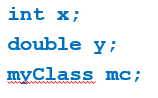
大多数基本的primitive data Type都不是class并与java高度相同

C++用户想定义自己的data type使用struct或者class

Struct也可以有data members,methods,constructors等，几乎与class一6A21一样，唯一区别在于，struct默认是Public

bool:Bolean 值是true or false

怎么样declare 变量



也可以 intx,y,z

C++用的是manifest typing显示输入策略：就是变量会给他一个type，你只能给他这个·type的value，php这种就不是

但是C++允许type coercion 类型强制，隐形或显性的改变其type，因此C+=j叫做weakly typed language，弱类型语言

Coercion : force sb to do sth

这个叫做隐式转换implicit type casting或者自动转换 automatic type conversion

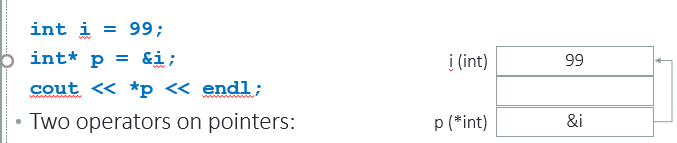


Pointer variable

所有变量都会包含一个值，例如Integer啥的

C++有一种特殊变量叫做Pointer variable,特包含了一个内存地址，而这个地址会转到一个所point的变量

对于任意变量T，他的pointer就是T\*, T\*里面包含了object T的地址



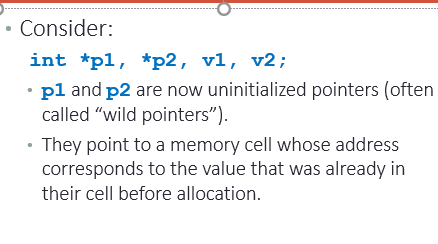
Dereferencing operator:\*,解应用 \*p代表着 这个pointer 所指向的object，就是解开P的意思

Int\* p代表你在定义一个指向int的指针

Address operator， &，&i 代表包含这个变量i的address 地址

Int\* p 与int \*p是一样的

我们更倾向于用后者，这样就可以和标准变量区分开也就是说对pointer来说，int是指定他能point的变量范围，而对于普通变量，Int是给他一个种类

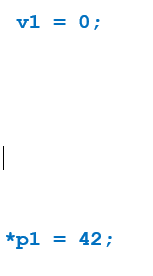


这一步来说

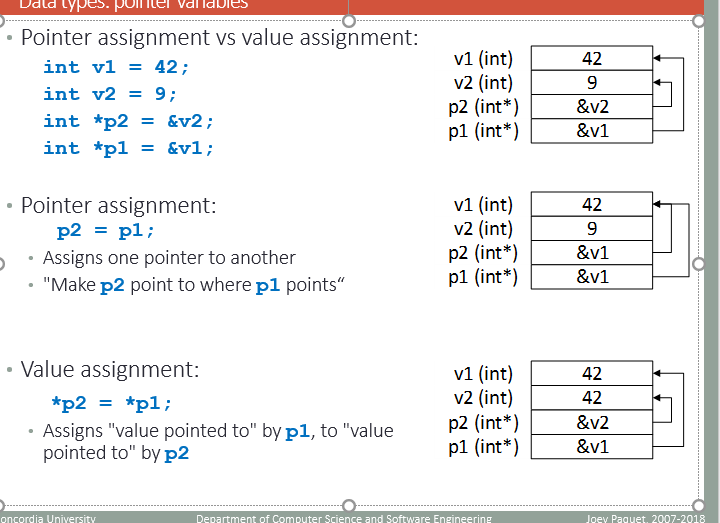
P1p2被定义成指针，v1v2被定义成int

而p1p2也不是完整指针，叫做uninitialized pointers(wild pointers),因为他们没有指向具体东西

， p1 等于v1的地址

再说一遍，\*是解应用，解开p1所指的variable

他们指向的都是同一个memory，如果改变\*p1或者v1，都会导致另外一方的改变



P2=p1与\*p2=\*p1的区别

第一个是改变P2指向，原来的9是不会变得

第二个是改变p2所指的variable的值

Dynamic variables动态变量

需要使用pointer

通过new来分配allocate

通过delete来解散deallocate

在program运行之前，你需要自己alloocate 他们

C++没有garbage collector

Local variable当地变量 //只有function激活时才有效，因此存在slack里

不是dynamic，通常在描述某一个function时使用（想一下 默认Namespace）,当最后一个花括号结束，也就是function结束时失效

|  |
| --- |
| A |
| B |
| C |
| D |
| E |

比如main占CDE部分的memory，当运行到functionX1时，分配给他B，运行到functionx2时，分配给他A，用完再回收

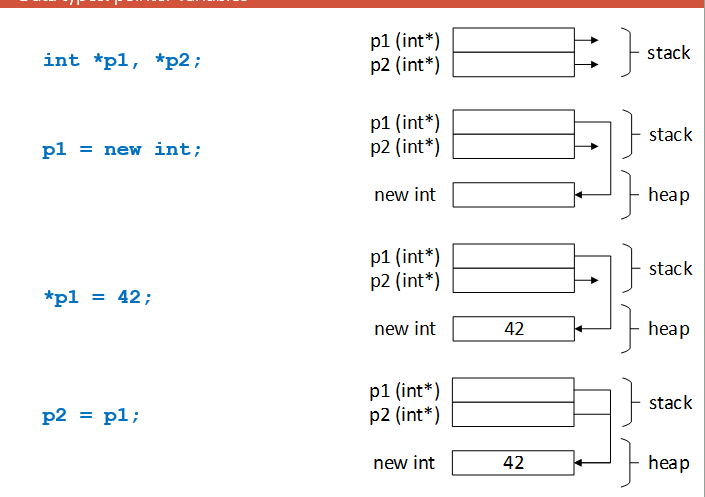
New 操作创造dynamically allocated values， 可以被pointer所指

被放在heap里（普通变量放在stack里），或者runtime system的freestore里//自由存储

程序员必须管好他，你必须要delete them

换句话说new就指定了一个地方你一直存着，你自己需要collect

但问题在于function内的Pointer还在slack里，当slack被删掉，那被指的东西实际在heap里还是存在，所以需要用delete把heap清除



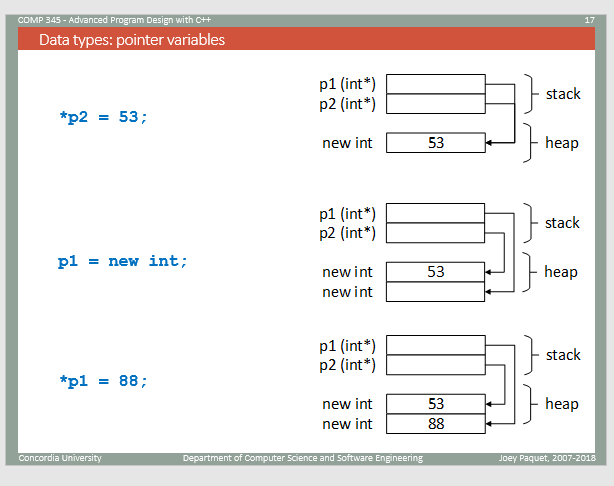
P1 p2都是Pointer

P1=new int；创造了一个动态变量，让p1指向他

给这个变量赋值

P2=p1，改变p2,

这些过程中Pointer始终是stack



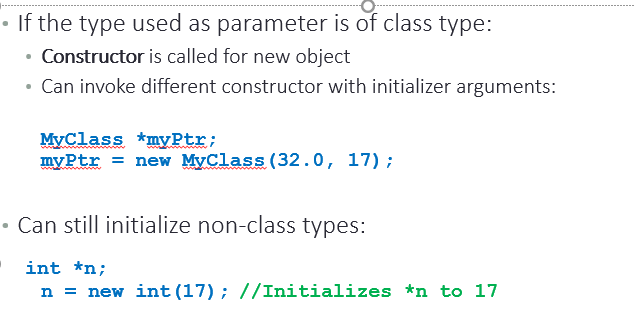
这时p1p2指的是一个东西，所以p2改成53所有的都是53

而new int会开拓一个新int在heap下面

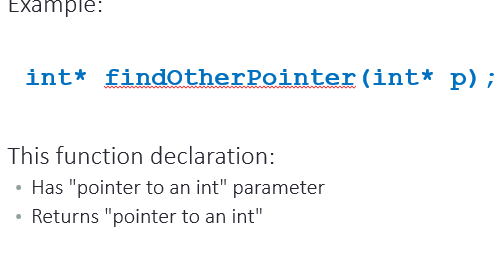
这时p1相当于格式化了，重新指向

Pointer也可以用来专门指向一个class type的object,

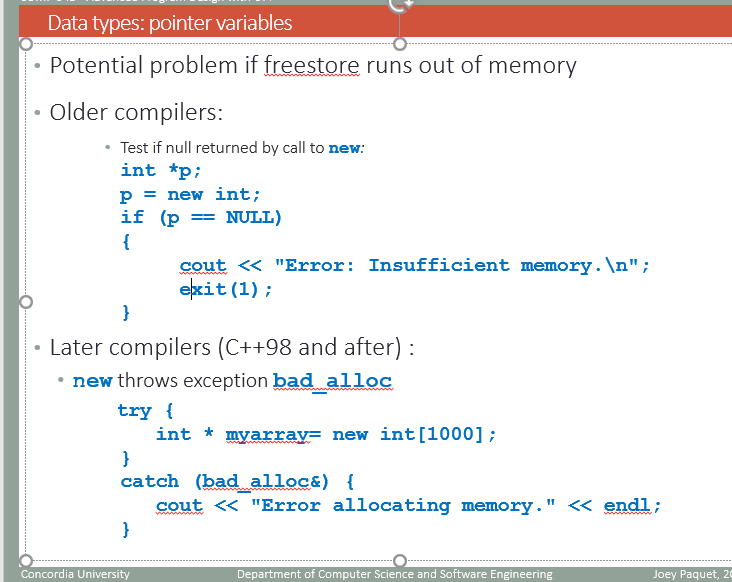
先指定类型，再用constructor 来initialize初始化



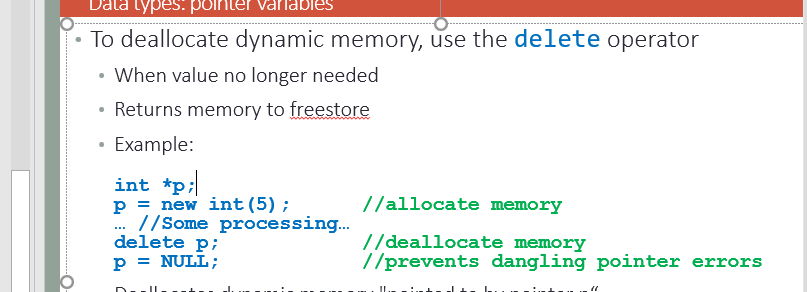
Pointer也可以用来作为从参数



有一个指向Int的Pointer p作为参数，最后会return一个指向Int的pointer  
因为new出来的动态变量存在heap或者freestore里，所以有可能用尽freestore

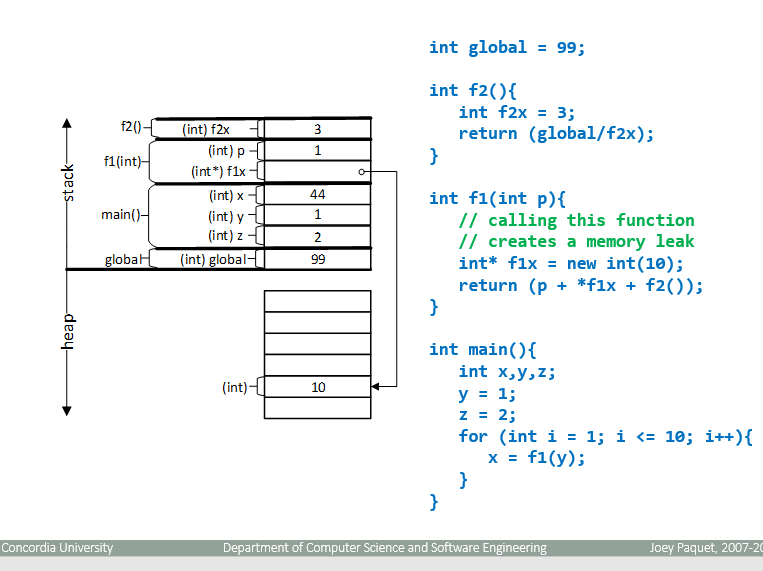


这也是我们为什么要写delete的原因，如果一个值不再被需要，我们应该释放freestore



Int\* p是wild pointer，什么都不指向

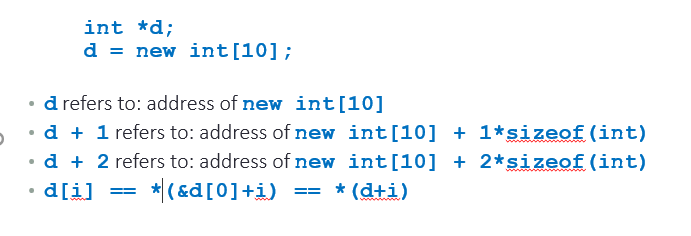
Delete p； 是dangling Pointer，实际上还是指着原来的位置，这一步只是为了解放内存，如果之后不用p，那就没必要Null,但是如果还要用P作为variable，那么还要让他=Null



在f1里，我们应该在return之前delete pointer

我们可以对Pointer进行arithmetic operator 代数运算

用在array里



1.指针如果没有指向同一个array，那么指针间的加减是没有意义的，只是单纯memory address的加减

2.指针变量加减一个整数

C++规定，一个指针变量加/减一个整数是将该指针变量的原值(是一个地址)和它指向的变量所占用的内存单元字节数相加或相减。如p+i代表这样的地址计算：p+i\*d，d为p所指向的变量单元所占用的字节数。这样才能保证p+i指向p下面的第i个元素。

因此实际上是往下数几个

3.  两个指针变量可以相减，如果两个指针变量指向同一个数组的元素，则两个指针变量值之差是两个指针之间的元素个数，假如p1指向a[1]，p2指向a[4]，则p2-p1=(a+4)-(a+1)，但p1+p2，指针相加并无实际意义。

4.两个指针比较，指向前面元素的指针小于后面一个指针

Reference

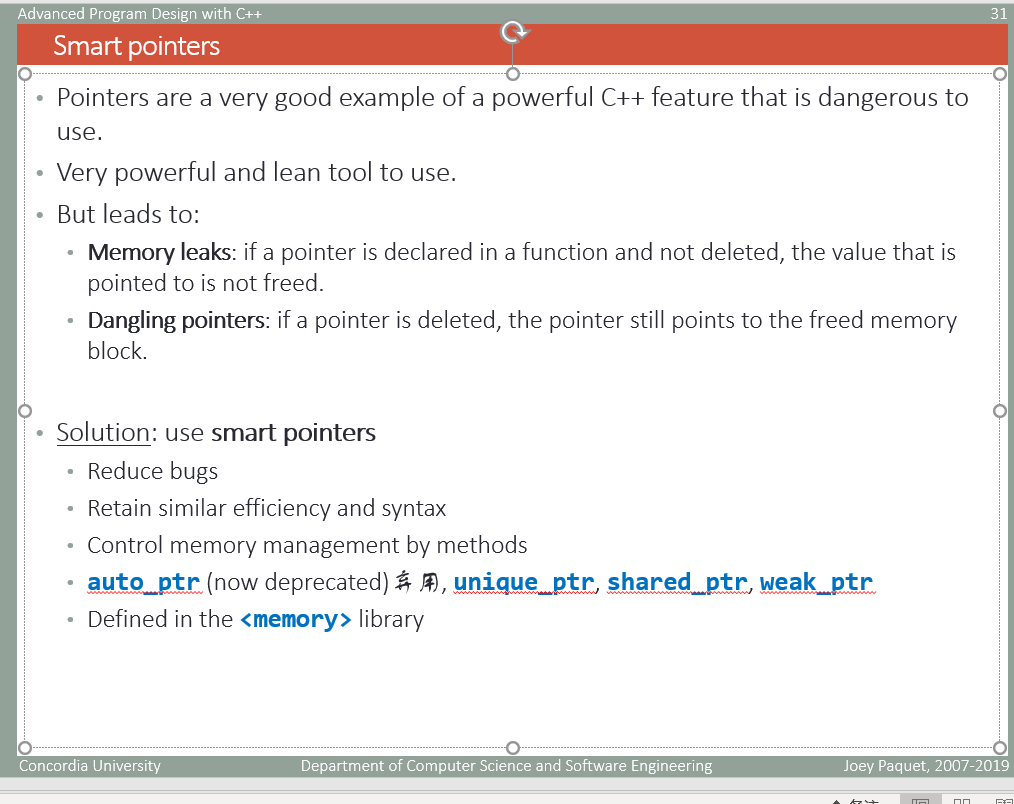
Pointer有许多优点，但是因为他也有很多复杂性，

我们用reference pointer作为pointer variable可以消除许多pointer的劣势，当然他们的power也相对降低

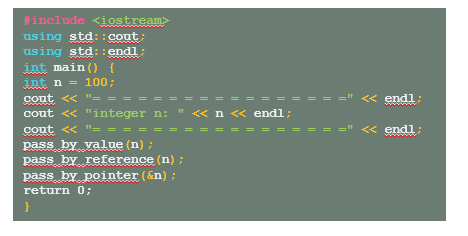
1. Reference不能用pointer运算
2. 任何reference的操作都会实际影响到他们所指向的variable
3. 因此reference必须要初始化并且之后不能改变
4. r就是v1的同盟，&R会return v1的Pointer,没有Pointer能指向reference

创造同盟关系，同盟不是指针，而是一个完全等价的东西

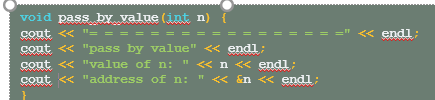
Smart pointers



参数传递 parameter passing



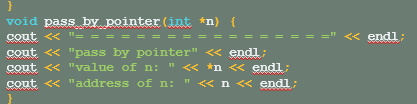
By value



会创造一个n的复制品，这个值对function是local的，对复制n的改变不会改变原来的n

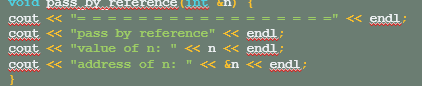
By pointer

会复制一个Pointeer



在function改变所指的value会改变真实的value

By reference



结果上是和Pointer传递一样的，就是不能改变原始Pointer所指向的地方

坏处，不能接受 null，因为你建立不了Null的同盟

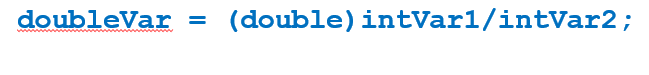
好处，可以接受还没命名的value

Explicit type casting

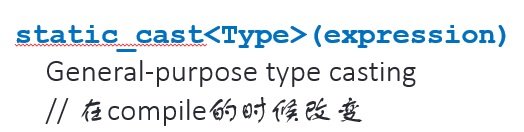
C++用的格式

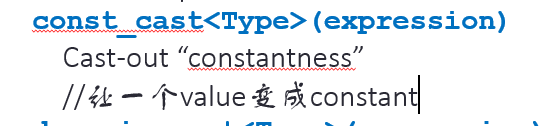
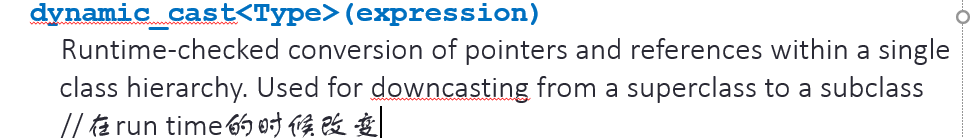
,intvar会被转换成一个double

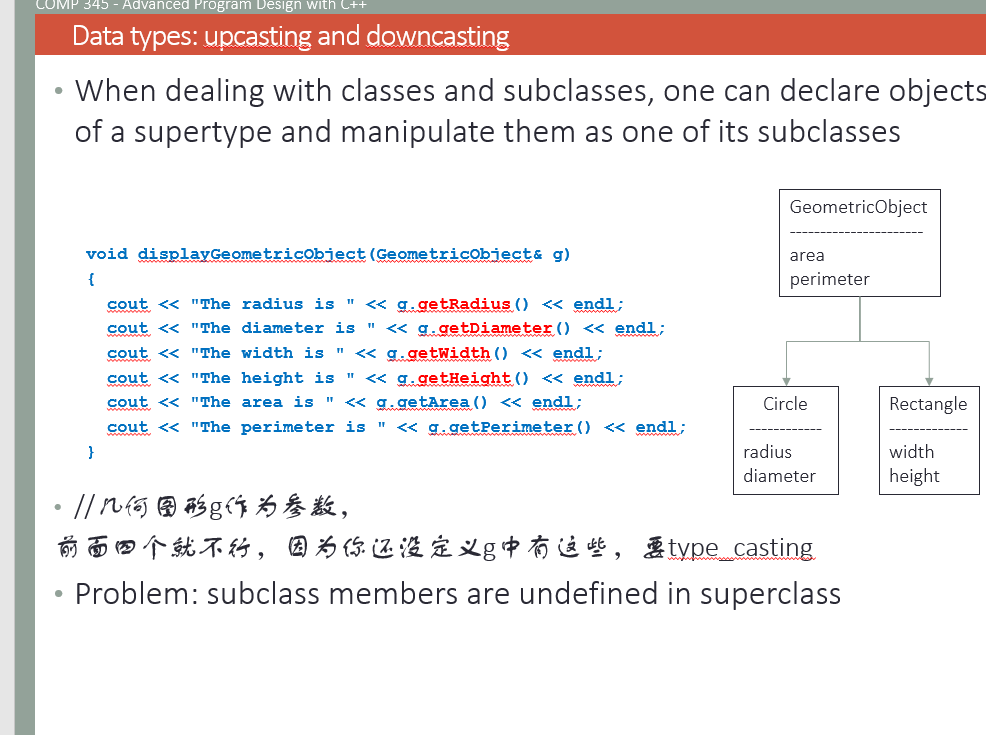
intVar1 IntVar2 还是会被改成double，然后再除



与这个一模一样



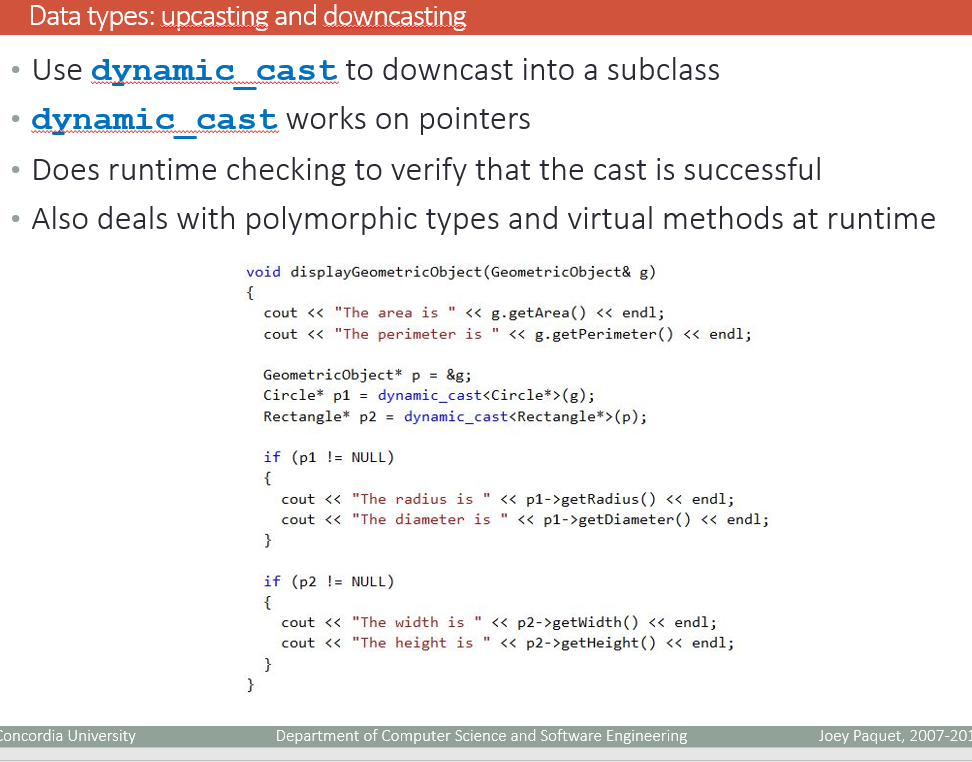




不行，没有包含子类参数



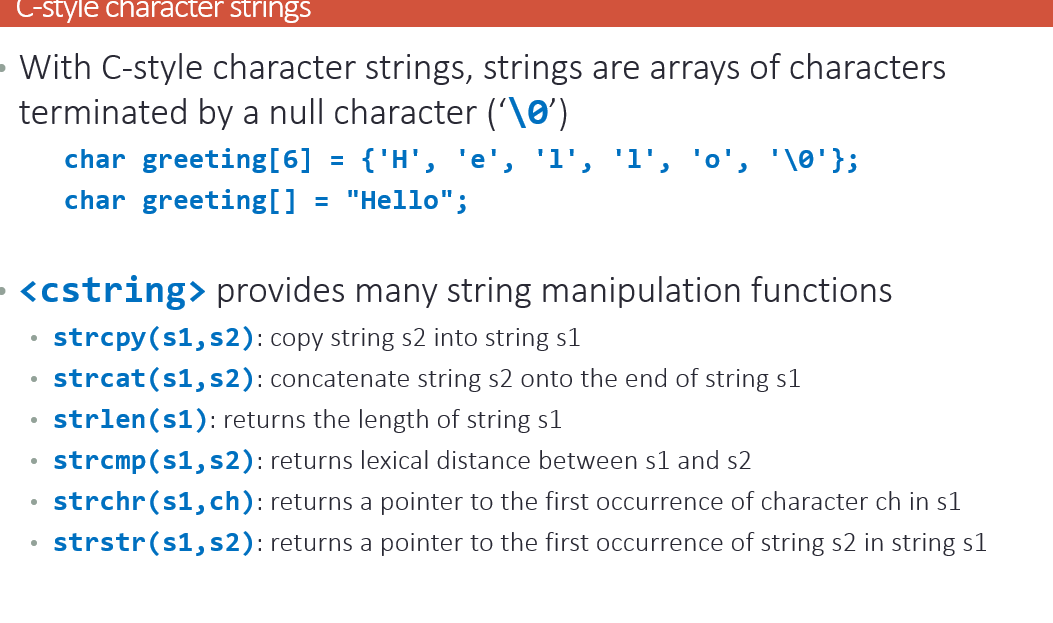
Compile是可以的，但是不会成功运行，如果一个object编写的时候并没有包含这些，那么static\_cast并不会让他有额外的这些属性



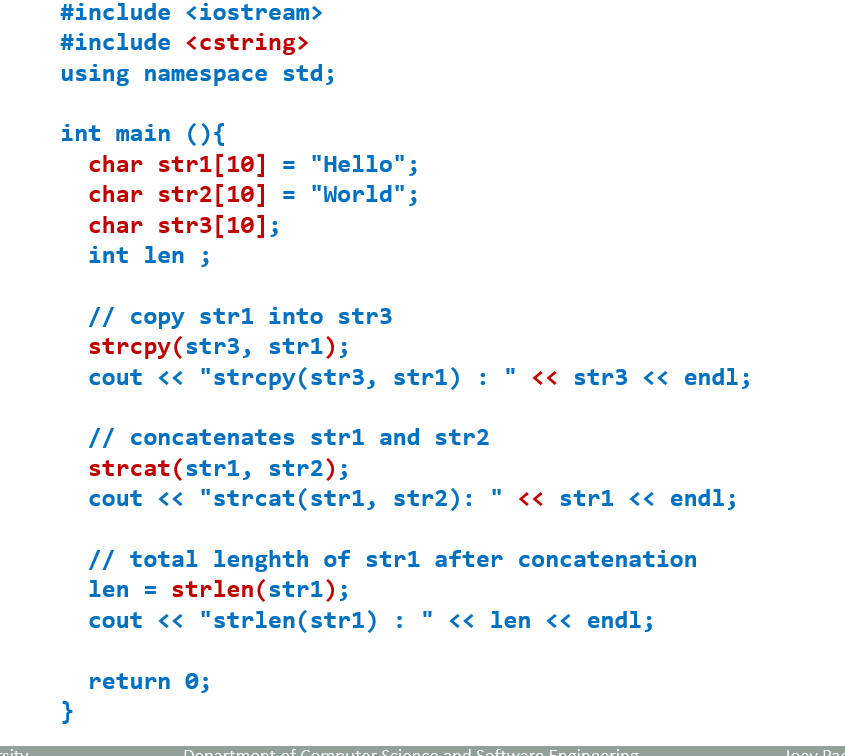
Dynamic cast作用在pointer上

事实上是可以work的，但你要尽量避免，

C++有两种方法来表示string

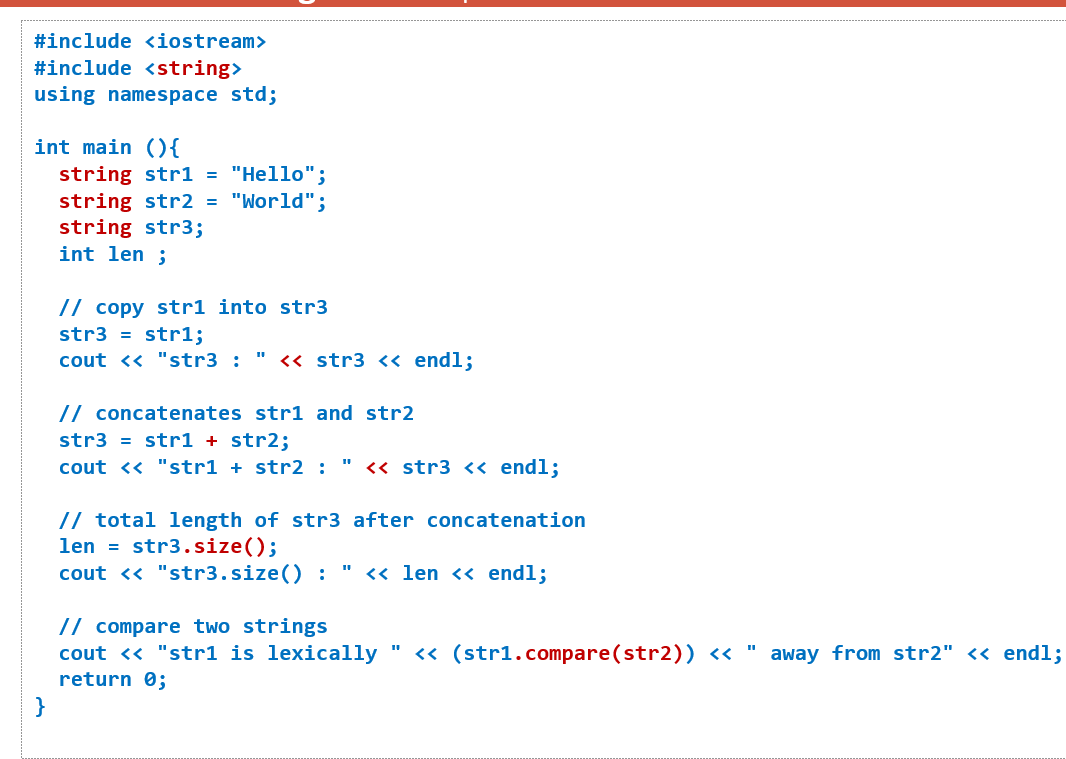


1. 建立在c上的char string，实际上是一个char array



Copy必须使用cstring里的strcpy

C++有自带string library



Concentrat用+,copy用=，length用size，比较用compare