# 第4章 函数与预处理

# 4.1 概述

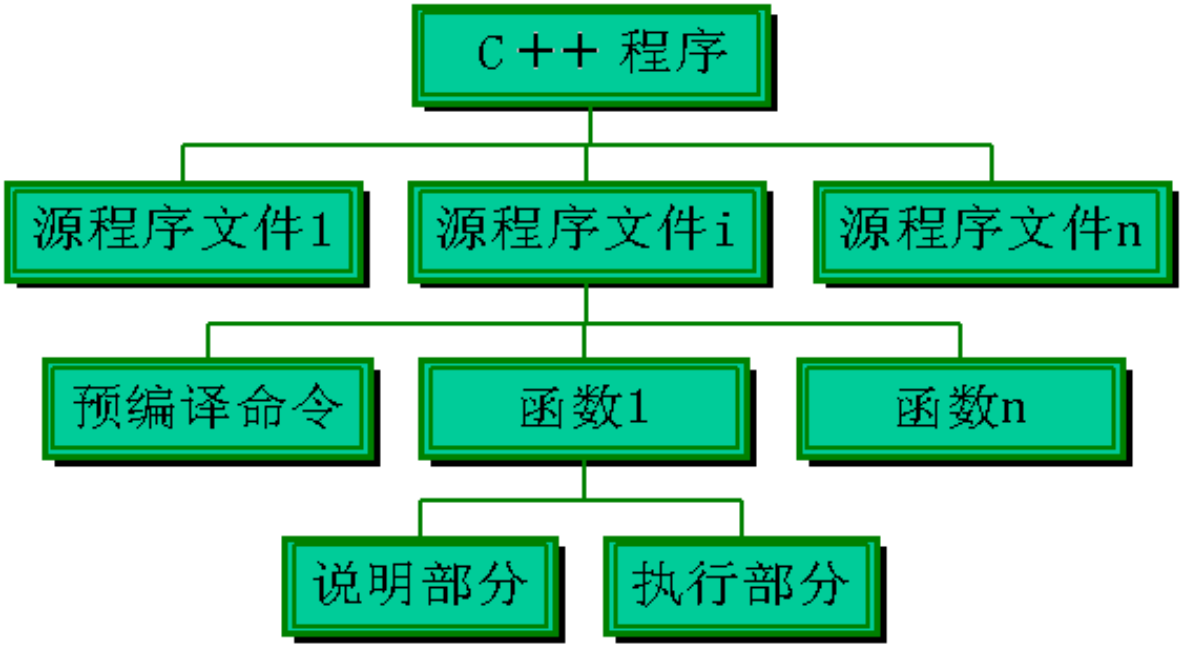
## ◆模块化程序设计

- 基本思想：将一个大的程序按功能分割成一些小模块
- 特点：
  - 各模块相对独立、功能单一、结构清晰、接口简单
  - 控制了程序设计的复杂性
  - 提高元件的可靠性
  - 缩短开发周期
  - 避免程序开发的重复劳动
  - 易于维护和功能扩充
- 开发方法：自上向下,逐步分解，分而治之

## 例 在主函数中调用其他函数。

```
#include <iostream>   //预处理命令
using namespace std;
int max(int x,int y)   //定义函数，函数值为整型，形式参数x，y为整型
{    int z;             //定义本函数中用到的变量z为整型
     if(x>y) z=x;
     else  z=y;
     return(z);         //将z的值返回，通过max带回调用处
}                       //max函数结束
int main( )                 //主函数
{    int a,b,m;             //变量声明
     cin>>a>>b;             //输入变量a和b的值
     m=max(a,b);            //调用max函数，将得到的值赋给m
     cout<<"max="<<m<<endl;     //输出大数m的值
     return 0;
}                           //主函数结束
```

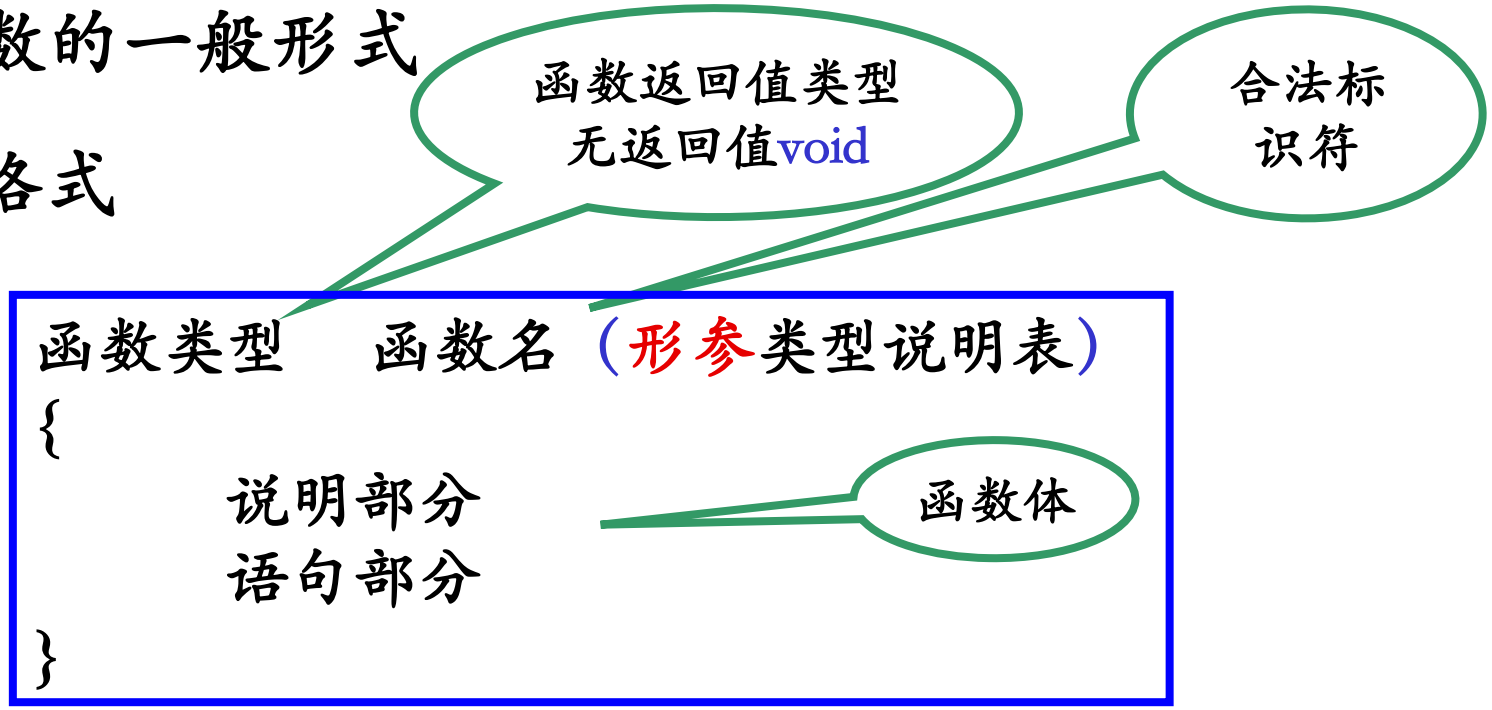
## ◆C++是模块化程序设计语言




- C++是函数式语言
- 必须有且只能有一个名为main的主函数
- C++程序的执行总是从main函数开始，在main中结束
- 函数不能嵌套定义,可以嵌套调用

# 4.2定义函数的一般形式

◆一般格式




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

# 4.3 函数参数和函数的值

## ◆形参与实参

- 形式参数：定义函数时函数名后面括号中的变量名
- 实际参数：调用函数时函数名后面括号中的表达式
- 说明：
  - 实参必须有确定的值
  - 形参必须指定类型
  - 形参与实参类型一致，个数相同
  - 若形参与实参类型不一致，自动按形参类型转换———函数调用转换
  - 形参在函数被调用前不占内存；函数调用时为形参分配内存；调用结束，内存释放

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

例 比较两个数并输出大者

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

运行情况如下：

2 3↙

Max is 3

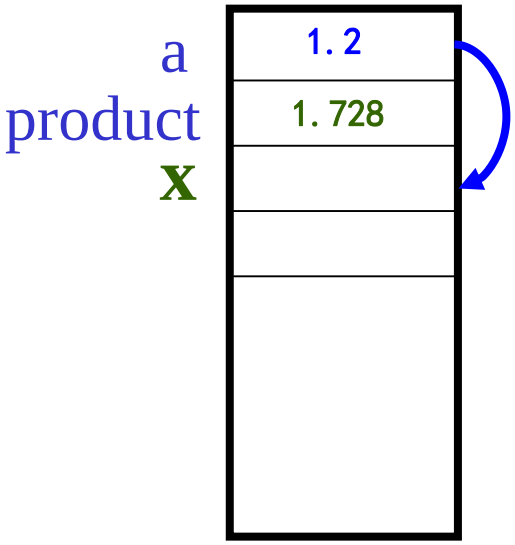
```
#include <iostream>
using namespace std;
int max(int  x, int  y)
{   int z;
    z=x>y?x:y;
    return(z);
}
int main()
{   int a,b,c;
    cin>>a>>b;
    c=max(a,b);
    cout<<"Max is”<<c<<endl;
    return 0;
}
```

形参

实参

# 例：计算x的立方

```
#include <iostream>
using namespace std;
float cube(float  x)
{
        return(x*x*x);
}
int main()
{   float  a, product;
    cout<<"Please input value of a:";
    cin>>a;
    product=cube(a);
    cout<< "Cube of " <<a<< "is " <<product<<endl;
    return 0;
}
```



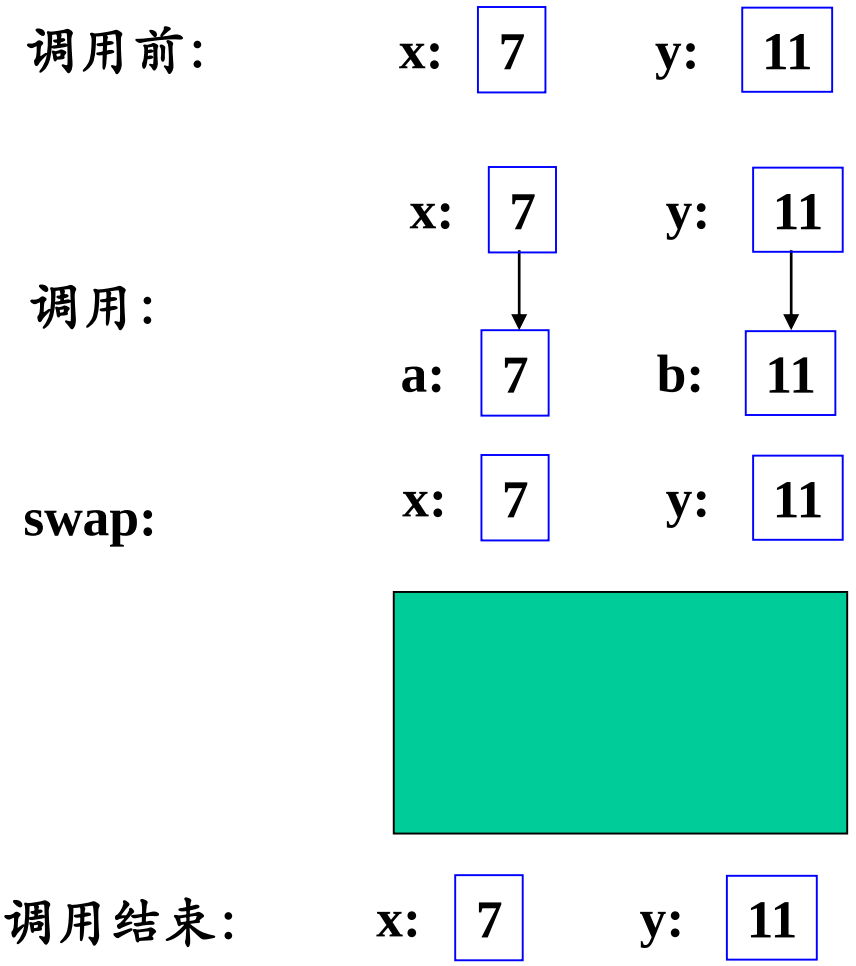
## 参数传递方式

### 值传递方式

- 方式：函数调用时,为形参分配单元,并将实参的值复制到形参中；调用结束，形参单元被释放，实参单元仍保留并维持原值
- 特点：
  - 形参与实参占用不同的内存单元
  - 单向传递

# 例 交换两个数

```
#include <iostream>
using namespace std;
void swap(int a,int b)
{   int temp;
    temp=a; a=b; b=temp;
}
int main( )
{   int x=7,y=11;
    cout<<"swapped:\n“<<endl;
    swap(x,y);
    cout<<x<<y<<endl;
}
```



# 函数的返回值

## 返回语句

- 形式： return(表达式);
  或 return 表达式;
- 功能：使程序控制从被调用函数返回到调用函数中，同时把返回值带给调用函数
- 说明：
  - 函数中可有多个return语句
  - 若无return语句，遇}时，自动返回主调用函数
  - 若函数类型与return语句中表达式值的类型不一致，按前者为准，自动转换------函数调用转换
  - void型函数

例 无返回值函数

```
void swap( )
{
    cout<<“Results: ”;
}
```

# 例: 函数返回值类型转换

```
#include <iostream>
using namespace std;
int max(float x, float y)
{   float z;
    z=x>y?x:y;
    return(z);
}
int main()
{   float a,b;
    int c;
    cin>>a>>b;
    c=max(a,b);
    cout<<"Max is " <<c<<endl;
    return 0;
}
```

运行结果：2.6 , 8.9
Max is 8

# 4.4 函数的调用

- 函数调用一般形式

  函数名([实参表列]);

  如： c=max(a,b);

- 说明：
  - 实参与形参个数相等，类型一致，按顺序一一对应
  - 实参表求值顺序，因系统而定（Visual C++，Turbo C 自右向左）

- 函数调用方式
  - 函数语句：

    例　printstar( );
  - 函数表达式：

    例　m=max(a,b)*2;
  - 函数参数：

    例 m=max(a,max(b,c));

# 函数说明

- 对被调用函数要求:
  - 必须是已存在的函数
  - 库函数: #include <*>
  - 用户自定义函数: 函数类型说明
- 函数说明
  - 一般形式: 函数类型 函数名(形参类型 [形参名],.....);
  - 作用: 告诉编译系统函数类型、参数个数及类型，以便检验
  - 函数定义与函数说明不同
  - 函数说明位置: 程序的数据说明部分（函数内或外）
  - 下列情况下，可不作函数说明
    - 被调用函数定义出现在主调函数之前

# 例 函数说明举例

```
#include<iostream>
using namespace std;
int  main()
{   float add(float x,float y);  /*function declaration*/
    float a,b,c;
    cin>>a>>b;
    c=add(a,b);
    cout<<"sum is “<<c<<endl;
    return 0;
}
float add(float x, float y)
{   float z;
    z=x+y;
    return(z);
}
```

# 例 函数说明举例

```
#include<iostream>
using namespace std;
float add(float x, float y)
{   float z;
    z=x+y;
    return(z);
}
int  main()
{   float a,b,c;
    cin>>a>>b;
    c=add(a,b);
    cout<<"sum is “<<c<<endl;
    return 0;
}
```

被调函数出现在主调函数之前，不必函数说明

```
#include <iostream>
using namespace std;
long   sum(int a, int b);
long   factorial(int n);
void main()
 {    int n1,n2;        long a;
      long   sum(int a, int b);
       cin>>n1>>n2;
       a=sum(n1,n2);
       cout<<a; }
long sum(int a,int b)
{    long   factorial(int n);
     long c1,c2;
     c1=factorial(a);
     c2=factorial(b);
    return(c1+c2);
}
```

文件包含编译预处理命令

函数类型说明

实参

函数调用

函数定义

形参

函数调用

函数返回值

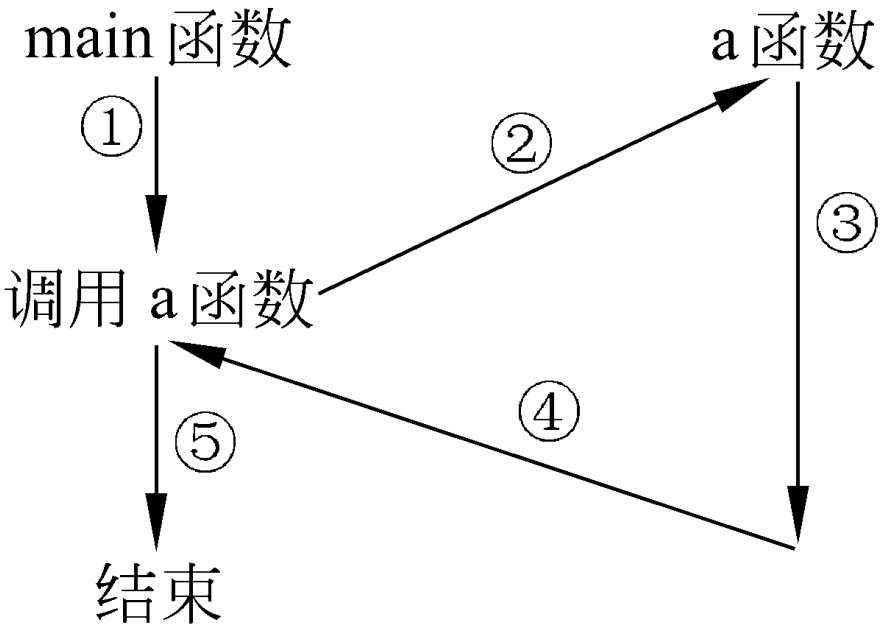
```
long factorial(int n)
{    long rtn=1;
     int i;
     for(i=1;i<=n;i++)
          rtn*=i;
     return(rtn);
}
```

# 作业：

$P_{122}$ 2, 3

# 4.5 内置函数

调用函数时需要一定的时间和空间的开销。下图表示函数调用的过程：



## 内置函数(内联函数)

- 在编译时将所调用函数的代码直接嵌入到主调函数中，而不是将流程转出去。
- 指定内置函数，只需在函数首行的左端加一个关键字

```
inline int max(int,int, int);
```

例4.4 函数指定为内置函数。

```
#include <iostream>
using namespace std;
inline int max(int,int, int);      //函数声明，注意左端有inline
int main( )
{  int i=10,j=20,k=30,m;
   m=max(i,j,k);
   cout<<"max="<<m<<endl;
   return 0;
}
inline int max(int a,int b,int c)      //定义max为内置函数
{  if(b>a) a=b;                        //求a,b,c中的最大者
    if(c>a) a=c;
    return a;
}
```

实参代替形参，被置换成

```
if (j>i) i=j;
if(k>i) i=k;
m=i;
```

**注意：inline可以写在声明函数或定义函数时**

- 优点：内置函数节省运行时间
- 缺点：增加了目标程序的长度。
- 对内置函数的要求：
  - 规模很小，一般为5个语句以下
  - 使用频繁的函数
  - 不包括复杂的控制语句，如循环语句和switch语句

# 4.6 函数的重载

- 函数的重载:

  一组概念相同，处理对象（参数）不同的过程，出于方便编程的目的，用同一个函数名字来命名的技术称为**函数重载**

- 重载是不同的函数，以参数的类型，个数和顺序来分辨

- 函数体可以相同也可以不同

例4.5 求3个数中最大的数（考虑整数、双精度数、长整数）。

```
#include <iostream>
using namespace std;
int  main( )
{ int max(int a,int b,int c);                    //函数声明
  double max(double a,double b,double c);   //函数声明
  long max(long a,long b,long c);              //函数声明
  int i1,i2,i3,i;
  cin>>i1>>i2>>i3;
  i=max(i1,i2,i3);                                    //
  cout<<"i_max="<<i<<endl;
  double d1,d2,d3,d;
  cin>>d1>>d2>>d3;
  d=max(d1,d2,d3);
  cout<<"d_max="<<d<<endl;
  long g1,g2,g3,g;
  cin>>g1>>g2>>g3;
  g=max(g1,g2,g3);
  cout<<"g_max="<<g<<endl;
  return 0;   }
```

```
int max(int a,int b,int c)
{  if(b>a) a=b;
   if(c>a) a=c;
   return a;
}
double max(double a,double b,double c)
{  if(b>a) a=b;
   if(c>a) a=c;
   return a;
}
long max(long a,long b,long c)
{  if(b>a) a=b;
   if(c>a) a=c;
   return a;
}
```

运行情况如下：

```
185   -76   567↙
i_max=567
56.87  90.23  -3214.78↙
d_max=90.23
67854  -912456  673456↙
g_max=673456
```

# 例4.6 参数个数不同。

```
#include <iostream>
using namespace std;
int main( )
{
    int max(int a,int b,int c);
    int max(int a,int b);
    int a=8,b=-12,c=27;
    cout<<"max(a,b,c)="<<max(a,b,c)<<endl;
    cout<<"max(a,b)="<<max(a,b)<<endl;
    return 0;
}
```

```
int max(int a,int b,int c)
{
    if(b>a) a=b;
    if(c>a) a=c;
    return a;
}
int max(int a,int b)
{
    if(a>b) return a;
    else return b;
}
```

# 4.7 函数模板

◆定义函数模板的形式:

template < typename T> 或 template <class T>

通用函数定义 通用函数定义

●在调用函数时系统会根据实参的类型来取代模板中的虚拟类型，从而实现了不同函数的功能。

◆注意:

它只适用于函数的参数个数相同而类型不同，且函数体相同的情况，如果参数的个数不同，则不能用函数模板。

例4.7 将例4.5程序改为通过函数模板来实现。

```
#include <iostream>
using namespace std;
template<typename T>
T max(T a,T b,T c)
{
   if(b>a) a=b;
   if(c>a) a=c;
   return a;
}
```

```
int max(int a,int b,int c)
{if(b>a) a=b;
 if(c>a) a=c;
 return a;
}
```

```
int main( )
{int i1=185,i2=-76,i3=567,i;
  double d1=56.87,d2=90.23,d3=-3214.78,d;
  long g1=67854,g2=-912456,g3=673456,g;
  i=max(i1,i2,i3);
  d=max(d1,d2,d3);
  g=max(g1,g2,g3);
  cout<<"i_max="<<i<<endl;
  cout<<"f_max="<<f<<endl;
  cout<<"g_max="<<g<<endl;
  return 0;
}
```

# 4.8 有默认参数的函数

◆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( )
{   add(10,20); //10+20
    add(10);  //10+6
    add();  //5+6
}
```

◆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
```

例4.8 求2个或3个正整数的最大数，用带有默认参数的函数实现。

```
#include <iostream>
using namespace std;
int main( )
{   int max(int a, int b, int c=0);
    int a,b,c;
    cin>>a>>b>>c;
    cout<<"max(a,b,c)="<<max(a,b,c)<<endl;
    cout<<"max(a,b)="<<max(a,b)<<endl;   //c取默认值
    return 0;
}
```

```
int max(int a,int b,int c)
{
    if(b>a) a=b;
    if(c>a) a=c;
    return a;
}
```

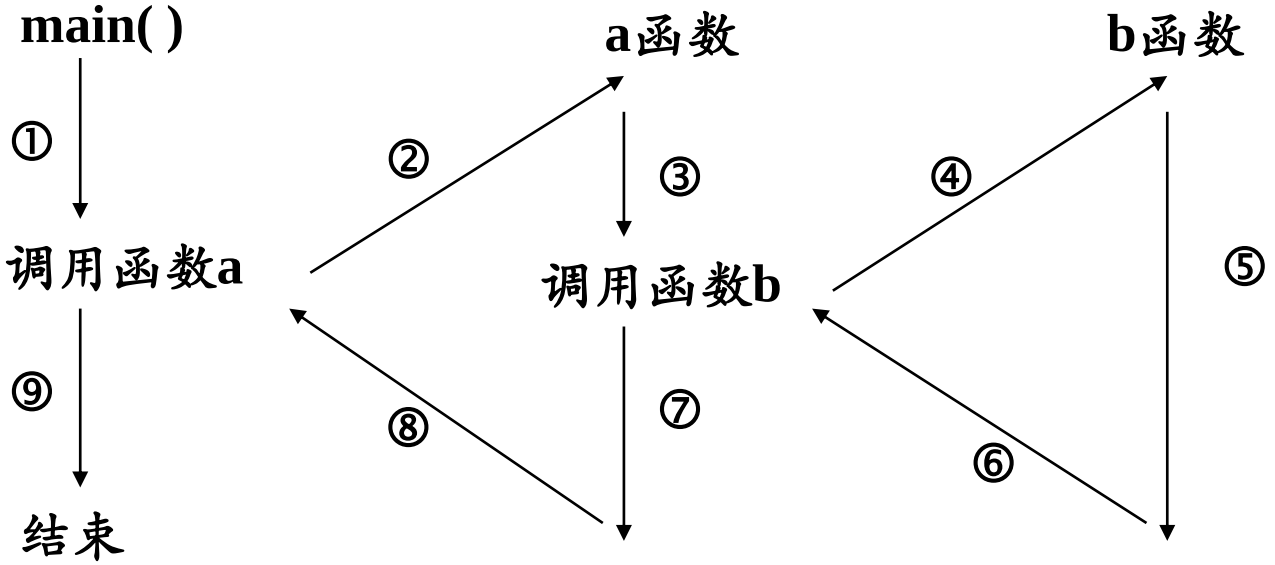
运行情况如下：

```
14  -56  135↙
max(a,b,c)=135
max(a,b)=14
```

# 4.9 函数的嵌套调用

## ◆嵌套调用

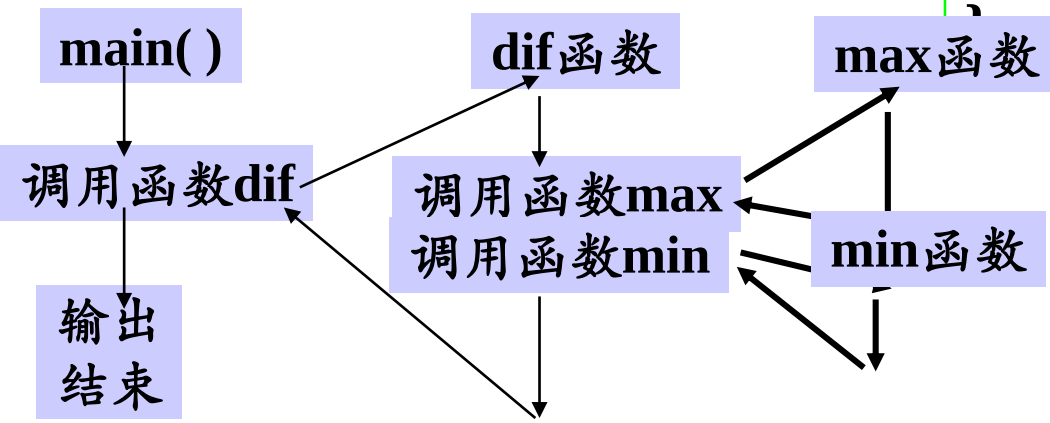
C＋＋规定：函数定义不可嵌套，但可以嵌套调用函数



# 例　求三个数中最大数和最小数的差值

```
#include <iostream>
using namespace std;
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);
}
void main()
 {  int a,b,c,d;
    cin>>a>>b>>c;
    d=dif(a,b,c);
    cout<<"Max-Min:“<<d<<endl;
}
```

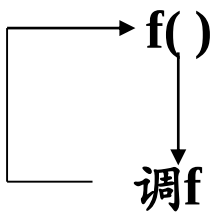


# 4.10 函数的递归调用

- 定义：函数直接或间接的调用自身叫函数的递归调用

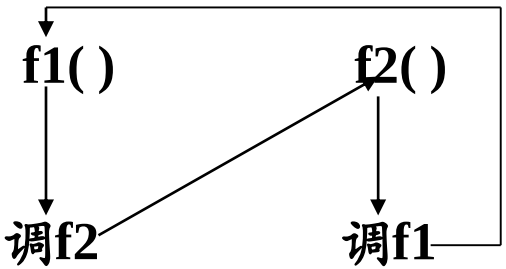
```
int  f(int x)
{    int y,z;
      ……
     z=f(y);
      ……

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);
}
```



## 说明

- C++编译系统对递归函数的自调用次数没有限制
- 每调用函数一次，在内存堆栈区分配空间，用于存放函数变量、返回值等信息，所以递归次数过多，可能引起堆栈溢出

例： 求n的阶乘

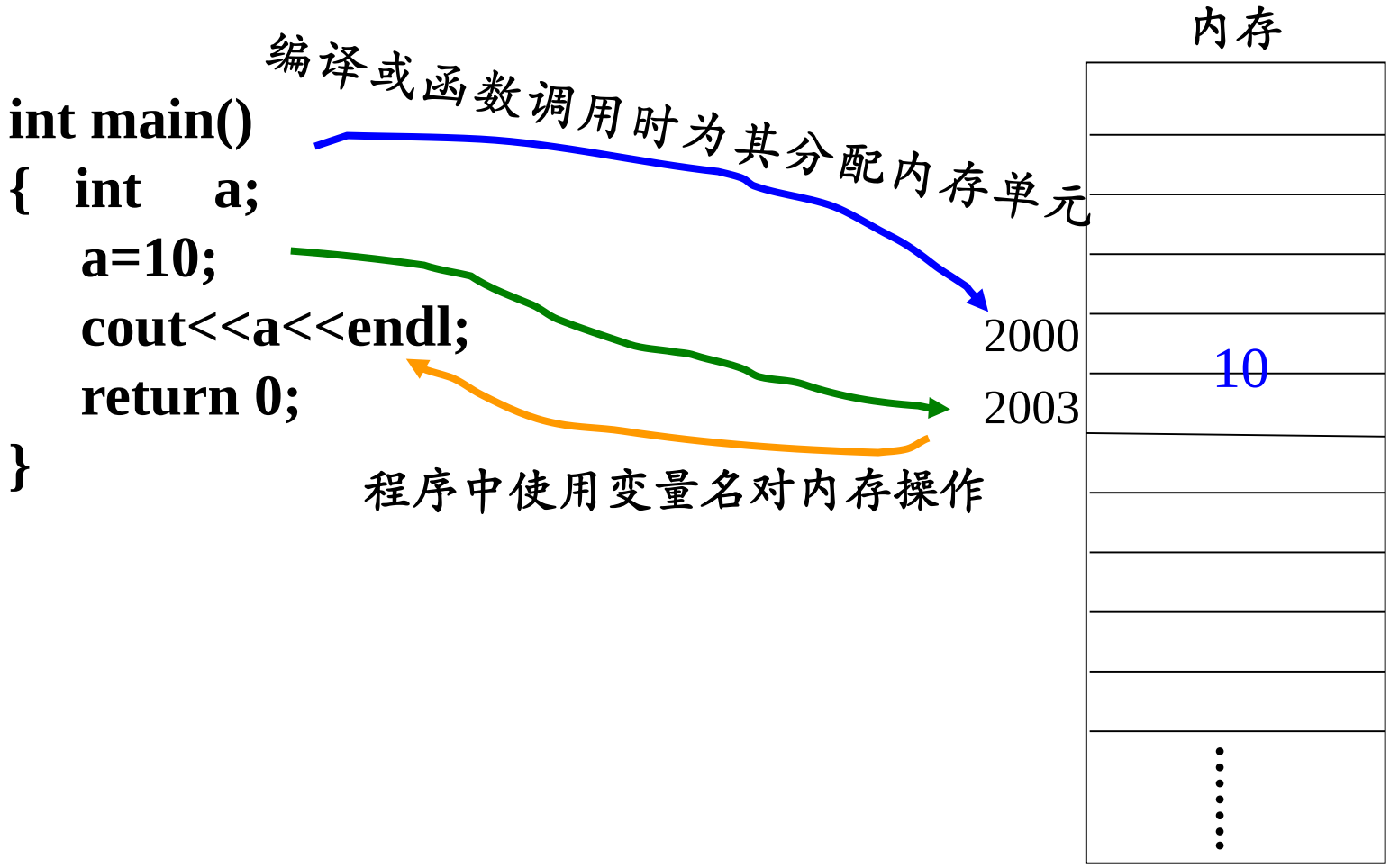
$$n!=\begin{cases}1 & (n=0,1)\\ n\cdot(n-1)! & (n>1)\end{cases}$$

```
#include <iostream>
using namespace std;
int fac(int n)
{   int f;
    if(n<0)  cout<<"n<0,data error!“<<endl;
    else if (n==0||n==1)  f=1;
    else f=fac(n-1)*n;
    return(f);
}
int main()
{   int n, y;
    cout<<"Input a integer number:“<<endl;
    cin>>n;
    y=fac(n);
    cout<<n<<"! =“<<y<<endl;
    return 0;
}
```

# 4.12 变量的存储类别

## ◆概述

### 变量是对程序中数据的存储空间的抽象



# 概述

- 变量是对程序中数据的存储空间的抽象
- 变量的属性
  - 数据类型：变量所持有的数据的性质（操作属性）
  - 存储属性
    - 存储类型：寄存器、静态存储区、动态存储区
    - 生存期：变量在某一时刻存在----静态变量与动态变量
    - 作用域：变量在某区域内有效----局部变量与全局变量

- 变量的存储类型
  - auto -----自动型
  - register-----寄存器型
  - static ------静态型
  - extern -----外部型
- 变量定义格式: [存储类型] 数据类型 变量表;

```
如:  int  sum;
     auto  int  a,b,c;
     register  int  i;
     static  float  x,y;
```

# 局部变量与全局变量

## 局部变量---内部变量

- 定义：在函数内定义，只在本函数内有效
- 说明：
  - main中定义的变量只在main中有效
  - 不同函数中同名变量，占不同内存单元
  - 形参属于局部变量
  - 可在复合语句中定义有效的变量
  - 局部变量可用存储类型：auto　register　static（默认为auto）

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

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

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

a,b,c有效

x,y,i,j有效

m,n有效

## 例 不同函数中同名变量

```
#include <iostream>
using  namespace std;
int main()
{   int a,b;
    int sub( );
    a=3;    b=4;
    cout<<"main:" <<a<<“,”<<b<<endl;
    sub(  );
    cout<<"main:" <<a<<“,”<<b<<endl;
    return 0;
}
int sub( )
{   int a,b;    a=6;    b=7;
    cout<<" sub:" <<a<<“,”<<b<<endl;
}
```

运行结果:

```
main:a=3,b=4
sub:a=6,b=7
main:a=3,b=4
```

## 全局变量---外部变量

- 定义：在函数外定义，可为本文件所有函数共用
- 有效范围：从定义变量的位置开始到本源文件结束，及有extern说明的其它源文件

```
float   max,min;
float  average(float   array[],  int  n)
{   int i;   float  sum=array[0];
    max=min=array[0];
    for(i=1;i<n;i++)
    {   if(array[i]>max)  max=array[i];
        else if(array[i]<min)  min=array[i];
        sum+=array[i];
    }
    return(sum/n);
}
void main()
{   int i;   float ave,score[10];
    /*Input  */
    ave=average(score,10);
    cout<<"max=" <<max<<"min="<<
            min<<"average="<<ave<<endl;
}
```

max

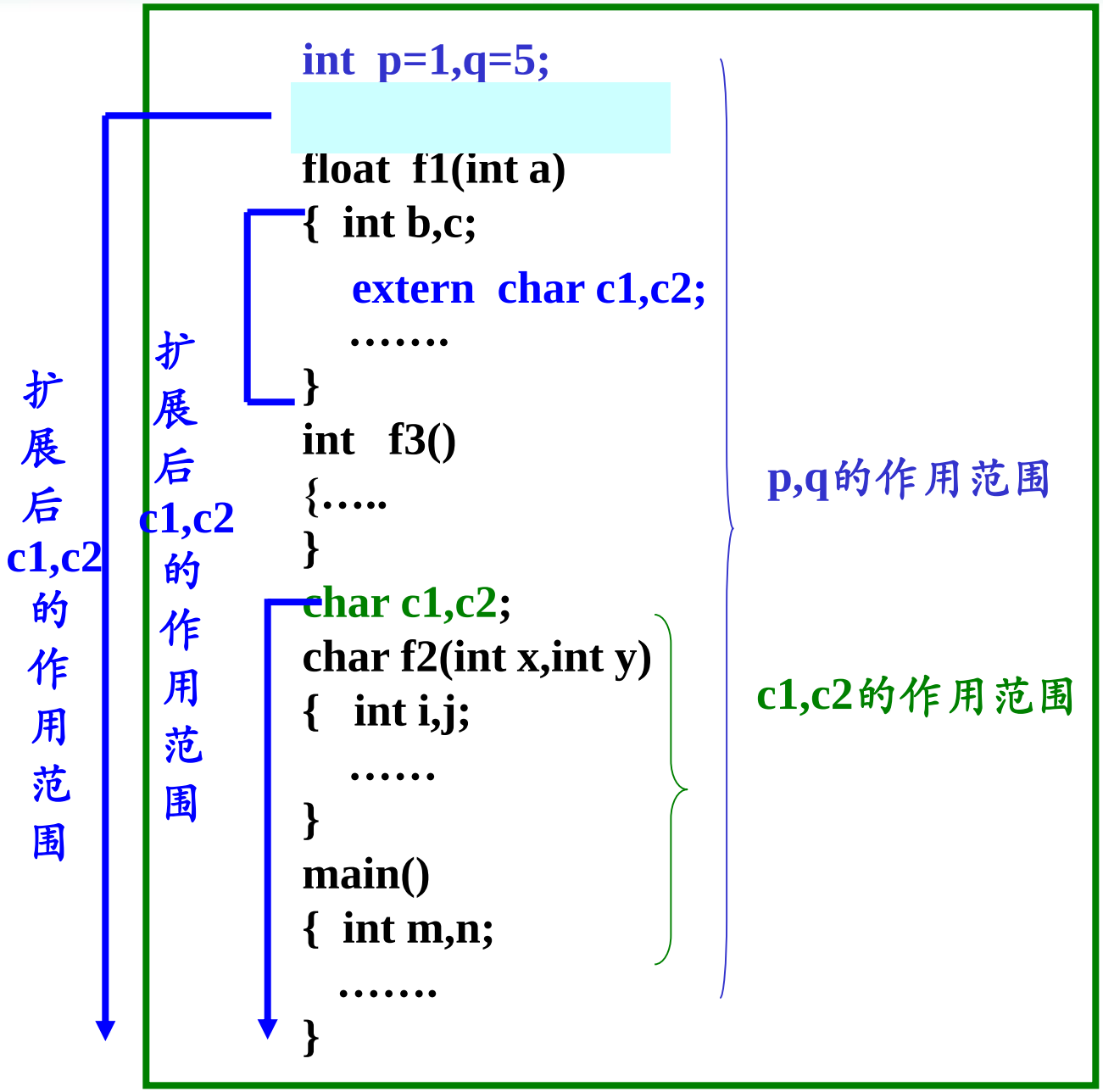
min

作用域

## 全局变量---外部变量

- 定义：在函数外定义，可为本文件所有函数共用
- 有效范围：从定义变量的位置开始到本源文件结束，及有extern说明的其它源文件
- 外部变量说明： extern 数据类型 变量表;
- 外部变量定义与外部变量说明不同

| | 定义 | 说明 |
|---|---|---|
| 次数: | 只能1次 | 可说明多次 |
| 位置: | 所有函数之外 | 函数内或函数外 |
| 分配内存: | 分配内存,可初始化 | 不分配内存,不可初始化 |


int p=1,q=5;
float f1(int a)
{ int b,c;
extern char c1,c2;
…….
}
int f3()
{…..
}
char c1,c2;
char f2(int x,int y)
{ int i,j;
……
}
main()
{ int m,n;
…….
}
扩展后c1,c2的作用范围
扩展后c1,c2的作用范围
p,q的作用范围
c1,c2的作用范围

## 例 外部变量定义与说明

```
#include<iostream>
using namespace std;
int max(int  x, int y)
{  int z;
   z=x>y?x:y;
   return(z);
}
void main()
{  extern int a,b;
   cout<<"max="<<max(a,b)<<endl;
}
int a=13,b=-8;
```

运行结果：max=13

- 若外部变量与局部变量同名，则外部变量被屏蔽

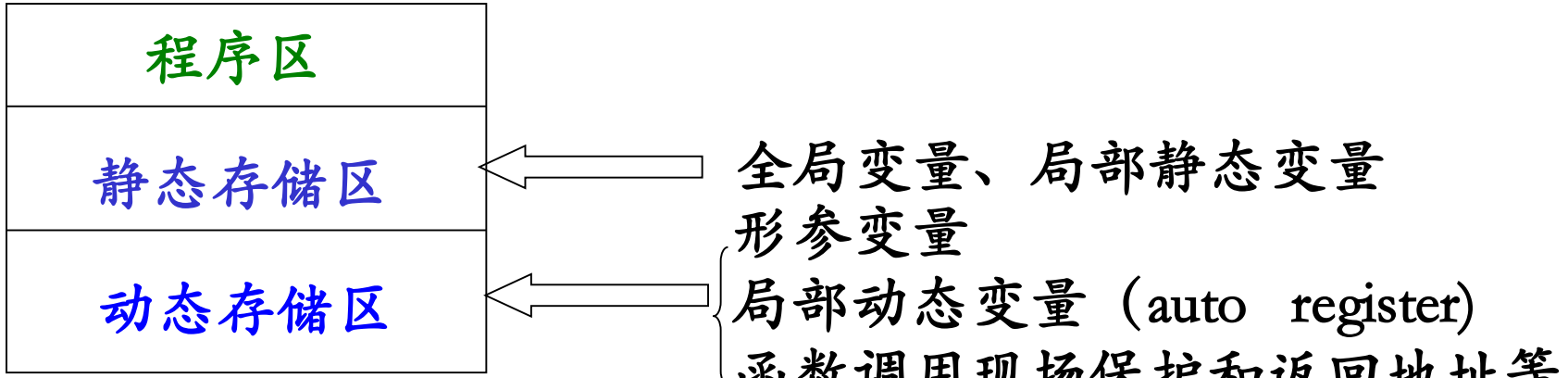
例 外部变量与局部变量

```
#include<iostream>
using namespace std;
int a=3,b=5;
max(int  a, int b)
{   int c;
    c=a>b?a:b;
    return(c);
}
int main()
{   int a=8;
     cout<<"max="<<max(a,b)<<endl;
     return 0;
}
```

运行结果：max=8

# 动态变量与静态变量

- 存储方式
  - 静态存储：程序运行期间分配固定存储空间
  - 动态存储：程序运行期间根据需要动态分配存储空间
- 内存用户区
- 生存期
  - 静态变量:从程序开始执行到程序结束
  - 动态变量:从包含该变量定义的函数开始执行至函数执行结束



# ◆变量存储类型

| | 局部变量 | | | 外部变量 | |
|---|---|---|---|---|---|
| 存储类别 | auto | register | 局部static | 外部static | 外部 |
| 存储方式 | 动态 | | 静态 | | |
| 存储区 | 动态区 | 寄存器 | 静态存储区 | | |
| 生存期 | 函数调用开始至结束 | | 程序整个运行期间 | | |
| 作用域 | 定义变量的函数或复合语句内 | | | 本文件 | 其它文件 |
| 赋初值 | 每次函数调用时 | | 编译时赋初值，只赋一次 | | |
| 未赋初值 | 不确定 | | 自动赋初值0或空字符 | | |

注：

- 局部变量默认为auto型
- register型变量个数受限,且不能为long, double, float型
- 局部static变量具有全局寿命和局部可见性
- 局部static变量具有可继承性
- extern不是变量定义,可扩展外部变量作用域

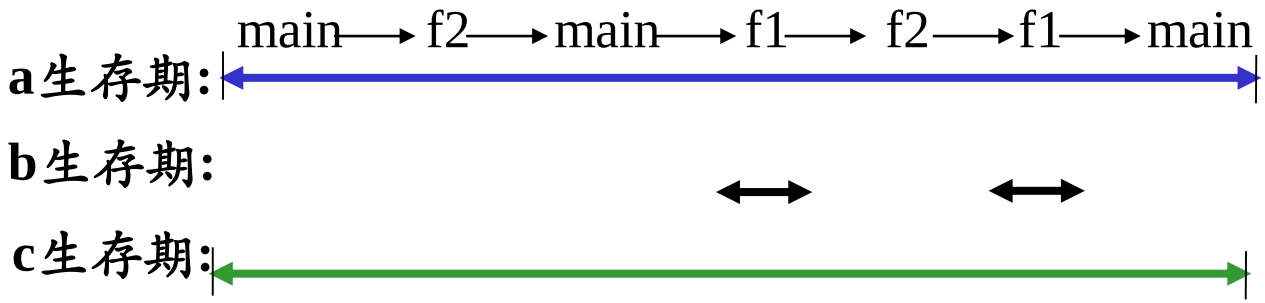
例　文件file1.c

```
int a;
main( )
{   .......
    .......
      f2;
    .......
      f1;
    .......
}
f1( )
{   auto int b;
    ..........
     f2;
    ........
}
f2( )
{   static int c;
    ..........
}
```

a作用域

b作用域

c作用域



## 例 局部静态变量值具有可继承性

```
#include<iostream>
using namespace std;
int main()
{   void  increment(void);
    increment();
    increment();
    increment();
    return 0;
}
void  increment(void)
{   int x=0;
    x++;
    cout<<x<<endl;
}
```

运行结果：1
1
1

```
#include<iostream>
using namespace std;
int main()
{   void  increment(void);
    increment();
    increment();
    increment();
    return 0;
}
void  increment(void)
{   static int x=0;
    x++;
    cout<<x<<endl;
}
```

运行结果：1
2
3

## 例 引用其它文件中的外部变量

```
int global;
extern float x;
main()
{   int local;
        .
        .
        .
}
```

file1.c

```
extern int global;
static int number;
func2()
{       .
        .
        .
}
```

file2.c

```
float x;
static int number;
func3()
{  extern int global;
        .
        .
        .
}
```

file3.c

# 例 引用其它文件中的变量，输出a×b和a的m次方

Ch7.cpp

```
#include <iostream>
using namespace std;
int a;
void main()
{  int power(int  n);
   int b=3,c,d,m;
   cout<<“Enter a and m:\n”;
   cin>>a>>m;
   c=a*b;
   cout<< a <<"*" <<b<<"="<<c<<endl;
   d=power(m);
   cout<<a<<"**" <<m<<" =" <<d<<endl;
}
```

Ch8.cpp

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

运行结果:

```
Enter a and m:
2,3
2*3=6
2**3=8
```

# 4.15 内部函数和外部函数

- 内部函数
  - 函数只能被本文件中其他函数所调用

内部函数定义： static 类型标识符 函数名(形参表)

- 外部函数
  - 函数能被其他文件调用

外部函数定义： [extern] 类型标识符 函数名(形参表)

- 在函数首部的最左端冠以关键字extern
- 如果在定义函数时省略extern,则默认为外部函数

如： extern int fun (int a, int b)

例4.15 输入两个整数，要求输出其中的大者。

file1.cpp（文件 1）

```
#include <iostream>
using namespace std;
int main( )
 {  extern int max(int,int);
    int a,b;
   cin>>a>>b;
   cout<<max(a,b)<<endl;
   return 0;
 }
```

file2.cpp（文件2）

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

运行情况如下：

```
7 -34↙
7
```

在计算机上运行一个含多文件的程序时，需要建立一个项目文件(project file)，在该项目文件中包含程序的各个文件。

# 4.16 预处理命令

- 作用： 对源程序编译之前做一些处理,生成扩展C源程序
- 种类
  - 宏定义 #define
  - 文件包含 #include
  - 条件编译 #if--#else--#endif等
- 格式:
  - “#”开头
  - 占单独书写行
  - 语句尾不加分号

## #include 包含指令

- 将一个源文件嵌入到当前源文件中该点处。
- #include<文件名>
  - 按标准方式搜索，文件位于C++系统目录的include子目录
- #include"文件名"
  - 首先在当前目录中搜索，若没有，再按标准方式搜索。

## #define 宏定义指令

- 定义符号常量，很多情况下已被const定义语句取代。
- 定义带参数宏，已被内联函数取代。

## #undef

- 删除由#define定义的宏，使之不再起作用。

## ◆ 条件编译指令#if 和 #endif

```
#if 常量表达式
    //当“常量表达式”非零时编译
    程序正文
#endif
……
```