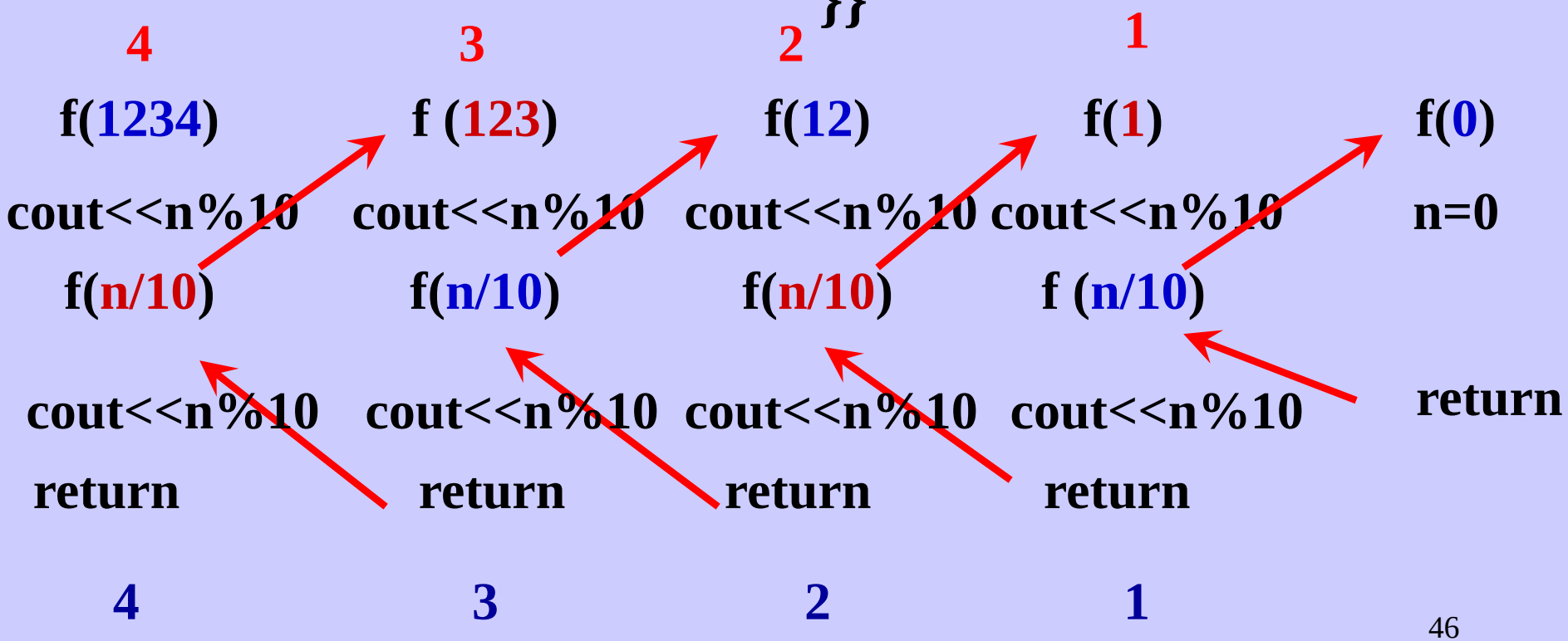
算法相同,层层调用,每层函数的变量所占内存单元不同。

```
void main (void)
{   int  i=5;
    cin>>i;
    f(i);
}
```

43211234

输入：1234

```
void f(int  n )
{if(n= =0)   return;
   else   {cout<<n%10;
           f(n/10);
   cout<<n%10; return;
}}
```



```
void recur(char c)
{  cout<<c;
   if(c<'5') recur(c+1);
  cout<<c;
}
```

```
void main(void)
{  recur('0');
}
```

输出：012345543210

```
void f(int n)
{  if(n>=10)
       f(n/10);
 cout<<n<<endl;
}
void main(void)
{  f(12345);
}
```

1

12

123

1234

12345

作用域和存储类

作用域是指程序中所说明的标识符在哪一个区间内有效，即在哪一个区间内可以使用或引用该标识符。在C++中，作用域共分为五类：块作用域、文件作用域、函数原型作用域、函数作用域和类的作用域。

块作用域

我们把用花括号括起来的一部分程序称为一个块。在块内说明的标识符，只能在该块内引用，即其作用域在该块内，开始于标识符的说明处，结束于块的结尾处。

在一个函数内部定义的变量或在一个块中定义的变量称为局部变量。

在函数内或复合语句内部定义的变量，其作用域是从定义的位置起到函数体或复合语句的结束。形参也是局部变量。

```
float  f1( int  a)
{  int  b,c;
   .....
}
```

a,b,c有效

```
void main(void )
{  int  m, n;
   .....
}
```

m,n有效

```
float  f2( int  x,  int y)
{  int  i, j;
   .....
}
```

x,y,i,j 有效

主函数main中定义的变量，也只在主函数中有效，同样属于局部变量。

不同的函数可以使用相同名字的局部变量，它们在内存中分属不同的存储区间，互不干扰。

```
void main(void)
{   int  x=10;
    {  int  x=20;
       cout<<x<<endl;        20
    }
    cout<<x<<endl;           10
}
```

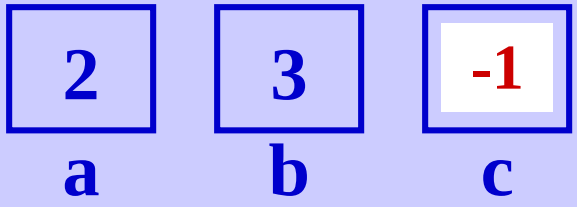
定义变量既是在内存中开辟区间

注意：

具有块作用域的标识符在其作用域内，将屏蔽其作用块包含本块的同名标识符，即

变量名相同，局部更优先。

```
void main(void)
{    int a=2, b=3, c=5;
     cout<<a<<'\t'<<b<<'\t'<<c<<endl;
     {       int  a, b=2;
             a=b+c;
             cout<<a<<'\t'<<b<<'\t'<<c<<endl;
     }
     c=a-b;
     cout<<a<<'\t'<<b<<'\t'<<c<<endl;
}
```




```
2       3       5
7       2       5
2       3       -1
```

```
void main(void)
{  int  a=1,b=2,c=3;
    ++a;                                                    a=2
   c+=++b;                                                  b=3,  c=6
   {   int  b=4, c;                                        b=4
       c=b*3;                                              c=12
       a+=c;                                               a=14
       cout<<“first:”<<a<<‘\t’<<b<<‘\t’<<c<<endl;          a=14,b=4,c=12
       a+=c;                                               a=26
      cout<<“second:”<<a<<‘\t’<<b<<‘\t’<<c<<endl;          a=26,b=4,c=12
    }
 cout<<“third:”<<a<<‘\t’<<b<<‘\t’<<c<<endl;               a=26,b=3,c=6
}
```

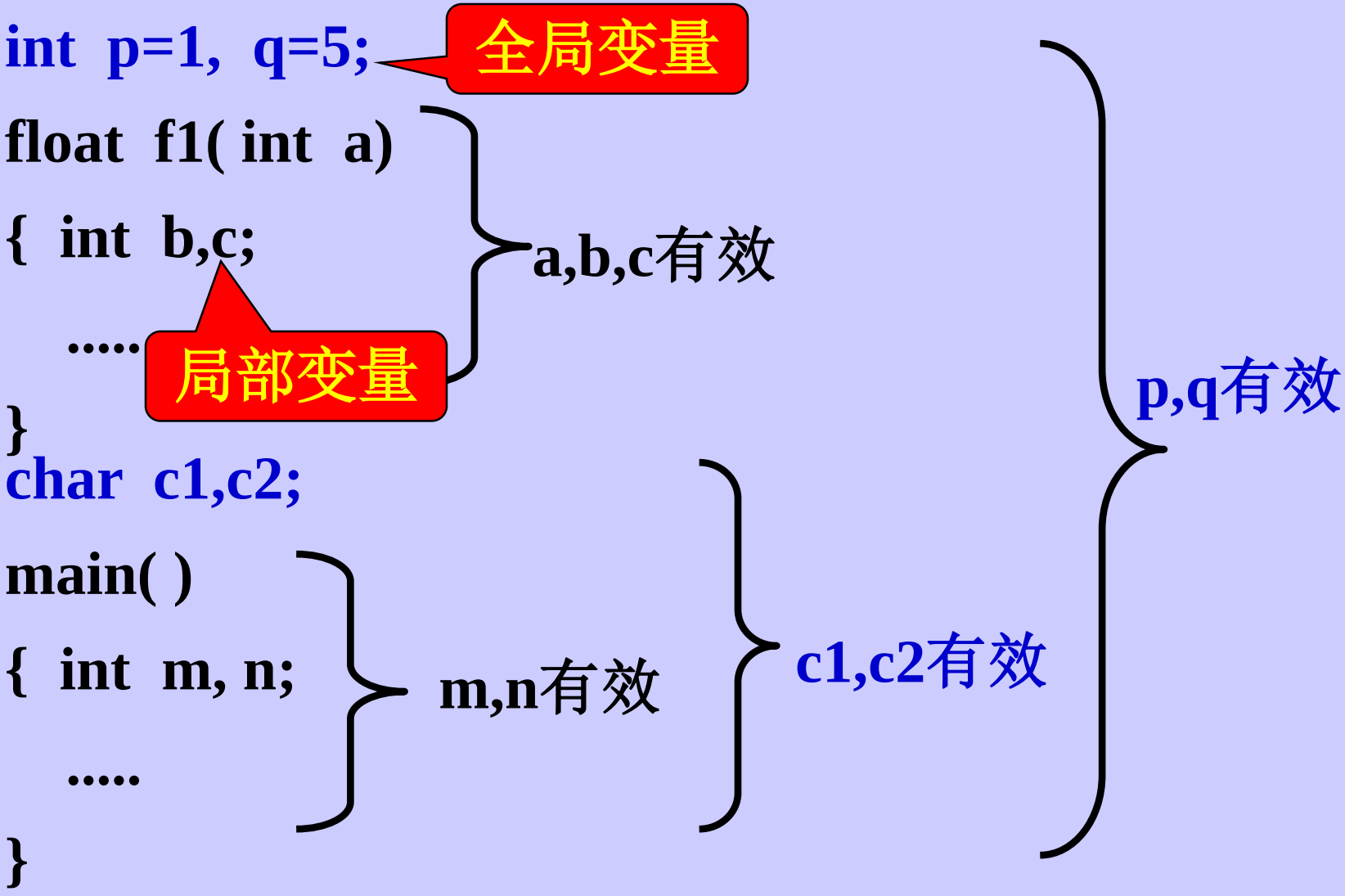
文件作用域

在函数外定义的变量称为全局变量。

全局变量的作用域称为文件作用域，即在整个文件中都是可以访问的。

其缺省的作用范围是:从定义全局变量的位置开始到该源程序文件结束。

当在块作用域内的变量与全局变量同名时，局部变量优先。



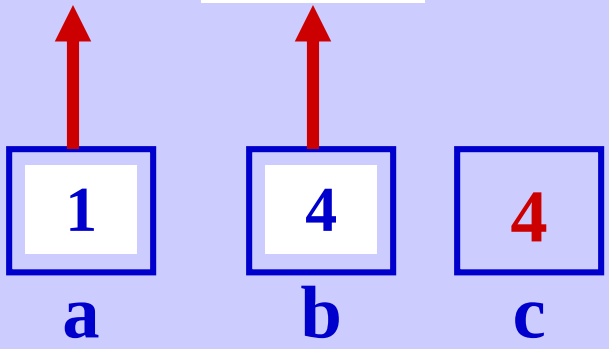
全局变量增加了函数间数据联系的渠道，在函数调用时可以得到多于一个的返回值。

```
int  min;
int  max (int x, int  y)
{   int  z;
     min=(x<y)?x : y;
     z=(x>y)? x : y ;
     return  z;
}
void main (void)
{    int   a,b,c;
     cin>>a>>b;
     c=max (a , b) ;
     cout<<“The max is”<<c<<endl;
     cout<<“ The min  is”<<min<<endl;
}
```

全局变量

函数值为4

min 在main( )和max( )中均有效，在内存中有唯一的存储空间。

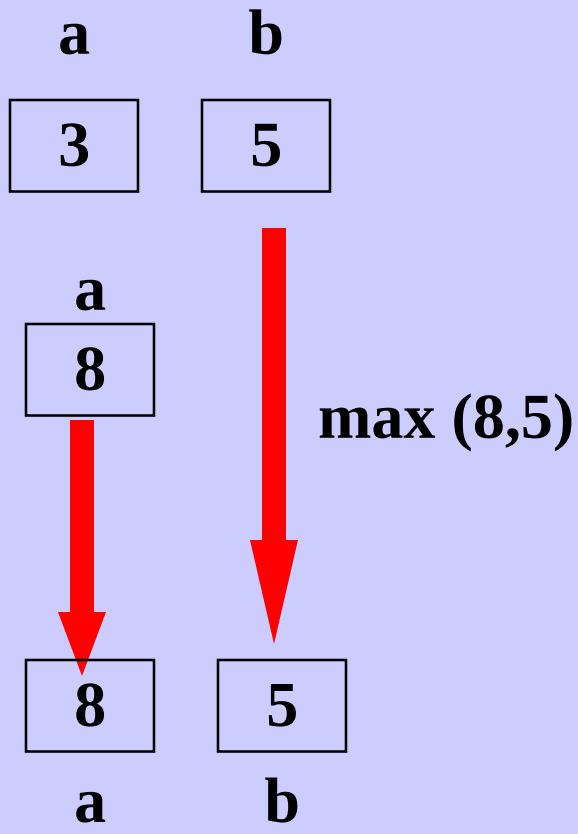


The max is 4

The min is 1

在同一个源文件中，外部变量与局部变量同名，则在局部变量的作用范围内，外部变量不起作用。

```
int  a=3, b=5;
int max(int  a, int  b)
{  int c;
   c=a>b? a:b;
   return  c;
}
void main(void)
{  int   a=8;
   cout<<max(a,b)<<endl;
}
```



输出：8

```
int  x;
void cude(void)
{  x=x*x*x ;
}
void main (void)
{   int  x=5;
    cude ( );
    cout<<x<<endl;
}
```

x为0

输出： 125

输出： 5

在块作用域内可通过作用域运算符“：：”来引用与局部变量同名的全局变量。

```
#include <iostream.h>
int  i= 100;
void main(void)
{
   int i , j=50;
   i=18;          //访问局部变量i
   ::i= ::i+4;    //访问全部变量i
   j= ::i+i;      //访问全部变量i和局部变量j
  cout<<”::i=”<<::i<<’\n’;
  cout<<”i=”<<i<<’\n’;
  cout<<”j=”<<j<<’\n’;
}
```

```
::i=104
i=18
j=108
```

函数原型作用域

在函数原型的参数表中说明的标识符所具有的作用域称为函数原型作用域，它从其说明处开始，到函数原型说明的结束处结束。

float tt(int x , float y); //函数tt的原型说明

由于所说明的标识符与该函数的定义及调用无关，所以，可以在函数原型说明中只作参数的类型说明，而省略参量名。

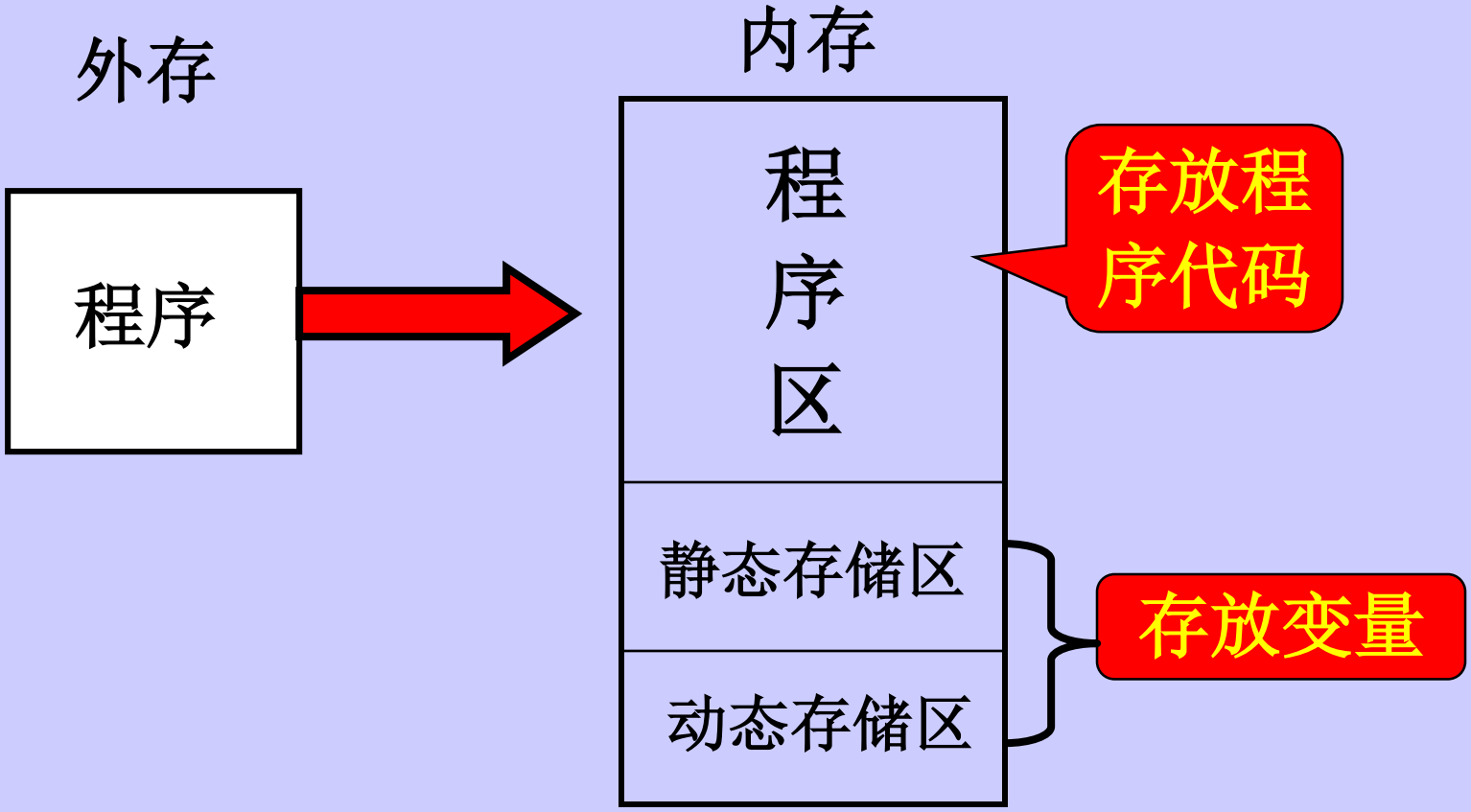
float tt (int , float);

```
int  i=0;
int workover(int i)
{  i=(i%i)*((i*i)/(2*i)+4);
    cout<<“i=“<<i<<endl;
    return i;
}
int rest (int i)
{   i=i<2?5:0;
    return i;
}
```

```
void main(void)
{  int i=5;
    rest(i/2);
    cout<<“i=“<<i<<endl;
    rest(i=i/2);
    cout<<“i=“<<i<<endl;
    i= rest(i/2);
    cout<<“i=“<<i<<endl;
    workover(i)
    cout<<“i=“<<i<<endl;
}
```

i=5
i=2
i=5
i=0
i=5

存储类



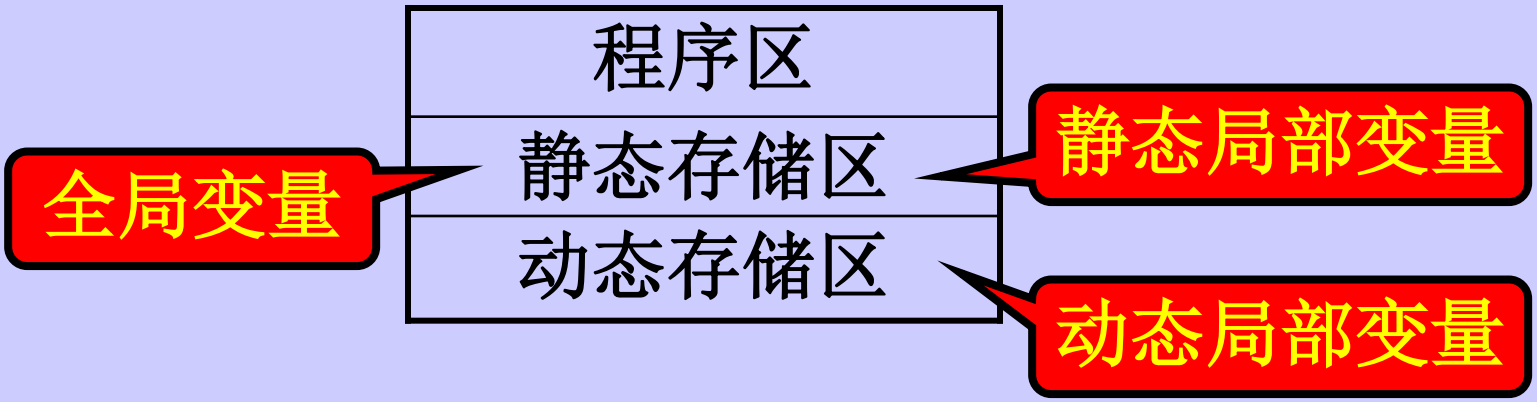
需要区分变量的存储类型

作用域 { 全局变量 / 局部变量

生存期 { 动态存储变量 / 静态存储变量

静态存储：在文件运行期间有固定的存储空间，直到文件运行结束。

动态存储：在程序运行期间根据需要分配存储空间，函数结束后立即释放空间。若一个函数在程序中被调用两次，则每次分配的单元有可能不同。



局部变量的分类

- 动态变量（auto）：默认，存储在动态区
- 寄存器变量（register）：在cpu内部存储
- 静态局部变量（static）：存储在静态区

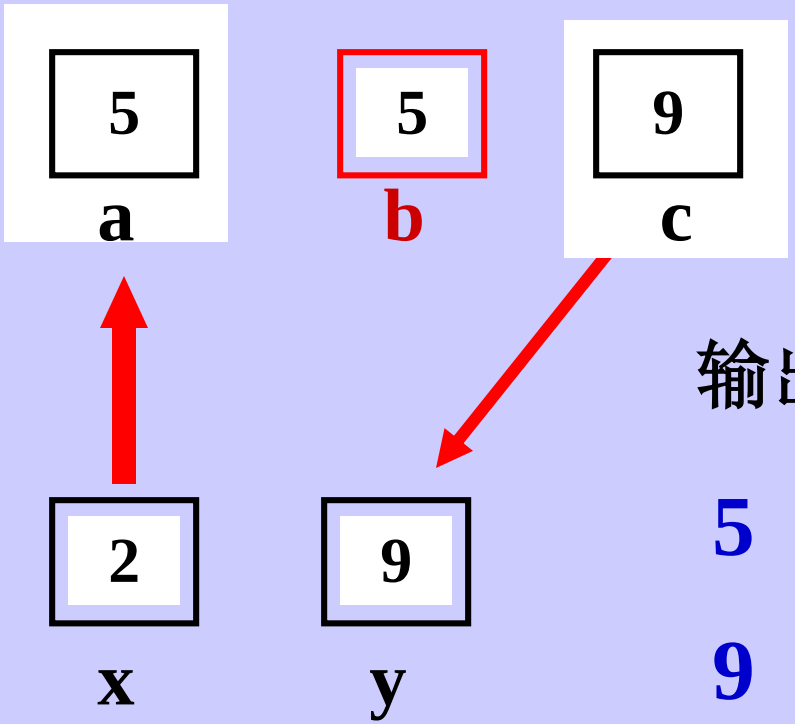
动态局部变量未被赋值时，其值为随机值。其作用域的函数或复合语句结束时，空间被程序收回。

程序执行到静态局部变量时，为其在静态区开辟存储空间，该空间一直被保留，直到程序运行结束。

由于存储在静态区，静态局部变量或全局变量未赋初值时，系统自动使之为0。

```
int fun(int a)
{  int c;
   static  int b=3;
  c=a+ b++;
  return c;
}
void main(void)
{   int x=2, y;
    y=fun(x);
   cout<<y<<endl;
   y=fun(x+3);
   cout<<y<<endl;
}
```

只赋一次初值



输出：

5

9

变量b是静态局部变量，在内存一旦开辟空间，就不会释放，空间值一直保留

```
int f (int  a)
{  int  b=0;
   static  int c=3;
   b=b+1;
   c=c+1;
   return  a+b+c;
}

void main(void)
{  int  a=2,i;
    for (i=0;i<3;i++)
        cout<<f(a)<<endl;
}
```

只赋一次初值

i=0 a=2
b=0, b=1 输出：7
c=3, c=4

i=1 a=2
b=0, b=1 输出：8
c=4, c=5

i=2 a=2
b=0, b=1 输出：9
c=5, c=6

7

8

9

```
int func (int a,  int b)
{  static  int  m=0, i=2;
    i+=m+1;
    m=i+a+b;
    return   m;
}
void main(void)
{  int  k=4, m=1, p;
   p=func(k, m);      cout<<p<<endl;
   p=func(k, m);      cout<<p<<endl;
}
```

func(4, 1)
a=4, b=1
m=0, i=2
i=3
m=3+4+1=8

func(4, 1)
a=4, b=1
m=8, i=3
i=3+8+1=12
m=12+4+1=17

输出：8,17

```
int q(int x)
{   int y=1;
    static int z=1;
    z+=z+y++;
    return x+z;
}
void main(void)
{  cout<<q(1)<<'\t';
    cout<<q(2)<<'\t';
    cout<<q(3)<<'\t';
}
```

4 9 18

全局变量的存储方式（extern　static）

全局变量是在函数的外部定义的，编译时分配在静态存储区，如果未赋初值，其值为0。

1、extern 存储类别

全局变量的默认方式，当在一个文件中要引用另一个文件中的全局变量或在全局变量定义之前要引用它时，可用extern作说明，相当于扩大全局变量的作用域。

2、静态（static）存储类别

它仅能在本文件中引用，即使在其它文件中用extern说明也不能使用。相当于限制了全局变量的作用域范围。

程序的作用是：给定b的值，输入a和m，求a×b和a^m的值。

文件file1.c中的内容为：

```
int   a;
void main(void)
{ extern  int  power (int);
    int  b=3, c, d, m;
    cin>>a>>m;
    c=a*b;
    cout<<a<<“*”<<b<<“=“<<c<<endl;
    d= power(m);
    cout<<a<<“**”<<m<<“=“<<d<<endl;
}
```

外部全局变量定义

外部全局变量说明

引用文件外定义的全局变量

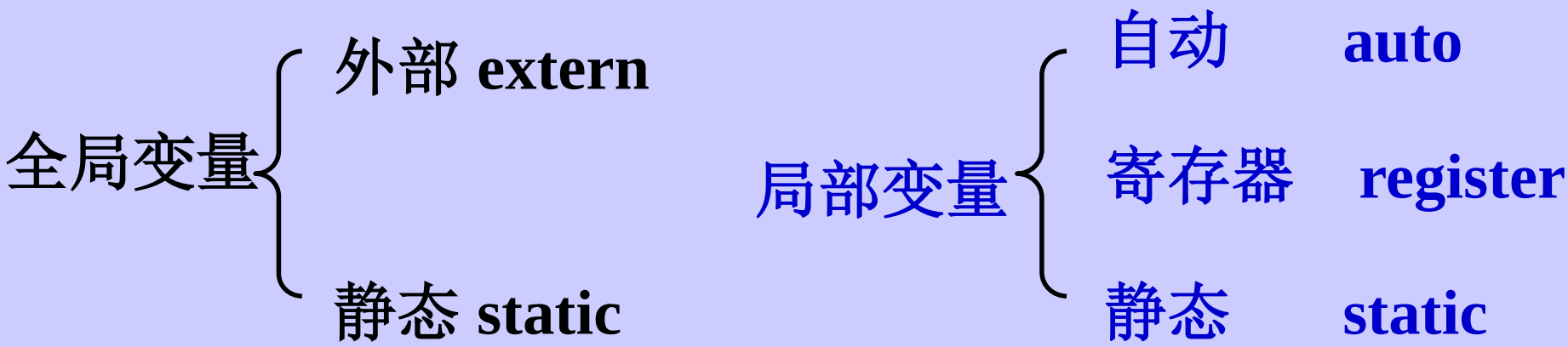
文件file2.c中的内容为：

```
extern  int   a;
int power (int n )
{  int  i, y=1;
    for (i=1; i<=n; i++)
        y*=a;
    return   y;
}
```

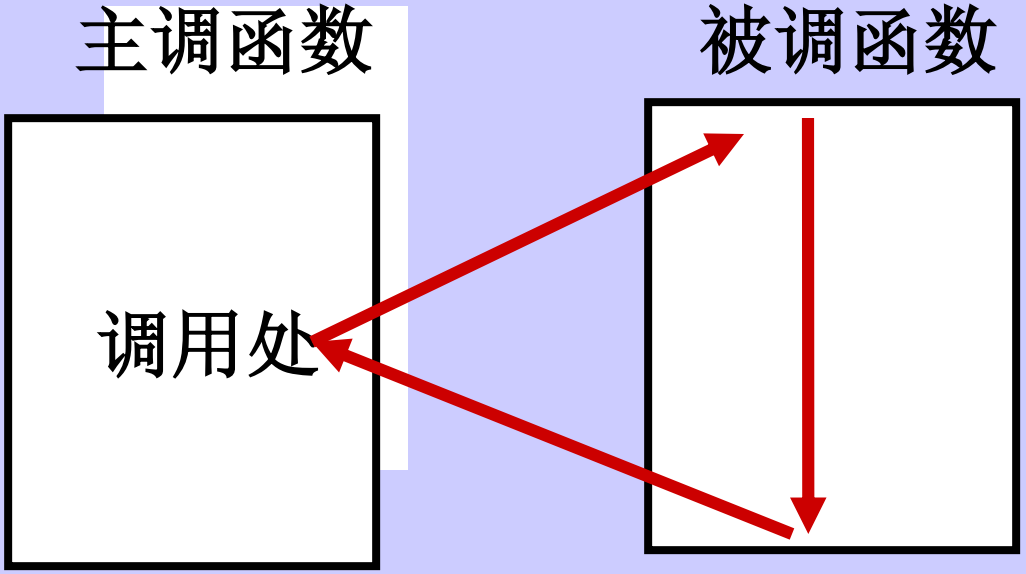
静态局部变量：static 在函数内部定义，存储在静态存储区，与auto对应，在别的函数中不能引用。

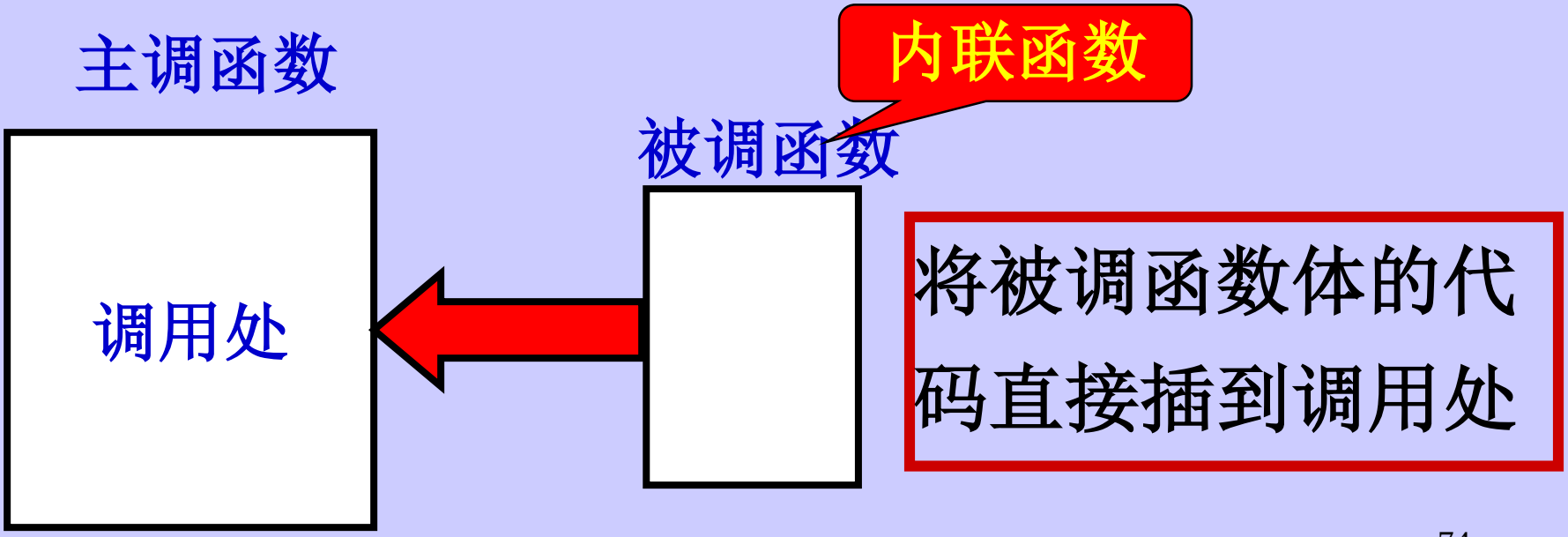
全局静态变量：static 在函数外部定义，只限在本文件中使用，与extern对应。

当变量名相同致使作用域相重时，起作用的是最近说明的那个变量。



内联函数





内联函数的实质是用存储空间（使用更多的存储空间）来换取时间（减少执行时间）.

内联函数的定义方法是，在函数定义时，**在函数的类型前增加修饰词inline。**

```
inline int max (int x, int y)
{   int z;
    z=(x>y)? x : y ;
    return z;
}
void main (void )
{   int a,b,c;
    cin>>a>>b;
    c=max (a+b , a*b) ;
    cout<<"The max is"<<c<<endl;
}
```

使用内联函数时应注意以下几点：

1、C++中，除在函数体内含有循环，switch分支和复杂嵌套的if语句外，所有的函数均可定义为内联函数。

2、内联函数也要定义在前，调用在后。形参与实参之间的关系与一般的函数相同。

3、对于用户指定的内联函数，编译器是否作为内联函数来处理由编译器自行决定。说明内联函数时，只是请求编译器当出现这种函数调用时，作为内联函数的扩展来实现，而不是命令编译器要这样去做。

4、正如前面所述，内联函数的实质是采用空间换取时间，即可加速程序的执行，当出现多次调用同一内联函数时，程序本身占用的空间将有所增加。如上例中，内联函数仅调用一次时，并不增加程序占用的存储间。

具有缺省参数值和参数个数可变的函数

在C++中定义函数时，允许给参数指定一个缺省的值。在调用函数时，若明确给出了这种实参的值，则使用相应实参的值；若没有给出相应的实参，则使用缺省的值。（举例说明）

```
int  fac(int n=2)
{   int t=1;
    for(int i=1;i<=n;i++)
          t=t*i;
   return t;
}
void main(void)
{
   cout<< fac( )  <<endl;
}
```

输出：720

输出：2

```
int  area(int long=4 , int width=2)
{   return  long* width;
}

void  main(void )
{    int  a=8,  b=6;
     cout<< area(a,b) <<endl;
    cout<< area(a) <<endl;
    cout<< area( ) <<endl;
}
```

```
48
16
8
```

使用具有缺省参数的函数时，应注意以下几点：

1.不可以靠左边缺省

```
int area(int long , int width=2)
int area(int long =4, int width)
```

错误！

2.函数原型说明时可以不加变量名

```
float v(float,float=10,float=20);
```

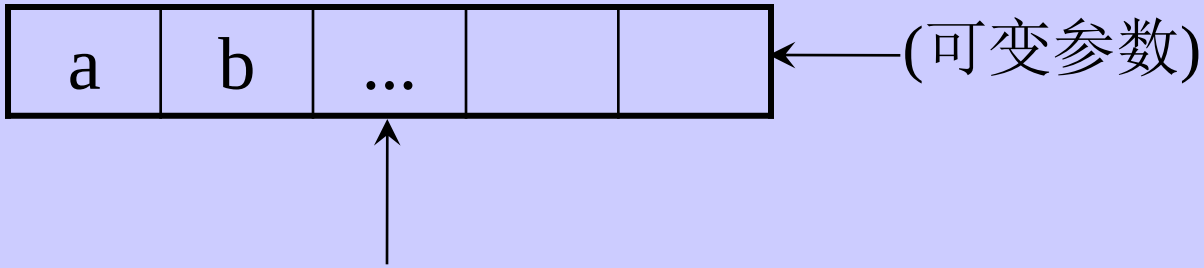
3.只能在前面定义一次缺省值，即原型说明时定义了缺省值，后面函数的定义不可有缺省值。

参数个数可变的函数

到目前为止，在定义函数时，都明确规定了函数的参数个数及类型。在调用函数时，实参的个数必须与形参相同。在调用具有缺省参数值的函数时，本质上，实参的个数与形参的个数仍是相同的，由于参数具有缺省值，因此，在调用时可省略。**在某些应用中，在定义函数时，并不能确定函数的参数个数，参数的个数在调时才能确定**。在C++中允许定义参数个数可变的函数。

首先，必须包含头文件“stdarg.h”，因为要用到里面的三个库函数va_start( )、va_arg( )和va_end( )。

其次，要说明一个va_list类型的变量。va_list与int，float类同，它是C++系统预定义的一个数据类型(非float)，只有通过这种类型的变量才能从实际参数表中取出可变有参数。如：va_list ap;



va_start(ap,b)：初始化

va_arg(ap,int)：依次取参数

va_end(ap)：正确结束

va_start():有两个参数，va_start(ap,b); b即为可变参数前的最后一个确定的参数。

va_arg():有两个参数，va_arg(ap,int)　int即为可变参数的数据类型名。

int temp;

temp=va_arg(ap,int);

va_end():完成收尾工作。va_end(ap);

在调用参数个数可变的函数时，必定有一个参数指明可变参数的个数或总的实参个数。如第一个参数值为总的实际参数的个数。

使用参数数目可变的函数时要注意以下几点：

1、在定义函数时，固定参数部分必须放在参数表的前面，可变参数在后面，并用省略号“...”表示可变参数。在函数调用时，可以没有可变的参数。

2、必须使用函数va_start()来初始化可变参数，为取第一个可变的参数作好准备工作；使用函数va_arg()依次取各个可变的参数值；最后用函数va_end()做好结束工作，以便能正确地返回。

3、在调用参数个数可变的函数时，必定有一个参数指明可变参数的个数或总的实参个数。

函数的重载

所谓函数的重载是指完成不同功能的函数可以具有相同的函数名。

C++的编译器是根据函数的实参来确定应该调用哪一个函数的。

```
int  fun(int a,  int b)
{   return a+b;  }

int  fun (int a)
{   return a*a;  }
```

```
void  main(void)
{  cout<<fun(3,5)<<endl;
   cout<<fun(5)<<endl;
}
```

8

25

1、定义的重载函数必须具有不同的参数个数，或不同的参数类型。只有这样编译系统才有可能根据不同的参数去调用不同的重载函数。

2、仅返回值不同时，不能定义为重载函数。

即仅函数的类型不同，不能定义为重载函数

```
int  fun(int a,  int b)
{   return a+b;  }

float   fun (int a,int b)
{  return   (float) a*a;  }
```

```
void  main(void)
{  cout<<fun(3,5)<<endl;
   cout<<fun(3,5)<<endl;
}
```

```
double sin(double x1,double x2)
{       return x1*x2;}
double sin(double x,int a)
{       return a+x;}
void main(void)
{       double x;
        cin>>x;
        cout<<sin(x)<<endl;;
        cout<<sin(x,x)<<endl;
        cout<<sin(x,10)<<endl;
}
```

sin(x,x)

sin(x,10)

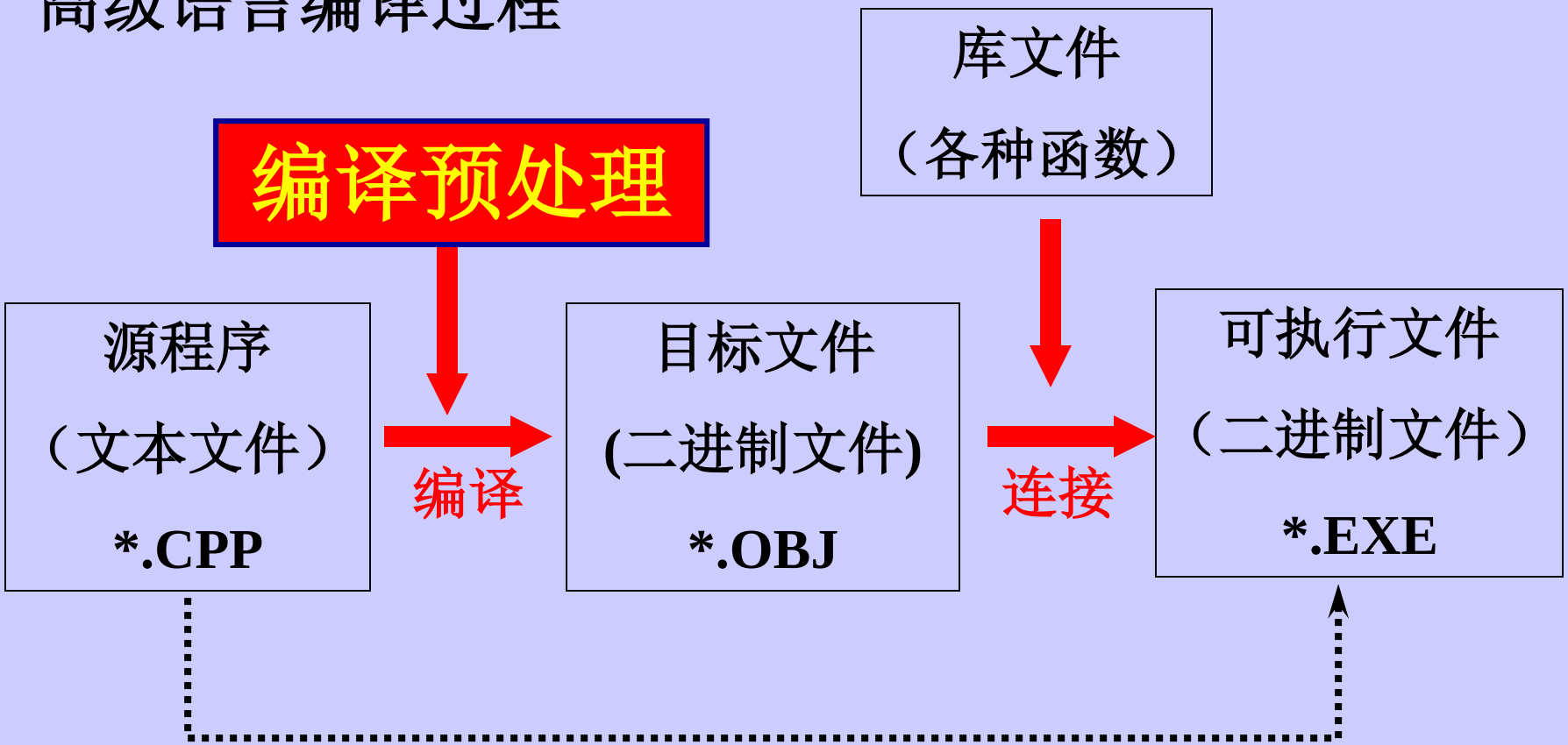
不同的参数类型

```
int add(int a,int b,int c)
{       return a+b+c;  }
int add(int a,int b)
{       return a+b;   }
void main(void)
{       cout<<"3+5="<<add(3,5)<<endl;
        cout<<"3+5+8="<<add(3,5,8)<<endl;
}
```

不同的参数个数

编译预处理

高级语言编译过程



C语言提供的编译预处理的功能有以下三种：

宏定义　　文件包含　　条件编译

宏定义

不带参数的宏定义

用一个指定的标识符（即名字）来代表一个字符串，以后凡在程序中碰到这个标识符的地方都用字符串来代替。

这个标识符称为宏名，编译前的替代过程称为“宏展开”。

 # define　标识符　字符串

```
#define  PRICE   30
void main(void)
{    int  num, total;  /*定义变量*/
     num=10;        /*变量赋值*/
    total=num*PRICE;
    cout<<"total="<<total<<endl;
}
```

编译前用30替代

编译程序将宏定义的内容认为是字符串，没有任何实际的物理意义。

注意：

1、宏展开只是一个简单的“物理”替换，不做语法检查，不是一个语句，其后不加分号“；”

2、#define命令出现在函数的外面，其有效范围为定义处至本源文件结束。可以用# undef命令终止宏定义的作用域。

```
#define G  9.8
void main(void )
{.....}
# undef   G
int max(int a,int b)
{..... }
```

3、对程序中用双引号括起来的字符串内容，即使与宏名相同，也不进行置换。

4、在进行宏定义中，可以用已定义的宏名，进行层层置换。

```
# define  R  3.0
# define   PI  3.1415926
# define   L   2*PI*R
# define   S    PI*R*R
void main(void)
{      cout<<“L=“<<L<<“  S=”<<S<<endl;
}
```

层层置换

层层置换

不置换

不置换

带参数的宏定义

#define　宏名(参数表)　字符串

```
#define     S(a, b)          a*b
...
float  x, y,area;
cin>>x>>y;
 area=S(x, y);    /*  area=x*y;  */
```

宏定义

形式参数

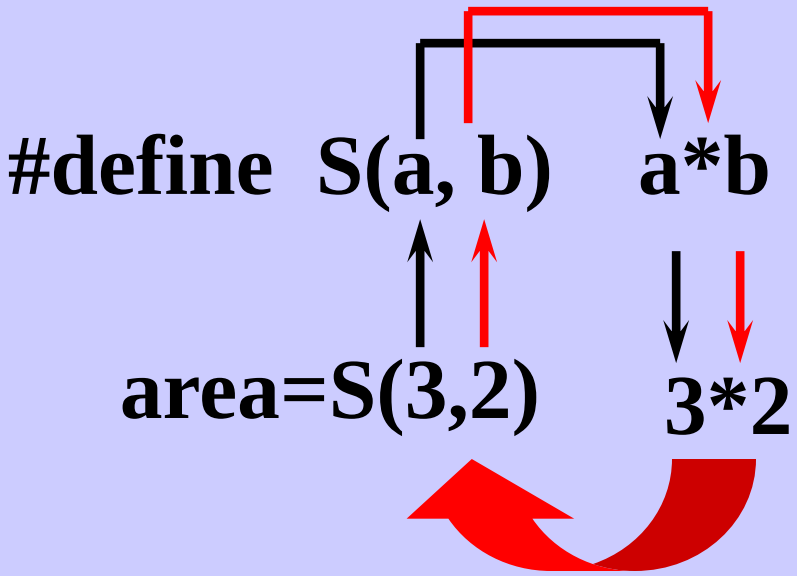
定义的宏

实参代入后还原

宏调用

实际参数

按#define命令行中指定的字符串从左至右进行置换宏名，字符串中的形参以相应的实参代替，字符串中的非形参字符保持不变。



机械地将实参代入宏定义的形参形式

S(a,b)等同于 a*b

S(3,2)等同于 3*2

```
#define  PI  3.1415926
#define  S(r)  PI*r*r
void main(void)
{  float  a,  area,  b;
    a=3.6; b=4.0;
    area=PI*a*a
   cout<<“r=“<<a<<“\narea=”<<area<<endl;
}
```

S(r) ⟶ PI*r*r

S(a) ⟶ PI*a*a

编译前机械替换，
实参形参一一对应

```
#define  PI  3.1415926

#define  S(r)  PI*r*r

void main(void)
{  float  a,  area,  b;
    a=1; b=2;
    area=S(a+b);
   cout<<“r=“<<a<<“\narea=”<<area<<endl;
}
```

S(r) → PI*r*r

S(a+b) → PI*a+b*a+b

S(a+b) → PI*(a+b)*(a+b)

错误

编译前机械替换，
实参形参一一对应

宏展开时实参不运算，不作语法检查

```
#define  S(r)  PI*(r)*(r)
```

定义宏时在宏名与带参数的括弧间不能有空格。

```
#define  S_ (r)   P*r*r
```

带参数的宏与函数调用的区别

相同：有实参、形参，代入调用。

不同之处：

1、函数调用先求表达式的值，然后代入形参，而宏只是机械替换。

2、函数调用时形参、实参进行类型定义，而宏不需要，只是作为字符串替代。

3、函数调用是在运行程序时进行的，其目标代码短，但程序执行时间长。而宏调用是在编译之前完成的，运行时已将代码替换进程序中，目标代码长，执行时间稍快。

一般用宏表示实时的、短小的表达式。

```
#define  A  3
#define B(a)  ((A+1)*a)
```

93

执行 x=3*(A+B(7)); 后， x的值为:

```
#define  neg(x)   ((-x)+1)
int  neg( int x)
{return  x+1;  }
void main(void)
{  int y;
   y=((-1)+1)
  cout<<“y=“<<y<<endl;
}
```

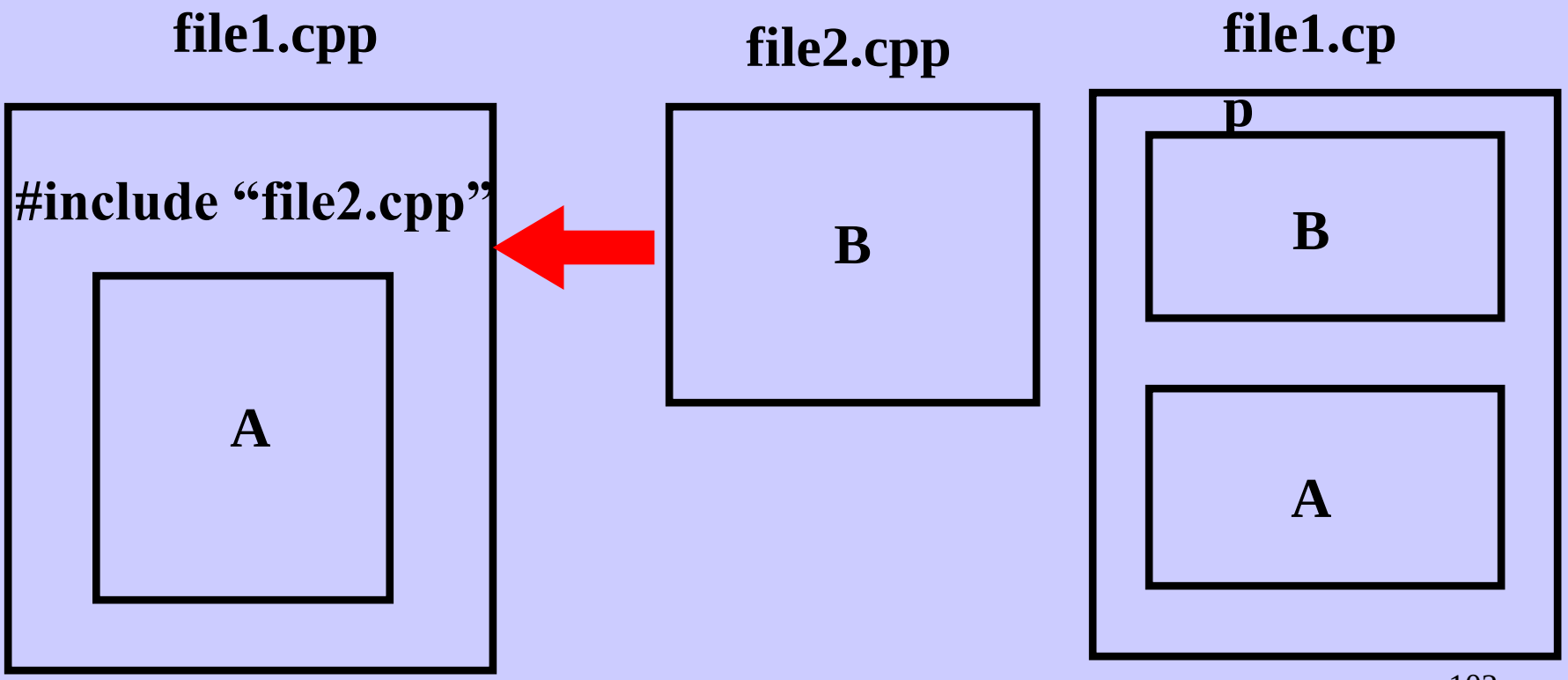
编译前机械替换，
实参形参一一对应

y=0

文件包含

一个源文件可以将另外一个源文件的全部内容包含进来，即将另外的文件包含到本文件之中。

 # include “文件名”



注意：

1、文件名是C的源文件名，是文本文件，后缀名可以任选。*.cpp *.h

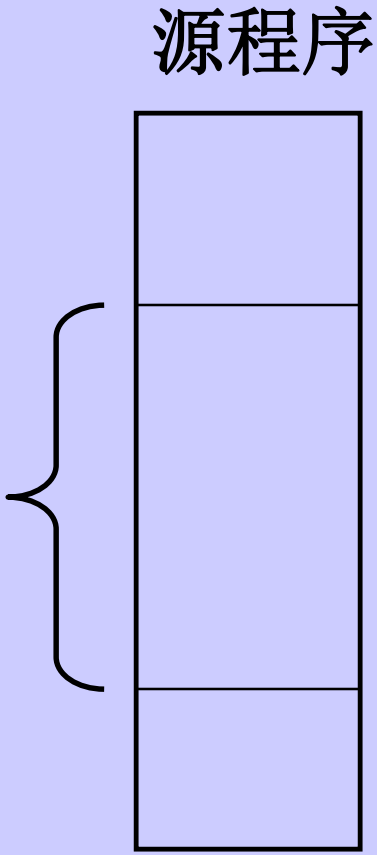
2、一个#include语句只能指定一个被包含文件。

3、文件名用双引号或尖括号括起来。

4、包含后所有源文件编译为一个可执行文件。

条件编译

C语言允许有选择地对程序的某一部分进行编译。
也就是对一部分源程序指定编译条件。



可以将部分源程序
不转换为机器码

条件编译有以下几种形式：

标识符

1、 # ifdef 标识符

程序段1

else

程序段2

end if

define DEBUG

.....

ifdef DEBUG

cout<<x<<'\t'<<y<<endl;

endif

当标识符已被定义过（用#define定义)，则对程序段1进行编译，否则编译程序段2.

2、 # ifndef 标识符

程序段1

else

程序段2

endif

define DEBUG

.....

ifndef DEBUG

cout<<x<<'\t'<<y<<endl;

endif

与形式1相反，当标识符没有被定义过（用#define定义)，则对程序段1进行编译，否则编译程序段2。

调试完后加#define DEBUG，则不输出调试信息。

3、 # if 表达式

程序段1

else

程序段2

endif

define DEBUG 1

.....

if DEBUG

cout<<x<<'\t'<<y<<endl;

endif

当表达式为真(非零)，编译程序段1，表达式为零，编译程序段2。

调试完后改为 #define DEBUG 0，则不输出调试信息。

采用条件编译后，可以使机器代码程序缩短。

以下程序的运行结果是：

```
#define  DEBUG
void main(void)
{   int  a=14, b=15,  c;
    c=a/b;
    # ifdef  DEBUG
    cout<<“a=“<<oct<<a<<“ b=“<<b<<endl;
    # endif
    cout<<“c=“<<dec<<c<<endl;
}
```

输出：a=16, b=17c=0

程序的多文件组织

而在设计一个功能复杂的大程序时，为了便于程序的设计和调试，通常将程序分成若干个模块，把实现一个模块的程序或数据放在一个文件中。**当一个完整的程序被存放在多于一个文件中时，称为程序的多文件组织。**

内部函数和外部函数

内部函数：函数只限于在本文件中调用，其它文件不能调用，用**static** 定义该函数。

```
static  float  fac( int  n)

{  .....    }
```

外部函数：函数的默认形式，可以被其它文件调用，用**extern** 定义该函数。调用时，在文件中用**extern** 说明。

```
void main(void)
{  extern  enter_string( );
   char   str[80];
   enter_string(str);
   ..........
}
```

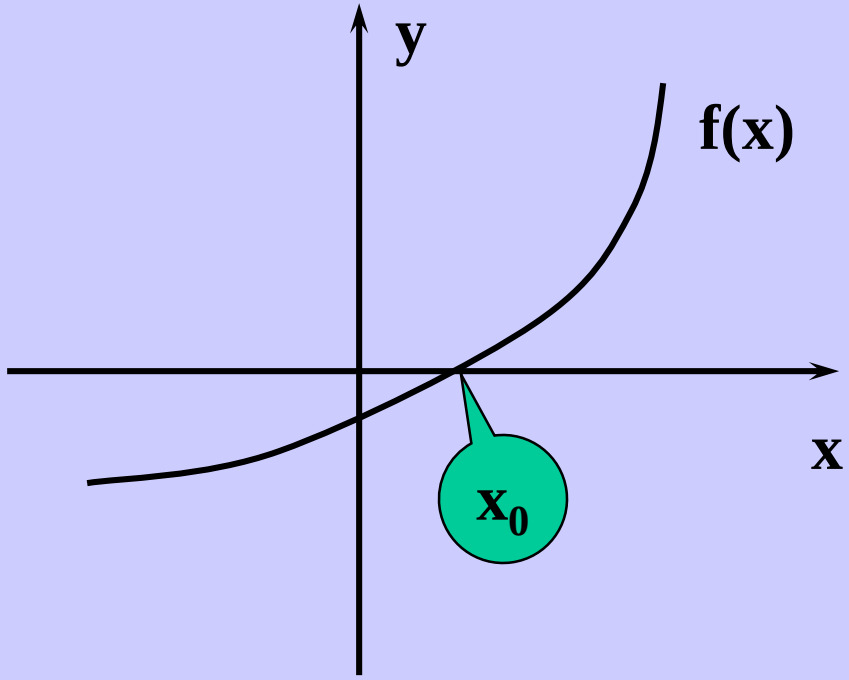
说明外部函数

补充算法

方程求解

1、牛顿切线法

只有为数不多的方程有精确解，一般都是用迭代方法近似求方程的解。方程f(x)=0的实数解实际上是曲线f (x)在x轴上交点的值。



1、任选一x值x1,在y1=f(x1)处做切线与x轴相交于x2处。

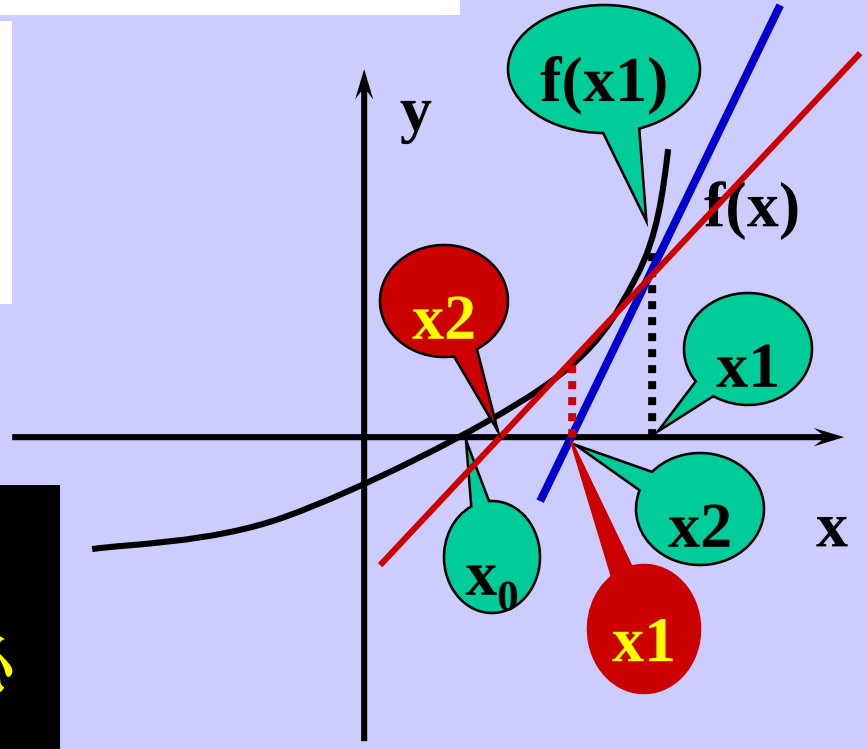
2、若|x2-x1|或|f(x2)|小于指定的精度，则令x1=x2，继续做

1。当其满足所需的精度时，x2就是方程的近似解。

根据已知点求其切线的公式为：

$$x2 = x1 - \frac{f(x1)}{f'(x1)}$$

这就是牛顿切线法。



牛顿切线法收敛快，适用性强，缺陷是必须求出方程的导数。

已知方程为f(x)=x*x-a时，用牛顿切线法求方程的解。给定初值x0，精度10^{-6}，算法编程如下。

```
cin>>x1; //从键盘输入x0
do
{   x0=x1;
    x1=x0-(x0*x0-a)/(2*x0) ;  //
 } while (fabs(x1-x0)>=1e-6) ;
cout>>"x="<<x1<<endl;
```

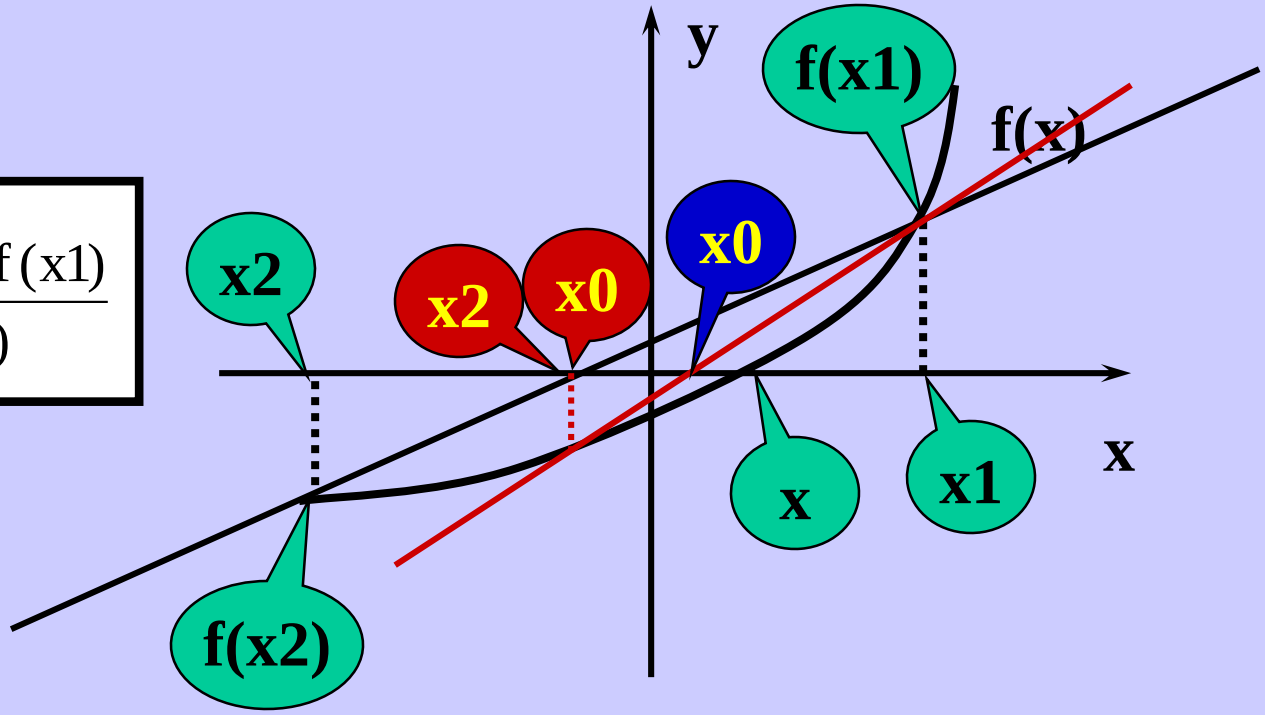
上一循环的新值成为本次循环的旧值

旧值算本次循环的新值

$$x1 = x0 - \frac{f(x0)}{f'(x0)}$$

2、弦截法

$$x0 = \frac{x1 * f(x2) - x2 * f(x1)}{f(x2) - f(x1)}$$



1、在x轴上取两点x1和x2, 要确保x1与x2之间有且只有方程唯一的解。

2、x1与x2分别与f(x)相交于y1=f(x1)、y2=f(x2)。

3、做直线通过y1、y2与x轴交于x0点。

4、若|f(x0)|满足给定的精度，则x0即是方程的解，否则，若f(x0)*f(x1)<0，则方程的解应在x1与x0之间，令x2=x0，继续做2。同理，若f(x0)*f(x1)>0，则方程的解应在x2与x0之间，令x1=x0，继续做2 ,直至满足精度为止。

用两分法求方程的根。 $x^3-5x^2+16x-80=0$

$$x0=\frac{x1*f(x2)-x2*f(x1)}{f(x2)-f(x1)}$$

```
#include <math.h>
float f (float x)
{return  x*x*x-5*x*x+16*x-80;
}
float  xpoint(float x1,float x2)
{  float  x0;
    x0=(x1*f(x2)-x2*f(x1))/(f(x2)-f(x1));
    return   x0;
}
```

输入x，输出f(x)

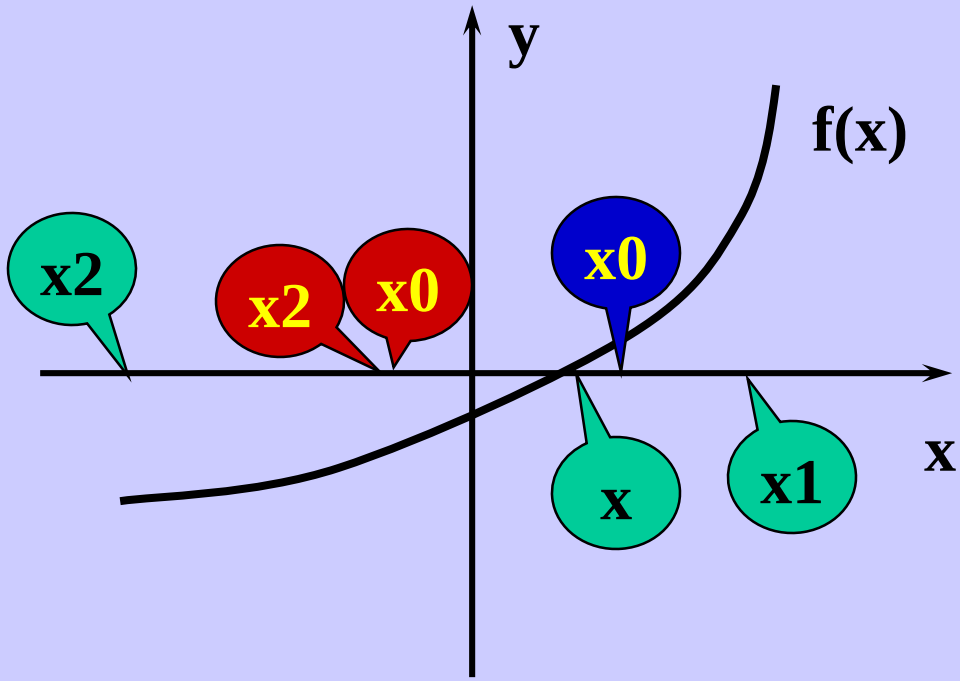
判断输入是否合法

输入x1,x2，输出x0

```
void main(void )
{ float x1,x2, x0, f0, f1, f2;
   do
   {  cout<<“Input x1, x2\n”;
      cin>>x1>>x2;
      f1=f(x1);   f2=f(x2);
   } while (f1*f2>0);
   do
   {  x0=xpoint(x1,x2);
     f0=f(x0);
  if ((f0*f1) >0) { x1=x0;f1=f0;}
  else {  x2=x0;  f2=f0;}
     }while (fabs(f0)>=0.0001);
   cout<<”x=”<< x0<<endl;
}
```

3、两分法

$x0=(x1+x2)/2$



1、在x轴上取两点x1和x2, 要确保x1与x2之间有且只有方程唯一的解。

2、求出x1,x2的中点x0。

3、若|f(x0)|满足给定的精度，则x0即是方程的解，否则，若f(x0)*f(x1)<0，则方程的解应在x1与x0之间，令x2=x0，继续做2。同理，若f(x0)*f(x1)>0，则方程的解应在x2与x0之间，令x1=x0，继续做2 ,直至满足精度为止。

用两分法求方程的根。 $x^3-5x^2+16x-80=0$

$x0=(x1+x2)/2$

```
#include <math.h>
float f (float x)
{return  x*x*x-5*x*x+16*x-80;
}
```

输入x，输出f(x)

判断输入是否合法

求x1与x2的中点

```
void main(void )
{ float x1,x2, x0, f0, f1, f2;
  do
  {  cout<<“Input x1, x2\n”;
     cin>>x1>>x2;
     f1=f(x1);   f2=f(x2);
  } while (f1*f2>0);
do
{  x0=(x1+x2)/2;
    f0=f(x0);
  if ((f0*f1) >0){ x1=x0;f1=f0;}
  else { x2=x0;f2=f0;}
  }while (fabs(f0)>=0.0001);
cout<<”x=”<< x0<<endl;
}
```

```
int q(int x)
{   int y=1;
    static int z=1;
    z+=z+y++;
    return x+z;
}
```

4 9 18

```
void main(void)
{  cout<<q(1)<<'\t';
    cout<<q(2)<<'\t';
    cout<<q(3)<<'\t';
}
```

下面函数pi的功能是：根据以下公式，返回满足精度（0.0005）要求的π的值，请填空。

$$\frac{\pi}{2}=1+\frac{1}{3}+\frac{1\,2}{3\,5}+\frac{1\,2\,3}{3\,5\,7}+\frac{1\,2\,3\,4}{3\,5\,7\,9}+\cdots$$

将后一项除以前一项，找规律

输入精度
输出π

```
#include “math.h”
double pi(double eps)
{  double  s, t;  int n;
  for ( t=1,s=0,n=1 ; t>eps; n++)
    {  s+=t;
       t=n*t/(2*n+1);
     }
    return  2*s ;
}
main( )
{  double x;
   cout<<“\nInput a precision:”;
   cin>>x;
   cout<< “π=“<<pi(x);
}
```

```
void f(int n)
{  if(n>=10)
       f(n/10);
 cout<<n<<endl;
}
void main(void)
{  f(12345);
}
```

```
1
12
123
1234
12345
```

```
void main(void)
{   char  s;  cin.get(s);
    while(s!='\n')
   {  switch(s-'2')
     {   case 0:
         case 1:  cout<<s+4;
         case 2: cout<<s+4;break;
         case 3: cout<<s+3;
         default: cout<<s+2; break;
     }
     cin.get(s);
   }cout<<endl;
}
```

输入：2347<CR>

545455555657