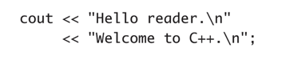
基础部分

#include必须放在最开头

#include <iostream>输入输出流，包含了有关输入输出的函数

 看起来就像右边这个语句<<流入了cout



<<插入运算符，cout，输出流对象，我们把这句话插入到cout输出流里

用命名空间std,如果你不加就要//std就是standard ，标准命名空间



>>提取运算符

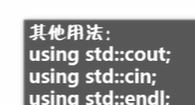


Endl相当于一个回车就是\n

：：是从属关系，cout是属于std的，你不加就要自己写

命名空间有点像家族的名称，代表这些东西是一个系列的，好处就是避免名称的重复

命名空间相当于java里的Package

，你要是不想写全部的，就要一个一个引用

C++中函数怎么写

int Af(int); 第一个int是返回类型，第二个Af是函数名字，第三个Int是参数类型

lecture1

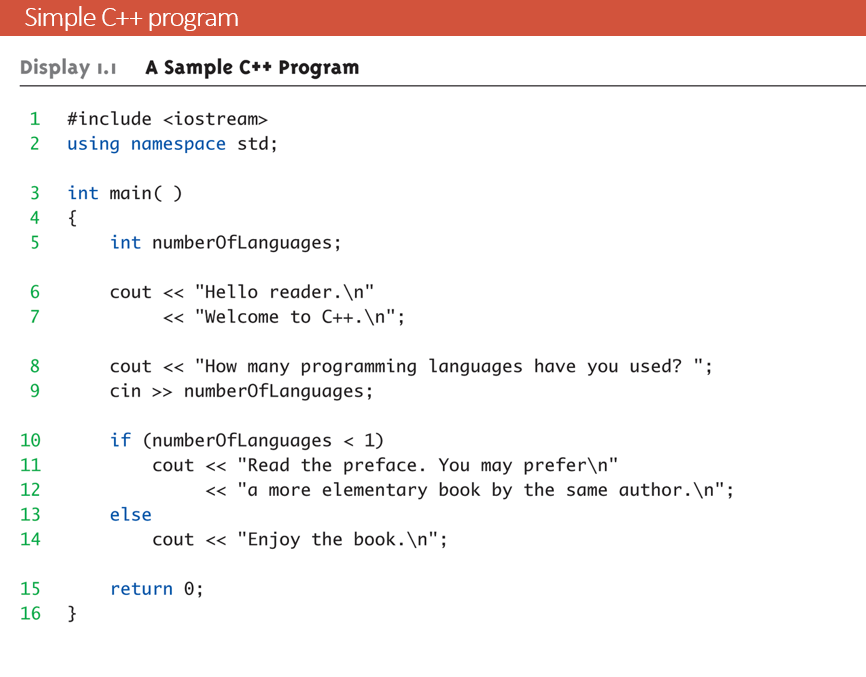
c++是C的superset

main() function是driver

一个C的的程序就是一个有效的C++程序

一个encapsulate封装其他function的class叫做object-oriented

那些不被封装的function叫做free function，例如main()



只有最简单的程序是一个file就可以运行的 ：self-contained

绝大部分程序都需要用其他文件里所描述的library，

Compiler会分别compile library 与program的code，然后用Linker来Link他们

与java不同，一个c++file可以包含多个class/function/data structure

这就导致了一个Physical design的技巧：结构设计，决定哪些entities(实体也就是上面三个东西) 需要被group

除了需要用Library，大型program被分为多个文件的原因是

1：speed up compilation 加速编译效率，只用重新compile上次有改动的文件

2：increase organization, decrease browsing time整个项目本身结构优化，更容易浏览，更有逻辑性，

3. facilitate促进 code reuse:把代码分割成不同module，更具重复使用性

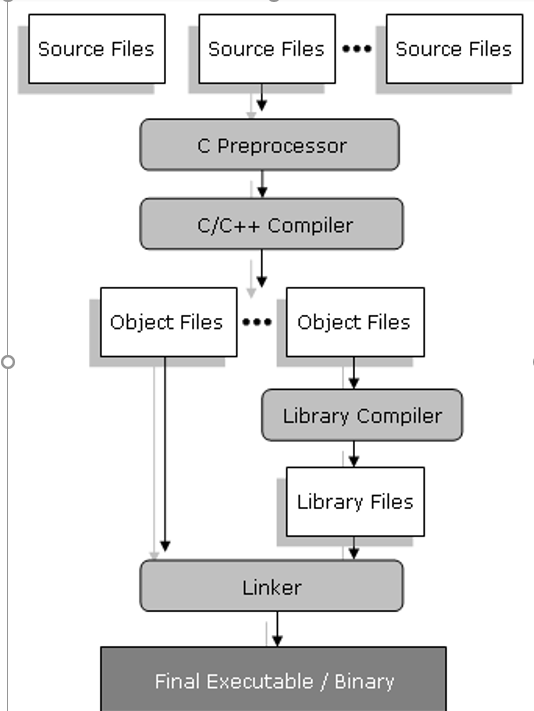
Compilation unit，CPP file

与java不同，一个file可以有多个function，数据结构，或者class

每一个compilation unit （cpp file） 都被独立编译成一个object file

这些object file通常会引用其他的object file

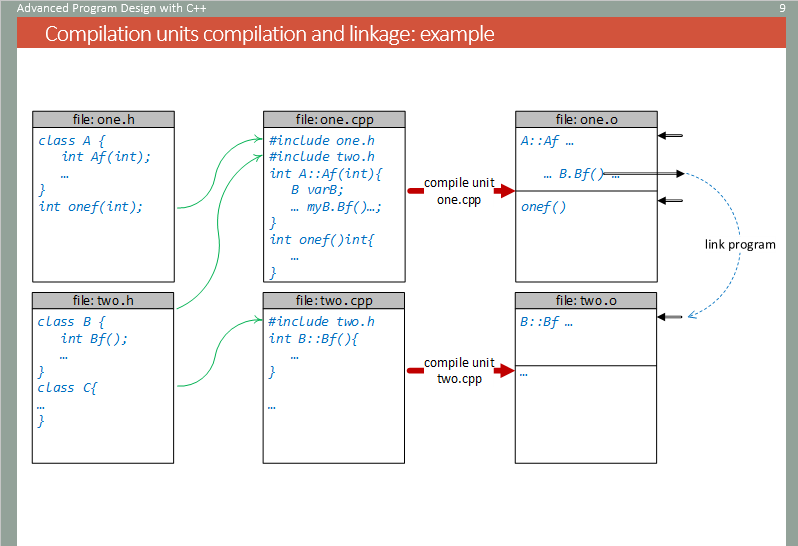
然后，linker尝试解析object file之间的交叉引用，以形成唯一的可执行文件

preprocessor预处理器//读取.h文件 并完整复制到对应位置

然后把不同的cpp file一对一转换成不同的object file

总的来说cpp 是source file，他们会被c++ compiler等编辑成object file，这些object file可能要互相引用，被引用的file就经由library compiler做成library file，再用linker组成最终executable file

看期中复习/博客.docx



可以看见.h 文件被复制到.cpp文件. include实际上就是展开

然后.cpp被转换成.o 文件， 然后Linking program就会比较.o文件， 然后Link起来，形成.exe文件

C++编译过程，

1. 预处理-》编译-》目标文件.o

Souce file到Cprocessor那一步会预处理，生成一个.ii文件

预处理（就是井号include）：#include<iostream> 所包含的.h文件全部包含了进来，编译器在后台生成了16000行代码在我们代码之前

C processor到C compiler哪一步，把.ii文件转成.s文件，也就是汇编文件

汇编文件：可以直接与内存CPU等打交道的计算机可以读取的底层文件

然后.s变成.o也就是目标文件，是二进制了已经

编译，

1. 链接，将object file与C++内置函数库相连，形成最终可执行的.exe文件

o与我们的编译库链接生成.exe文件

Compilation unit cpp文件 是一个程序里的separate part

存储在独立的file

被分别编译然后在连接起来

追求高度cohesive以及不要coupled

Cohesion:凝聚，一组element为了达到一个共同目的组合起来的程度

意思就是要每一个cpp文件都有他自己的意义，模块化

Coupling：一个cpp文件依赖其他cpp文件的程度，如果是high coupling，那么你想要改变一个component，无疑会导致连锁反应

Interface/HEADER FILE

.h 文件：

通常包含多个class的描述以及free functions和operators 的declare

他只是一个接口，忽略了具体的Implementation细节

也就是说.h文件时interface

Implementation file

.cpp文件

包含着对free member/function的具体描述

通用规则：ccp应该与对应的h是一个名字

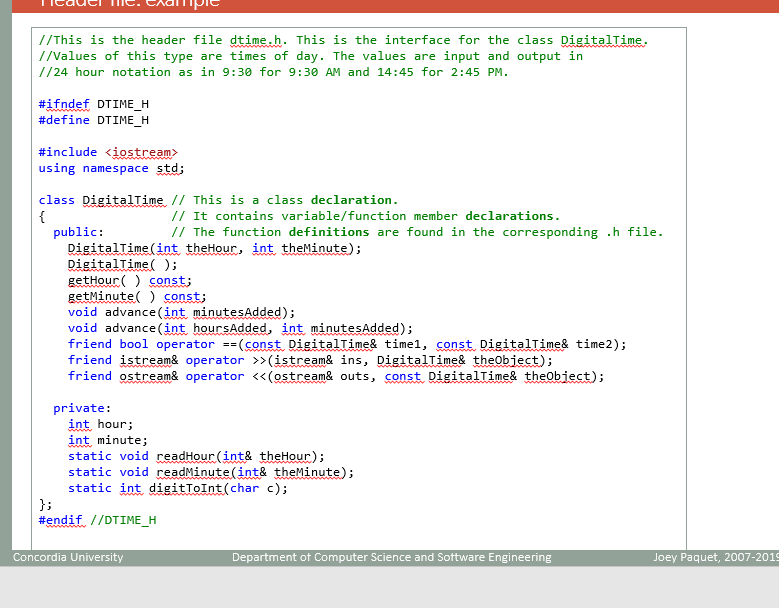
cpp应该#includes对应h

cpp文件永远不应该被#include //因为会被多次declare, obj file无法link 因为两个declaration不知道选哪一个

换句话说: 假设有 辅助cpp, 辅助cpp对应的 h， driver cpp

辅助cpp要include同名h

Driver cpp 要include 辅助.h



这个文件是dtime.h 也就是说他是header file

他是digitalTime的接口

#ifndef DTIME\_H 注意大写

头件的中的#ifndef，这是一个很关键的东西。比如你有两个C文件，这两个C文件都include了同一个头文件。而编译时，这两个C文件要一同编译成一个可运行文件，于是问题来了，大量的声明冲突。

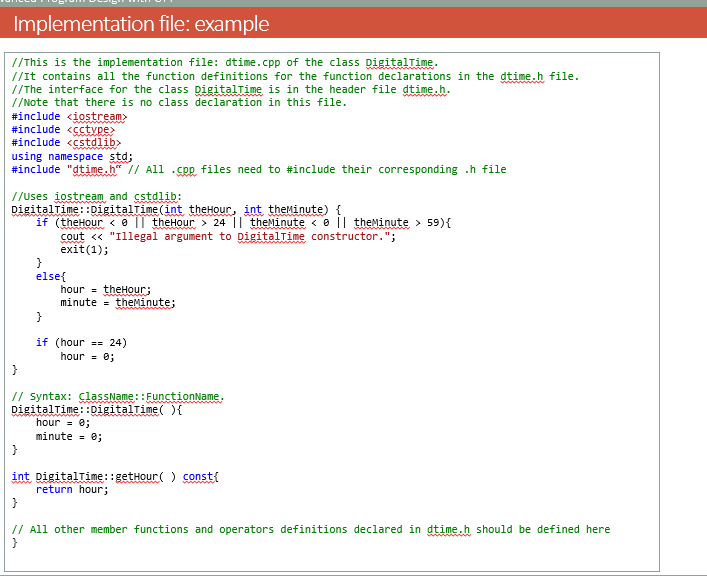
如果还没define,再define这个文件

前面两行是构造器

Const写在函数名面前修饰的是函数返回值，然而想这里写在后面代表他是一个常成员函数，不能修改对象任何成员，只能用只读操作

FRIEND 函数，理论上private 和protected不能从class外访问，，而friend关键字修饰过以后就可以

最后要写#endif



然后我们就要描述对应的function,

第一个构造器构造object

第二个是初级构造器

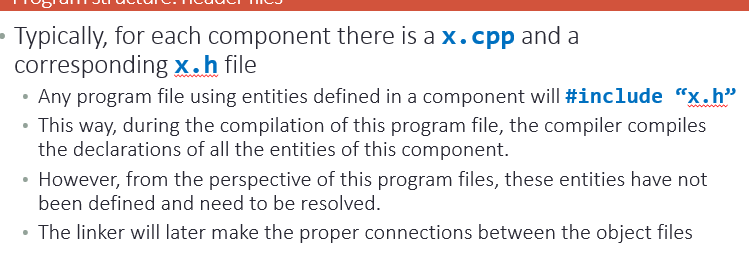
第三个是描述方法getHour()，注意.h文件可以不直接表明方法return的类型，可以去具体描述中再标明

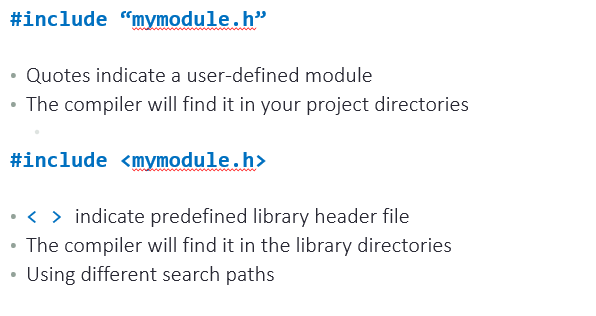
通常来说，每一个cpp文件有一个对应的h文件，任何program file(exe文件)试图用cpp中的entities都会导致x.h的使用

因此，在一个program file编译的时候,编译器会declare (h做的操作，就是声明变量)所有entity//preprocessing， 在.h文件declare，

然而在program file的角度来说，这些entity还没有被具体描述

Inker会合理的链接这些object file并且描述这些entity//然后cpp描述





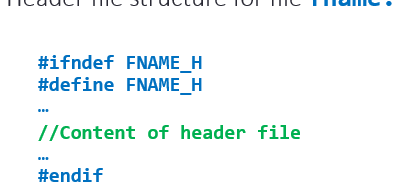
双引号是我们自己的h

书名号是C++内置library

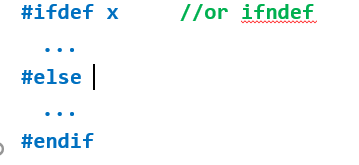
处理多余的#includes

一个header file可能被多个commponent所include

然而我们只能定义一次，不然多次定义造成混淆



也可以用else如果有别的需求



Includes

Header file通常会被include很多次

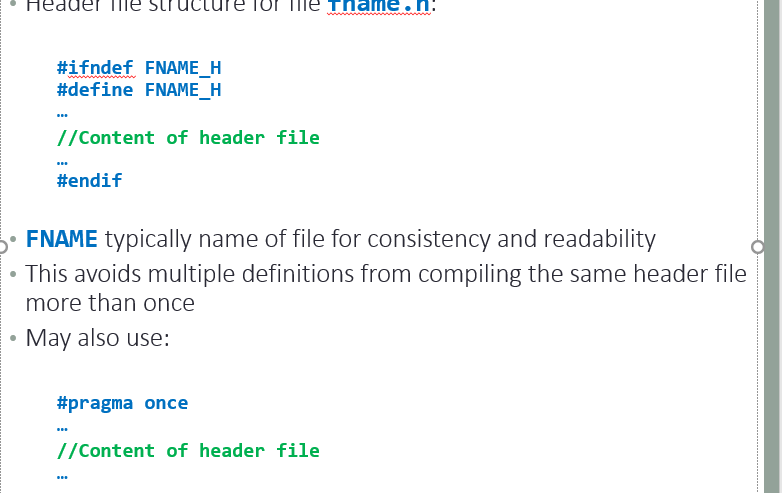
我们只能compile一次

否则会重复define

所以要用ifndef define啥的 来让compiler具体只阅读一次

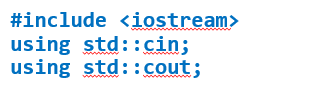
我的理解：

把每个.h都复制黏贴，然后因为if def ndef啥的，有第一次declare, 后面的都跳过了//ide里变成灰色，那么转化的obj file 实际上就只有一个对应function /METHOD/CLASS



两种格式

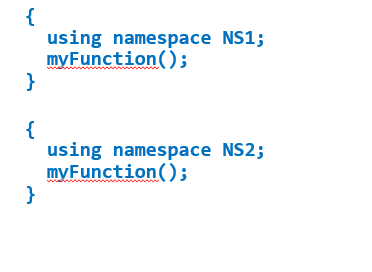
NameSpace ： 就是双冒号前的东西， class,library都可以是namespace，例如xxClass::aFunction 又例如std::cout ,这样不同class哪怕有名字相同的function,实际上也是不一样的，

一个大型的program可能用到许多class与function,有可能重名，namespace就用来解决这个问题，他就相当于一个家族

只要用using就可以直接激活某个namespace

但是我们更提倡使用 than using namespace，因为一旦激活就无法停止，你只能在不同的括号里用不同的namespace

假设NS1 NS2都有myFunction而且不同，我们如果非要用using nameSpace法，就只能花括号， 但最好别这么用，最好的方法是namespace::myFunction()

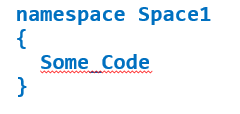
所有的代码都属于某些namespace

除非指定属于某些namespace， 不然就属于global namespace

一个global namespace

1. 不需要用using 这个词
2. 一直available，自动using
3. 但是却关不掉
4. 因此一个global namespace极易造成重名

怎么样创建一个namespace

，然后在some\_code处declare或define这些entities

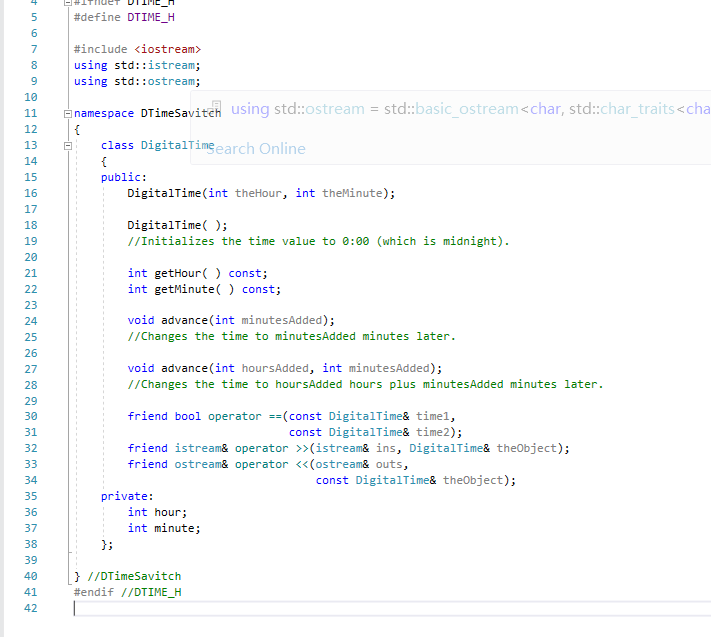
我们可以用这个space的两种方法

激活整个

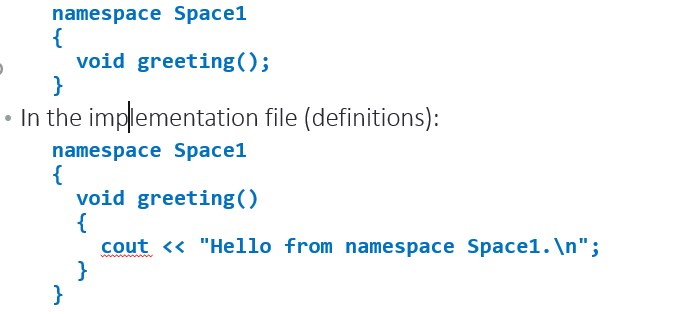
激活某一个

我们应该怎么使用namespace

对于h文件来说，整个包括就行

，

对于cpp file来说



我们首先要用相同的Namespace来描写那些只是declare的entities

但是我们这个cpp可能要使用一些辅助function，这样就可以用一个不命名的namespace

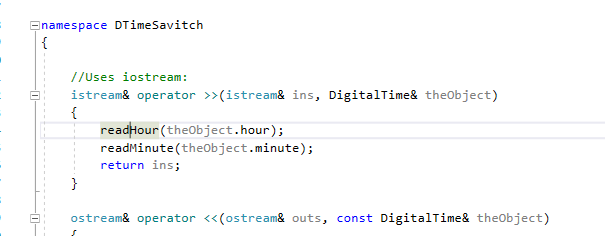
每一个cpp file都有他自己的unnamed namespace，就是namespace {

}

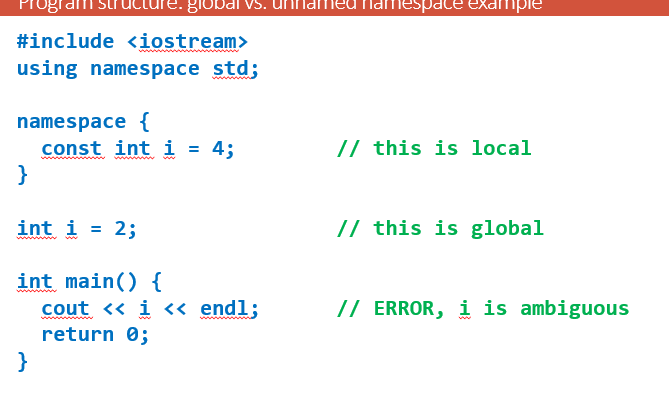
这种，你可以用它作为local，他的范围只在这个cpp



这些function可以在同一cpp的另一cpp中使用

又不影响其他文件，换句话说，unnamed namespace就是Local的compilation unit，他们只对当前cpp可见，可用

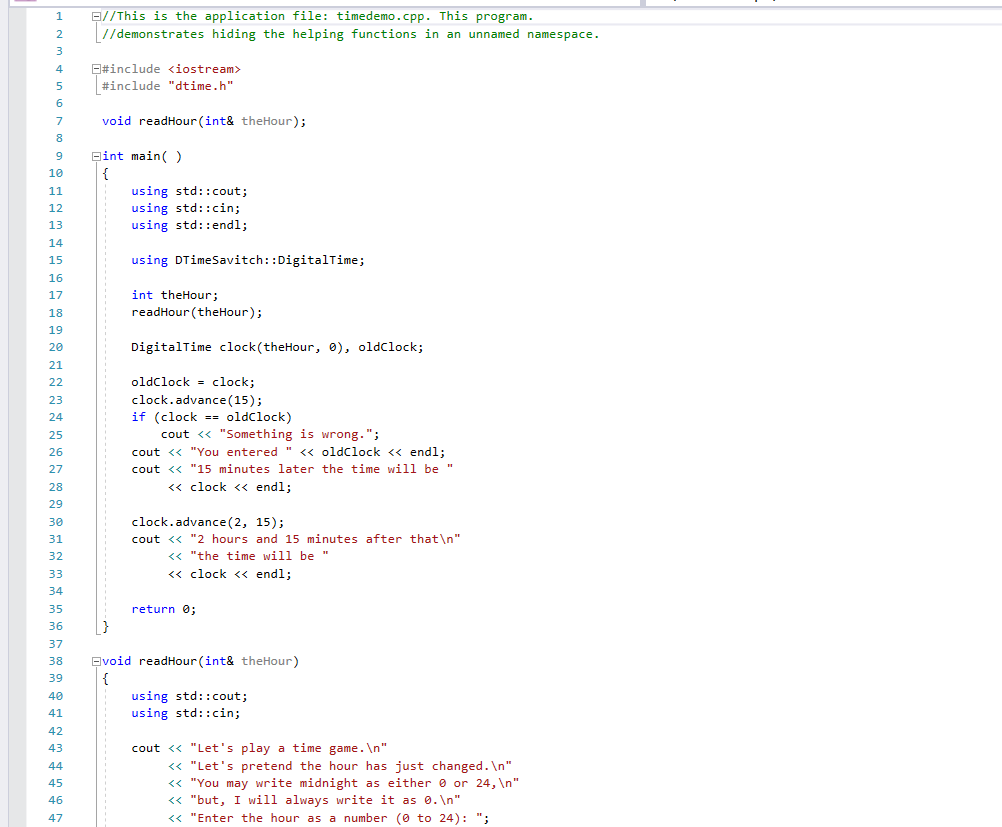
怎么创建global namespace，没有用namespace包裹的entiies



可以看见namespace里面包裹的是local

没用namespace的是global

DRIVER



Driver不需要使用namespace,只要include .h文件， 然后就可以用 username::xx

Lecture2