**Паттерн Наблюдатель (Observer)**

**1. Постановка задачи**

На основе паттерна Наблюдателя, написать программу для слежения за состоянием выбранного файла.  
   
Ограничимся двумя характеристиками:  
 1. Существует файл или нет;  
 2. Каков размер файла.  
   
Программа будет выводить на консоль уведомление о произошедших изменениях в файле.  
 Существует несколько ситуаций для наблюдаемого файла  
 1. Файл существует , файл не пустой - на экран выводится факт существования файла и его размер.  
 2. Файл существует, файл был изменен - на экран выводится факт существования файла, сообщение о том что файл был изменен и его размер.   
 3. Файл не существует - на экран выводится информация о том что файл не существует.  
   
В главной программе создаем объект для отслеживания состояния конкретного файла,  
 затем определяется объект наблюдатель, после чего связываем Наблюдателя с Источником.  
 Можно использовать бесконечный цикл, в котором будем обновлять состояние объекта Источника каждые 100 миллисекунд  
 (например так std::this\_thread::sleep\_for( std::chrono::milliseconds( 100 ) );).

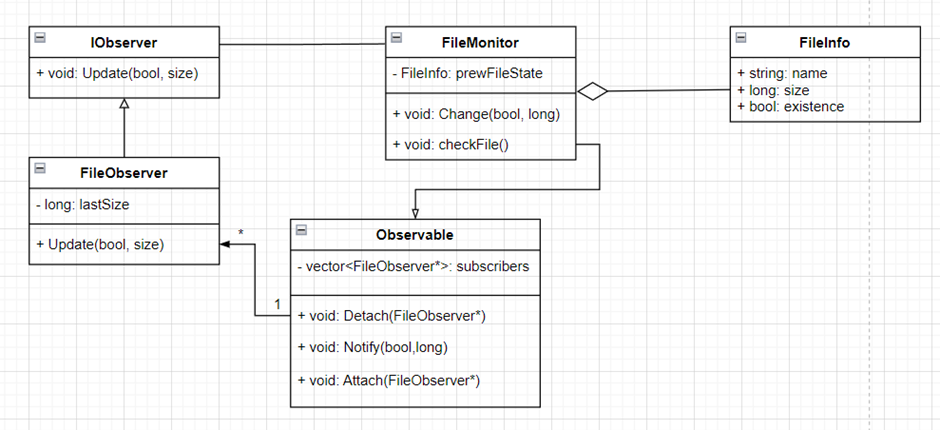
**2. Предлагаемое решение.**

Предлагаемое решение излагается на уровне идеи. Описывается общее решение задачи, с указанием структуры решения, в частности описание предлагаемых сущностей(классы), функций. Если в качестве решения предлагается архитектура ПО, она также описывается и представляется с помощью UML диаграммы. Если предложенная архитектура ПО базируется на одном их известных паттернов, то в отчете обязательно описывается назначение этого паттерна его архитектура, а также какие шаги были выполнены для применения этого паттерна для решения задачи. (Таким образом сдаете теоретическую часть, рассматриваемой темы + ваше решение)

Паттерн Наблюдатель — это поведенческий шаблон проектирования, который использует отношение "один ко многим". В этом отношении есть один наблюдаемый объект и множество наблюдателей. И при изменении наблюдаемого объекта автоматически происходит оповещение всех наблюдателей. То есть создаётся механизм подписки, позволяющий одним объектам следить и реагировать на события, происходящие в других объектах.

Этот паттерн предоставляет методы, с помощью которых “наблюдатели” могли подписываться и отписываться от наблюдаемого объекта (то есть добавлять себя в список подписчиков и убирать себя из него), паттерн Наблюдатель хранит внутри объекта издателя список ссылок на объекты подписчиков. Когда произойдет в наблюдаемом объекте что-то новое, то паттерн будет оповещать весь список подписчиков об этом специальным методом. Все подписчики имеют единый метод оповещения.

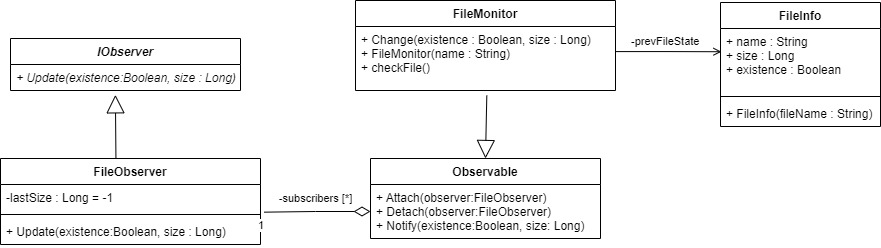
Создаём шаблонный класс наблюдателя, затем от него реализуем class **FileObserver,** в котором будут вызываться уведомления о каждом изменении файла. Реализуем class **FileInfo**, хранящий размер и существование файла.Также создаём class **FileMonitor,** хранящий предыдущее состояние файла, этот класс будет оповещать наблюдателей об изменении файла. Class **Observable,** отслеживающий всех наблюдателей (можно добавлять и удалять наблюдателей) и class **IObserver** , в котором методы вызывается при каждом изменении.



FileInfo(f)



Попытка исправить:



**3. Коды программ. Полностью прокомментированы, с включением объяснений решения.**

fileinfo.h

fileinfo.cpp

observable.h

observable.cpp

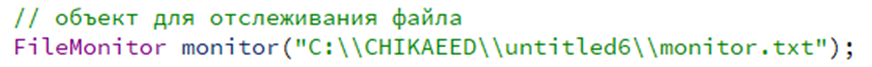
observer.h

observer.cpp

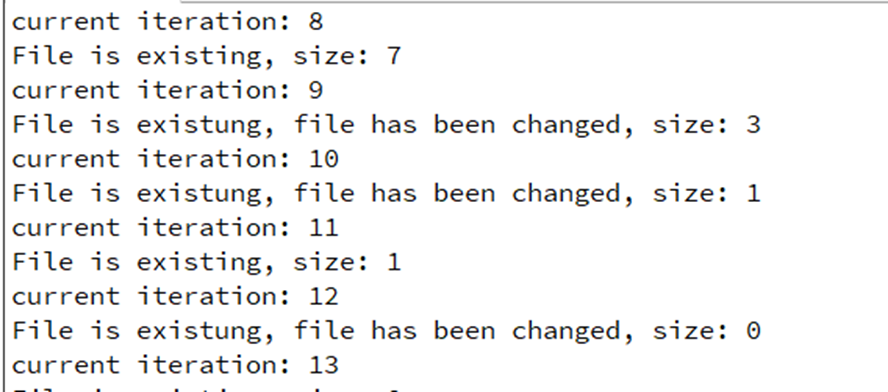
main.cpp

**4. Инструкция пользователя. В этом разделе демонстрируются варианты взаимодействия с вашим ПО с точки зрения пользователя. Если требуется, то предоставляете готовые примеры (входные файлы с исходными данными и т.д.).**

Пишем в скобки полный путь до файла, за которым пользователь будет наблюдать:



Дальше у нас запускается бесконечный цикл, который отслеживает состояние нашего файла.

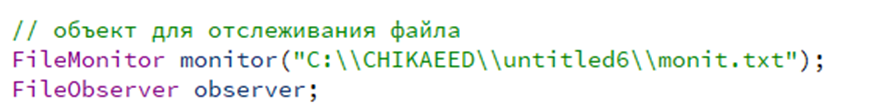


Написана текущая итерация, существует ли файл, изменен ли он и его размер.

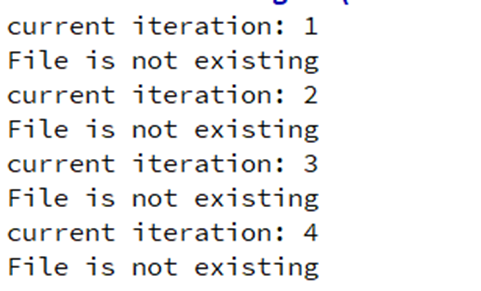
**5. Тестирование. В этом разделе описывается процесс тестирования, в частности все случаи (case1, case2) допустимые и недопустимые при которых ваше ПО выдает адекватную реакцию. Также, тестовые примеры, относительно которых проводилось тестирование.**

Проверка на отсутствие файла:

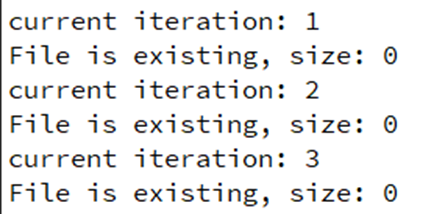
Пишем путь до файла, которого нет в папке проекта



Запускаем программу, запускается бесконечный цикл, оповещающий нас, что файла нет

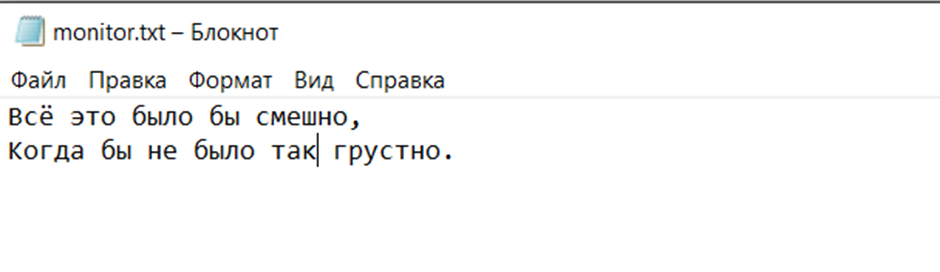


Теперь добавим существующий, но пустой файл:

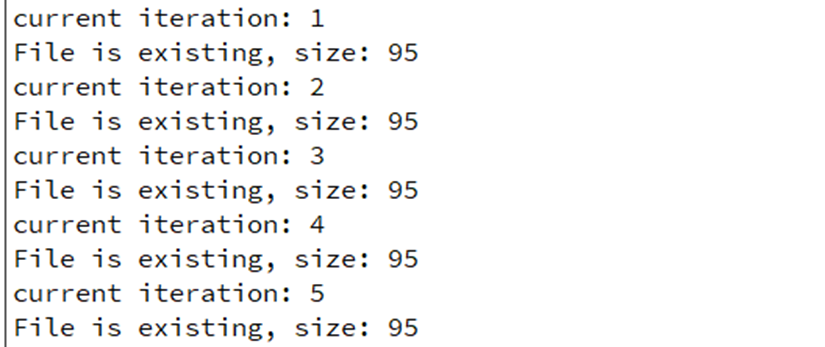


Работает корректно

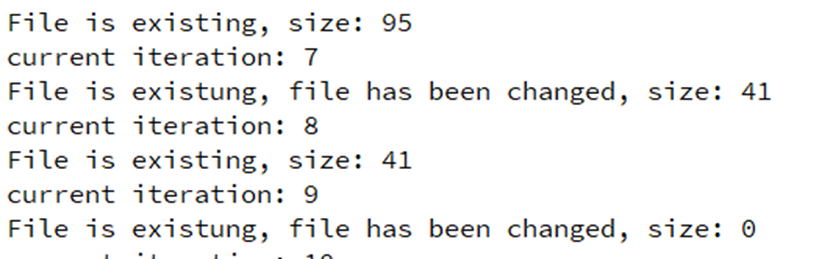
Добавим в файл цитату



Теперь запустим и проверим работу программы снова



А теперь запущу программу снова, во время работы программы удалю вторую строчку, сохраню файл, удалю первую строчку и сохраню файл, в консоли должны быть видны все изменения и в конце файл будет пустым:



А теперь во время работы программы я удалю файл

