metaProgramming

metaProgramming 的意思是写一个程序，把程序看作是将用来计算的数据。一个metaprogram是一个可以读写，分析，生成，改变问题的 program //基本，一个可以被加工的Program

metalanguage: metaprogram是用什么语言写的

object language: 一个用来表示被操作的program的语言 //目标语言，

如果metalanguage与object language相同，那么允许写可以自己改变的program，这叫做reflectiver programming

metaprogramming，一直是programming language的一部分，从1960的Lisp开始

reflective programming从1970年开始，通过开发系统例如small talk

当前许多动态语言dynamic language允许reflctive programming

template metaprogramming: 模板编程，是一种特殊的metaprogramming技巧

使用metalanguage来写一个模板

这个模板被pre-compiler使用，生成一个intermediate code(中间代码)，这段中间代码是一种特定的编程语言，然后会被main compiler所compile ，当然，Main compiler还compil.e常规代码

1.第一步,template首先由meta language所declare的

2.他在regular source code或另外一个 template中· 作为一段代表着特殊内容的代码，然后在实际创造object的时候会指定这些template，生成具体内容

3. 然后就会生成具体的内容  
4. 在生成具体内容后，如果有同样type的template，不会生成额外一份，而是用之前generate的代码

在C++中，metalanguage 是一种hybrid language混合语言，允许在metaprogram中使用object language，也允许在object language中使用metaprogram instance

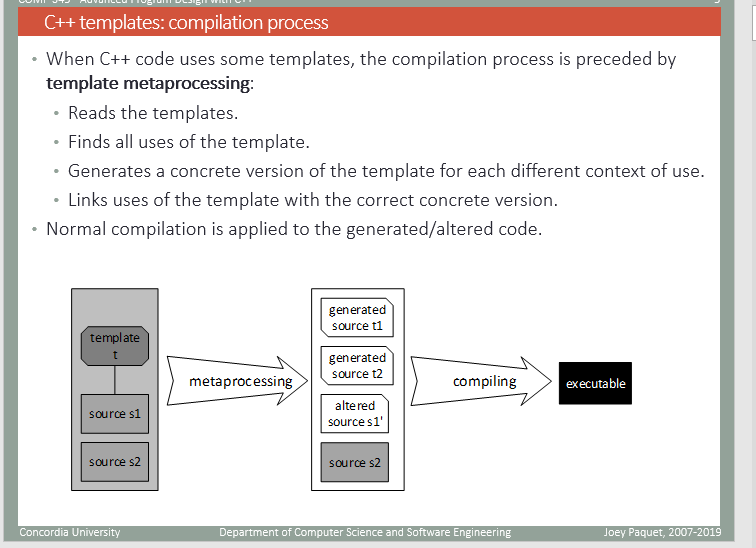
C++提供了这样一个性能，允许使用meta language来编写C++ TEMPLATE，这个meta language是C++ 的小变种,可以再template里面使用C++代码，混合着C++代码与metalanguage的这段代码叫做hybrid language

C++TEMPLATE主要用一段抽象type的code来表达一个通用模板

最常用的template就是c ontainer，例如STL

例如vector<xxx>

这个type就是一个template



当C++CODE使用template时，编译过程会进行 template metaprocessing预处理

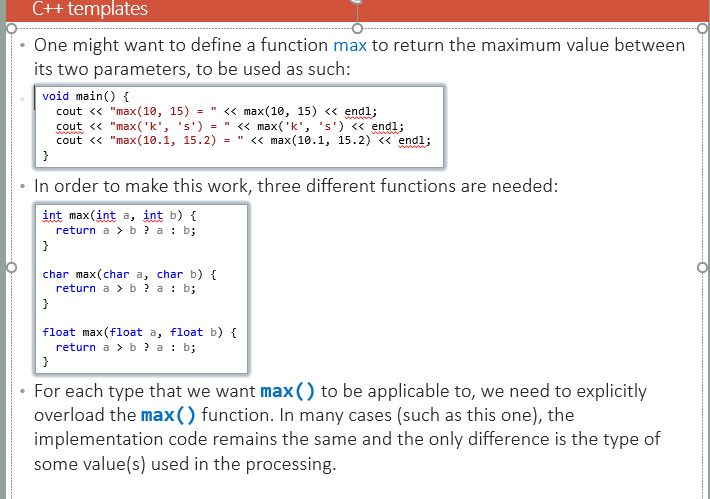
阅读这些template

找到所有用这些template的地方

生成一个实际版本的template对每一种不同的用法

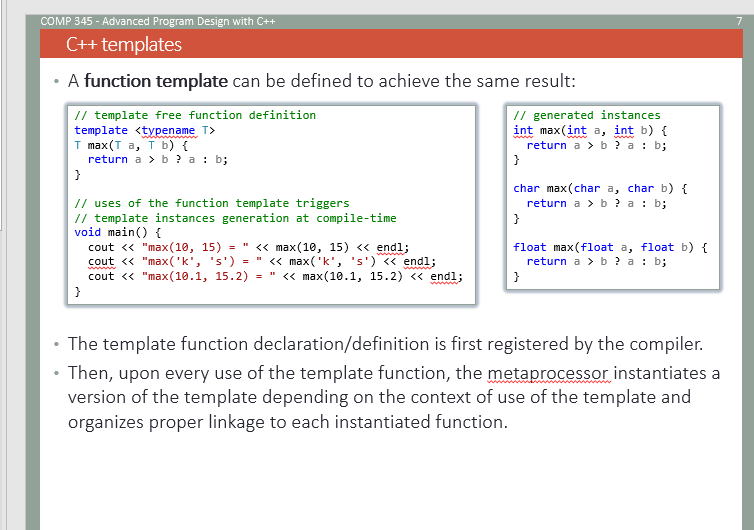
将每一个使用template的地方与对应的实际版本联系起来

然后这些generated code就像正常一样运行

+有的人相比较Max.

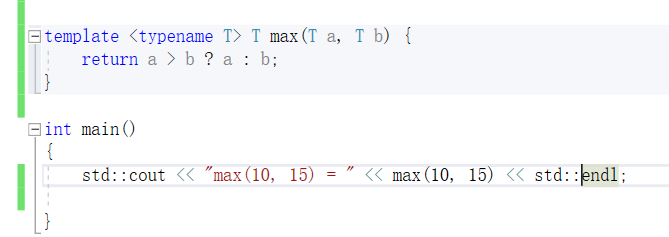
比较数字，float, char

但是这样我们就要安装三个东西



template就是完美替代

template<typename T> 就是先写一个template //实际上是一个前缀



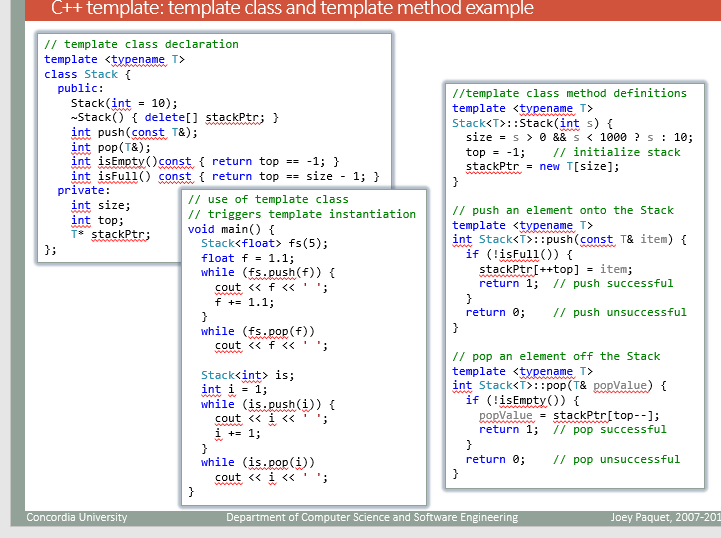
实际上template是一个前缀，告诉你这是一个template function，

但是我们通常把template 以及template 表放在第一行，

使用template的function放在第二行

通过template前缀，我们能先在compiler中登记template function,

然后与每次实际使用这个template function的时候， metaprocessor 会实例化一个template的具体版本（可以使用的C++语言的function）,这取决于call他的人是谁，然后将使用者与实例化的function 连接起来



template甚至可以描写class

在.h页面很简单，就加个前缀

在cpp页面相对复杂，因为template的T并不继承，所以我们还要template一次，告诉你这是template function

在Main driver中，格式与创造vector差不多

练手

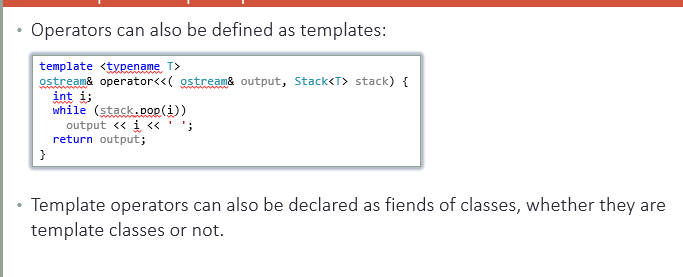
template <typename T>

class XX{  
}

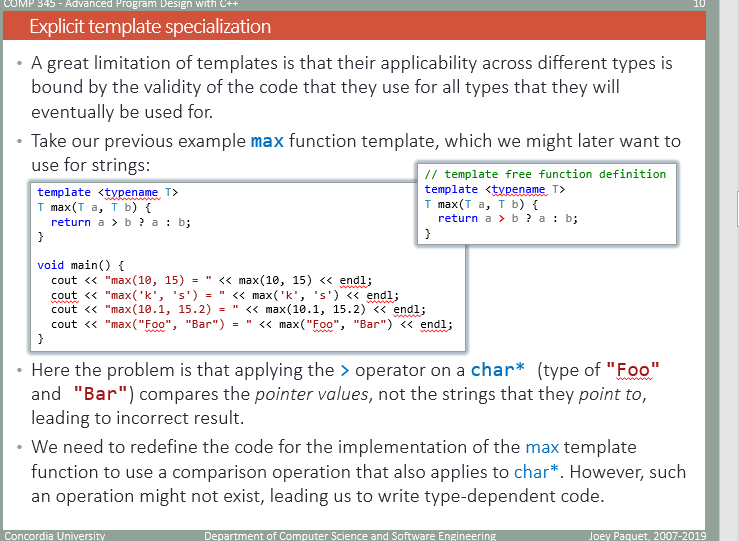
.CPP

TEMPLATE <typeName T>

XX<T>::XXXX正常写



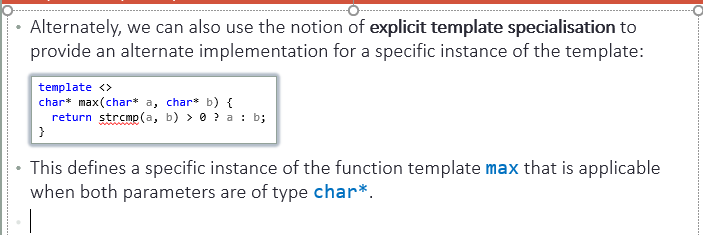
operator也可已定义成template



template有局限性·就是他们并不能带入所有类型的type， 我们要制定具体type范围

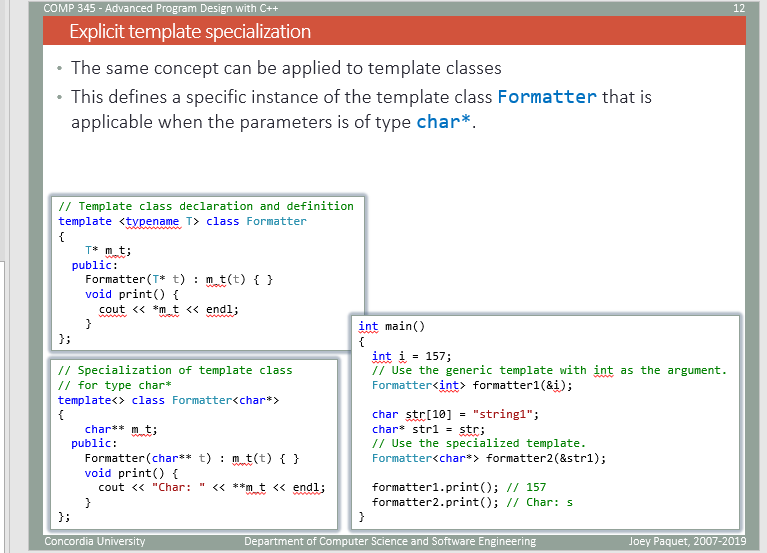
以Max 为例， 如果我们这里使用了char\*,例如“foo”, 它实际上是一个指针，也就是说他在比较两个指针的地址，会return错误的结果

我们需要重新描述使用char\* 的template，然而这有时候不可能，所以我们还是要用传统code



当然我们也可以使用 explict template specialisation的概念，//指定template类型

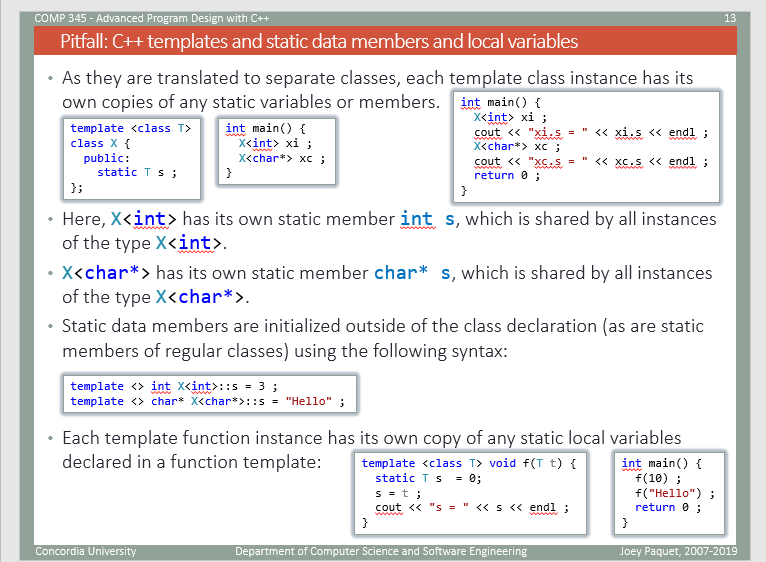
这描述了一个这描述了一个模板max的特殊实例，当两个parameter都是char\*的时候



第一个是符合大多数情况的模板

第二个是特例对于参数是char\*

当然这个是class类型·，所以char\*并不是参数，而是指代如果typeName是char\*的情况

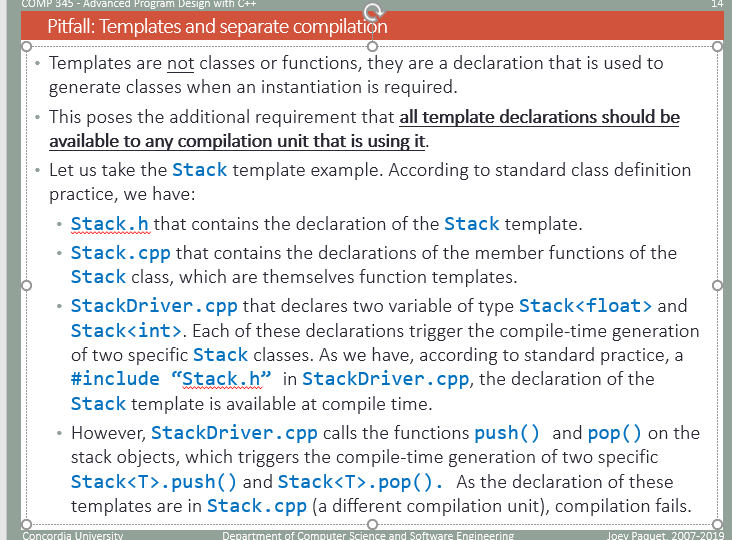


当它们都被转译成独立的class，每一个template class实例都有自己的copy，对于任何static variable 与member//原来同个class多个object的static共享，现在不同类别之间实际上是不同的static，不共享，但如果是同一类的话，，好贵还是共享

在class外面生成的static data member



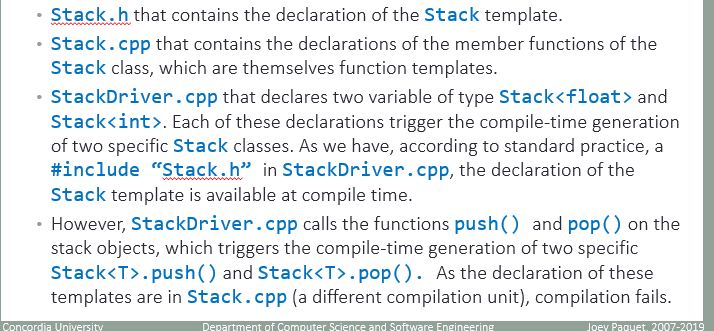
记住static通常要在外面实例化



template既不是class也不是function,它是一种declaration,当实际需要的时候生成对应class

这就有了额外要求，所有template declaration需要被任意cpp file 使用

以stack template为例



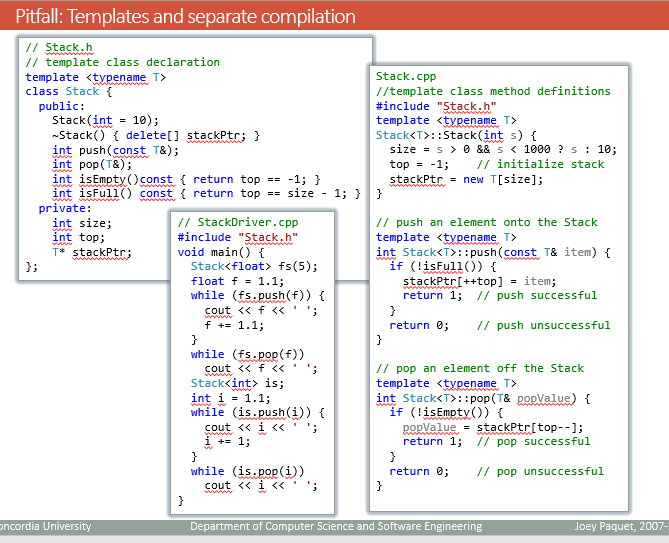
stack.h需要声明stack template

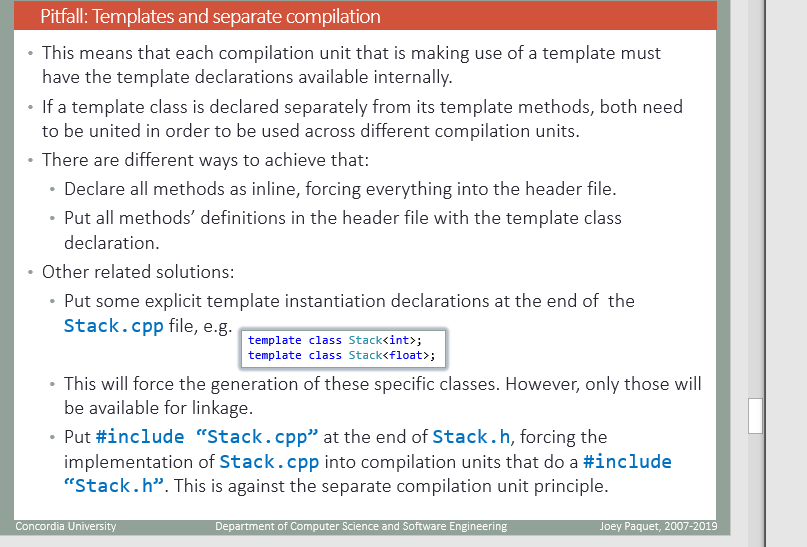
stack cpp包括了sttack class的具体描写

stack driver， 使用了两个stack 类型，每一次declare都会生成一个compile-time存在的stack class

如果根据老经验，#include "stack.h"在driver中就可以使用stack了

然而，stackDriver呼叫stack object的Push()与pop（）时，会使用的是两个特定类型的，实例化的push()与pop()，这些声明在stack.cpp中（另外一个compilation unit），运行失败



.

这就意味着每一个compilation unit必须内置模板声明部分

两种方法，让所有method都是inline，强行让所有东西在header file里

让method definition就写在header file里

或许其他方法：

在cpp的结束加上指定类型，这会强制生成对应的实例化class

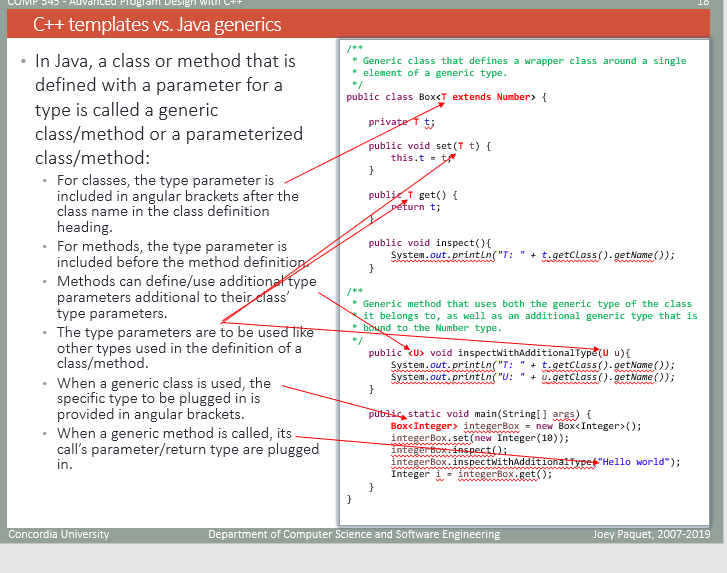
然后再在.h的结束后面加上include .cpp， 但这违反了不加cpp的原则’

C++ TEMPLATE与JAVA GENERICS

JAVA GENERICS与C++ TEMPLATE 语法上看起来很像，目的也相同，但是实现过程是不一样的，因此实际效果也不一样

java generics使用了叫做type erasure的技术//类型擦出， compiler 可以跟踪generic definition泛类描写， 因此在实际运行的时候使用同样的class 描述

C++ 用的是template metaprogramming， template由新的type parameter生成，完整的code 再被参数生成以后再compile，因此在run time会有多个描述



在java 中，使用generic type的叫做generic class/method

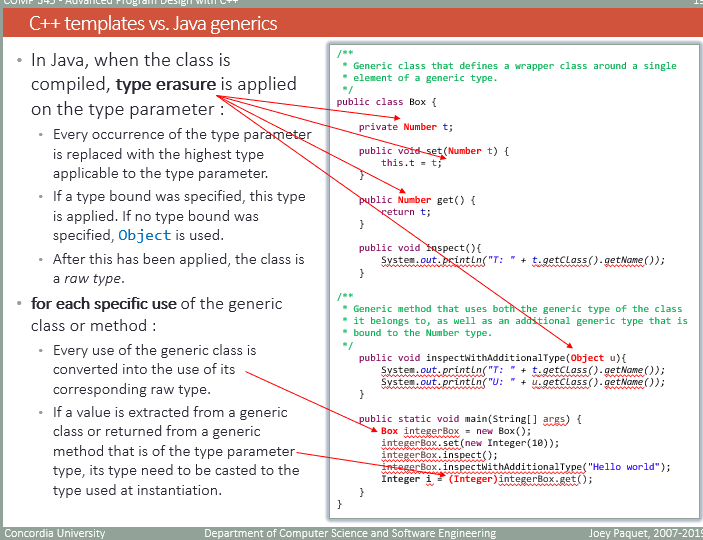
对于class来说,type在一个方括号angular brackets里 （后缀）

对于method来说，在public后面，（前缀）

method里面还能使用额外的，在声明class时没用过的类型（T与U）

当generic class被使用的时候，指定type会插入对应的方括号

当generic method被call时，通过参数指定对应 type



在JAVA 里，当一个class被compile ,type erasure应用到type parameter上：

这张图是与上图的两种不同写法，

type erasure就是泛类，例如Numbeer, Object

T extends Number 可以直接被Number替代

、Number是Highest type

当指定了一个较小类型的type，就会使用驾校类型的type

如果没指定，就会使用Object

如果使用Object,这个class就是raw type

C++的template让运行更加缓慢，尤其是大量template的实例化被需要的时候，

导致不必要的代码块如果过分使用explicit template insantiation

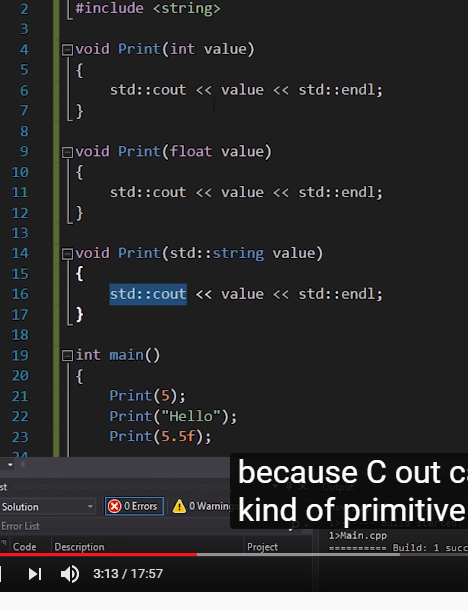
因为template的使用，要重新考虑怎样切割.h .cpp文件

erro reporting变得非常含糊，并且与源码无关

c++的template本质上是与JAVA C# 的泛类generic class不同的概念

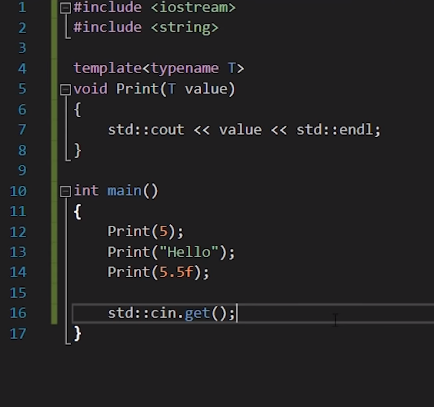
template

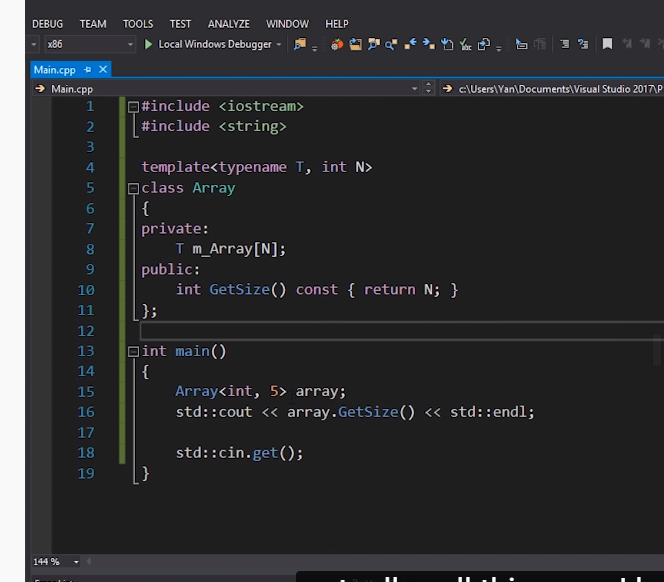
是一种模板



理论上我们当然可以把他写的非常复杂，

但有时候我们想要一种通用模板，





template就是创造一个模板，一个填空题，我们将不确定的部分不确定的部分有何用template Typename或指定的类型表示

然后就可以写出我们的模板

实例化的时候我们告诉他指定类型

这时compiler才会真正生成一个存在的funciton/class等等