数据结构与算法
DATA STRUCTURE

第二讲 数组，函数，C++的类和指针

信息管理与工程学院

2017 - 2018 第一学期

数组（array）

- 由相同类型的元素（element）的集合所组成的存储结构
- 分配一块连续的内存来存储。
- 利用元素的索引（index）可以计算出该元素对应的储存地址
- 索引小标从0开始,

数组种类

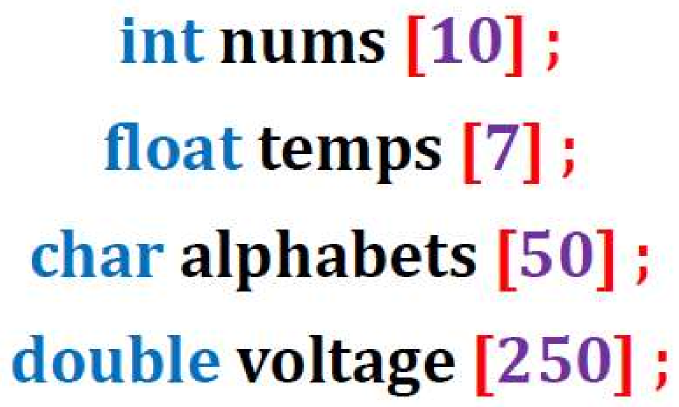
- 一维数组

- 多维数组（包括二维数组）

一维数组声明

```
datatype name [size];
```

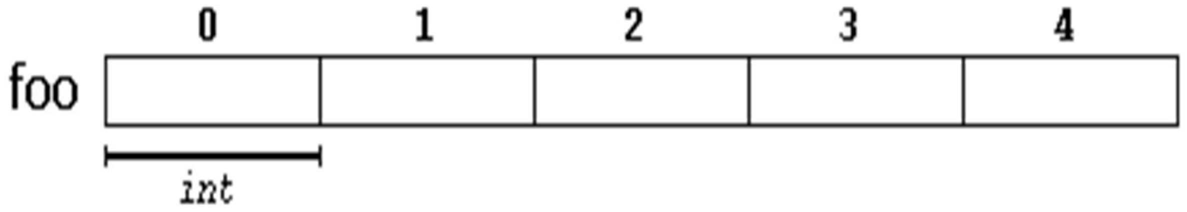
- Datatype就是数据类型，包括int，char，etc。
- Name是标识符，就是程序员自己命名的
- size是数组大小



```
int nums [10] ;
float temps [7] ;
char alphabets [50] ;
double voltage [250] ;
```

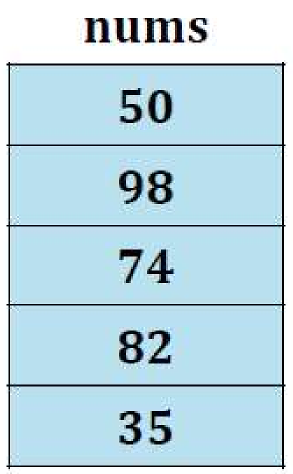
一维数组内存分配

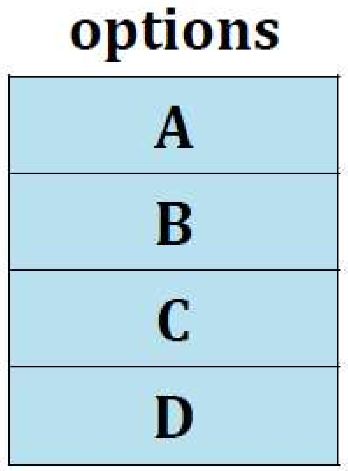
- 比如 int foo[5];



初始化

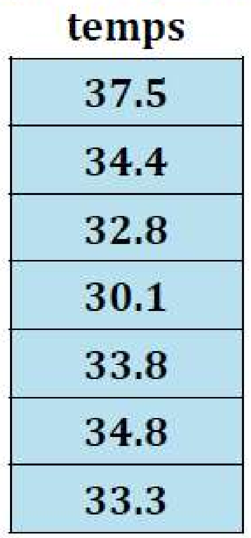
```
int nums [5] = { 50,98,74,82,35};   char options [4] = { 'A' , 'B' , 'C' , 'D' };
```



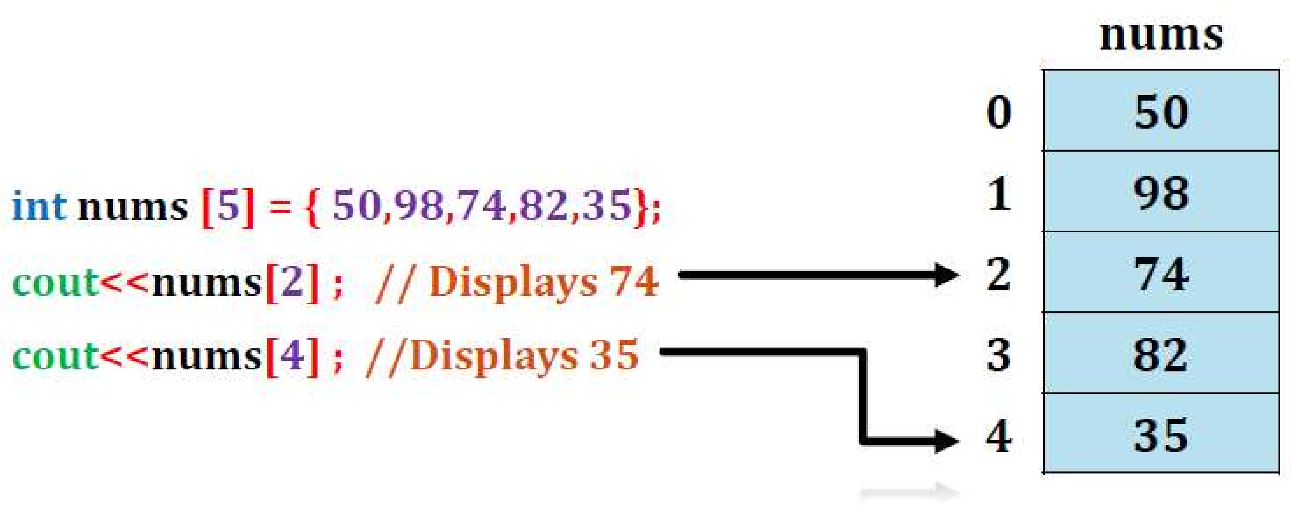


初始化

```
float temps [7] = { 37.5,34.4,32.8,30.1,33.8,34.8,33.3};
```



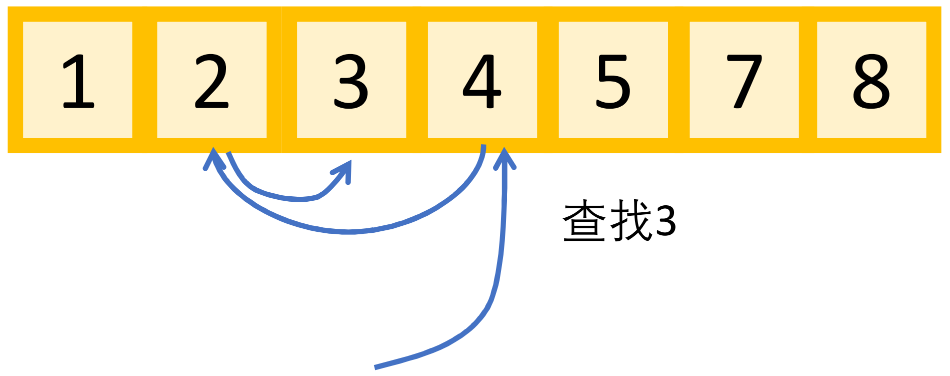
读取数组元素



```
int nums [5] = { 50,98,74,82,35};
cout<<nums[2] ;  // Displays 74
cout<<nums[4] ; //Displays 35
```

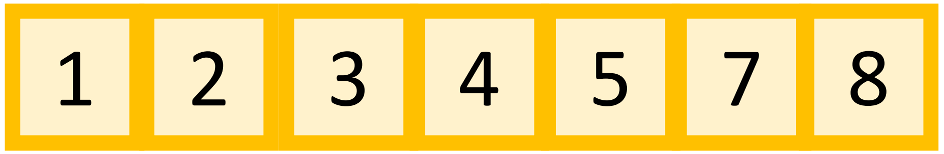
二分法查找

- O(log(n)) search, O(1) select:



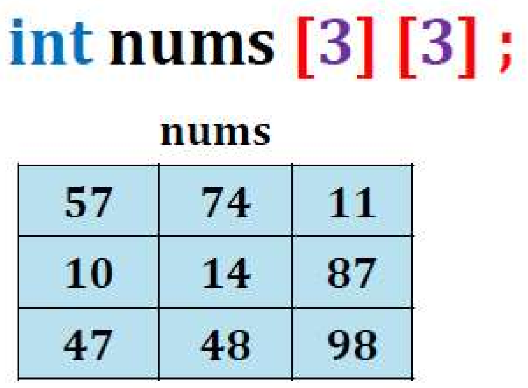
二分法查找

- O(log(n)) search, O(1) select:



二维数组

- 有两个维度(row, column)的索引index
- datatype name [rowsize][columnsize];



```
int nums [3] [3] ;
```

nums

57	74	11
10	14	87
47	48	98

初始化

```
int nums [3] [3] = {{57,74,11},{10,14,87}, {47,48,98}};
```

nums

57	74	11
10	14	87
47	48	98

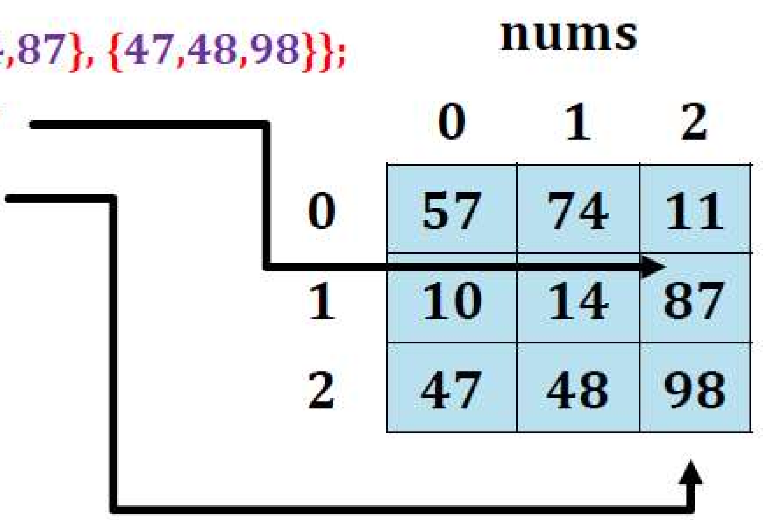
```
char alphabets [2] [3] = {{ ‘C’ , ’D’ , ’W’ }, { ‘1’ , ’?’ , ’V’ }};
```

alphabets

C	D	W
1	?	V

读取二维数组

```
int nums [3] [3] = {{57,74,11},{10,14,87}, {47,48,98}};
cout<<nums [1] [2] ;  // Displays 87
cout<<nums [2] [2] ; //Displays 98
```



	0	1	2
0	57	74	11
1	10	14	87
2	47	48	98

作业1

- Rotate an array of n elements to the right by k steps.
- For example, with n = 7 and k = 3, the array [1,2,3,4,5,6,7] is rotated to [5,6,7,1,2,3,4]
- 进一步如要要求空间复杂度O(1)，除了原来分配的数组，怎么解？

作业2

- 给定一个二维数组，输出数据按螺旋形顺序
- 比如

  [1, 2, 3]
  [4, 5, 6]
  [7, 8, 9]

  输出[1, 2, 3, 6, 9, 8, 7, 4, 5]

函数（function）

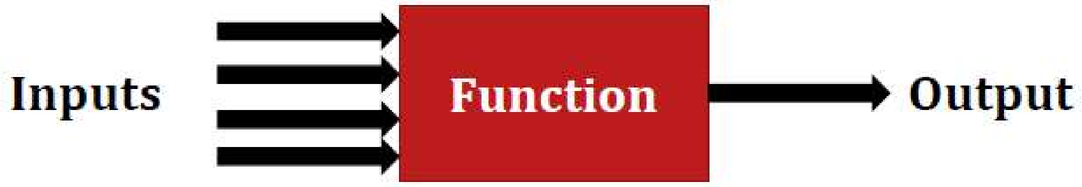
- 执行某种任务的代码块
- 单个可重用块中封装常用操作
- 函数里定义了形参，调用方传递实参给形参
- 函数可以设返回值
- 每一个C++程序必须有main（），自动调用

函数种类

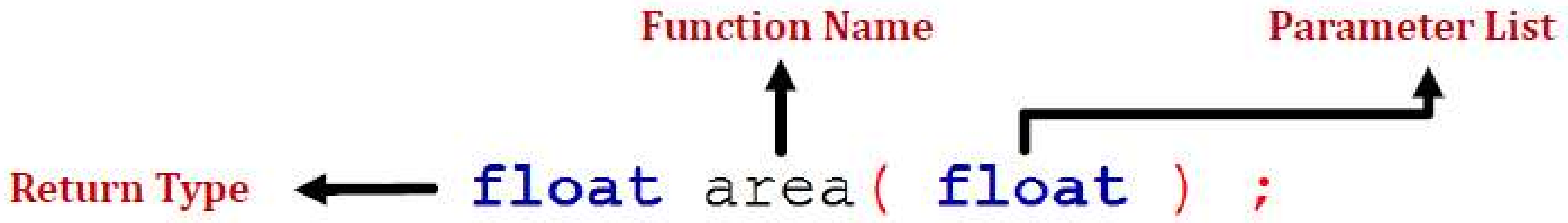
- 预先定义的函数，从标准库里读取
  - cout, cin, sin, cos
- 用户定义的函数

用户定义的函数

- 包括**4**个部分**:** 函数名、形式参数表、返回类型和函数体。
  - 函数的使用者通过函数名来调用该函数；
  - 调用时把实际参数传送给形式参数表作为数据的输入；
  - 通过函数体中的处理程序实现该函数的功能；
  - 最后得到返回值作为输出（也可以不返回值）。

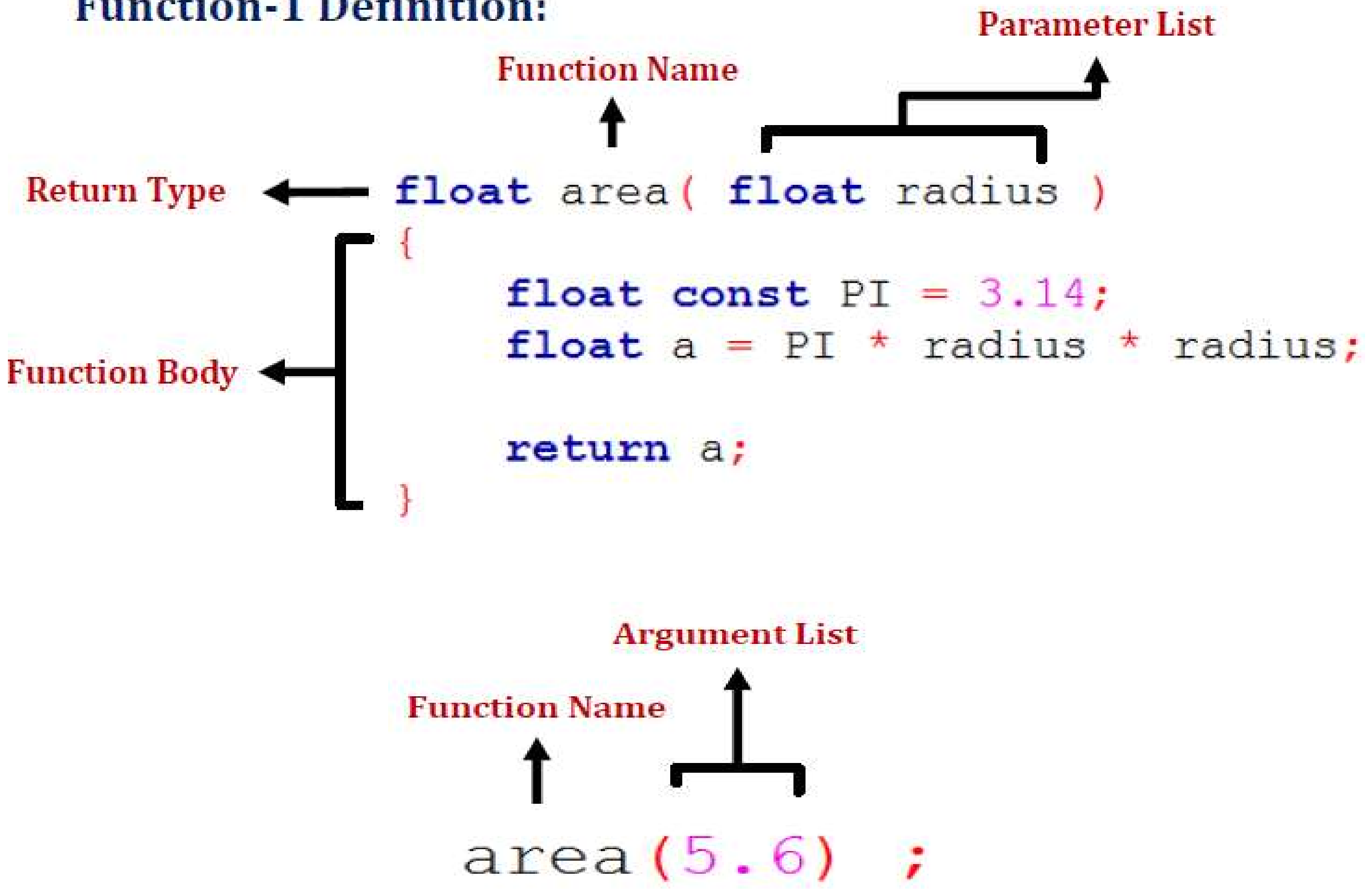


函数声明



函数定义

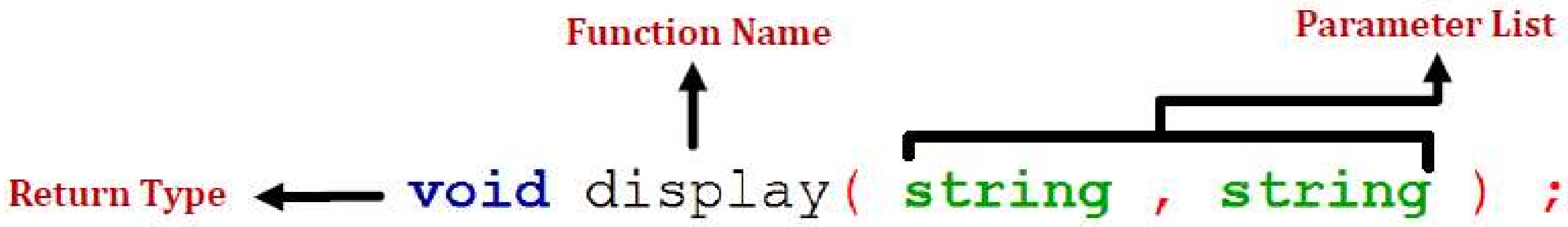
Function-1 Definition:



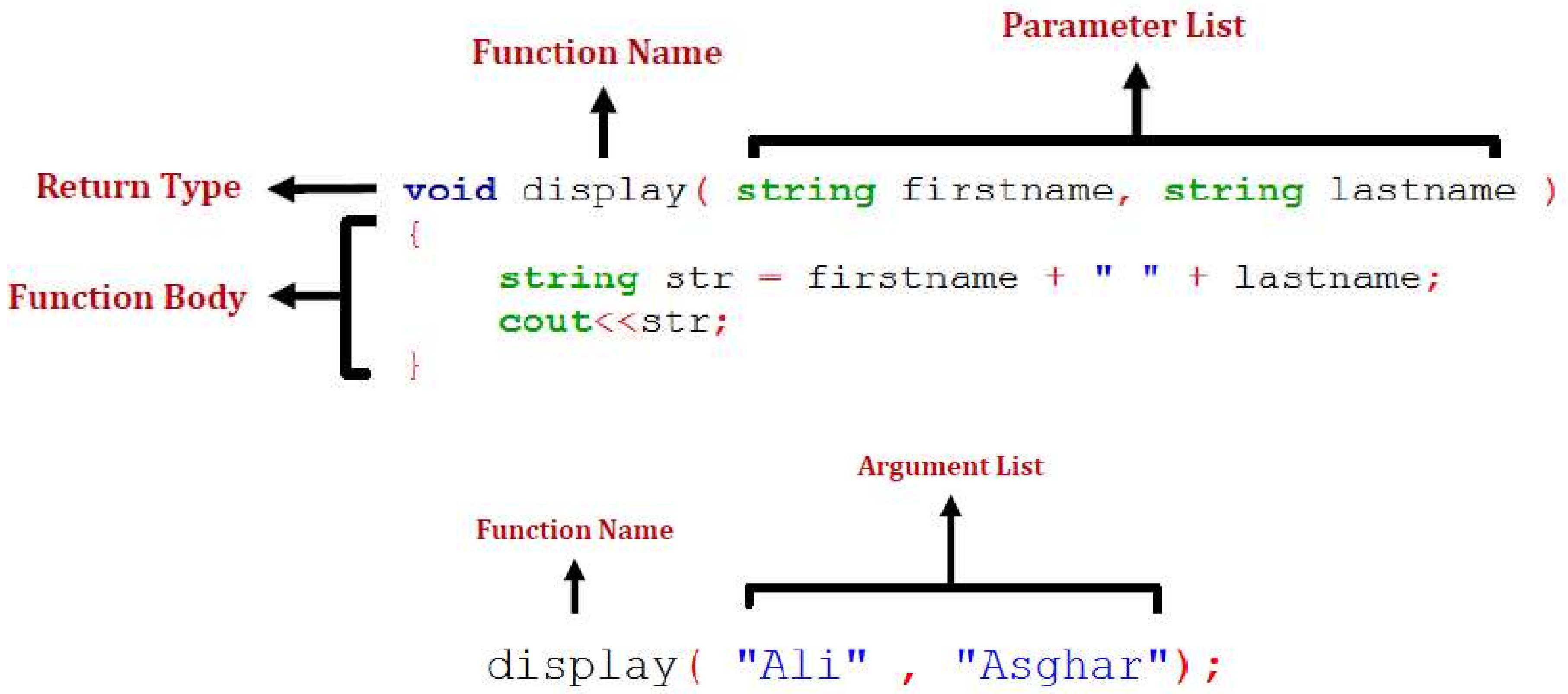
```
float area( float radius )
{
      float const PI = 3.14;
      float a = PI * radius * radius;

      return a;
}
```

```
area(5.6) ;
```

函数声明


Function Name
Parameter List
Return Type
void display(string , string) ;

函数定义



```
void display( string firstname, string lastname )
{
        string str = firstname + " " + lastname;
        cout<<str;
}
```

```
display( "Ali" , "Asghar");
```

传递实参

- Pass by value
  - 内置数据类型都可以
  - 实参复制到形参，在执行过程中实参不会改变
- Pass by reference
  - 用户定义数据类型应考虑
  - 一般来说，应该使用const datatype &

```
string concatenate (const string& a, const string& b)
{
  return a+b;
}
```

函数递归调用

n! = n * (n-1) * (n-2) * (n-3) ... * 1

```cpp
// factorial calculator
#include <iostream>
using namespace std;

long factorial (long a)
{
  if (a > 1)
   return (a * factorial (a-1));
  else
   return 1;
}

int main ()
{
  long number = 9;
  cout << number << "! = " << factorial (number);
  return 0;
}
```

```
9! = 362880
```

数组作为形式参数 （一）

```
int processArray(int arg[], int size) {
    ….
}

Int main() {
  Int myArray[10];
  processArray(myArray, 10);
}
```

数组作为形式参数（二）

```
int processArray(int arg[][15], int size) {
    ….
}

int main() {
    Int myArray[10][15];
    processArray(myArray, 10);
}
```

内联(inline)函数

- 在函数定义前加上一个inline前缀就成为内联函数。编译程序在编译时将会把这个函数语句直接插入到普通代码中，因而减少了与函数调用和参数传递有关的系统开销。
- 直接插入代码所需要的空间比不直接插入的调用方式所需要的空间要多，这取决于函数定义的大小。
- 除了加上inline保留字外，内联函数的定义与其它任何函数定义的方式一样。

```
inline string concatenate (const string& a, const string& b)
{
   return a+b;
}
```

内联和宏命令

- 工作原理类似，都是执行一段简单的函数任务，但在预编译阶段直接展开，而不是像函数一样要额外开销

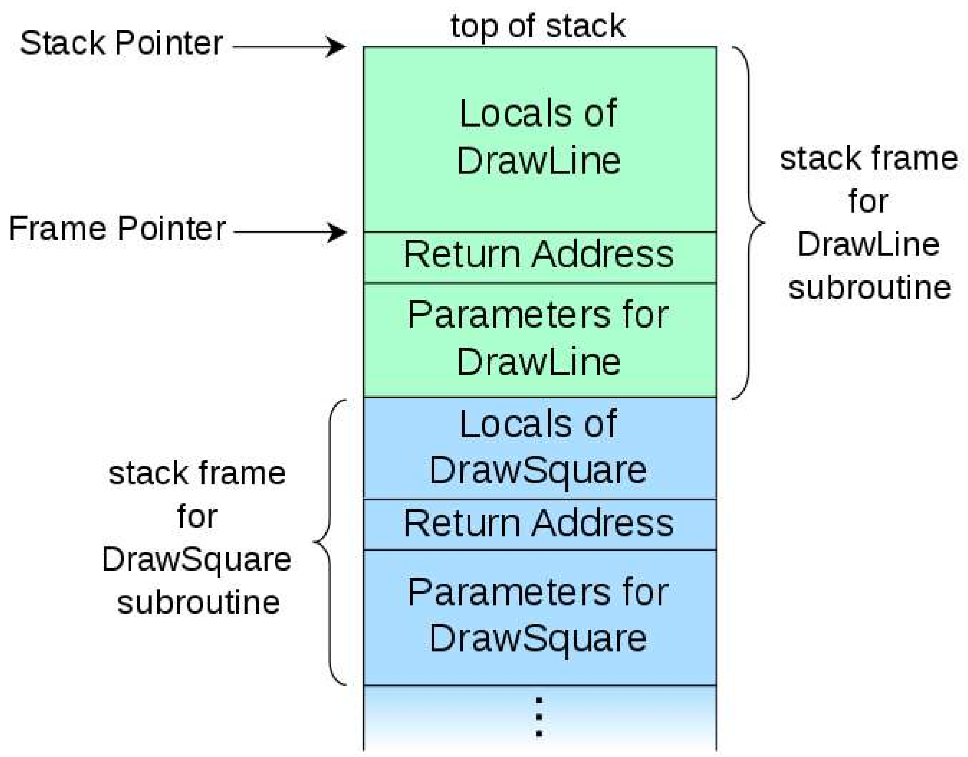
- 内联可以完全取代宏命令，改进了它的缺点

```
inline int MAX(x, y){
    return x > y ? x : y;
}
#define MAX(x, y) x > y ? x : y;
```

函数额外开销

- 函数调用时要把一些变量压到stack
- 但是stack大小有限，比如1M

```
int foo() {
  return foo();
}
```



Struct

- 可以让你把数据放到一个地方
- 进一步可以把 函数放到struct，
- 可以， 不过只能在C++。这就是OO

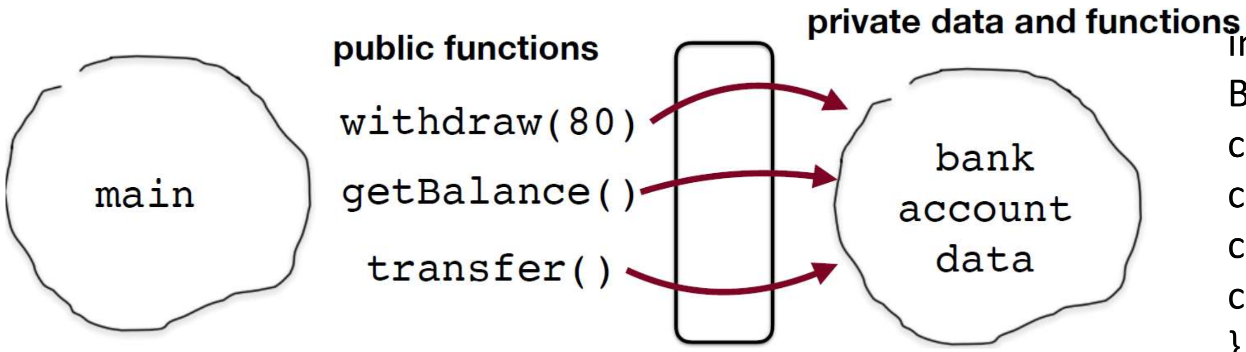
```
struct Lunchable {
        string meat;
        string dessert;
        int numCrackers;
        bool hasCheese;

        int CountCalories();
        };
```

C++类（class）

- 和struct差不多，private vs public
- 可以自定义一个数据类型
- 比如
  - 学生注册系统需要存储学生信息
  - 手机系统要保存联系人信息

类：封装

- 和struct不同在于private
- 封装就是把内部的数据和外部的接口分割开来
- 一般来说data都是private的， 函数可以public
- 和外部的接口越少越好



```
int main() {
BankAccount checking("Bob", 42);
checking.withdraw(80);
cout << checking.getBalance() << endl;
checking.balance = 81.2345;
checking.setBalance(81.2345);
}
```

类成员

- 成员变量
  - private
- 成员函数
  - 可以直接访问数据变量
- 构造函数
  - 初始化成员变量
  - 可以有输入参数

类interface

- .h
  - 可以变成别人引用
  - 声明declaration
- .cpp
  - 实现
  - 定义Definition
  - 转化为bin

头文件定义

避免重复引用。

新的写法

```
// classname.h
#pragma once

class ClassName {
    // class definition
};
```

老的写法

```
// classname.h
#ifndef _CLASSNAME_H
#define _CLASSNAME_H

class ClassName {
    // class definition
};

#endif
```

头文件格式.h file

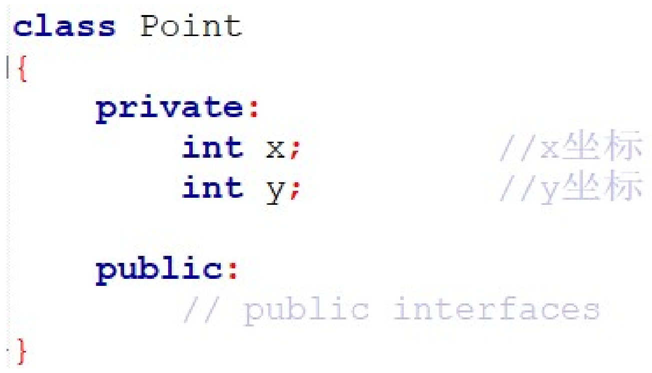
```
// in ClassName.h
class ClassName {
public:
        ClassName(parameters); // constructor
        returnType func1(parameters); // member functions
        returnType func2(parameters); // (behavior inside
        returnType func3(parameters); // each object)
private:
        type var1; // member variables
        type var2; // (data inside each object)
        type func4(); // (private function)
};
```

Point class

- 什么数据 (private)
- 什么构造函数 (public)
- 什么功能 (public / private)
- Keep it simple

Point class

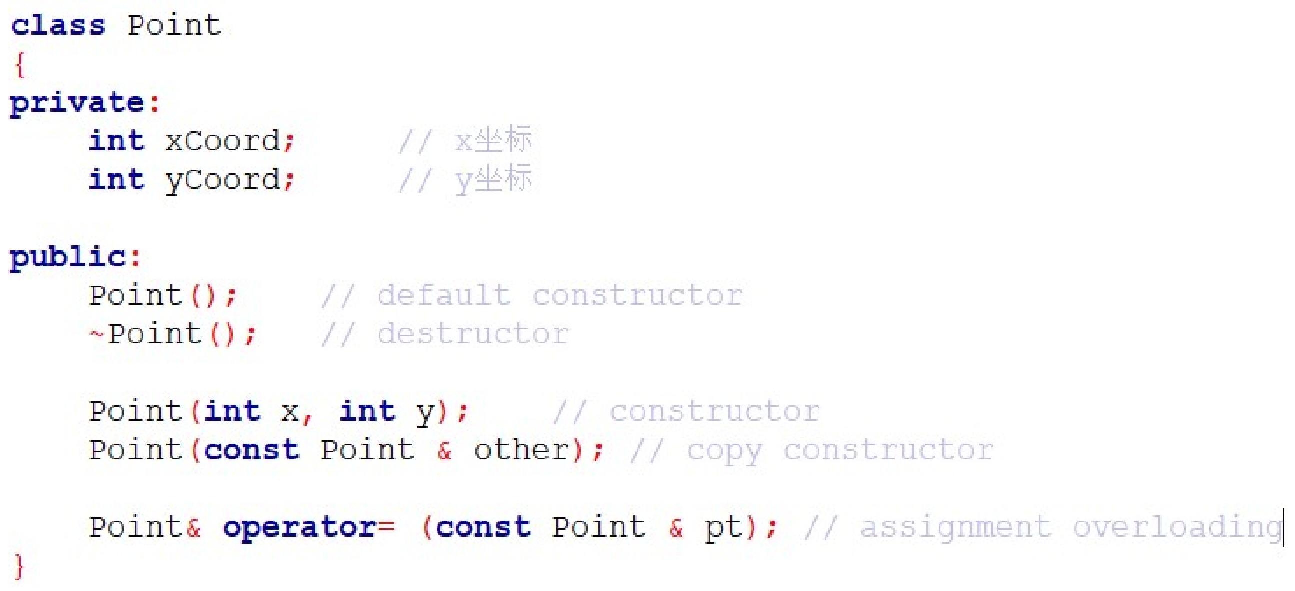
- 数据



```
class Point
{
    private:
        int x;          //x坐标
        int y;          //y坐标

    public:
        // public interfaces
}
```

Point类

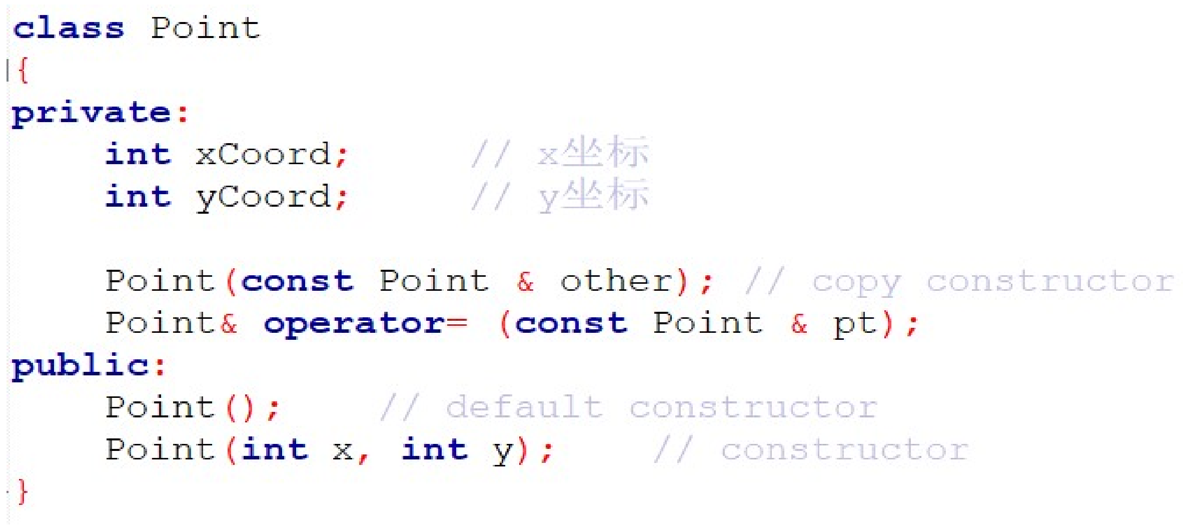
- 隐含的构造函数和方法



```cpp
class Point
{
private:
    int xCoord;     // x坐标
    int yCoord;     // y坐标

public:
    Point();    // default constructor
    ~Point();   // destructor

    Point(int x, int y);    // constructor
    Point(const Point & other); // copy constructor

    Point& operator= (const Point & pt); // assignment overloading
}
```

Point类

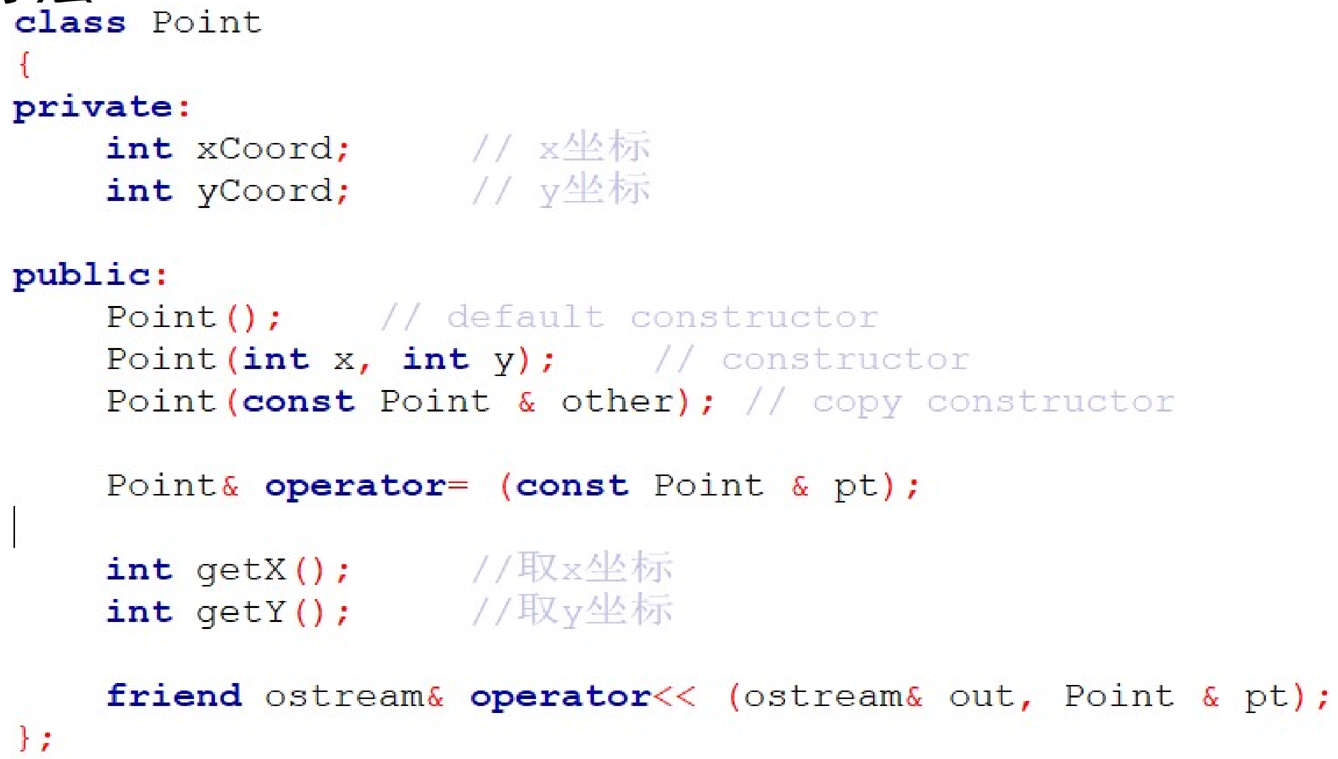
- 禁止类复制和赋值



```
class Point
{
private:
    int xCoord;      // x坐标
    int yCoord;      // y坐标

    Point(const Point & other); // copy constructor
    Point& operator= (const Point & pt);
public:
    Point();     // default constructor
    Point(int x, int y);     // constructor
}
```

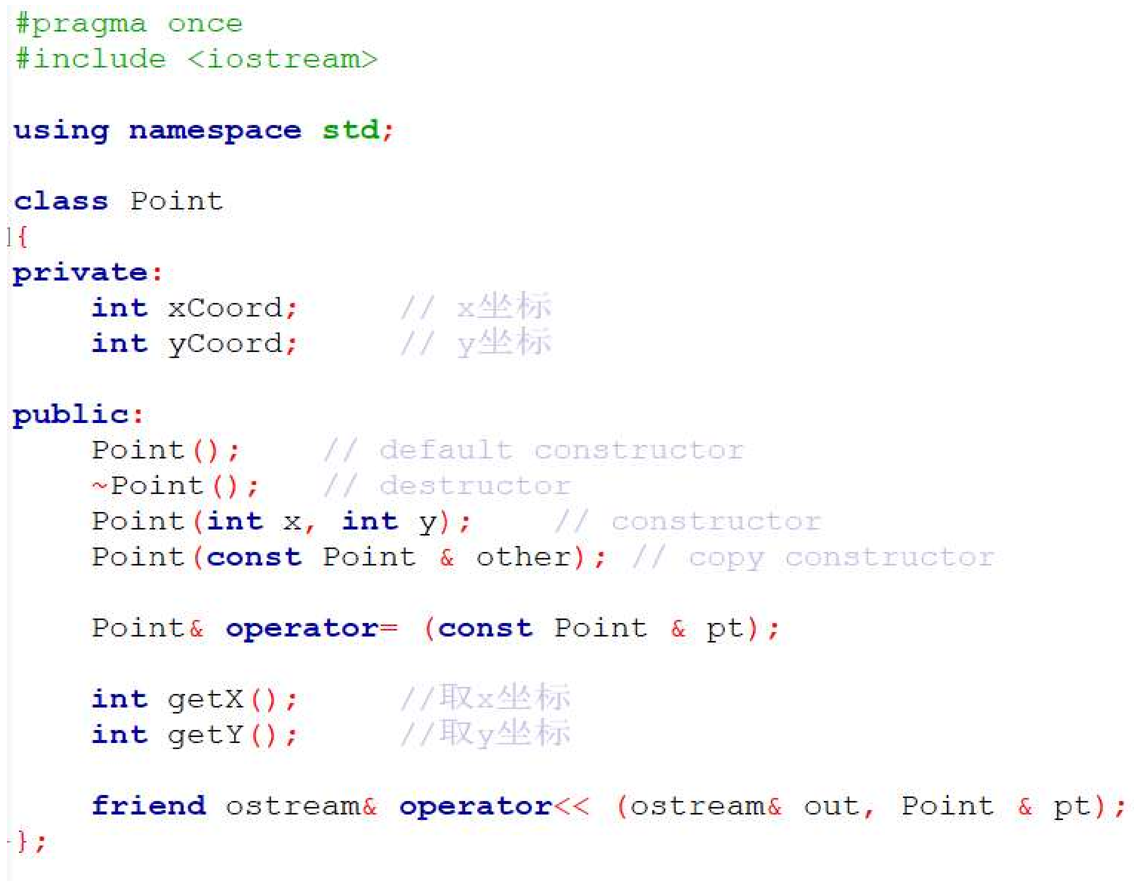
Point类

- 公开方法



```
class Point
{
private:
    int xCoord;      // x坐标
    int yCoord;      // y坐标

public:
    Point();     // default constructor
    Point(int x, int y);     // constructor
    Point(const Point & other); // copy constructor

    Point& operator= (const Point & pt);

    int getX();      //取x坐标
    int getY();      //取y坐标

    friend ostream& operator<< (ostream& out, Point & pt);
};
```

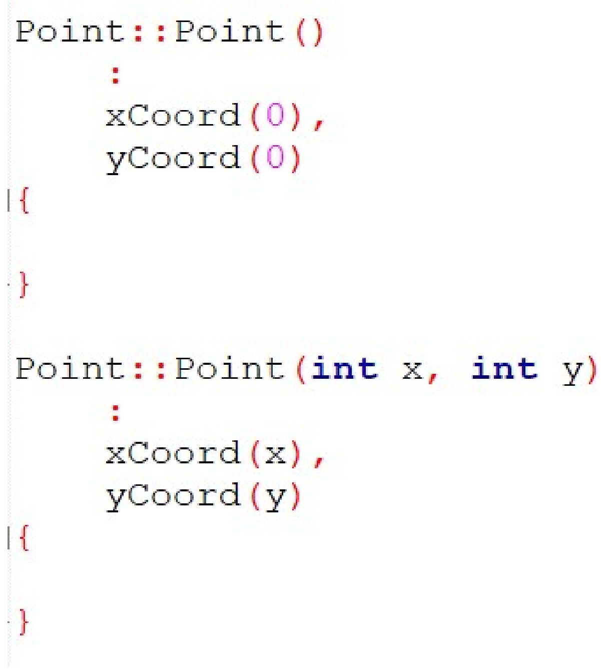
最后, Point类声明point.h



```
#pragma once
#include <iostream>

using namespace std;

class Point
{
private:
    int xCoord;     // x坐标
    int yCoord;     // y坐标

public:
    Point();    // default constructor
    ~Point();   // destructor
    Point(int x, int y);    // constructor
    Point(const Point & other); // copy constructor

    Point& operator= (const Point & pt);

    int getX();     //取x坐标
    int getY();     //取y坐标

    friend ostream& operator<< (ostream& out, Point & pt);
};
```

类定义Point.cpp

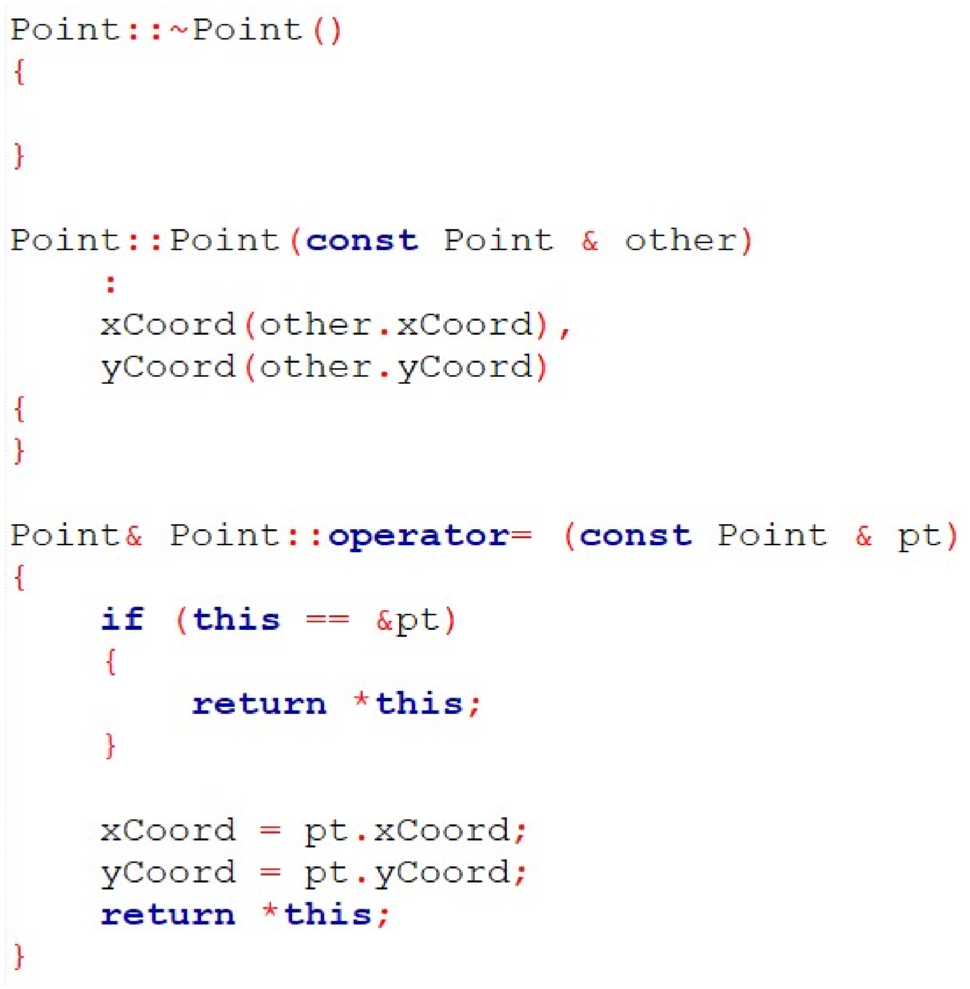
// purpose: the default constructor
// to init (x, y)
// arguments: none
// return value: none
// (constructors don't return anything)



```
Point::Point()
    :
    xCoord(0),
    yCoord(0)
{

}

Point::Point(int x, int y)
    :
    xCoord(x),
    yCoord(y)
{

}
```

```
#include “Point.h”
Main()
{
   Point ptStart;
   Point ptEnd(1, 2);
   ….
}
```

Point.cpp

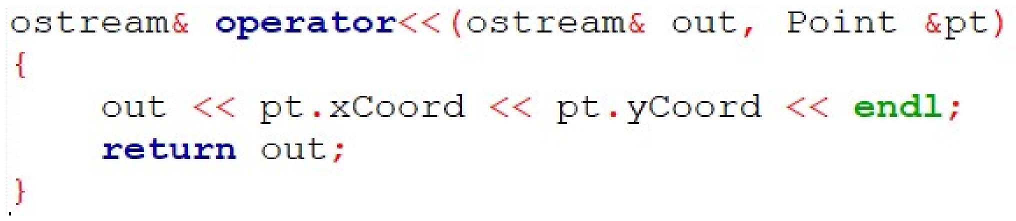


```
Point::~Point()
{

}

Point::Point(const Point & other)
    :
    xCoord(other.xCoord),
    yCoord(other.yCoord)
{
}

Point& Point::operator= (const Point & pt)
{
    if (this == &pt)
    {
        return *this;
    }

    xCoord = pt.xCoord;
    yCoord = pt.yCoord;
    return *this;
}
```

```
#include “Point.h”
Main()
{
    Point ptStart;
    Point ptEnd(1, 2);
    Point ptNew = ptStart;
    GetDist(Point a, Point b);
    ….
}
```

Point.cpp

// purpose: To overload the << operator
// for use with cout
// arguments: a reference to an outstream and the
// point we are using
// return value: a reference to the outstream



```
ostream& operator<<(ostream& out, Point &pt)
{
    out << pt.xCoord << pt.yCoord << endl;
    return out;
}
```

```
#include "Point.h"
Main()
{
    Point ptStart;
    cout << ptStart << endl;
    ….
}
```

友元(friend)函数

- 在类的声明中可使用保留字friend定义友元函数。

- 友元函数实际上并不是这个类的成员函数，它可以是一个常规函数，也可以是另一个类的成员函数。如果想通过这种函数存取类的私有成员和保护成员，则必须在类的声明中给出函数的原型，并在该函数原型前面加上一个friend。

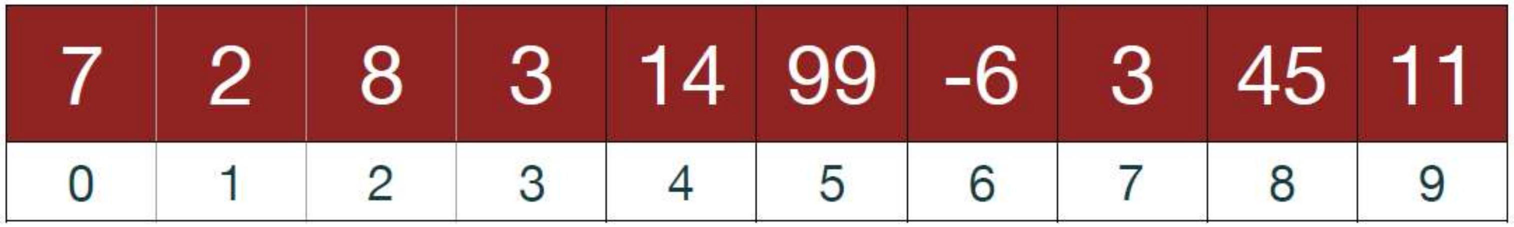
- 参看Point类的声明，有两个重载操作符<< 与>>，它们都被声明为友元函数。

类对象（object）

- 建立类的对象（亦称为实例化）时采用的方式类似于定义C变量的方式，可以自动地，或静态地，或通过动态分配来建立。建立一个Point类实例的语句是：
- Point p(6, 3);　自动地
- Point q;　自动地
- static Point s(3, 4);　静态地
- Point *pStart = new Point(1, 1);　通过动态分配

指针

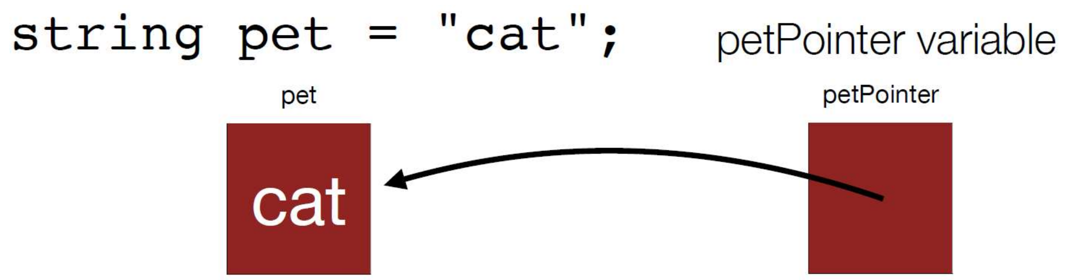
- 计算机内存是连续的字节块，每个字节都带有索引
- 像数组？
- address



7	2	8	3	14	99	-6	3	45	11
0	1	2	3	4	5	6	7	8	9

指针是什么

- 内存地址
- 由操作系统决定
- 我们并不在乎内存地址本身的值，我们更在意地址指向的值

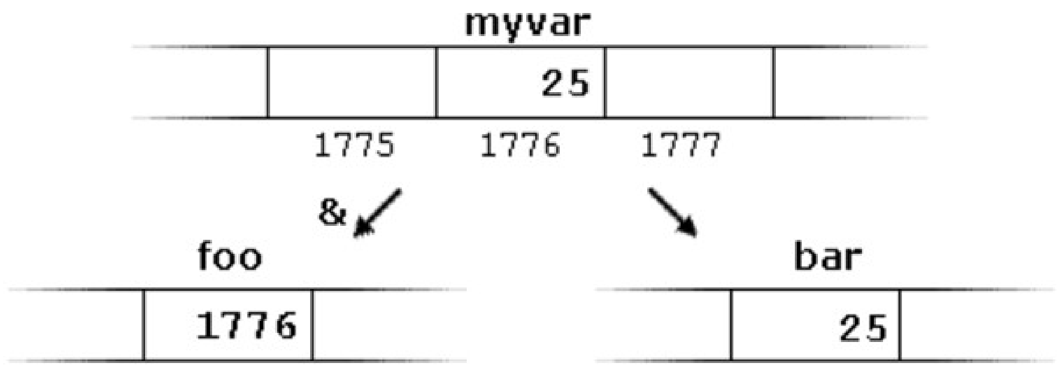


关于指针

- 内存每个字节都有地址，所以每个变量都有地址
- 每个地址在内存里都是唯一的
- 计算机（compiler)知道每个变量地址
- 计算机也能找到一个地址对应的变量
- 地址对应一中数据类型，64bit 8字节；32bit 4字节

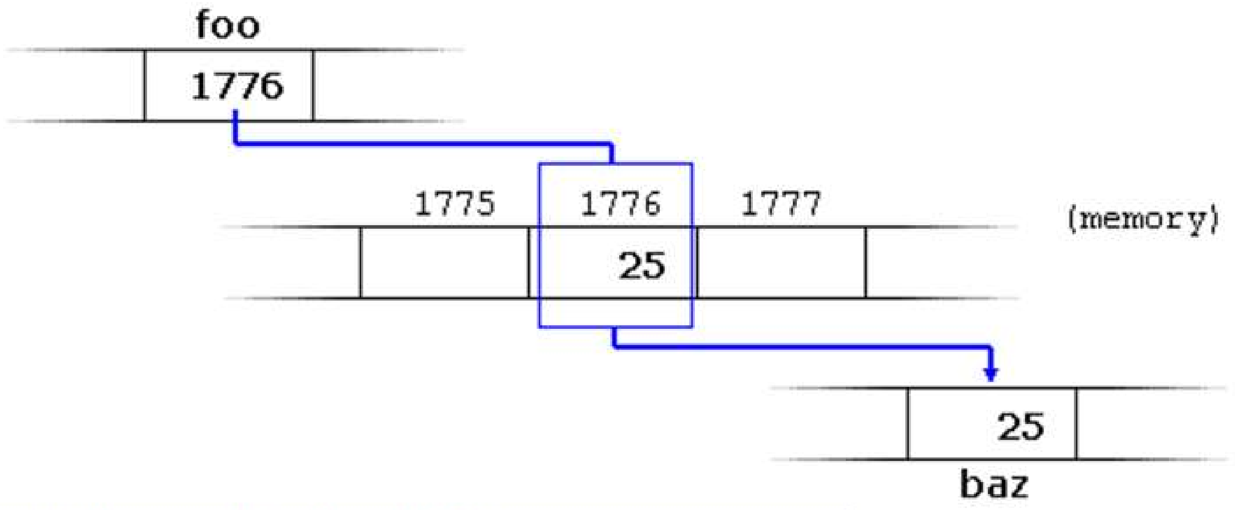
取地址&

- int myvar = 25；
- int * foo = &myvar；
- bar = myvar;



取指针值*

- baz = *foo



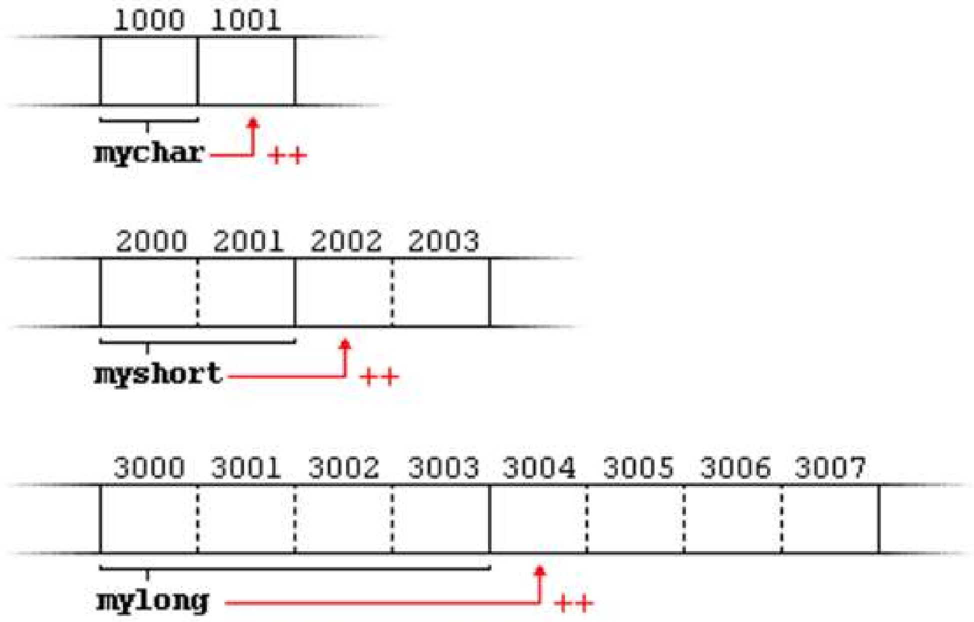
```
baz = foo;    // baz equal to foo (1776)
baz = *foo;   // baz equal to value pointed to by foo (25)
```

指针和数组

```cpp
// more pointers
#include <iostream>
using namespace std;

int main ()
{
  int numbers[5];
  int * p;
  p = numbers;  *p = 10;
  p++;  *p = 20;
  p = &numbers[2];  *p = 30;
  p = numbers + 3;  *p = 40;
  p = numbers;  *(p+4) = 50;
  for (int n=0; n<5; n++)
    cout << numbers[n] << ", ";
  return 0;
}
```

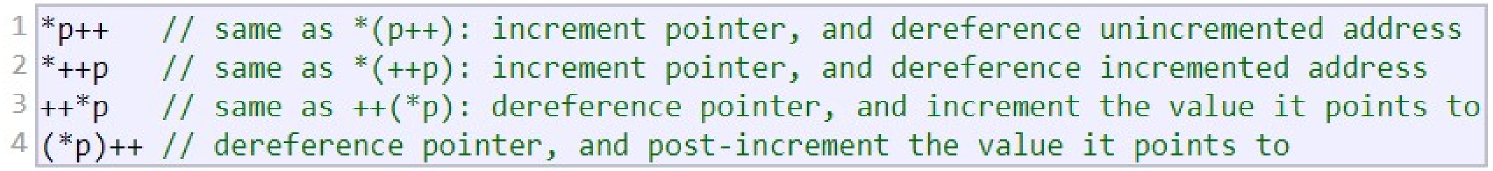
```
10, 20, 30, 40, 50,
```

指针运算


1000
1001
mychar
++
2000
2001
2002
2003
myshort
++
3000
3001
3002
3003
3004
3005
3006
3007
mylong
++

指针运算

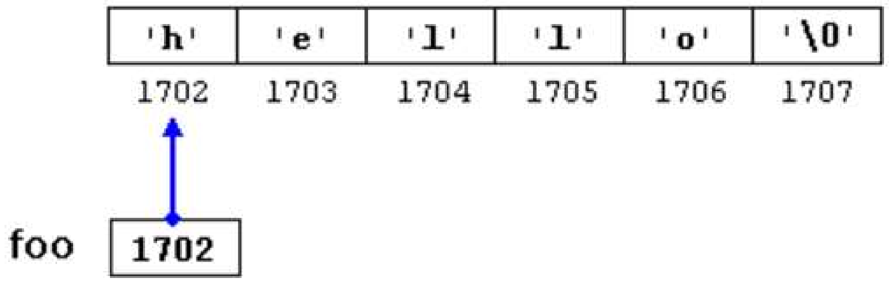
- *p++
- *++p
- ++*p
- (*p)++



```
*p++   // same as *(p++): increment pointer, and dereference unincremented address
*++p   // same as *(++p): increment pointer, and dereference incremented address
++*p   // same as ++(*p): dereference pointer, and increment the value it points to
(*p)++ // dereference pointer, and post-increment the value it points to
```

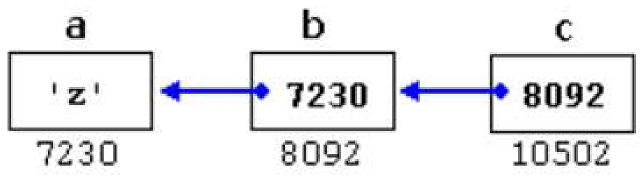
指针和string

- const char * foo = “hello” ;

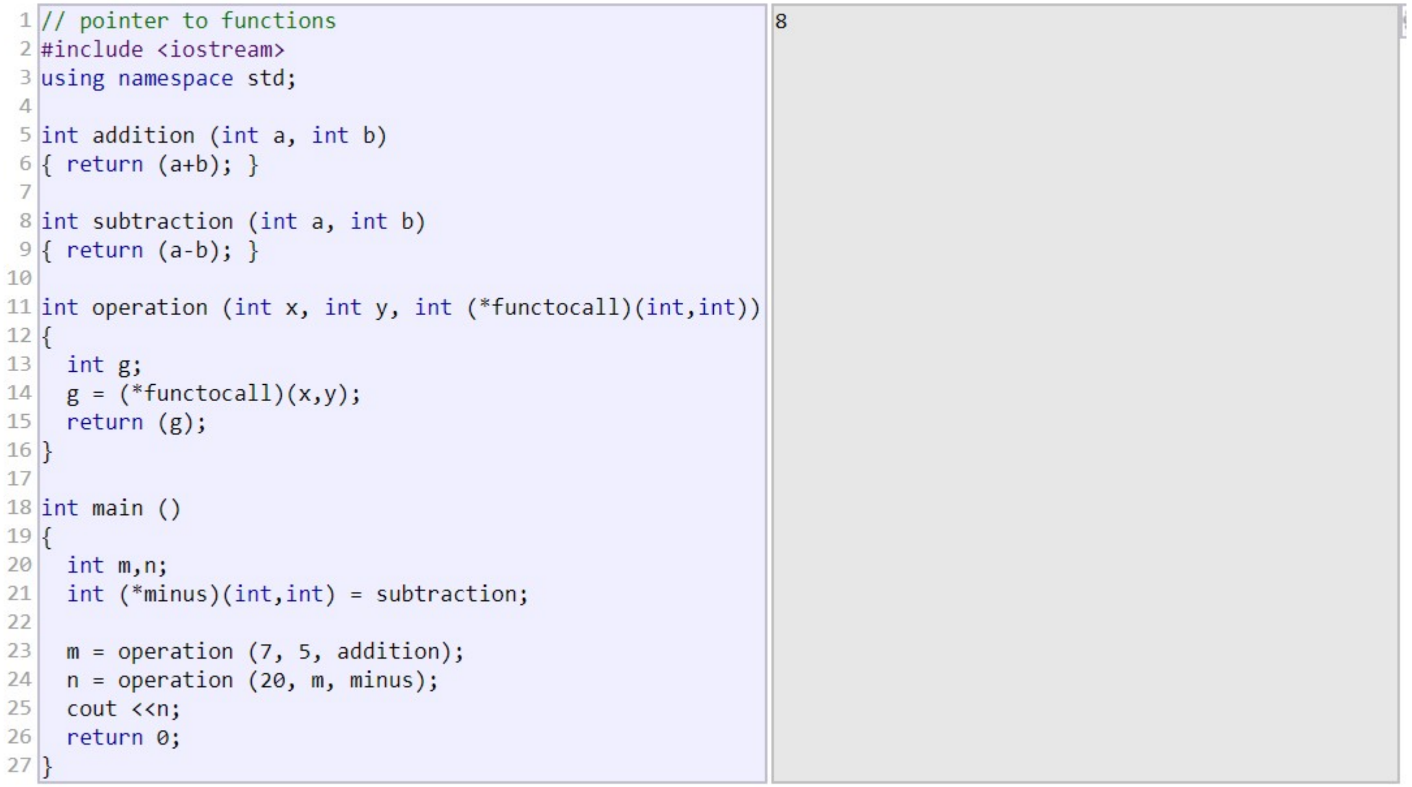


指针的指针

- char a
- char * b;
- char ** c;
- a = ‘z’;
- b = &a;
- c = &b;



函数指针



```
// pointer to functions
#include <iostream>
using namespace std;

int addition (int a, int b)
{ return (a+b); }

int subtraction (int a, int b)
{ return (a-b); }

int operation (int x, int y, int (*functocall)(int,int))
{
  int g;
  g = (*functocall)(x,y);
  return (g);
}

int main ()
{
  int m,n;
  int (*minus)(int,int) = subtraction;

  m = operation (7, 5, addition);
  n = operation (20, m, minus);
  cout <<n;
  return 0;
}
```

```
8
```

回到类例子

rectangle.h:

```
#pragma once

class Rectangle {
public:
    Rectangle(double width = 1, double height = 1); // constructor
    ~Rectangle(); // destructor (more on this later)

    double area();
    double perimeter();
    double getHeight();
    double getWidth();

private:
    double *height; // pointer to a double
    double *width;  // pointer to a double
};
```

```
int main() {
    Rectangle r(3,4);
    cout << "Width: " << r.getWidth() << ", ";
    cout << "Height: " << r.getHeight() << endl;

    cout << "Area: " << r.area() << endl;
    cout << "Perimeter: " << r.perimeter() << endl;
```

no problem...

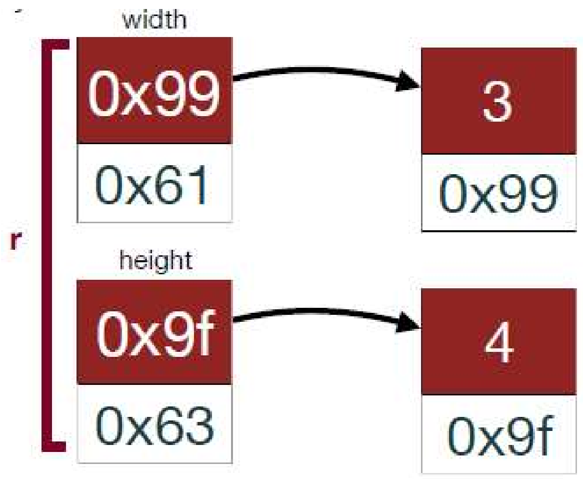
```
   {
        // let's make a copy:
        Rectangle r2 = r;

    }

   cout << "Width: " << r.getWidth() << ", ";
```

crash!

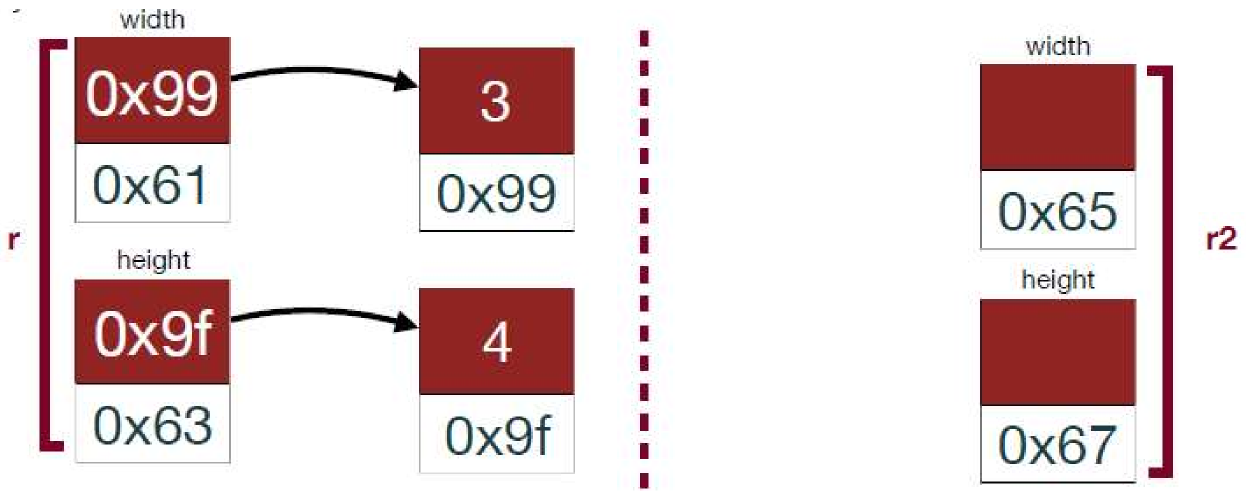
rectangle.cpp:

```
#include "rectangle.h"

Rectangle::Rectangle(double width, double height) { // constructor
    this->width = new double;
    this->height = new double;
    *(this->width) = width;
    *(this->width) = height;
}

Rectangle::~Rectangle() { // destructor
    delete height;
    delete width;
 {

double Rectangle::area() {
    return *width * *height;
}

double Rectangle::perimeter() {
    return 2 * *width + 2 * *height;
}

double Rectangle::getHeight() {
    return *height;
}

double Rectangle::getWidth() {
    return *width;
}
```

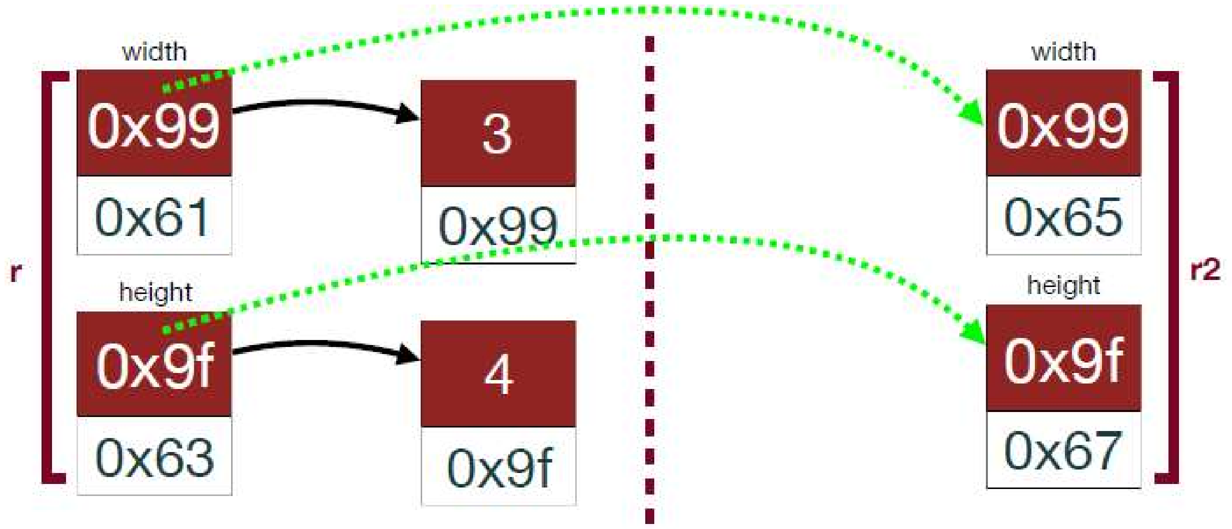
```
int main() {
    Rectangle r(3,4);

}
```



```
int main() {
     Rectangle r(3,4);
  { Rectangle r2 = r; }
}      ...
```

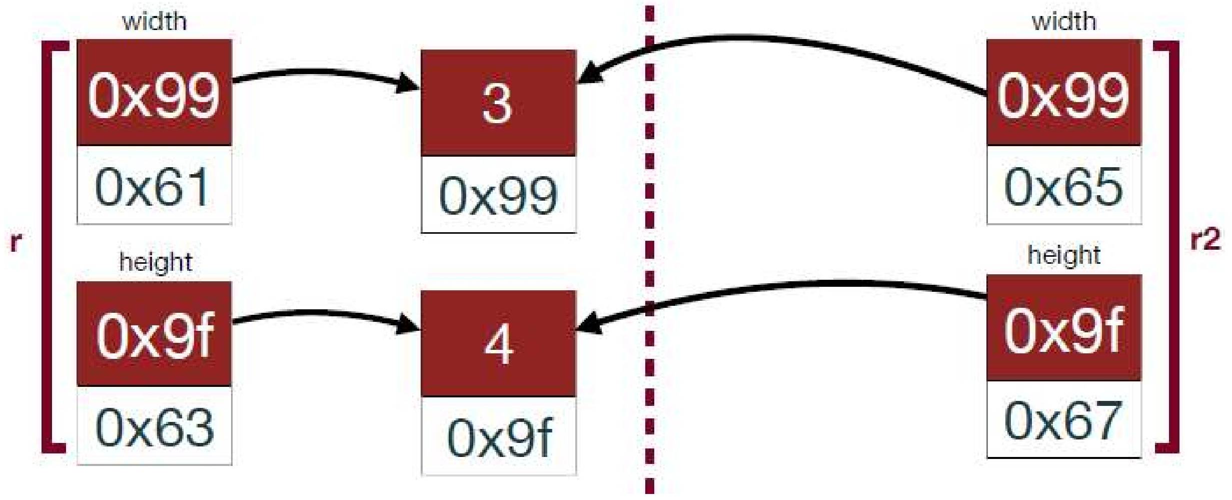


```
int main() {
    Rectangle r(3,4);
  { Rectangle r2 = r; }
}      ...
```



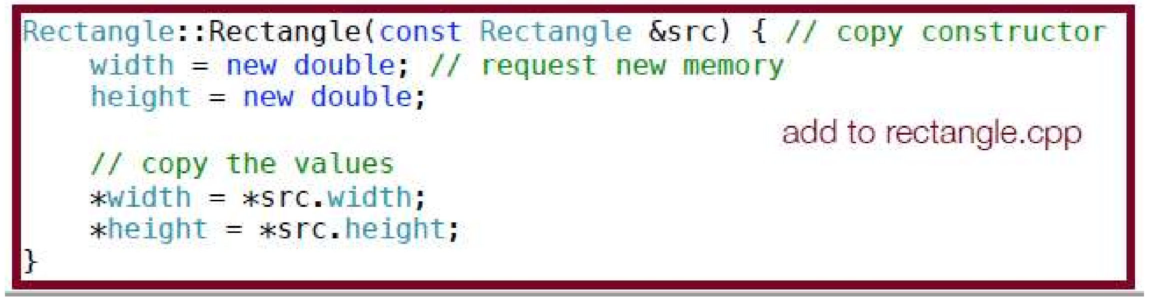
问题：

```
int main() {
    Rectangle r(3,4);
  { Rectangle r2 = r;  }
}       ...
```



修正：rule of three

- 定义了destructor，那么必须定义copy constructor 和assignment overloading



```
Rectangle::Rectangle(const Rectangle &src) { // copy constructor
    width = new double; // request new memory
    height = new double;

    // copy the values
    *width = *src.width;
    *height = *src.height;
}
```

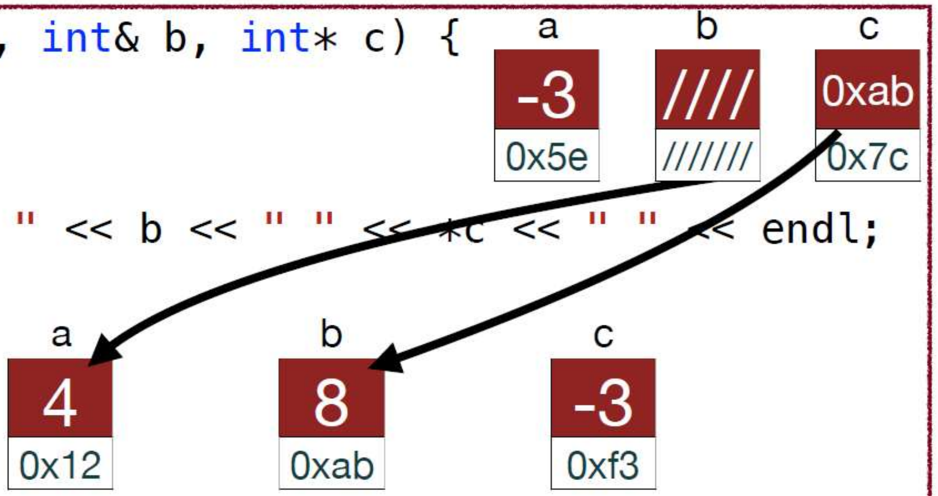
add to rectangle.cpp

总结

- 指针就是内存地址， 也是一种数据类型
- 必须指向一种数据类型，int *， char *， void *
- 声明指针用*
- 取地址用&
- 取指向的值用*
- 空指针nullptr
- 两个指针可以指向同一个地址

```
void mystery(int a, int& b, int* c) {
    a++;
    (*c)--;
    b += *c;
    cout << a << " " << b << " " << *c << " " << endl;
}

int main() {
    int a = 4;
    int b = 8;
    int c = -3;
    cout << a << " " << b << " " << c << " " << endl;
    mystery(c, a, &b);
    cout << a << " " << b << " " << c << " " << endl;
    return 0;
}
```

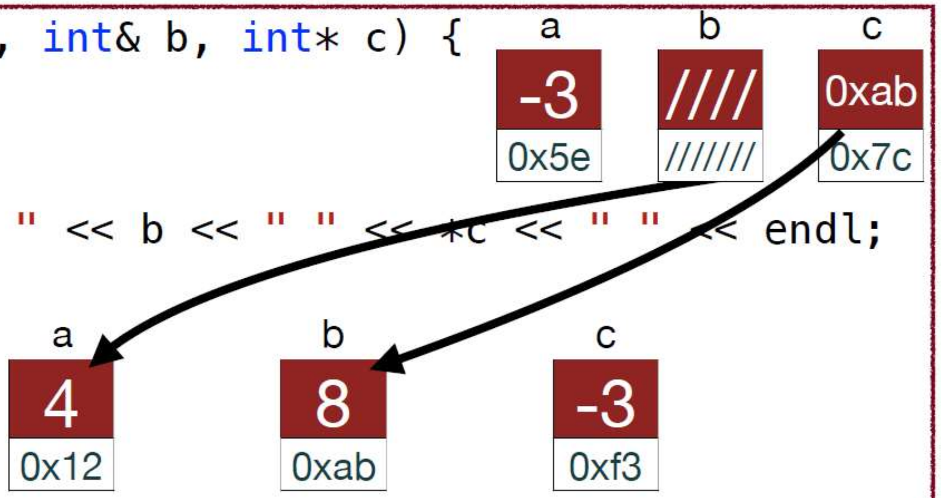
```
void mystery(int a, int& b, int* c) {
    a++;
    (*c)--;
    b += *c;
    cout << a << " " << b << " " << *c << " " << endl;
}

int main() {
    int a = 4;
    int b = 8;
    int c = -3;
    cout << a << " " << b << " " << c << " " << endl;
    mystery(c, a, &b);
    cout << a << " " << b << " " << c << " " << endl;
    return 0;
}
```



Answer:

```
4    8   -3
```

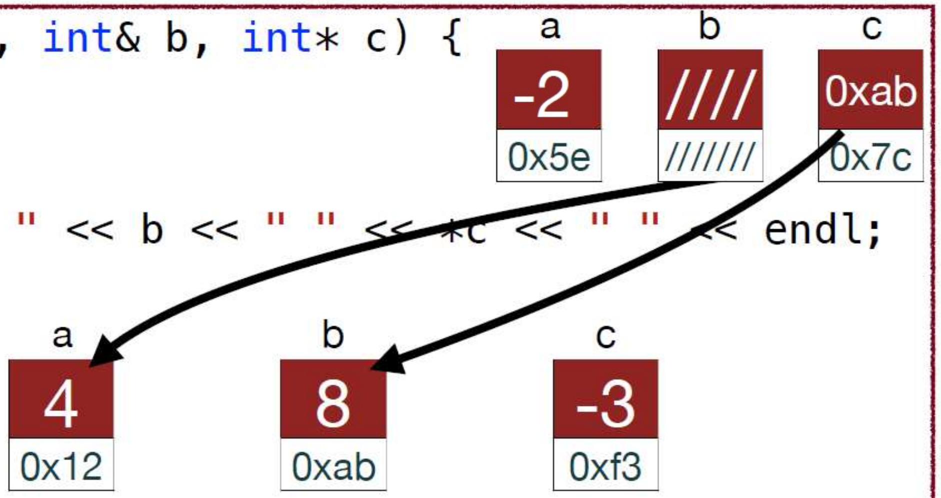
```
void mystery(int a, int& b, int* c) {
    a++;
    (*c)--;
    b += *c;
    cout << a << " " << b << " " << *c << " " << endl;
}

int main() {
    int a = 4;
    int b = 8;
    int c = -3;
    cout << a << " " << b << " " << c << " " << endl;
    mystery(c, a, &b);
    cout << a << " " << b << " " << c << " " << endl;
    return 0;
}
```



Answer:

```
4   8   -3
```

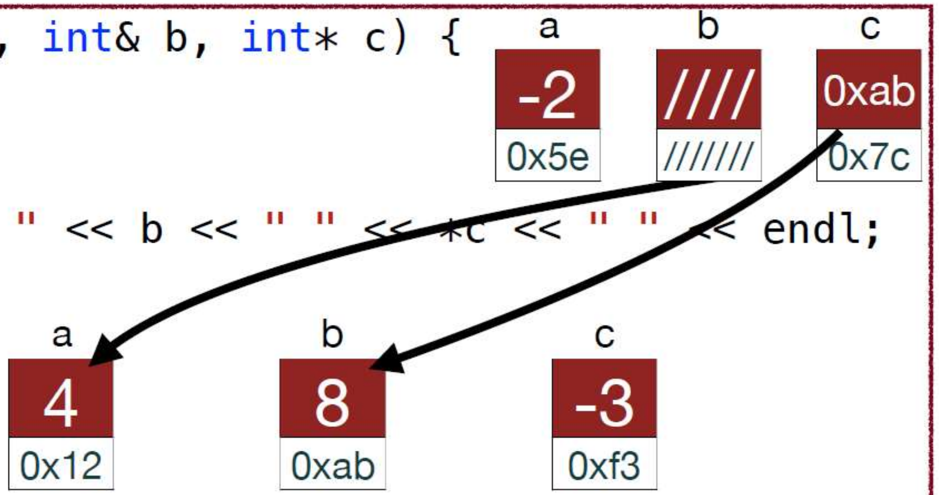
```
void mystery(int a, int& b, int* c) {
    a++;
    (*c)--;
    b += *c;
    cout << a << " " << b << " " << *c << " " << endl;
}

int main() {
    int a = 4;
    int b = 8;
    int c = -3;
    cout << a << " " << b << " " << c << " " << endl;
    mystery(c, a, &b);
    cout << a << " " << b << " " << c << " " << endl;
    return 0;
}
```



Answer:

```
4    8   -3
```

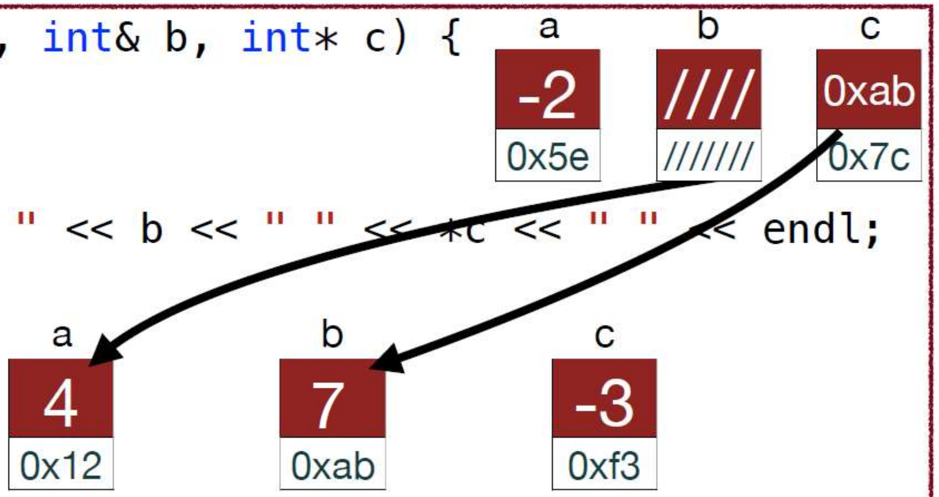
```
void mystery(int a, int& b, int* c) {
    a++;
    (*c)--;
    b += *c;
    cout << a << " " << b << " " << *c << " " << endl;
}

int main() {
    int a = 4;
    int b = 8;
    int c = -3;
    cout << a << " " << b << " " << c << " " << endl;
    mystery(c, a, &b);
    cout << a << " " << b << " " << c << " " << endl;
    return 0;
}
```



Answer:

```
4   8   -3
```

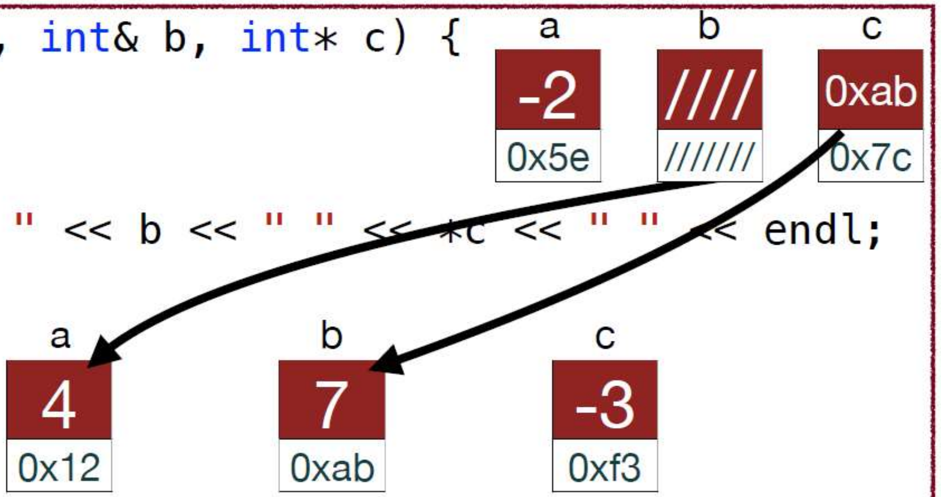
```
void mystery(int a, int& b, int* c) {
    a++;
    (*c)--;
    b += *c;
    cout << a << " " << b << " " << *c << " " << endl;
}

int main() {
    int a = 4;
    int b = 8;
    int c = -3;
    cout << a << " " << b << " " << c << " " << endl;
    mystery(c, a, &b);
    cout << a << " " << b << " " << c << " " << endl;
    return 0;
}
```



Answer:

```
4    8   -3
```

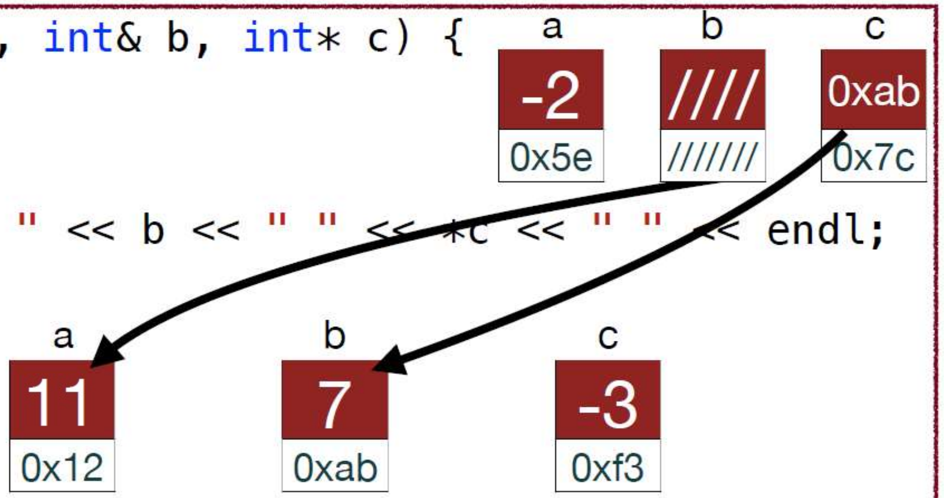
```
void mystery(int a, int& b, int* c) {
    a++;
    (*c)--;
    b += *c;
    cout << a << " " << b << " " << *c << " " << endl;
}

int main() {
    int a = 4;
    int b = 8;
    int c = -3;
    cout << a << " " << b << " " << c << " " << endl;
    mystery(c, a, &b);
    cout << a << " " << b << " " << c << " " << endl;
    return 0;
}
```



Answer:

```
4   8   -3
```

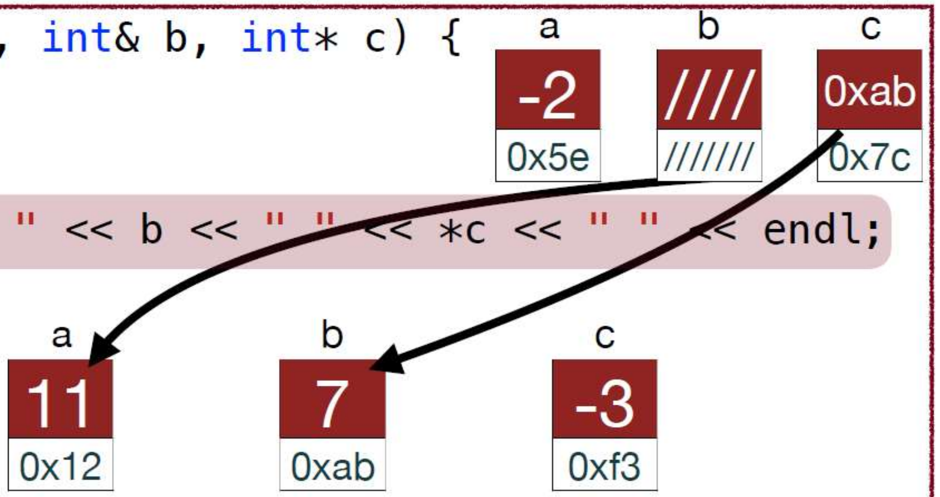
```
void mystery(int a, int& b, int* c) {
    a++;
    (*c)--;
    b += *c;
    cout << a << " " << b << " " << *c << " " << endl;
}

int main() {
    int a = 4;
    int b = 8;
    int c = -3;
    cout << a << " " << b << " " << c << " " << endl;
    mystery(c, a, &b);
    cout << a << " " << b << " " << c << " " << endl;
    return 0;
}
```



Answer:

```
4    8   -3
```

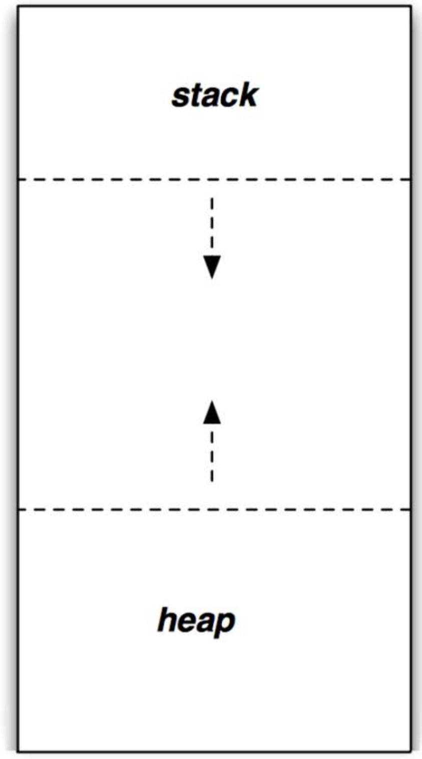
```cpp
void mystery(int a, int& b, int* c) {
    a++;
    (*c)--;
    b += *c;
    cout << a << " " << b << " " << *c << " " << endl;
}

int main() {
    int a = 4;
    int b = 8;
    int c = -3;
    cout << a << " " << b << " " << c << " " << endl;
    mystery(c, a, &b);
    cout << a << " " << b << " " << c << " " << endl;
    return 0;
}
```



Answer:

```
 4    8   -3
-2   11    7
```

动态内存分配



为什么要分配动态内存

- 可以在程序运行中一直保留一片内存，直到我们自己释放
- 可以在需要的时候就能申请一片内存
- 全局变量缺点：所有函数都能用，申请的内存也是固定的

分配动态内存操作符：new

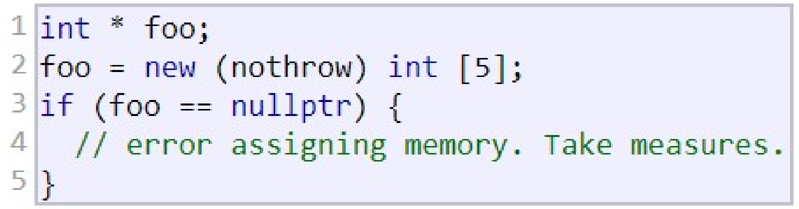
```
type *variable = new type; // allocate one element
```
or
```
type *variable = new type[n]; // allocate n elements
```

```
int *anInteger = new int; // create one integer on the heap

int *tenInts = new int[10]; // create 10 integers on the heap
```

第二个例子直接分配了一个大小为10的int类型的连续内存块，也就是数组。

分配内存出错

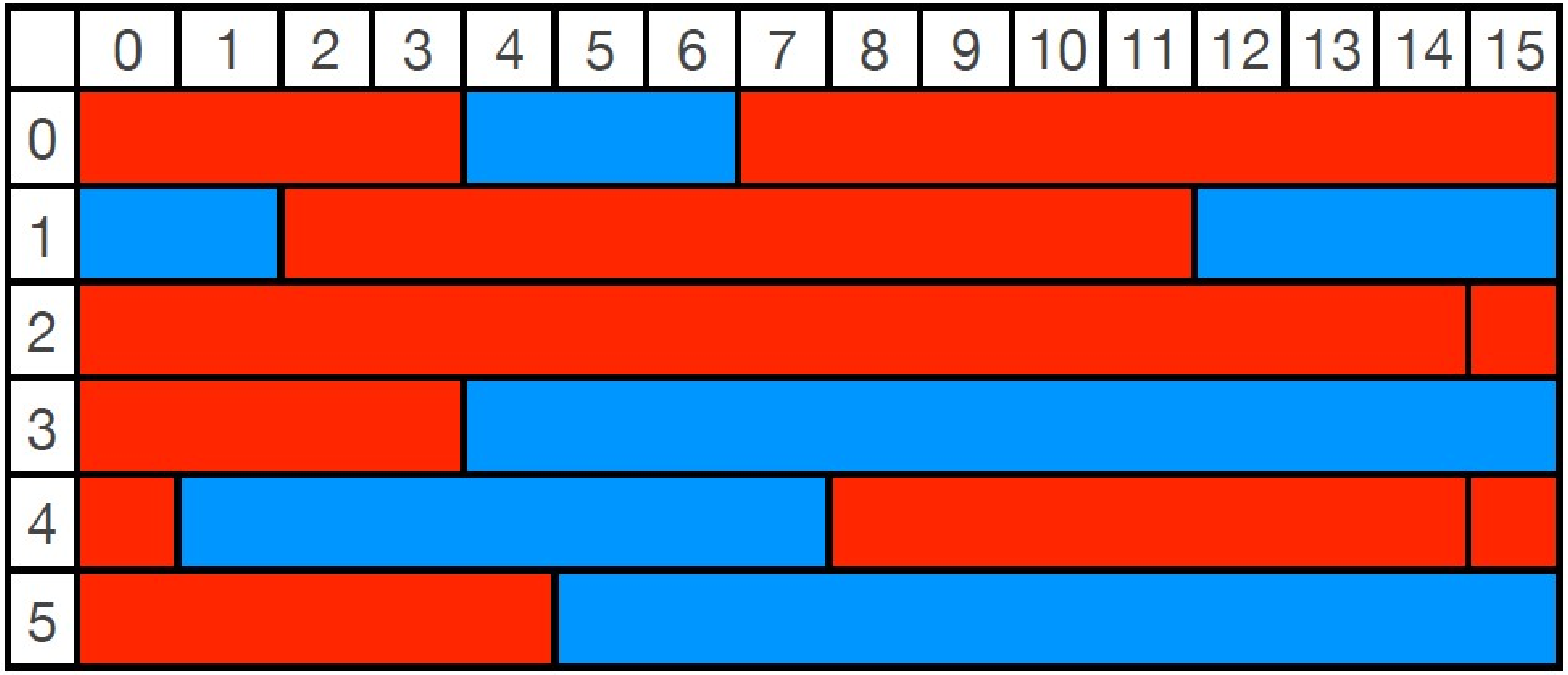
- 如果不能分配内存，就是throw exception
- 除非用nothrow



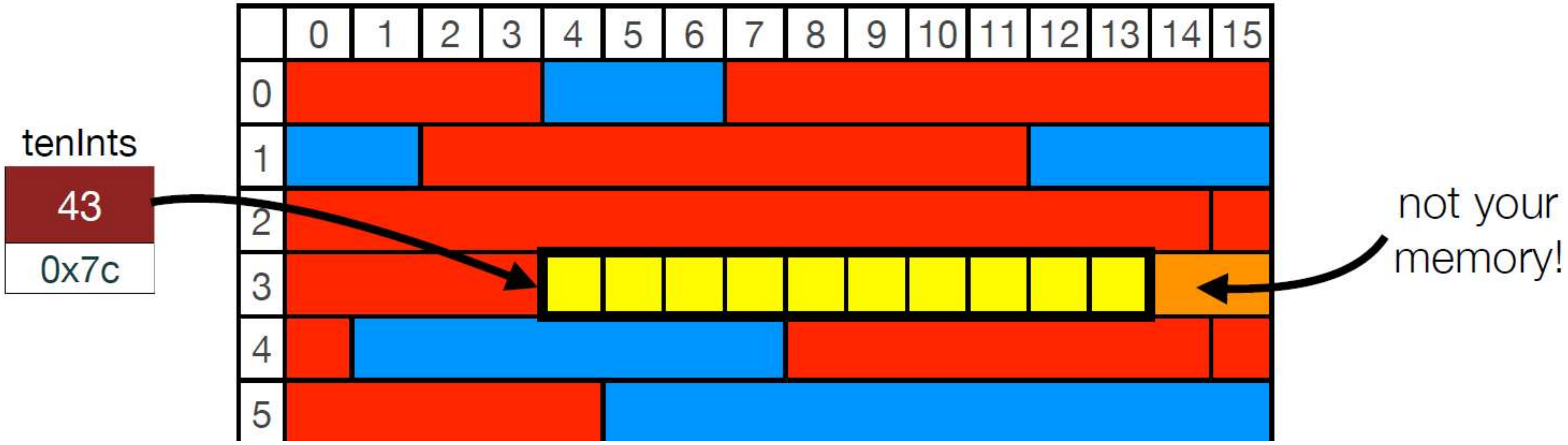
```cpp
int * foo;
foo = new (nothrow) int [5];
if (foo == nullptr) {
  // error assigning memory. Take measures.
}
```

```cpp
int firstArray[10]; // create a static array on the stack;
                    // size of 10 is known at compile time

int *secondArray = new int[10]; // create 10 integers on the
                                // heap. Dynamically allocated.
// fill the arrays with values
for (int i=0; i < 10; i++) {
    firstArray[i] = i*2; // evens
    secondArray[i] = i*2 + 1; // odds
}
```

```
int *tenInts = new int[10]; // create 10 integers on the heap
```

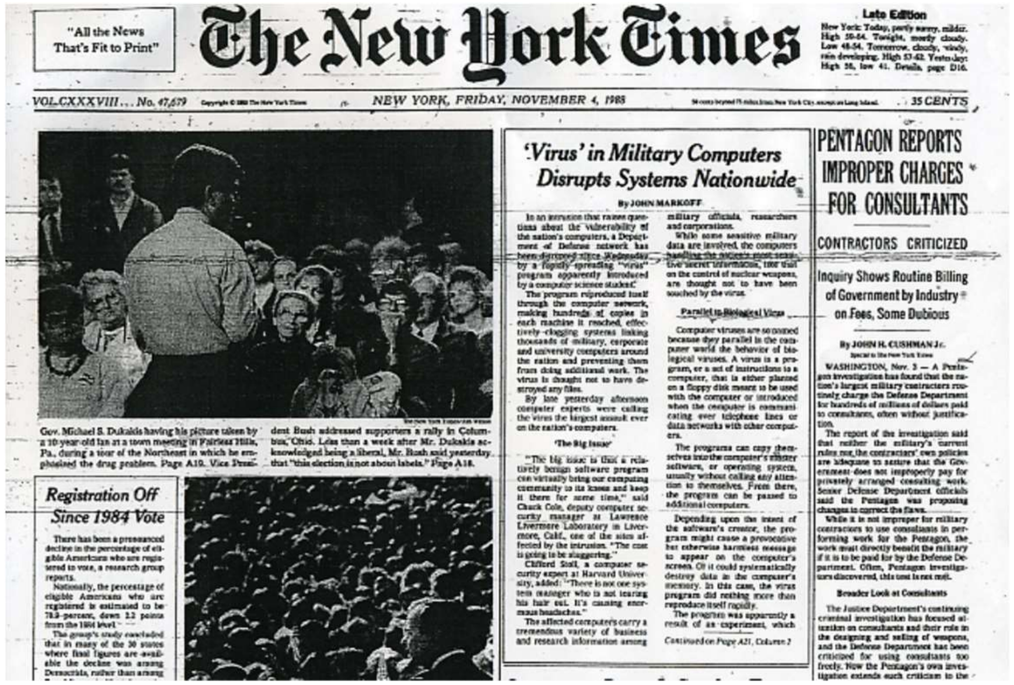


```
int *tenInts = new int[10]; // create 10 integers on the heap
```



Buffer overflow

- 1988，　worm virus



"All the News That's Fit to Print"

The New York Times

Late Edition

New York: Today, partly sunny, milder. High 50-64. Tonight, mostly cloudy. Low 48-54. Tomorrow, cloudy, windy, rain developing. High 57-62. Yesterday: High 58, low 41. Details, page D16.

VOL.CXXXVIII . . . No. 47,679 NEW YORK, FRIDAY, NOVEMBER 4, 1988 35 CENTS

'Virus' in Military Computers Disrupts Systems Nationwide

By JOHN MARKOFF

In an intrusion that raises questions about the vulnerability of the nation's computers, a Department of Defense network has been disrupted since Wednesday by a rapidly spreading "virus" program apparently introduced by a computer science student.

The program reproduced itself through the computer network, making hundreds of copies in each machine it reached, effectively clogging systems linking thousands of military, corporate and university computers around the nation and preventing them from doing additional work. The virus is thought not to have destroyed any files.

By late yesterday afternoon computer experts were calling the virus the largest assault ever on the nation's computers.

'The Big Issue'

"The big issue is that a relatively benign software program can virtually bring our computing community to its knees and keep it there for some time," said Chuck Cole, deputy computer security manager at Lawrence Livermore Laboratory in Livermore, Calif., one of the sites affected by the intrusion. "The cost is going to be staggering."

Clifford Stoll, a computer security expert at Harvard University, added: "There is not one system manager who is not tearing his hair out. It's causing enormous headaches."

The affected computers carry a tremendous variety of business and research information among military officials, researchers and corporations.

While some sensitive military data are involved, the computers handling the nation's most sensitive secret information, like that on the control of nuclear weapons, are thought not to have been touched by the virus.

Parallel to Biological Virus

Computer viruses are so named because they parallel in the computer world the behavior of biological viruses. A virus is a program, or a set of instructions to a computer, that is either planted on a floppy disk meant to be used with the computer or introduced when the computer is communicating over telephone lines or data networks with other computers.

The programs can copy themselves into the computer's master software, or operating system, usually without calling any attention to themselves. From there, the program can be passed to additional computers.

Depending upon the intent of the software's creator, the program might cause a provocative but otherwise harmless message to appear on the computer's screen. Or it could systematically destroy data in the computer's memory. In this case, the virus program did nothing more than reproduce itself rapidly.

The program was apparently a result of an experiment, which

Continued on Page A21, Column 2

PENTAGON REPORTS IMPROPER CHARGES FOR CONSULTANTS

CONTRACTORS CRITICIZED

Inquiry Shows Routine Billing of Government by Industry on Fees, Some Dubious

By JOHN H. CUSHMAN Jr.

Special to The New York Times

WASHINGTON, Nov. 3 — A Pentagon investigation has found that the nation's largest military contractors routinely charge the Defense Department for hundreds of millions of dollars paid to consultants, often without justification.

The report of the investigation said that neither the military's current rules nor the contractors' own policies are adequate to assure that the Government does not improperly pay for privately arranged consulting work. Senior Defense Department officials said the Pentagon was proposing changes to correct the flaws.

While it is not improper for military contractors to use consultants in performing work for the Pentagon, the work must directly benefit the military if it is to be paid for by the Defense Department. Often, Pentagon investigators discovered, this test is not met.

Broader Look at Consultants

The Justice Department's continuing criminal investigation has focused attention on consultants and their role in the designing and selling of weapons, and the Defense Department has been criticized for using consultants too freely. Now the Pentagon's own investigation extends such criticism to the

Gov. Michael S. Dukakis having his picture taken by a 10-year-old fan at a town meeting in Fairless Hills, Pa., during a tour of the Northeast in which he emphasized the drug problem. Page A10. Vice President Bush addressed supporters a rally in Columbus, Ohio. Less than a week after Mr. Dukakis acknowledged being a liberal, Mr. Bush said yesterday that "this election is not about labels." Page A18.

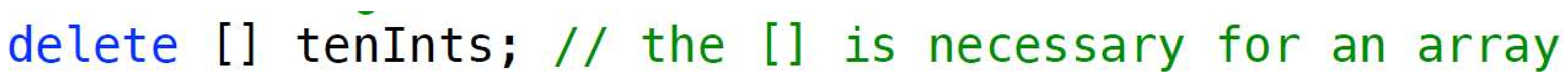
Registration Off Since 1984 Vote

There has been a pronounced decline in the percentage of eligible Americans who are registered to vote, a research group reports.

Nationally, the percentage of eligible Americans who are registered is estimated to be 70.9 percent, down 2.2 points from the 1984 level.

The group's study concluded that in many of the 30 states where final figures are available the decline was among Democrats, rather than among

delete



```
delete [] tenInts; // the [] is necessary for an array
```

- 释放内存
- 注意不是删除变量，而是释放变量指向的内存
- 不能再使用
  - Int * pBuffer = new int[10];
  - Delete [] pBuffer;
  - pBuffer = new int [20];
  - …..
  - Delete [] pBuffer;

一种方法：

```
const int INIT_CAPACITY = 10000000;

class Demo {
public:
    Demo(); // constructor
    string at(int i);
private:
    string *bigArray;
};

Demo::Demo() {
    bigArray = new string[INIT_CAPACITY];
    for (int i=0;i<INIT_CAPACITY;i++) {
        bigArray[i] = "Lalalalalalalalala!";
    }
}
```

```
string Demo::at(int i) {
    return bigArray[i];
}

int main()
{
    for (int i=0;i<10000;i++){
        Demo demo;
        cout << i << ": " << demo.at(1234) << endl;
    }
    return 0;
}
```

```
Demo::~Demo() {
    delete[] big_array;
}
```