第2讲

C与C++的差异

主讲人：赵文彬

本次课主要内容

- 程序基本框架
- 输入输出
- 动态数组
- 引用
- 函数
- string

动态数组

- C语言
  - 申请
    - malloc
    - calloc
    - realloc
  - 释放
    - free
- C++
  - 申请
    - new

    new 类型(初值)　变量
    new 类型[长度]　数组
  - 释放
    - delete

    delete 指针变量　（对变量）
    delete []指针变量　（对数组）

动态数组

- 举例
  - int * a = new int
    - 开辟一个存放整数的存储空间，返回一个指向该存储空间的地址（即指针）
  - int *a = new int(100)
    - 开辟一个存放一个整数的空间，并指定初值为100，返回一个指向该存储空间的地址

动态数组

- 举例
  - char * pt=new char[10]
    - 开辟一个存放字符数组（包括10个元素）的空间，返回字符数组首元素的地址
    - delete []pt;
  - new int[5][4]
    - 开辟一个存放二维整数型（大小为5*4）的空间，返回首地址
    - 很少这么用！

动态数组

- 举例
  - float *p = new float(3.14159)
  - delete p;
- 注意
  - 用new分配数组空间时不能指定初值。
  - 若分配不成功，则返回一个空指针NULL

逆序输出

```
#include <iostream>
#include <new>
using namespace std;
int main ()
{   int i,n;
    int * p;
   cout << "How many numbers would you like to type? ";
   cin >> n;
   p= new  int[n]; //动态申请空间
   if (p == NULL)//申请是否成功判断
  {  cout << "Error: memory could not be allocated";
      return 1;  }
  for (i = 0; i < n; i++)       cin >> p[i];
   cout << "You have entered in inverse ordering: ";
   for (i = n - 1; i >= 0; i--)    cout << p[i] << ", ";
   delete[ ] p; //释放动态申请空间
   return 0; }
```

动态数组

```
#include <iostream>
using namespace std;
void main()
{   int **a;
    int row,col,i,j;
    cout << "请输出矩阵的行数和列数： ";
    cin >> row >> col;
    a = new int* [row] ; //申请空间
    for(i = 0; i < row;i++)        *(a + i) = new int[col];
    cout << "请输出" << row << "行" << col << "列矩阵： " << endl
    for(i = 0; i < row; i++)
        for(j = 0; j < col; j++)
          cin >> a[i][j];
```

动态数组

```
    cout << "矩阵的转置为：" << endl;
    for(j = 0; j < col; j++)
    {
        for(i = 0; i < row; i++)
          cout << a[i][j] << " ";
        cout << endl;
    }
    //释放空间
    for(i = 0; i < row; i++)
        delete [] a[i];
    delete [] a;
}
```

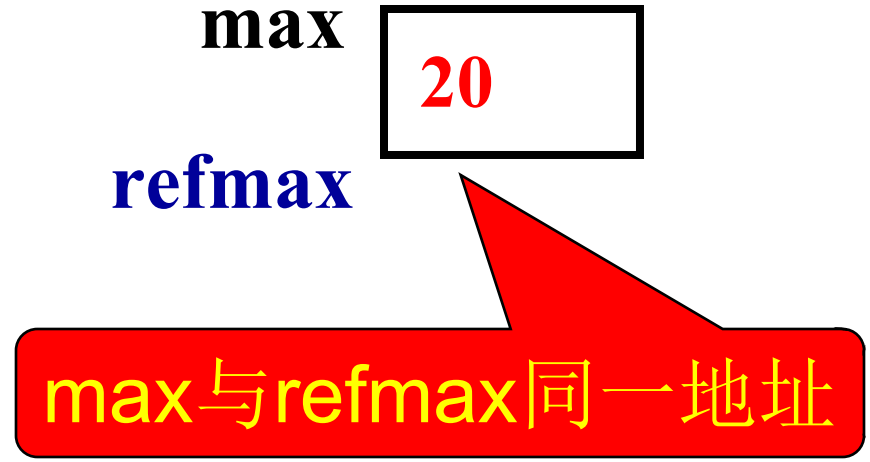
引用

- 对变量起另外一个名字 (外号)，这个名字称为该变量的引用。

  <类型> &<引用变量名> = <原变量名>;

  - 其中原变量名必须是一个已定义过的变量

```
int   max ;
int  &refmax=max;
max=5 ;
refmax=10;
refmax=max+refmax;
```



引用

- 说明
  - 引用在定义的时候要初始化。

```
int  &refmax;
```

错误，没有具体的引用对象

```
int  &refmax=max;
```

max是已定义过的变量

  - 对引用的操作就是对被引用的变量的操作。
  - 引用类型变量的初始化值不能是一个常数。

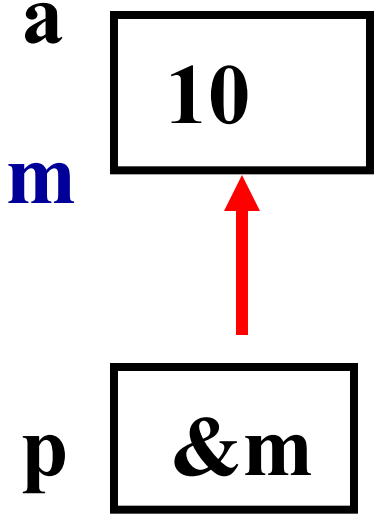
```
如：int  &ref1 = 5;    // 是错误的。
int  &ref=i;
```

引用

- 说明
  - 引用同变量一样有地址，可以对其地址进行操作，即将其地址赋给一指针。

```
int   a, *p;
int  &m=a;
```

&是变量的引用



&是变量的地址

```
p=&m;
*p=10;
```

引用

- 可以用动态分配的内存空间来初始化一个引用变量。

```
float  &reff = * new float ; //用new开辟一个空间，取一个别名reff
 reff= 200;   //给空间赋值
 cout << reff ;        //输出200
 delete &reff;         //收回这个空间
```

这个空间只有别名，但程序可以引用到。

```
float  *p,  a;
p=new  float;
```

```
float  a=* new  float;
```

错误！

引用

- 指针与引用的区别：
  - 指针是通过地址间接访问某个变量，而引用是通过别名直接访问某个变量。
  - 引用必须初始化，而一旦被初始化后不得再作为其它变量的别名。

引用

当&a的前面有类型符时（如int &a），它必然是对引用的声明；如果前面无类型符（如cout<<&a），则是取变量的地址。

引用

- 以下的声明是非法的
  - 建立数组的引用 int & a[9];
  - 建立指向引用的指针 int & *p;
  - 建立引用的引用 int & &px;

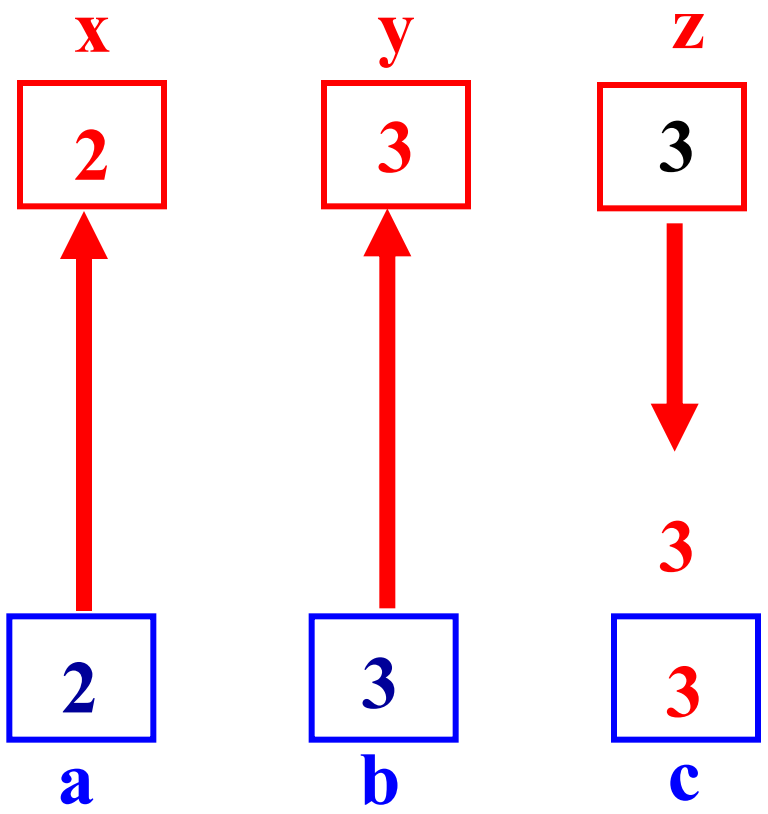
函数

- 引用传递

- 之前学过的参数传递方式
  - 值传递
  - 地址传递

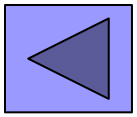
值传递

```
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;
}
```



```
#include <iostream>
using namespace std;
void duplicate (int a, int b, int c)
{
    a *= 2;
    b *= 2;
    c *= 2;
}
void main ()
{
    int x = 1, y = 3, z = 7;
    duplicate (x, y, z);
    cout << "x=" << x << ", y=" << y << ", z=" << z;
}
```

x = 1, y = 3, z = 7



数组名作为函数参数

```
void main ()
{  float aver_score (float a[10]);
   float x[10],aver;
   cout<<“Input 10 scores:\n”;
   for (int i=0;i<10;i++)
      cin>>x[i];
   aver=aver_score(x);
   …
}
```

地址传递

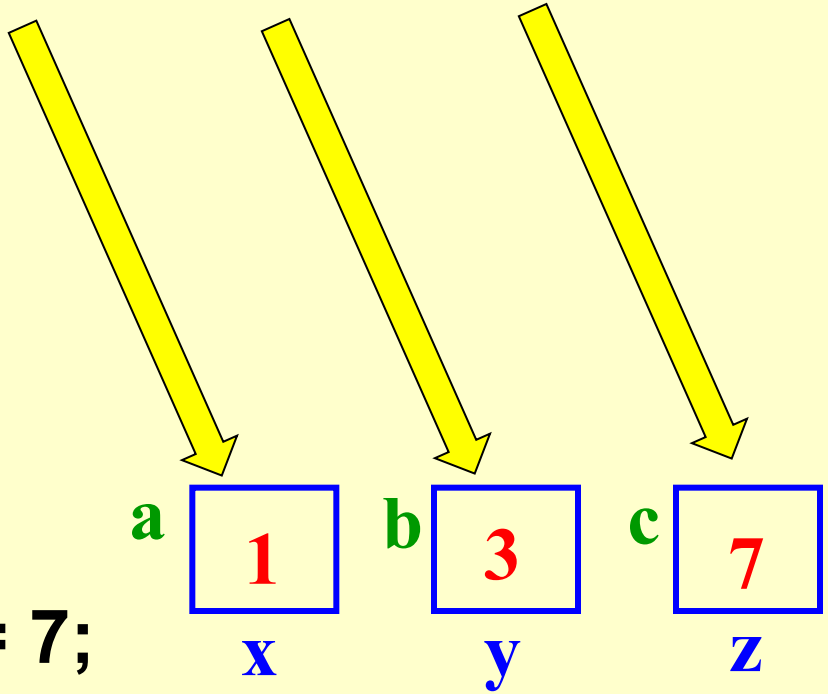
```
float aver_score (float a[10])
{  float sum=0,aver;
   for (int i=0; i<10;i++)
      sum+=a[i];
   aver=sum/10;
   return aver;
}
```

```
#include <iostream>
using namespace std;
#define N 5
void maxAndMin(int a[], int *max_p, int *min_p)
{ *max_p = a[0];
  *min_p = a[0];
  for(int i = 1; i < N; i++)
  {   if(a[i] > *max_p)    *max_p = a[i];
      if(a[i] < *min_p)    *min_p = a[i];}}
void main( )
{  int a[N],i,max,min;
   cout << "Please inter " << N << " integers:" << endl;
   for(i = 0; i < N; i++)  cin >> a[i];
   maxAndMin(a, &max, &min);
   cout << "The maximum is: " << max << endl << "The
minmum is: " << min << endl;}
```

地址传递

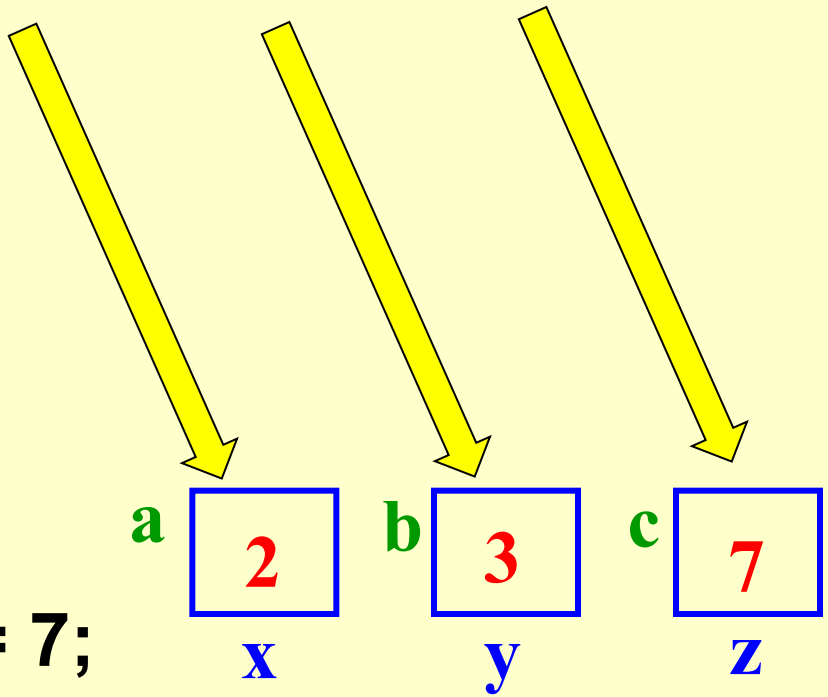
引用传递

```
#include <iostream>
using namespace std;
void duplicate (int& a, int& b, int& c)
{
    a *= 2;
    b *= 2;
    c *= 2;
}
void main ()
{
    int x = 1, y = 3, z = 7;
    duplicate (x, y, z);
    cout << "x=" << x << ", y=" << y << ", z=" << z;
}
```



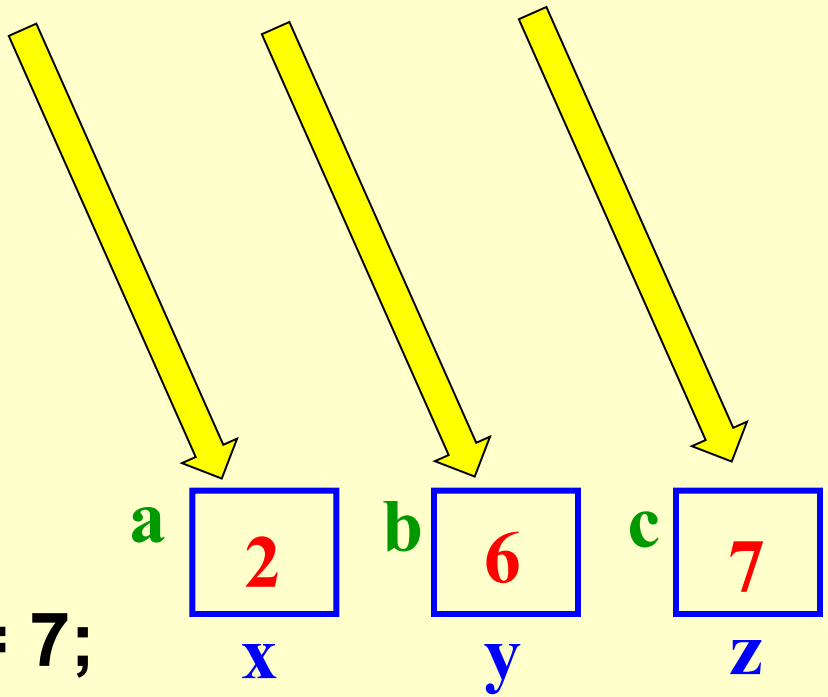
引用传递

```
#include <iostream>
using namespace std;
void duplicate (int& a, int& b, int& c)
{
    a *= 2;
    b *= 2;
    c *= 2;
}
void main ()
{
    int x = 1, y = 3, z = 7;
    duplicate (x, y, z);
    cout << "x=" << x << ", y=" << y << ", z=" << z;
}
```



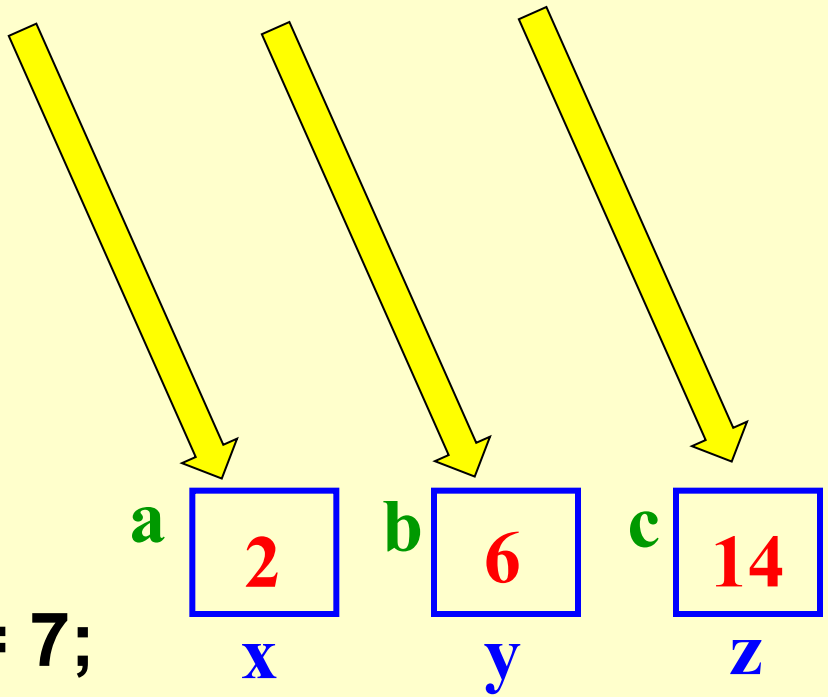
引用传递

```
#include <iostream>
using namespace std;
void duplicate (int& a, int& b, int& c)
{
    a *= 2;
    b *= 2;
    c *= 2;
}
void main ()
{
    int x = 1, y = 3, z = 7;
    duplicate (x, y, z);
    cout << "x=" << x << ", y=" << y << ", z=" << z;
}
```



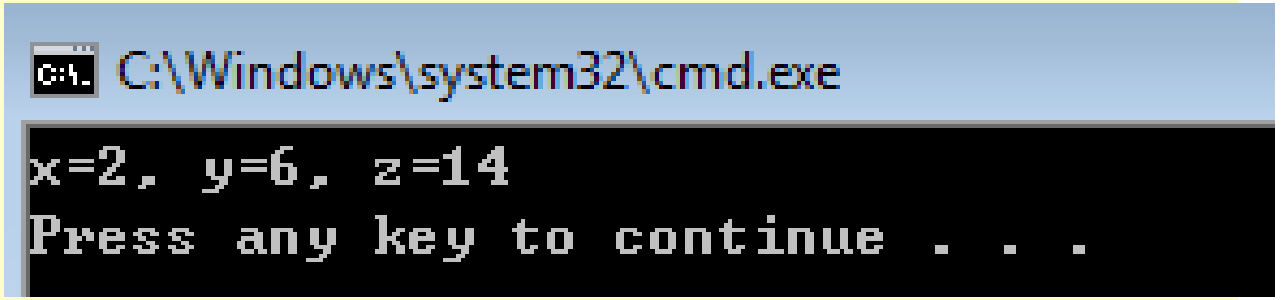
引用传递

```
#include <iostream>
using namespace std;
void duplicate (int& a, int& b, int& c)
{
    a *= 2;
    b *= 2;
    c *= 2;
}
void main ()
{
    int x = 1, y = 3, z = 7;
    duplicate (x, y, z);
    cout << "x=" << x << ", y=" << y << ", z=" << z;
}
```

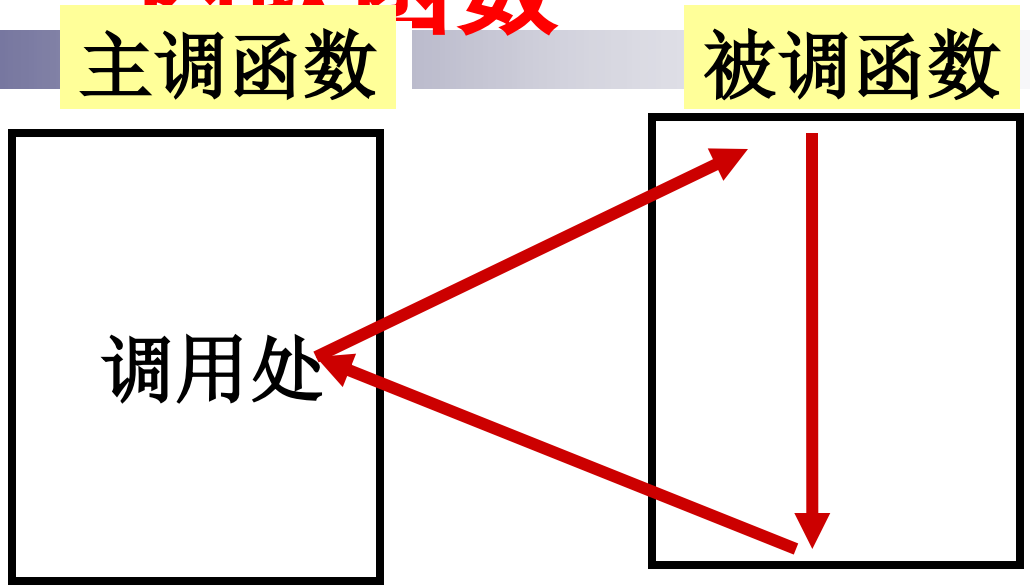


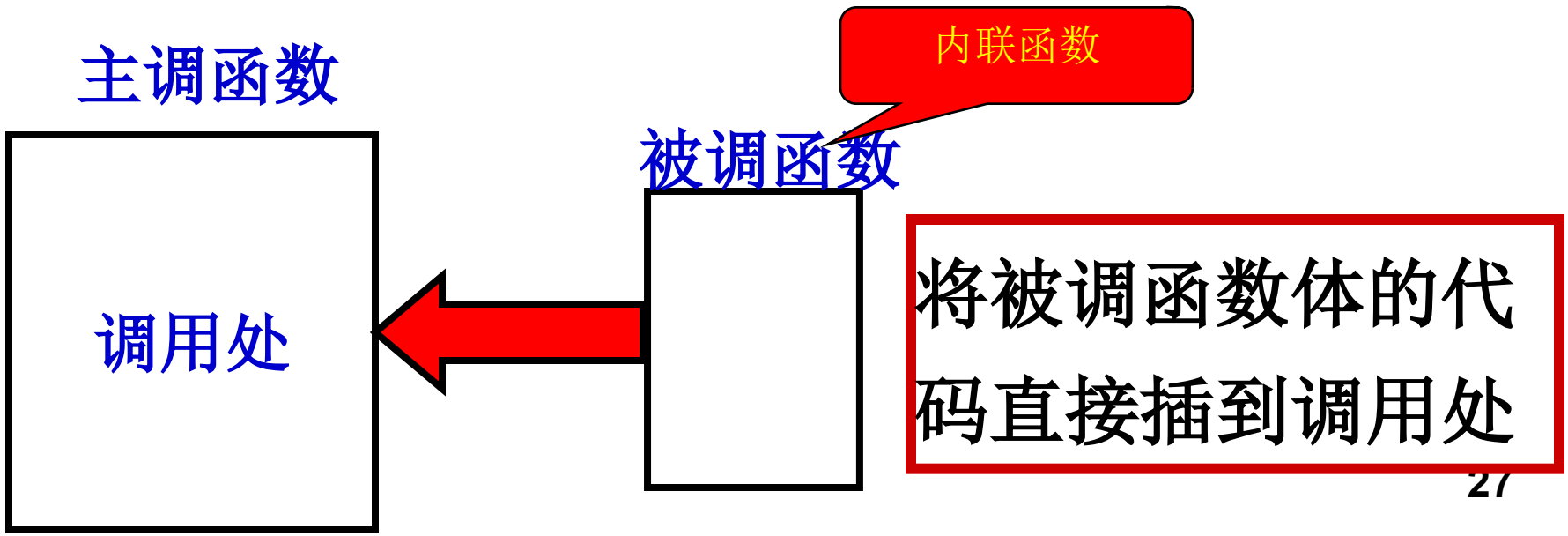
引用传递

```
#include <iostream>
using namespace std;
void duplicate (int& a, int& b, int& c)
{
    a *= 2;
    b *= 2;
    c *= 2;
}
void main ()
{
    int x = 1, y = 3, z = 7;
    duplicate (x, y, z);
    cout << "x=" << x << ", y=" << y << ", z=" << z;
}
```



内联函数





内联函数

- 实质
  - 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）.
- 定义方法
  - 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;
}
```

内联函数

- 注意
  - 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。
  - 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。
  - 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。

内联函数

- 注意
  - 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。
  - 内联函数一般用于调用较少，代码较短，较简单的函数。

具有缺省参数值的函数

- 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。

具有缺省参数值的函数

```
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

具有缺省参数值的函数

```
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

具有缺省参数值的函数

- 注意
  - 不可以靠左边缺省

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

错误！

  - 函数原型说明时可以不加变量名

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

  - 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

函数的重载

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

8

25

函数的重载

- 条件
  - 参数个数不同
  - 参数类型不同
  - 参数顺序不同
- 注意
  - 仅返回值不同时，不能定义为重载函数。

函数的重载

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

float   fun (int a,int b)
{  return   (float) 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

函数的重载

```
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

string

- 字符串变量
  - 定义　string 变量名
  - 初始化
    - string s1(“Hello!”); string s2(s1);
    - string s1(“Hello!”, 2);
    - string s1(“Hello!”, 2, 1);
    - char cstr[ ] = “Hello”; string str(cstr);
    - char cstr[ ] = “Hello”; string str(cstr, 2);
    - string s(4, ‘c’);

string

- 字符串变量的赋值
  - 可以用赋值语句将字符串常量赋给字符串变量。

```
string str1;
str1="hello";
```

```
string str1("hello"),str;
str=str1;
```

```
string word="then";
word[2]='e';
```

string

- 字符串变量的运算
  - +: s + t 将字符串s与t连接成一个新的字符串
  - =: s = t 将字符串t的值赋值给s
  - +=： s += t 等价于s = s + t
  - ==, !=, >, <, >=, <=： 按照字典顺序比较字符串的大小
  - []: s[i] 字符串s的下标为i的元素

Sort of string

```
#include <iostream>
#include <string>
using namespace std;
void main()
{
   string str1,str2,str3;
   cout << "Please input two strings:";
   cin >> str1;
   cin >> str2;
   if(str2 > str1)str3 = str2;
   else      str3 = str1;
   cout << str3 << endl;
}
```

string

- string类型的其它函数
  - unsigned int length() const；//返回字符串中字符的个数
  - unsigned int find(const basic_string &str) const; //查找并返回str在本字符串中出现的位置
  - string substr(unsigned int pos, unsigned int n) const; //取子串，取本字符串中位置pos开始的n个字符，构成新的字符串
  - string & insert(unsigned int p0, const char *s); //将s所指向的字符串插入在本字符串中位置p0之前

string


```
V
{
    cout << "str = " << str << endl;
    cout << "str1 = " << str1 << endl;
    cout << "str2 = " << str2 << endl;
    cout << "The length of str is:" << str.length() << endl;
    cout << str.find(str2) << endl;
    cout << str.substr(9, 2) << endl;
}
```

小　结

- 动态内存的分配
- 引用
- 函数
  - 引用传递
  - 引用类型函数（即返回值为引用类型）
  - 内联函数
  - 形参具有默认值的函数
  - 函数的重载
- string

小　结

- 作业
  - 用C++写如下程序
    - 用引用传递的方式求得一列数（个数由用户指定）中的最大值与最小值。
    - 自定义简单函数，实现函数重载。
    - 自定义有缺省值的函数，在主函数中进行调用测试
    - 利用string类型判断字符串是否对称
    - 利用string类型分离文件名和文件类型