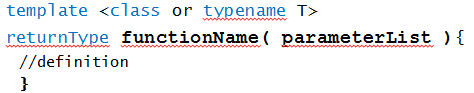
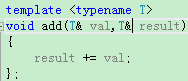
实现代码相同，支持数据类型不同：1.函数模板

2.类型转换的操作符重载

函数模板：提供了一类函数的抽象，即代表一类函数。函数模板实例化后生成具体函数。

**普通函数**和**类的成员函数**可以声明为函数模板

格式：

例子：

一般来说，模板函数在需要的时候才会生成

一般在调用时，函数模板才会实例化，如会生成模板函数

【PS. 若想这样直接传入具体的参数而不是变量名，写函数模板的时候只能传入形参，而不能传实参，这样编译器会在调用add那是提示“没有与参数列表匹配的函数模板”。原因是因为传入具体参数的时候是没有办法再改变的，没有地址可以传入？】

形参跟实参的概念混淆了…

形参指的是这个函数头里面的a和b，称为形参列表，而实参是指你传进去的参数，比如你传的1.0和2.0就是实参。

这里的问题你的想法没有错，就是因为它不是变量，不占有自己的内存空间，所以没办法传引用进去。

这里就要再次传递一下传参的方式了：

一般来说，传参的方式有两种：1、传值；2、传引用。

也有人说还有一种叫传指针…实际上传指针就是传指针的值，所以把它当成传值吧。

传值跟传引用很好区分，你上面的是传值，而是传引用，很明显这两个的区别就是：传值是传副本进去，函数里面改变参数的值，函数体外是不会有所改变的，因为并没有把参数本身传递进去，函数体内改变的是副本的值，，传引用是传它本身进去，所以函数体内对参数做了改变，函数体外也会有相应的改变。引用是变量的别名，拥有相同的内存空间，所以要穿引用至少要有内存空间才能传，所以你不能传一个常数进去。

模板函数的参数分为：**函数实参**和**模板实参**



括号内的参数为函数实参

【PS. 在第二调用里面，因为声明是T 是int，浮点型的1.2会强制性类型转换为int型的1，因为输出结果为4】

模板函数的模板实参在下列情况下，不能省略：

1. 从函数实参获得的信息有矛盾

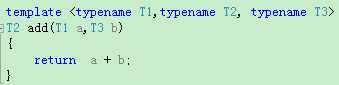
例如，因为1.2是浮点型，3是整型，产生歧义。如果没有模板实参，编译器会报错：

【因为函数模板的两个传入参数应该相同，这里编译器没有办法生成两个不同类型参数的模板函数。而如果有模板实参，传入参数会强制性类型转换为相应类型】

2. 需要获得特定类型的返回值

3. 虚拟类型参数没有出现在模板函数的形参表中

这是多个类型被参数化，有多义性，一般不省略模板实参

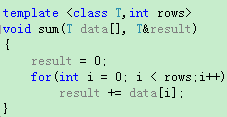
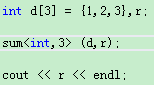


输出结果为

【根据结果来看，第一个调用不能获得的参数是T3，因此就根据传入参数自动补充T3为double型】

4. 函数模板含有常规形参

如果常规参数的信息无法从模板函数的实参表中获得，则在调用时必须显式的给出对应于常规参数的模板实参。

例如函数模板为，调用时

常规参数的模板实参3不能省略，否则编译器会报错

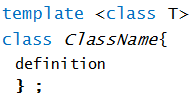
类模板

1.将数据类型参数化

2.用于处理多种不同数据类型

3.功能相同，数据类型不同

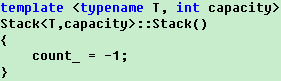
4.代码复用

定义时格式：

PS. 不一定要用T，可以自己定义参数名字

生成对象调用时：

类外定义函数时：



PS. 在类模板前已经声明，在类里面写成员函数不需要加template

注意类名字后面的模板实参不能漏

形参的名字不能在模板内部重用，也就是说一个名字在一个模板中只能使用一次:



模板的声明和定义中参数的名字可以不同：

声明时

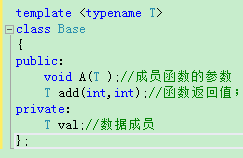
定义时可以

可以参数化的数据类型：

1.数据成员的类型

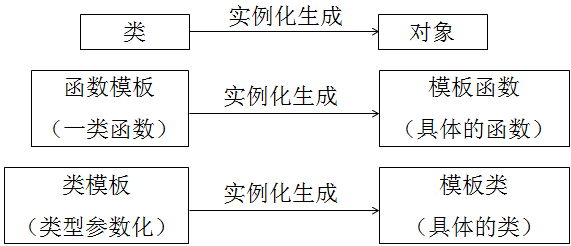
2.成员函数的参数的类型

3.返回值的类型

如

PS 在模板声明中，“class”和“typename”是相同的

类和对象、函数模板和模板函数、类模板和模板类之间的关系



PS 在模板类里面可能有模板函数，关系可能有嵌套

Non-type Parameters 非类型类模板参数（？）

可以在模板中使用非类型类模板参数

默认参数可以是：常整数（包括枚举）

指向外部链接对象的指针

浮点数，指向内部链接对象的指针则不行

【关于这里，我首先不是很明白什么叫“Non-type Parameters”，PPT上给的例子是;

这里是指这个int是指定了类型的，是非类型模板参数。



是跟函数模板里的常规实参差不多意思么？

是

然后关于指向内部链接对象的指针和内部链接对象指针，百度了一下，发现依然很难理解。

简单来说，指向内部链接对象的指针是指假如在类里面定义了一个数组，指向这个数组的指针就是指向内部链接对象的指针，

相反，如果数组是在类外定义的，指向这个数组的指针是指向外部链接的指针？】

后面会说内部链接对象跟外部链接对象的区别，你看了再返回来解决你这个问题…



这个会在类定义中出现，在编译时就产生数组，用指针动态开辟数组，生成类的时候给出元素数量。

【额，不这样写就不是在编译时产生数组？在编译时产生数组的好处是啥？】

编译时产生数组就是那种不是动态分配的，你就理解成不是用malloc和new来分配的数组，用new和malloc的是动态分配的，是要到运行的时候才知道有没有分配这些内存的。

编译时分配数组是分配在栈内存的，内存管理由编译器来搞定，而动态分配是要程序员自己来进行创建和删除的，就像链表结点的创建和删除一样…编译时就分配好，运行的时候就不用对它分配空间，自然就不需要在运行时因为分配空间而占用CPU了…

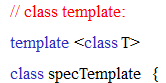
接下来是关于内部链接对象和外部链接对象了：

你就先简单理解为外部链接对象就是上面用静态分配也就是编译时就分配好的来分配的对象，内部链接对象就是用动态分配来分配的对象。

非类型模板参数要求在编译或者链接的时候就要求知道它对应传进去的参数的地址，而不是等到运行时才知道，所以要在编译的时候就知道了对象的地址…

重写类模板

* 应用场景：即想使用模板，同时又需要对一个特殊类型做不同的实现。

：这是一般的类模板

：这是重写后的

Template<> 表示是同样名字的模板的特例

这里两个函数是表明除了char外的调用第一个模板，char调用第二个模板

这个叫模板的特化哦…就是对于某些类型，你要用这个模板函数，但是又有特殊处理，所以就自己把这个函数给特殊化了，遇到这种类型的时候就不去调用原来的模板函数了，而是调用特化后的函数。

四种继承：

1.普通类继承模板类

2.模板类继承了普通类（非常常见）

3.类模板继承类模板

4.模板类继承类模板，即继承模板参数给出的基类

继承哪个基类由模板参数决定

例子如下：

有普通基类A

普通基类B

类模板C

派生类Derived

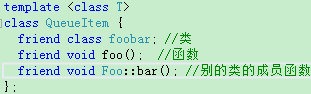
继承的时候：

模板与类

·可以为类模板的友元：

一、非模板友元类或友元函数：

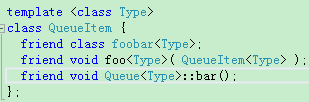
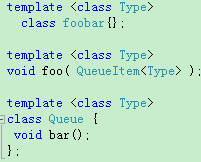
1. 全局函数 2.别的类的成员函数 3.友元类



注意：1.类和函数作为友元不需要在类之前就声明或定义

2.别的类的成员函数在声明为友元前，此类必须已经被定义

二、绑定的（bound）友元类模板或函数模板



这时foobar()，foo() 和Queue本身是模板，作为友元函数时与类型有关，与模板类联系在一起

【就是说foobar<int>只是类QueueItem<int>的友元函数，而与类QueueItem<double>没有关系】

注意：在一个模板可以被用在一个类模板的友元声明中之前，它的声明或定义必须先被给出

类模板与静态成员

·非模板类的静态成员与所有实例化的对象相关

·Each class (int, float, etc.) has its own copy of static data members

【就是实例化后类T<int>和T<double>的静态成员之间没有关联？】