Задание 3

Задача 1. Рассмотрим следующую задачу (Producer–consumer problem). В программе определены 2 потока, один из которых записывает элементы в общую очередь, а другой эту очередь читает и обрабатывает.

```cpp
#include <iostream>
#include <thread>
#include <queue>

int main() {
    size_t count = 0;
    bool done = false;
    std::queue<int> items;

    std::thread producer([&]() {
        for (int i = 0; i < 10000; ++i) {
            // ... some code may be here ...
            items.push(i);
            count++;
        }
        done = true;
    });

    std::thread consumer([&]() {
        while (!done) {
            while (!items.empty()) {
                items.pop();
                // ...
                count--;
            }
        }
    });

    producer.join();
    consumer.join();
    std::cout << count << std::endl;
}
```

Необходимо решить проблемы синхронизации доступа к очереди, счетчику и булевому флагу. Исправьте код соответствующим образом.

Задача 2. Реализуйте интерфейс классов `IProduct` и `IShop`, которые представляют собой товары и магазины, и поддерживают следующие возможности:

- Наследники класса `IProduct` (конкретные товары) должны уметь прикреплять себя к конкретным магазинам (`product.Attach(shop)`), отписывать магазинов от себя (обратная операция `product.Detach(shop)`). Прикрепление означает, что магазин начинает следить за товаром и продавать его по заданной в продукте цене `product.GetPrice()` (проставляется в конструкторе). Здесь `shop` — указатель на магазин, который может быть `nullptr`.
- Каждый продукт имеет методы `StartSales()` и `StopSales()`, которые означают старт и остановку продаж соответственно во всех магазинах, которые их продают. При этом в деструкторе товаров всегда есть вызов `StopSales()`. После `StopSales()` продажи больше не начинаются.

- Каждый продукт имеет метод `ChangePrice(double price)`, который изменяет его цену. В этом случае подписанные магазины обязаны начать продавать его по новой цене.
- Каждый конкретный магазин может закрыться по своему усмотрению (это означает удаление соответствующего объекта), и не обязан сообщать продуктам об этом.

Основной сценарий работы такой системы — многопоточный, в котором есть два основных параллельных треда. В первом из них (не основном) есть последовательность операций с продуктами вида

```
// class A : public IProduct {/*...*/}
// class B : public IProduct {/*...*/};
// class C : public IProduct {/*...*/};
A prod1(15);
prod1.StartSales();
prod1.Attach(shopPtr1);
prod1.Attach(shopPtr2);
B prod2(13);
prod2.StartSales();
prod2.Attach(shopPtr3);
prod2.Attach(shopPtr1);
prod1.Detach(shopPtr2);
prod2.Detach(shopPtr3);
prod2.ChangePrice(12.99);
prod1.ChangePrice(16);
C prod3(45);
prod3.Attach(shopPtr1);
```

Во основном треде до создания первого треда сконструированы магазины, а во время работы второго они занимаются продажей товаров и могут закрываться насовсем.

(1) Реализуйте указанные взаимодействия (нужно самостоятельно придумать простые реакции магазинов на изменения цен, и действия «продаж» — как проявляет себя магазин в треде). API классов кроме указанных требований свободно для реализации. Воспользуйтесь рассмотренными на занятиях методами синхронизации и паттернами проектирования.
(2) Продемонстрируйте работу программы в многопоточной среде.
(3) С использованием любой библиотеки тестирования напишите unit-тесты для нескольких конкретных взаимодействий.

Решение задач нужно прислать (предпочтительнее ссылка на github, и т. п.) на `vkonovodov@gmail.com`. Решение должно быть оформлено в виде компилирующегося кода.

Срок сдачи задания — 30 декабря 2018 г. 23:59.