题目描述

目前国内疫情形势较为缓解，在公共场合佩戴口罩的公共安全意识逐渐普及，并且不少国家也研发出了有效的疫苗。但从国际形势看，疫情形势依然严峻，因此我们不能放松警惕，依旧要科学合理地预防和应对。

基于上述考虑，南京市卫生健康委员会决定为南京各单位提供口罩和疫苗的**有偿**供应，委员会请你帮助实现一个防疫物品购买系统，管理各单位的采购订单。对于每个订单，有如下的几个要求：

- 订单分**口罩订单**和**疫苗订单**两种，口罩和疫苗的订单都有不同的种类，每种口罩和疫苗订单都有名称、价格、购买数量等属性。
- 由于生产力的关系，口罩供应量非常大，而疫苗供应量有限，因此疫苗订单存在**购买限制**，即每种疫苗都有一个最大总量，当库存不足时便会购买失败。
- 详细的要求将在接口说明中详述。

你需要实现如下三个类：

- 口罩订单类Mask
- 疫苗订单类Vaccine
- 为了将口罩和疫苗的订单分类整合以计算总量，需要实现一个模板类CombinedOrder用于合并同类型的订单

具体描述如下：

口罩订单Mask

Mask代表一个口罩的订单，每个订单只能购买一种口罩。

```
// 构造函数
Mask(const string& name, int price, int number);
/**
 * 说明：
 * 参数包含口罩类型名、口罩价格、采购数量。
 * 例如：Mask n95("N95", 20, 100)表示该订单购买了100个N95口罩，而N95口罩的单价为20元。
 */

// 成员函数，返回口罩类型名称
string GetName();

// 成员函数，返回口罩单价
int GetPrice();

// 成员函数，返回订单购买该口罩的数量
int GetNumber();

// 成员函数，返回订单总价（单价*数量）
int GetCost();
```

疫苗订单Vaccine

Vaccine代表一个疫苗的订单，每个订单只能购买一种疫苗。疫苗存在**库存总量**，意为仓库中一开始的总存量，并非在下订单之后的剩余库存。

```
// 构造函数
Vaccine(const string& name, int price, int number, int bound);
/**
 * 说明：
 * 参数包含疫苗类型名、疫苗价格、采购数量和库存总量。
 * 例如：Vaccine kexing("Kexing", 300, 100, 200)表示该订单购买了100支科兴疫苗，而科兴疫苗的单价为300元。同时，系统中科兴疫苗的库存总量为200。
 * 可能存在初始化时采购数量大于库存总量的情况，但不需要在此处处理异常。
 */

// 成员函数，返回疫苗类型名称
string GetName();

// 成员函数，返回疫苗单价
int GetPrice();

// 成员函数，返回订单购买该疫苗的数量
int GetNumber();

// 成员函数，返回该疫苗的库存总量
int GetBound();

// 成员函数，返回订单总价（单价*数量）
int GetCost();
```

合并订单类CombinedOrder：模板类

CombinedOrder为模板类，例如`CombinedOrder<Mask> maskOrders`即为用于统计Mask订单的合并订单类，包含一系列的Mask订单并按照类别分类合并统计。疫苗同理。

```
// 构造函数，初始化对象
CombinedOrder();

// 成员函数，将新订单合并入已有订单
bool AddOrder(T& order);
/*
 * 说明：
 * 将某一订单加入CombinedOrder。若列表中已经有这个订单对应产品类型的订单，则将新订单的订购数量累加到已有的订单。返回true。
 * 对于疫苗订单，新订单加入后某产品累加购买量超过库存总量，以及新订单购买量本身就超过库存总量，这两种情况会导致订购失败，此时返回false，并且CombinedOrder内对应订单的购买数量不变。
 */

// 成员函数，计算目前该合并订单的所有物品的总价格
int GetTotalCost();

// 成员函数，重载[]操作符，获取合并订单中的单个订单
T &operator[](int index);
/*
 * 说明：
 * []下标对应的类别顺序为order录入系统的顺序，例如录入A,B,C,A,D五个口罩订单，则A,B,C,D四类订单的下标分别为0，1，2，3。
 */
```

除了题目中给定的需要特殊处理的情况以外，以下情况不需要特殊考虑：

- 加入同一合并类的订单，如果商品名称相同，其单价和库存量是一致的
- 不会有诸如把Mask订单加入Vaccine合并类的操作
- 不会试图获取合并订单中没有的东西（例如，合并订单中只有两类却调用order[2]）

此外，在CombinedOrder.h中，请实现以下两个全局方法：

```
// 模板函数，比较两个同类订单的价格
template<class T>
bool IsLessThan(T t1, T t2);
/*
 * 说明：
 * 比较规则为，订单t1的总价严格小于t2的总价则为true，否则为false
 */

// 模板函数，排序合并订单
template<class T>
void SortOrder(CombinedOrder<T> &order);
/*
 * 说明：
 * 排序对象为CombinedOrder内部的统计订单列表，排序规则为按每种订单（合并后）的总价从小到大排。测试用例中不会出现总价相同的情况。
 */
```

调用示例

```
// 创建若干个Mask和Vaccine对象
Mask mask1("Mask1",5,5);
Mask mask2("Mask2",10,1);
Mask mask3("Mask3",15,20);
Vaccine vaccine1("Vaccine1",300,2,10);
Vaccine vaccine2("Vaccine2",400,1,3);

// 打印信息
cout << mask2.GetName() << " " << vaccine1.GetBound() << endl;

// 创建两个合并订单类
CombinedOrder<Mask> order1;
CombinedOrder<Vaccine> order2;

// 加入订单
order1.AddOrder(mask1);
order1.AddOrder(mask2);
order1.AddOrder(mask3);
order2.AddOrder(vaccine1);
order2.AddOrder(vaccine2);

// 计算合并订单的总价
cout << order1.GetTotalCost() << endl;

// 根据下表获取合并订单中的特定类型
cout << order2[1].GetName() << endl;

// 比较两个订单的价格
cout << IsLessThan(mask1, mask2);

// 排序合并订单
```

```
SortOrder(order2);
```

注意⚠️

- 请正确处理头文件和实现文件之间的关系，文件、函数的命名严格按照给定要求，注意大小写。
- 将5个文件(Mask.h, Mask.cpp, Vaccine.h, Vaccine.cpp, CombinedOrder.h)打包成ZIP压缩包上传（ZIP包中不要包含文件夹或者其他文件）。由于类模板和函数模板的特殊性，CombinedOrder的函数定义直接写在.h文件中即可，不需要.cpp文件
- 请不要在你提交的代码中包含main函数