#第一节

-1. 死循环 for(;;);

#第二节

-1. object = entity 就是东西，it may be visible or invisible, in programming languages object is variable. 对象就是变量

-2. 对象 = 属性 + 服务 ； 内置有属性的数据，外置有对外的功能和接口

-3. 面向过程：按时间顺序； 面向对象：存在什么东西来分析；

-4. class 就是高级的struct，struct只有数据，而class不只有数据还有操作

#第三节

-1. 对象与对象之间的交互是消息，对象的状态不由外力直接改变的，而是让对象自己做不做动作

-2. 所有东西都可以通过总结他们相同的特征来归为一类，类定义了对象长什么样，对象是一个那个类的东西或实例

-3. OOP Characteristics

1)Everything is an object.

2)A program is a bunch of objects telling each other what to do by sending messages.(而不是Howtodo)

3)Each object has its own memory made up of other objects.

4)Every object has a type.(fish is fish, bird is bird.)

5)All objects of a particular type can receive the same messages.(所有可以接收相同消息的对象可以被分为同一类，从个体的同性即对象的接口)

-4. 接口的作用：消息交流和保护作用，就是松耦合

-5. 内部的具体实现需要保护起来

-6. Encapsulation封装

Bundle data and methods dealing with these data together in an object.

Hide the details of the data and the action.

Restrict only access to the publicized methods.

#第四节

-1. 每一个类的定义都建议定义成两个文件：一个头文件.h和一个body文件.ccp

头文件里存放这个类的所具有的函数原型（包括构造函数和析构函数）、还有声明数据成员

-2. 两个连着的冒号叫做域的解析符

#第五节

-1. Definition of a class

Separated .h and .cpp files are used to define one class.

Class declaration and prototypes in that class are in the header file(.h).

All the bodies of these functions are in the source file(.cpp).

-2. the header files

If a function/class is declared in a header file, you must include the header

file everywhere the function/class is used and where the function/class is defined.

-3. the class header files are telling you how to used and what you can use.

-4. #include

#include is to insert the included file into the .cpp file at where the #include statement is.

#include "xx.h" : first search in the current directory, then the directories declared somewhere.

#include <xx.h> : search in the specified directoires, like "includes" or other 系统目录

#include <xx> : same as #include <xx.h>

-5. Standard header file structure

#ifndef HEADER\_FLAG

#define HEADER\_FLAG

// type declaration here...

#endif //HEADER\_FLAG

-6. Tips for header

One class declaration per header file;

Associated with one source file in the same prefix of file name;

The contents of a header file is surrounded with #ifndef #define #endif

#第六节

-1. Abstract

Abstraction is the ability to ignore details of parts to focus attention on a higher level of a problem.

-2. 在做抽象时尽量在没有或者很少代价的情况下为后来的改进入口做铺路

#第七节

-1. Fields, Parameters, local variables

All three kinds of variable are able to store a value that is appropriate to their defined type;

Fields are defined outside constructors and methods;

Fields are used to store data that persists throughout the life of an object.

As such, they maintain the current state of an object. They have a lifetime that lasts as long as their object lasts;

Fields have class scope: their accessibility extends throughout the whole class, and so they can be used within any of the

constructors or methods of the class in which they are defined.

-2. Fields 就是字段，就是类的成员变量

#第八节

-1. this : the hidden parameter

"this" is a hidden parameter for all member functions, with the type of the class;

void Point::print()

->(can be regarded as)

void Point::print(Point \*p)

-2. this : pointer to the caller

Inside member functions, you can use this as the pointer to the cariable that calls the function.

"this" is a natural local variable of all class member functions that you can not define, but can use it directly.

#第九节

-1. 构造函数: Guaranteed initialization with the constructor

If a class has a constructor, the compiler automatically calls that constructor at the point an object is created,

before clinet programmers can get their hands on the object;

The name of the constructor is the same as the name of the class.

-2. 构造函数没有返回类型，但可以有参数，并且当被定义一个对象时就会被调用；

-3. 析构函数: The destructor，在这个对象要结束时被调用，不能有参数，而且不能有返回类型

In C++, cleanup is as important as initialization and is therefore guaranteed with the destructor;

The destructor is named after the name of the class with a leading tilde(~). The destructor never has any arguments.

#第十节

-1. 即使是C语言也可以在随意地方定义新变量；

-2. 空间是进了大括号的时间就分配了，但是构造函数是在运行到那一行的时候才会运行；

-3. 没有构造的时候是不能够析构的；

-4. 以下的是一种构造函数，需要一个初始化变量输入a

struct Y { float f; int i; Y(int a) };

Y y1[] = {Y(1), Y(2), Y(3) }; //ok

Y y2[2] = { Y(1) }; // not ok

#第十一节

-1. new与delete

new int; new Stash;// Stash是一个类名称，这里调用构造函数 new int[10];

delete p; delete[] p;

-2. Tips for new and delete

> Don't use delete to free memory that new didn't allocate;

> Don't use delete to free the same block of memory twice in succession;

> Use delete [] if used new [] to allocate an array;

> Use delete (no brackets) if used new to allocate a single entity;

> It's safe to apply delete to the null pointer (nothing happens);

#第十二节

-1. 类与结构对比：class defaults to private; struct defaults to public;

#第十三节

-1. Initialization list 与 assignment的区别

> Student::Student(string s):name(s) {} //initialization, before constructor, 初始化，推荐都用这个

> Student::Student(string s) { name = s;} //assignment, 赋值 = 初始化 + 赋值

#第十四节

-1. 组合：construct new object with existing objects, it is the relationship of "has-a"，就是用已有的对象拼出一个新的对象

#第十五节

-1. 继承：Inheritance is to take the existing class, clone it, and then make additions and modifications to the clone，就是用类去拼出新的类

-2. Inheritance allows sharing of design for Member data\ Member functions\ Interfaces接口就是public那些

-3. 只要private中有成员变量就要在构造函数中用初始化列表的方式初始化：

class A{

public:

A():i(0) { cout<<"A::A()"<<endl; }

~A() { cout<<"A::~A()"<<endl; }

private:

int i;

}

#第十六节

-1. 尽量用已有的代码来写新的代码；

-2. 子类继承父类后，子类声明了对象会调用父类的构造函数；另外父类的构造函数被调用时初始化时要用initialization list来初始化

-3. 顺序是：父类构造、子类构造、子类析构、父类析构

#第十七节

-1. To define a function with an arguement list, defaults must be added from right to left.

int harpo( int n, int m=4, int j=5 );

int chico( int n, int m=6, int j); //illeagle

-2. 尽量不要用default values

#第十八节

-1. 每一个程序都是一个堆栈，栈里面有两个东西，一个是本地变量，一个是返回地址；

-2. inline函数在头文件和主文件内都要加关键字；

-3. 使用inline时要把函数的body放在使用的该函数的主文件里头；

-4. Tradeoff of inline functions

> Body of the called function is to be inserted into the caller;

> This may expand the code size;

> But deduces the overhead of calling time;

> So it gains speed at the expenses of space;

> In most cases, it's worth;

> It is much better than macro in C, it checks the types of the parameters;

-5. 递归不能用inline

-6. inline or not：

> Inline:

>> Small functions, 2 or 3 lines;

>> Frequency called functions, e.g. inside loops

> Not inline:

>> Very large functions, more than 20 lines

>> Recursive functions

#第十九节

-1. pointers and const

> char \* const q = "abc"; // q is const

\*q = 'c'; // OK

q++; // Error

> const char \*p = "ABCD";

// (\*p) is a const char

\*p = 'b'; // ERROR

p++; // OK

-2. const写在 \* 的前面就是对象是const，写在 \* 后面就是指针是const；

-3. 传一个参数的地址进一个函数比传一个参数的值更加方便省时间，但是最好要加const

-4. char \*s = “hello world”; //这是一个const

char s[] = “hello world”; //这不是const

#第二十节

-1. 如果一个对象前有const，那么此对象的所有值不能被修改；

-2. 一般的面向对象编程编写有三个文件夹，第一个是声明(.h)，第二个是函数的body(.cpp)，第三个是主程序(.cpp)

-3. Const member function usage

> repeat the const keyword in the definition as well as the declaration:

>> int get\_day() const;

>> int get\_day() const {return day};

> Function members that do not modify data should be declared const;

> const member functions are safe for const objects

-4. 有相同的函数名和参数一致的情况下，若带const结尾的函数则使用优先，并且函数重载；

-5. 在一个函数后面有const说明这个函数不修改成员变量，在原型和定义都要加const

int get\_day() const;

int get\_dat() const { return day; }

意味着this is const

-6.

const int size1;

int array[size1]; //error

enum { size2 = 100};

int array[size]; //ok

#第二十一节

-1. Declaring references

> char c;

> char \*p = &c; // a pointer to a character 要习惯把星号贴到p前

> char &r = c; // a reference to a character

可以认为 r 是 c 的别名，当我们想用 c 时我们可以用 r ，我们永远可以把 c 和 r 当成等效

-2. pointers vs. references

> References:

>> can't be null;

>> are dependent on an existing variable, they are an alias for an variable;

>> can't change to a new "address" location;

> Pointers:

>> can be set to null;

>> pointer is independent of existing objects;

>> can change to point to a different address;

-3. 没有reference的reference

int x;

int &a = x; // ok

int &b = a; // not ok

int &b = x; // ok too

-4. cpp数据可以放在堆栈、堆、全局数据区等等

-5. 引用必须要初始化，bindings don’t change at run time, unlike pointers

-6. int x = 3;

int &y = x;

const int &z = x; //不能用z去修改x

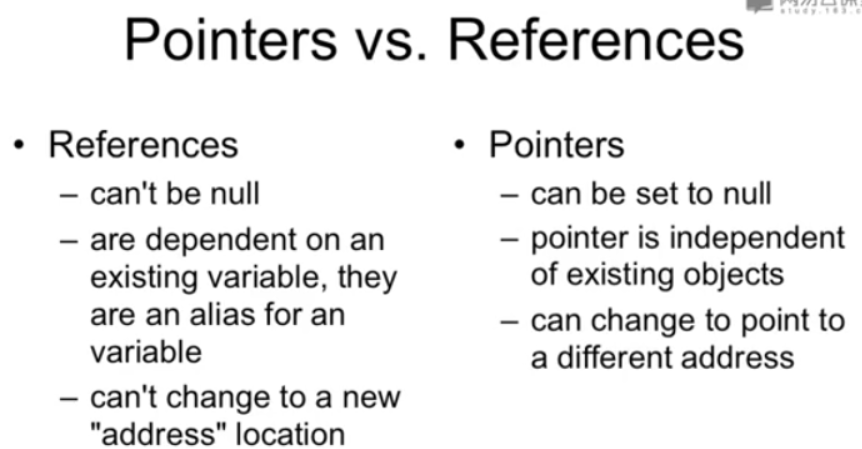
-7. the target of a reference must have a location !

look:

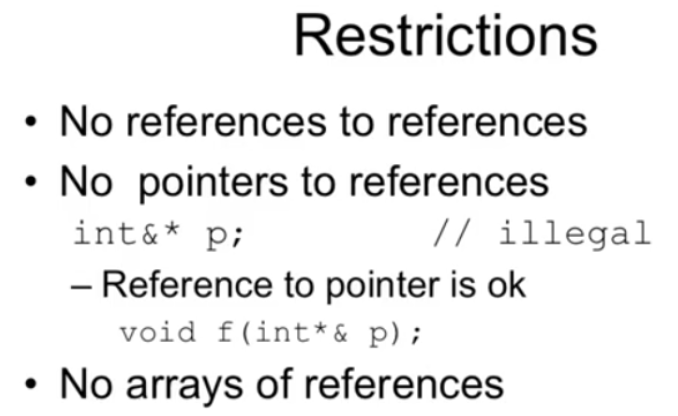
void fun(int &);

fun (i \*3); // error, because i\*3 hasn’t a location

-8.



-9.

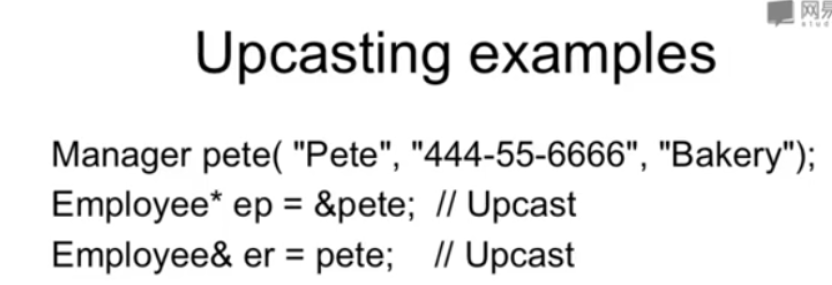


#第二十二节

-1. 向上模型：Upcasting

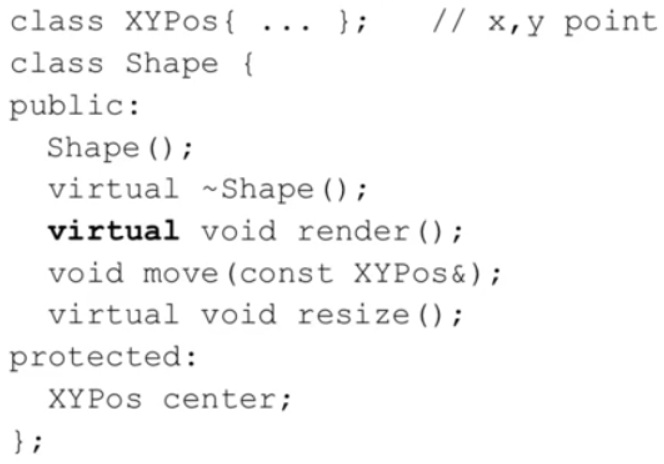
Upcasting is the act of converting from a Derived reference or pointer to a base class reference or pointer.

-2.

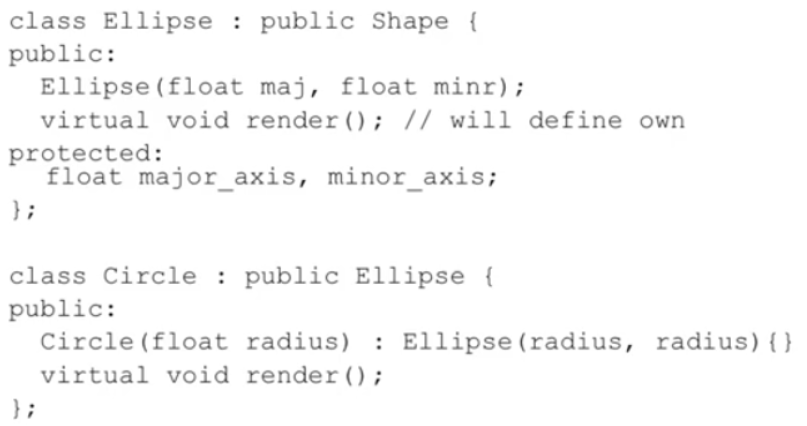


#第二十三节

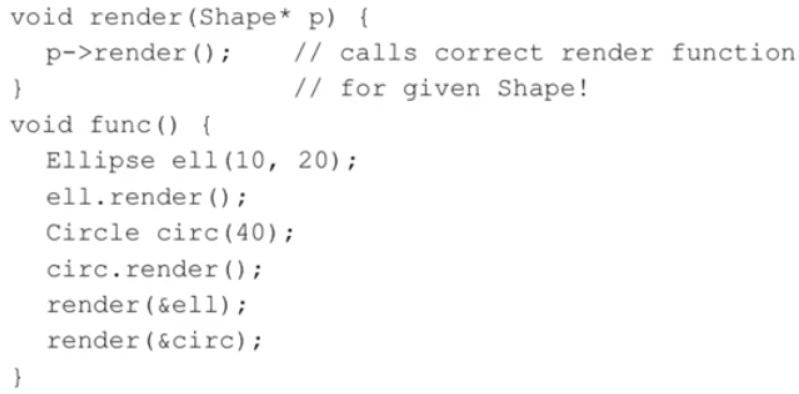
-1. 关于virtual 的应用，若此类的子类有同名的函数（就像重载那样），那么这个类的函数和子类的函数是有联系的



前代里面有定义带virtual的函数，子类后辈相同的函数名的函数无论是否写virtual均是virtual的，因此推荐写上



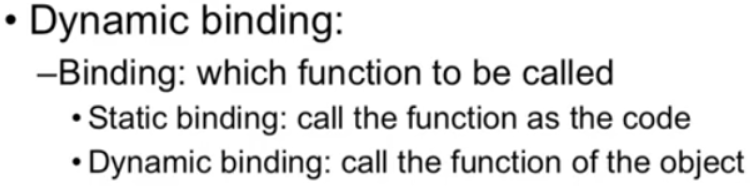
-2. 多态性/多态对象



// p就是一个多态对象

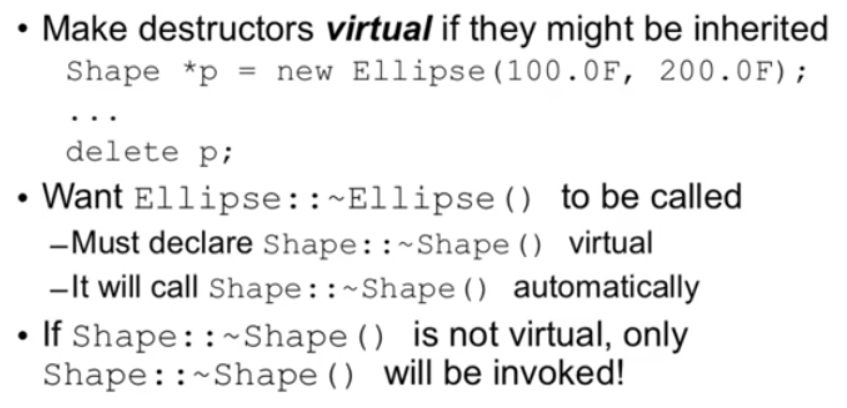
-3. upcast： take an object of the derived class as an object of the base one.//就像上面的Ellipse可以被当成shape使用

-4. 动态绑定

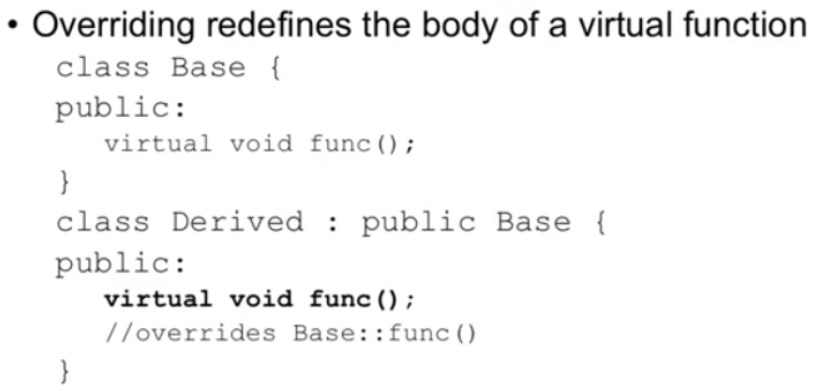


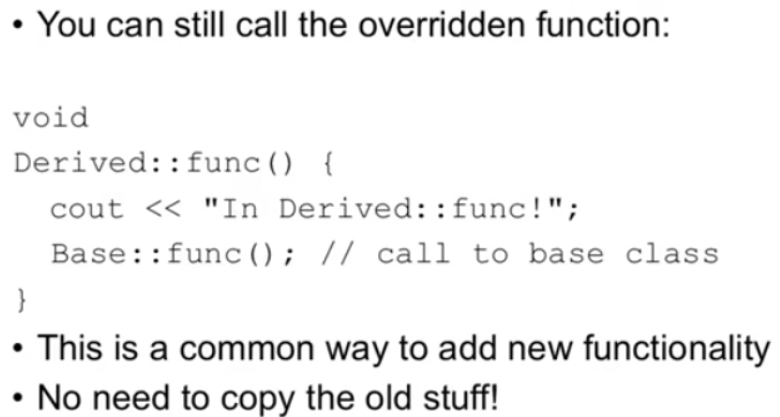
#第二十四节

-1. 如果一个类内有函数存在virtual，那么它的析构函数也一定要是virtual

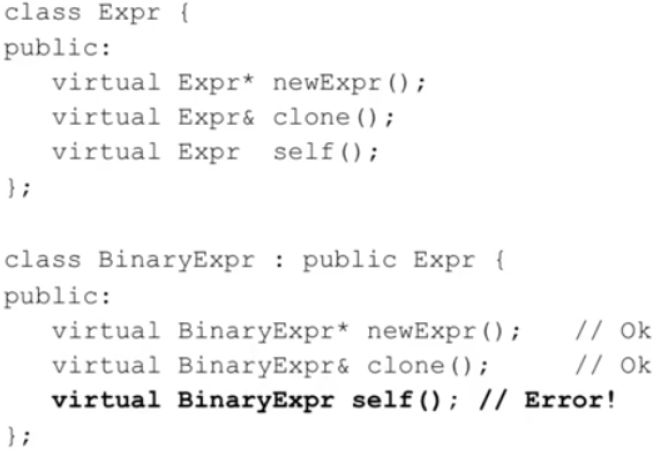


-2. 覆盖overriding



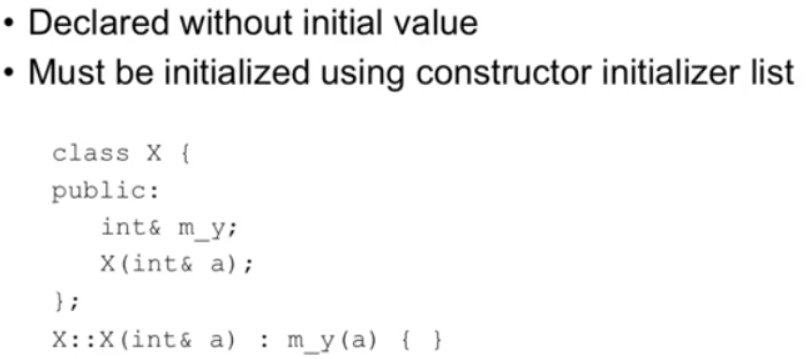


-3. Relaxation example// 因为只有通过指针或引用才能引起upcast的关系

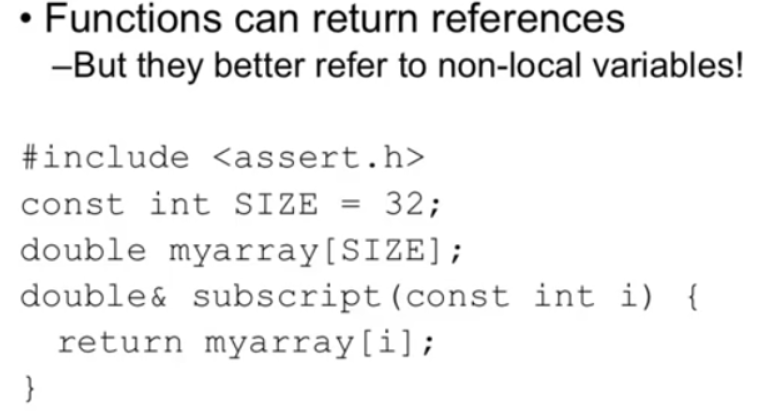


#第二十五节

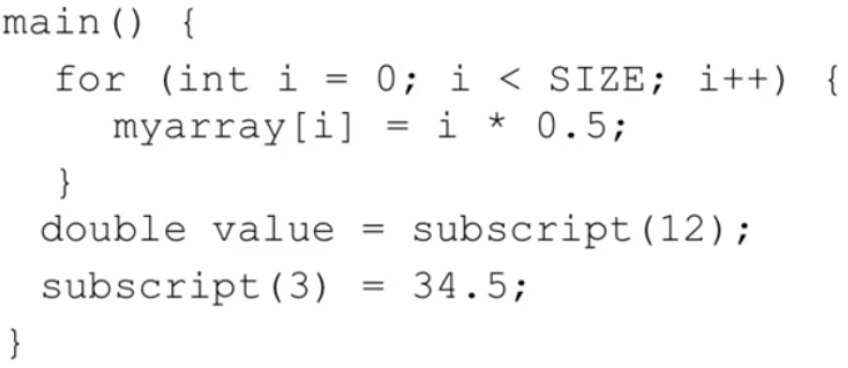
-1. references as class members



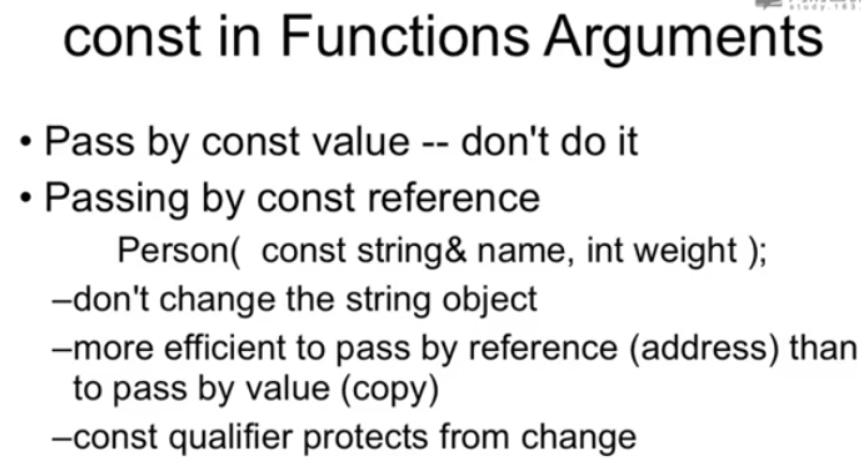
-2. 当一个函数要返回一个引用时，在return那里是一个直接的变量，不用星号或其他



一个返回值是reference的函数可以做为左值



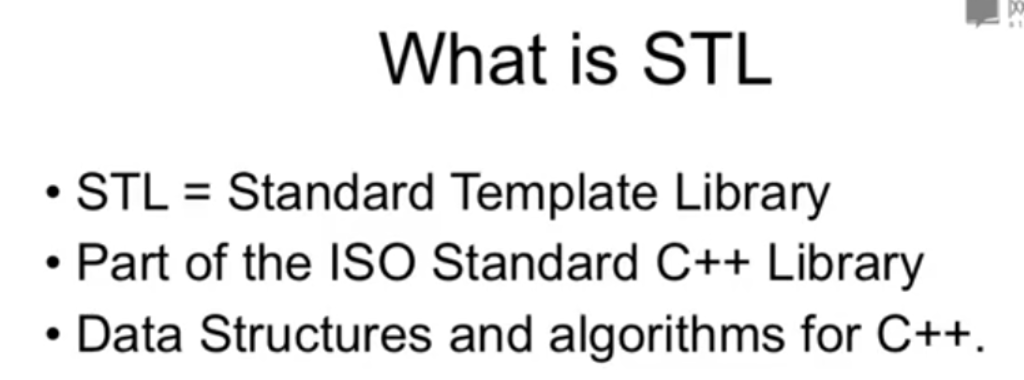
-3. 传入参数进入函数的推荐！



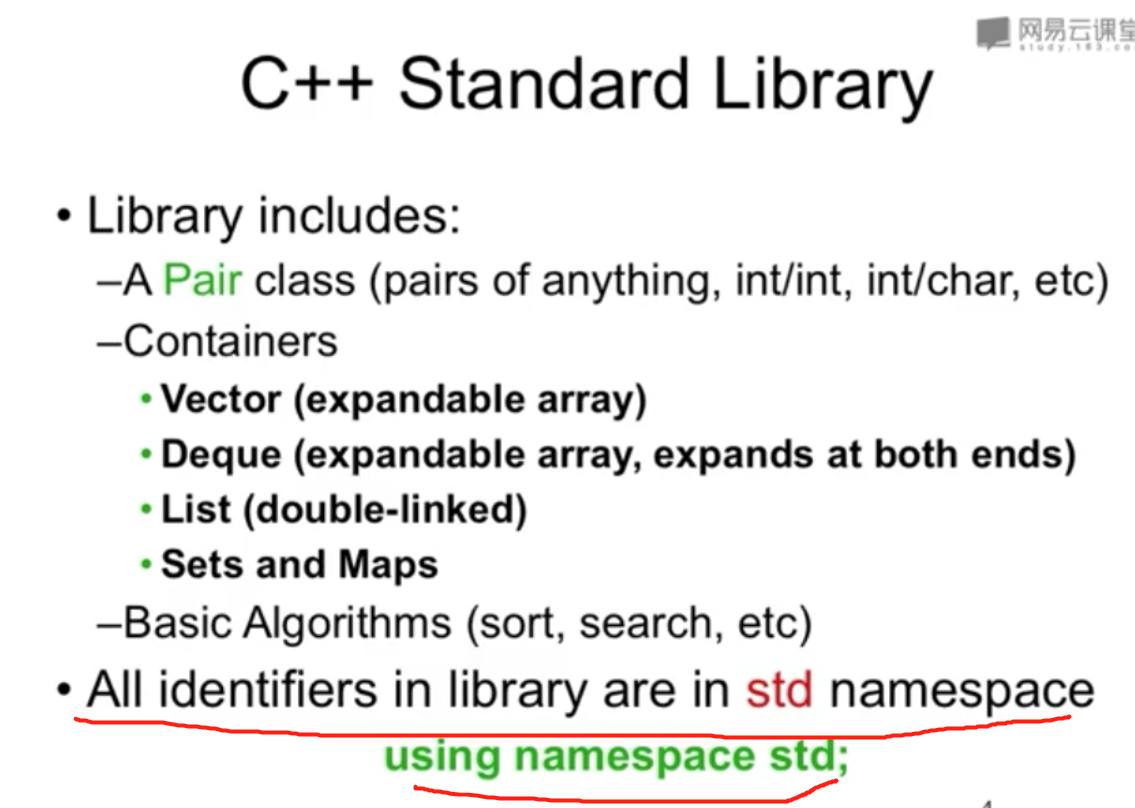
-4. 当一个函数返回的是一个对象时，有可能可以被充当左值

#第四十一节

-1.



-2.

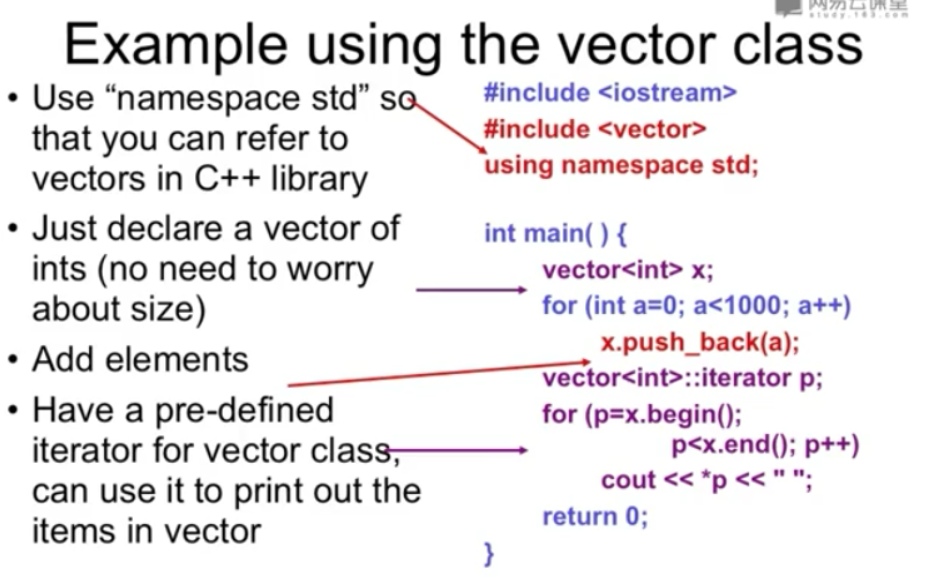


-3.

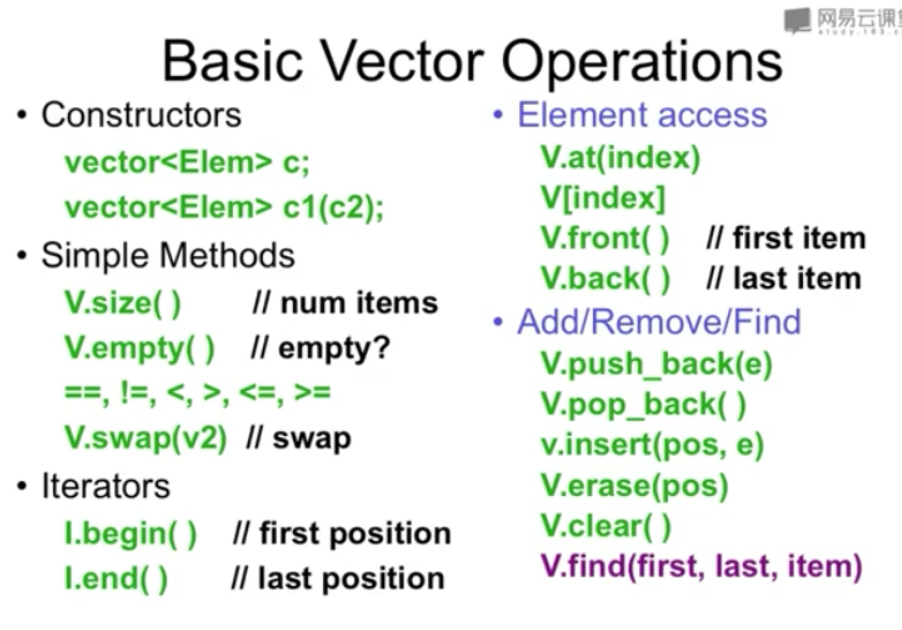
The three parts of STL: containers\algorithms\Iterators

Top 3 data structures: map\vector\list

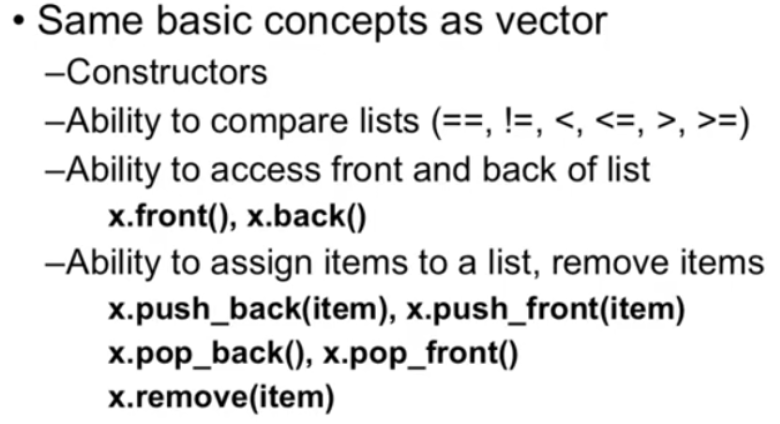
-4. 这里第二个for最好也要用用“!=”来判断



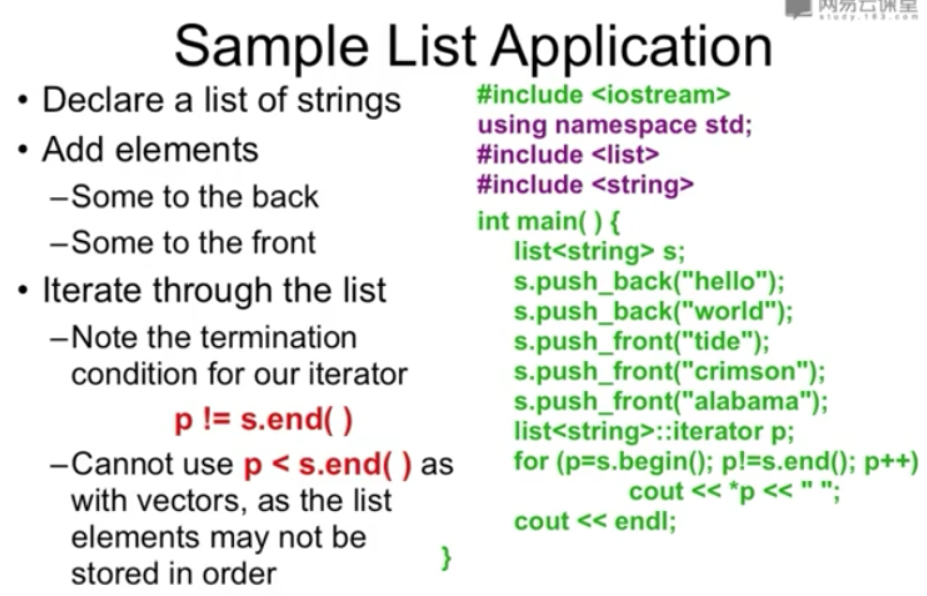
-5.



-6. List class



-7. 注意这里要用不等于！！



-8. map的用法

#include <map>

#include <string>

using namespace std;

map<string, float> price;

price[“apple”] = 0.75;

price[“coke”] = 0.42;

string item;

double total = 0;

while( cin>>item )

{ total += price[item]; }