Злобин Глеб, гр. 931920

***1. Постановка задачи***

На основе паттерна Наблюдателя, написать программу для слежения за состоянием выбранного файла. Программа должна выводить следующую информацию:

* Существует файл или нет
* Каков размер файла

Программа должна 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.

***2. Предлагаемое решение***

Ввиду того, что в условии требуется реализовать программу с использованием паттерна Наблюдателя, дадим определение этого понятия. ****Наблюдатель**** — это поведенческий шаблон проектирования, который создаёт механизм подписки, позволяющий одним объектам следить и реагировать на события, происходящие в других объектах.

Паттерн Наблюдатель предлагает хранить внутри объекта издателя список ссылок на объекты подписчиков, причём издатель не должен вести список подписки самостоятельно. Он предоставит методы, с помощью которых подписчики могли бы добавлять или убирать себя из списка. Когда в издателе будет происходить важное событие, он будет проходиться по списку подписчиков и оповещать их об этом, вызывая определённый метод объектов-подписчиков.

Все подписчики должны следовать общему интерфейсу и иметь единый метод оповещения.

Можно провести аналогию из жизни: при подписке на газету/журнал издательство само будет присылать новые номера, т.е. не нужно ездить и проверять, не вышел ли новый номер.

Наблюдатель будет реализован с использованием фабричного метода, решающий схожую проблему, что и метод использования шаблонных классов (который мы рассматривали в 1-й лабораторной) - использование функций/класса для разных типов данных. Напомним, что шаблонные классы позволяют работать одному классу с разными типами данных (получается некий «параметрический» класс). Фабричный метод решает эту проблему несколько иначе: он подразумевает создание целых отдельных классов, являющихся наследниками общего класса-интерфейса, и в каждом из которых функции определяются совершенно по-разному. Этот метод более гибкий.

В то же время шаблонные классы у нас также будут применены: сначала мы определим файл с классом самого Наблюдателя в классическом представлении, являющимся шаблонным классом, т.к. мы можем создавать наблюдателей для разных целей (файл observable.h, класс Observable с параметром FileObserver); в другом файле (FileObserver.h) мы определим именно наблюдателя для отслеживания файлов (класс FileObserver); в третьем файле (ConsoleFileObserver.h) мы применим фабричный метод и создадим класс-наследник от класса FileObserver для вывода наблюдений в консоль (класс ConsoleFileObserver). Можно создать и больше наследников для различных способов вывода информации, но ограничимся выводом в консоль.

Не забываем, что все методы всех классов лучше определять в одном файле вместе с объявлениями (.h), а не во внешнем файле (.cpp), т.к. это может привести к конфликту между определениями одних и тех же методов в компиляторе (компилятор забудет, что были использованы несколько разных экземпляров шаблона класса для разных типов данных, а не только один).

В отдельном файле (fileAbout.h) определена структура, переменные которой содержат информацию о файле. Переменная exists говорит о том, существует файл, или нет; переменная size содержит информацию о размере файла; перегрузка оператора != нужна для проверки того, различаются ли два файла, или нет (возвращается булевое значение).

Также отдельным файлом реализован класс FileMonitor - наследник самого первого класса Observable, посылающий наблюдателям сигнал об изменениях в файлах. По сути, он непосредственно отслеживает изменения в файлах («мониторит» файлы), и при их возникновении запускает нужные функции из файлового наблюдателя FileObserver.

Перечислим все имеющиеся функции в каждом классе:

Класс **Observable**:

* Деструктор, который объявлен виртуальным, чтобы его использовать при дальнейшем наследовании данного класса
* registerObserver - функция, создающая связь с имеющимися наблюдателями (принимающая одного наблюдателя в аргументе, а затем вызывающаяся рекуррентно).
* unregisterObserver - функция, обратная предыдущей
* Конструкор по умолчанию
* notifyObservers - функция для уведомления всех наблюдателей об изменениях (их число может быть произвольным), вызывается из класса FileMonitor при проверке изменений в файлах.
* Переменные m\_observers, m\_requests, m\_count для хранения наблюдателей, запросов на добавление наблюдателей и количества наблюдателей.

Класс FileObserver (интерфейс):

* Деструктор (снова виртуальный)
* Функции onFileChanged и fileStatus, сделанные виртуальными по той же причине, что и деструкторы (будут использоваться при наследовании). Изначально (в данном интерфейсе) функции не определены, будут конкретизироваться позже в конкретном наблюдателе.

Класс ConsoleFileObserver (наследник класса FileObserver) - класс конкретизирует определённые ранее функции, представляя на выходе консольный наблюдатель.

Класс FileMonitor:

* Конструктор, принимающий в качестве аргумента имя/путь к отслеживаемому файлу. В нём инициализируются переменные обоих сравниваемых файлов - как при предыдущем, так и при текущем состоянии (которое было принято в аргументе).
* checkFile - функция, отслеживающая изменения в файлах. Вызывает функции из других классов (посылает «сигналы»), выполняющие дальнейшие действия наблюдателя. Если изменения в файле есть - вызывает функцию onFileChanged и также перезаписывает переменную файла с предыдущим состоянием, если изменений нет - то вызывает функцию fileStatus, которая просто выводит информацию о файле.
* getFileAbout - функция узнаёт непосредственно информацию о файлах (используя инструментарий библиотеки stat), вызывается из функции checkFile. Возвращает переменную структуры fileAbout.

***3. Коды программ***

Коды прилагаются в файлах:

FileObserver.h

observable.h

ConsoleFileObserver.h

fileAbout.h

fileMonitor.h

Main.cpp

Проверка их работы осуществляется в файле «main.cpp».

***4. Инструкция пользователя***

Для работы наблюдателя нужно объявить (в файле main) переменные типа FileMonitor (которая будет отслеживать изменения, и в аргументе которой нужно указать путь к файлу) и переменную наблюдателя, выводящий результаты наблюдений нужным нам образом (мы реализовали только ConsoleFileObserver, т.е. наблюдателя, выводящего результат в консоль). Путь к файлу можно задать двумя способами: 1 - если файл находится в папке с проектом, то в 14 строке файла main.cpp дописать дальнейший путь к нему (например: + “Input/file1.txt”), 2 - задать полный путь к файлу (если он находится не в папке с проектом) в 21 строке (но тогда её нужно раскомментировать, а 13-17 строки - закомментировать). Первый способ я посчитал более грамотным, поэтому его оставил в качестве основного, а 2-й закомментировал.

На данный момент переменные наблюдателя созданы (одна из них получает сигналы, другая - выполняет конкретные действия), теперь нужно их связать, используя функцию registerObserver. Далее можно выполнять проверку изменений в файлах неограниченное число раз, как вариант - можно сделать бесконечный цикл, вызывающий функцию checkFile каждые n секунд (используя библиотеку <thread>). Можно запустить наблюдения для нескольких файлов сразу, при этом привязав их к одному наблюдателю (т.е. создать несколько переменных FileMonitor и одну переменную ConsoleFileObserver) - тогда результаты наблюдений для нескольких файлов будут выводиться одновременно. Можно привязать и каждый файл к своему наблюдателю - программа допускает все эти случаи.

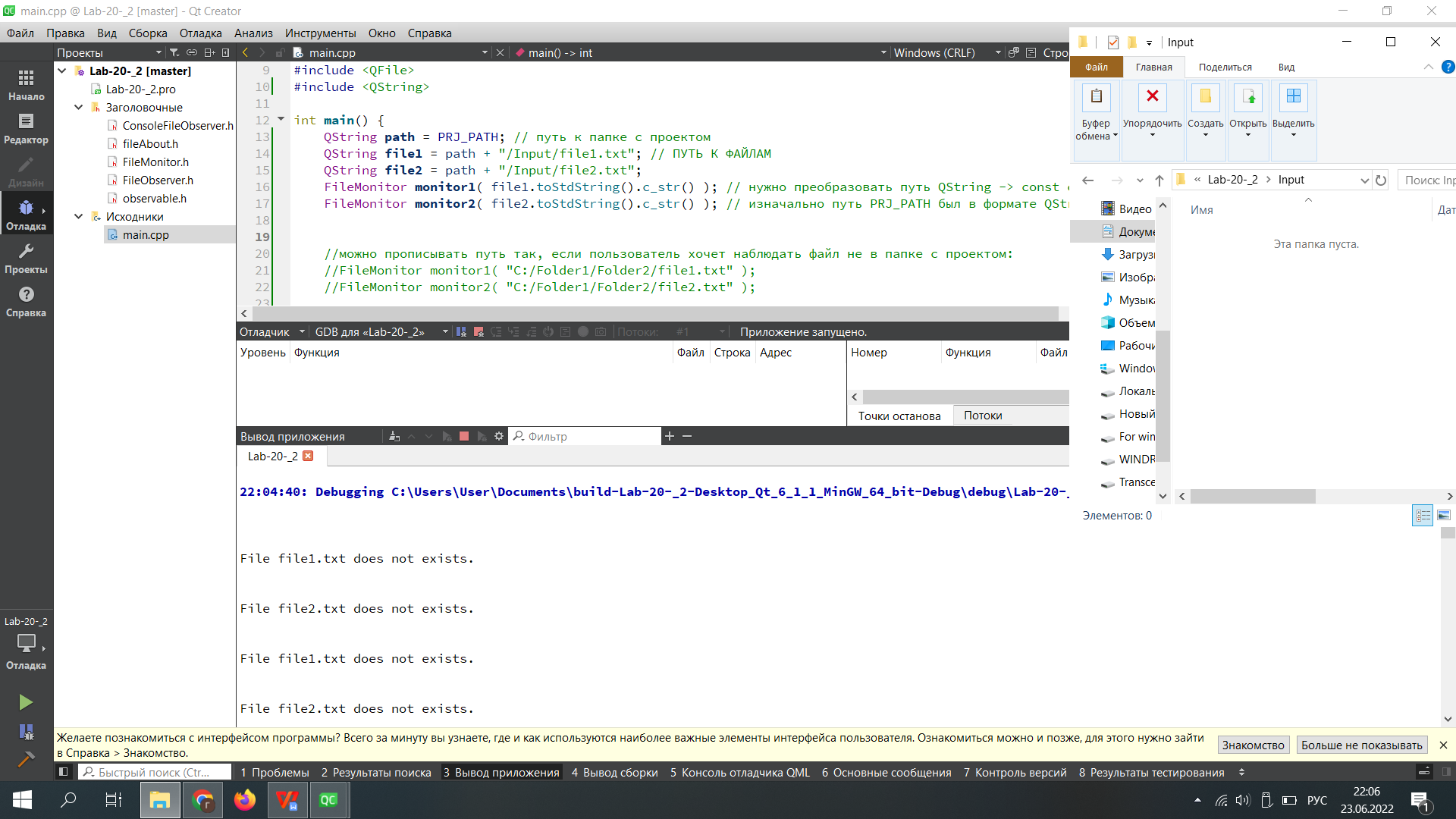
***5. Тестирование***

Нужно создать файлы, из которых будут считываться данные. Заранее созданы файлы file1.txt и file2.txt в папке Input, находящейся в папке с проектом, и в main-е прописаны пути к ним при создании переменных FileMonitor (16 и 17 строки).

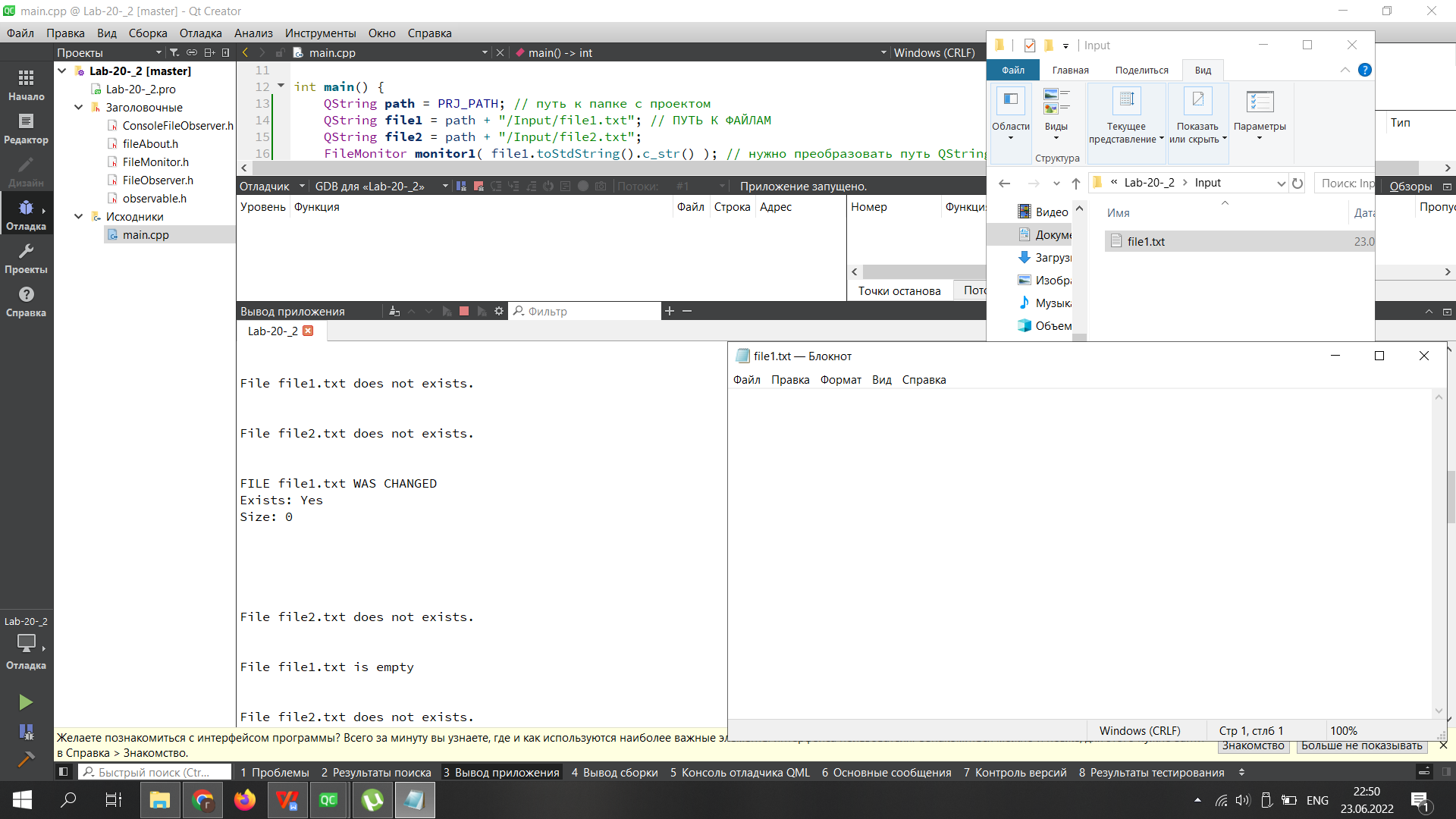
Путь к файлам можно задать и произвольный, используя 2-й способ задания в 21-22 строках main.cpp.

Поскольку программа отслеживает и выводит все изменения автоматически, достаточно один раз запустить отладчик, и в любые моменты времени вносить изменения в файл. Все варианты тестирования будут продемонстрированы скриншотами текстового документа и вывода программы. В процессе работы отладчик не будет выключаться (чтобы улавливались изменения файлов).

**Case 1:** Пусть изначально файлы file1.txt и file2.txt не будут существовать - вот, что будет выводить программа через каждый определённый промежуток времени:

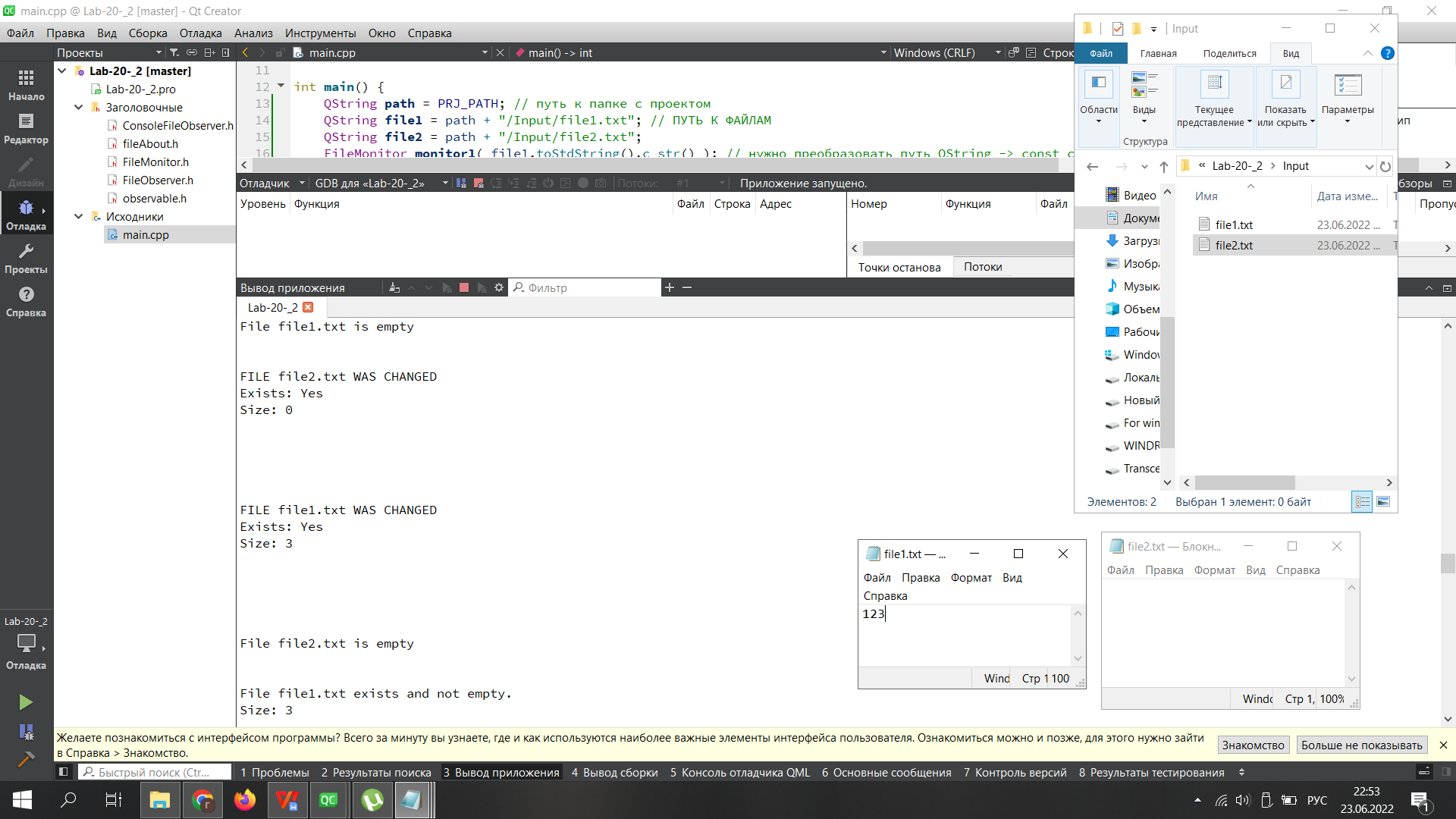


**Case 2:** Создадим один файл (пока пустой):



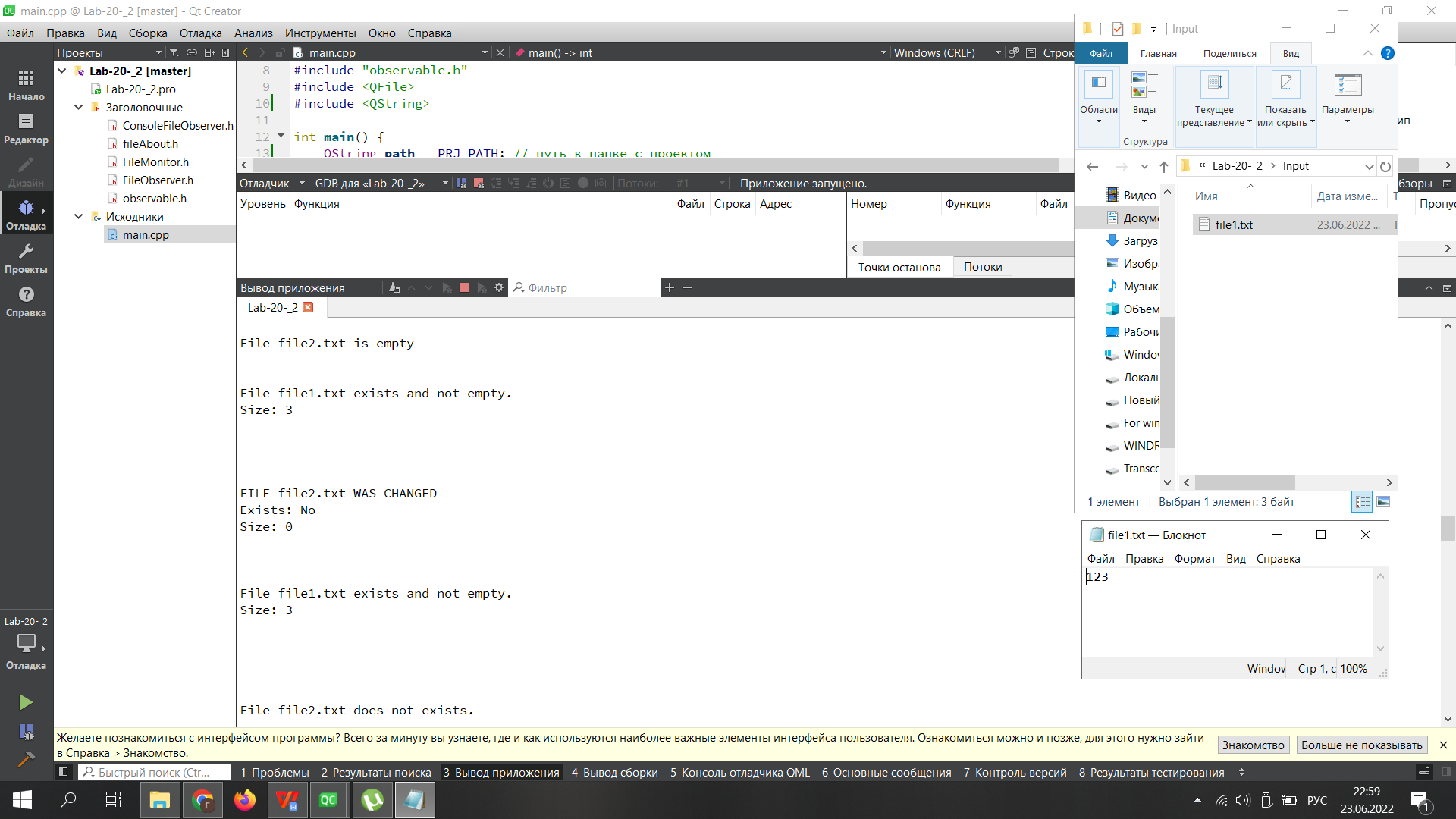
Сначала вывелось уведомлении об изменении 1-го файла (он создался), затем стало выводиться сообщение, что он пустой. Для 2-го файла всё выводится без изменений.

**Case 3**: Одновременно (почти одновременно) создадим 2-й файл (также пустым) и внесём 3 случайных символа в 1-й файл:



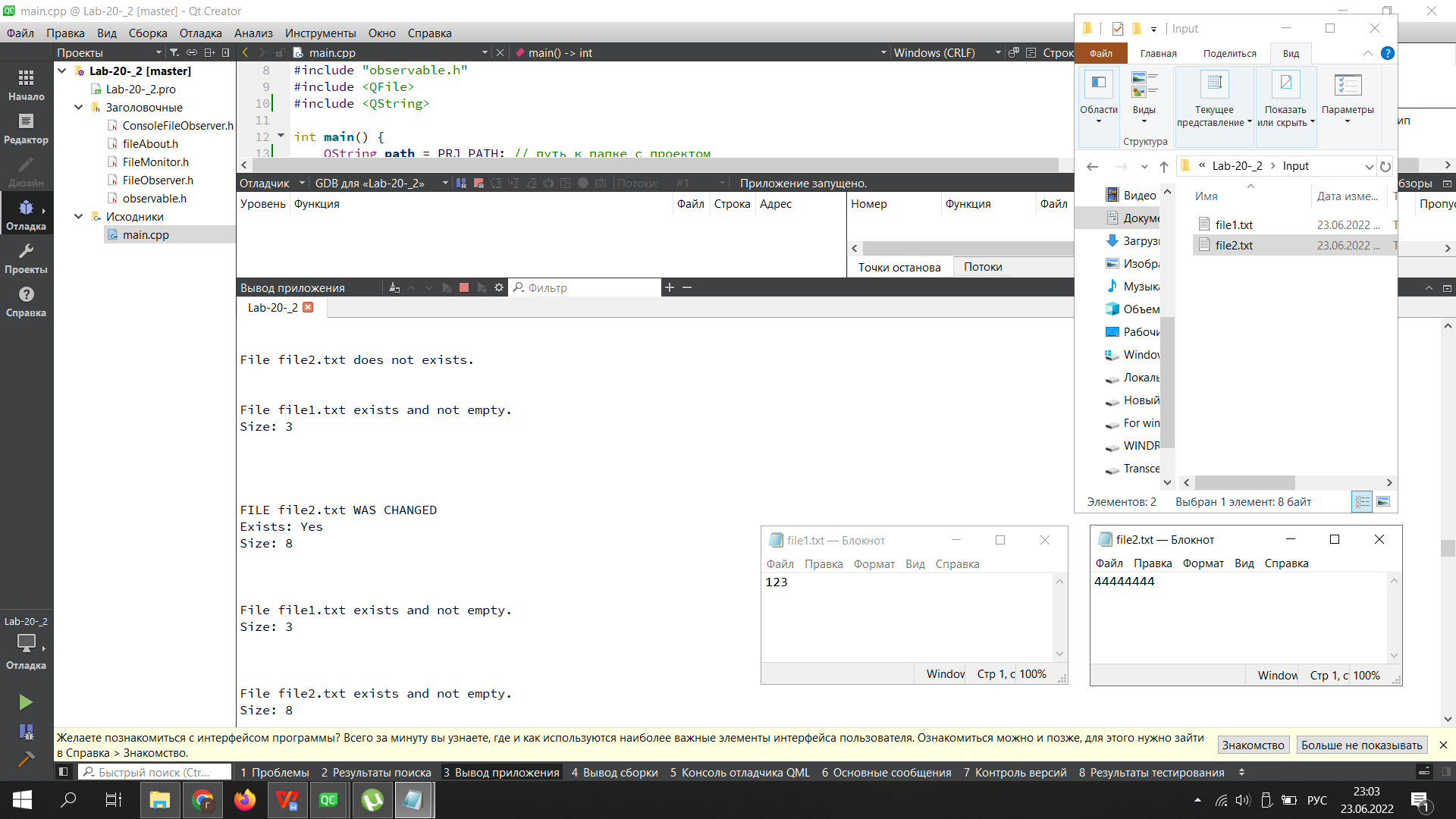
Вывелись уведомления об изменениях обоих файлов, корректно указывающие информацию об их существовании и размере. Затем циклически стала выводиться информация об этих файлах.

**Case 4**: Удалим 2-й файл:



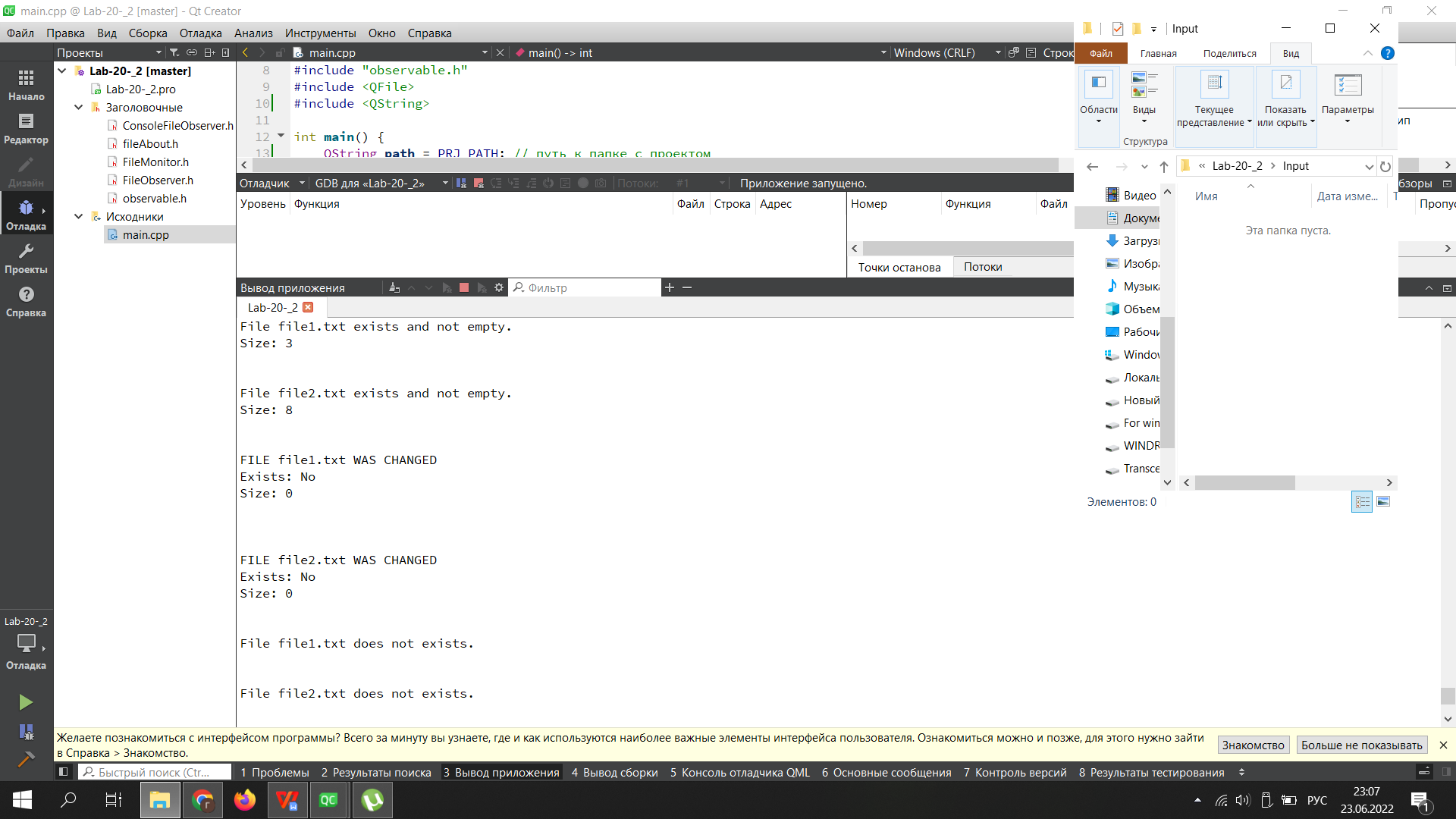
Вывелось уведомление об изменении 2-го файла (он больше не существует), и затем циклически стала корректно выводиться информация об обоих файлах (1-ый - всё так же имеет длину 3, 2-й - не существует).

**Case 5**: Создадим 2-й файл, в котором сразу будут содержаться символы:

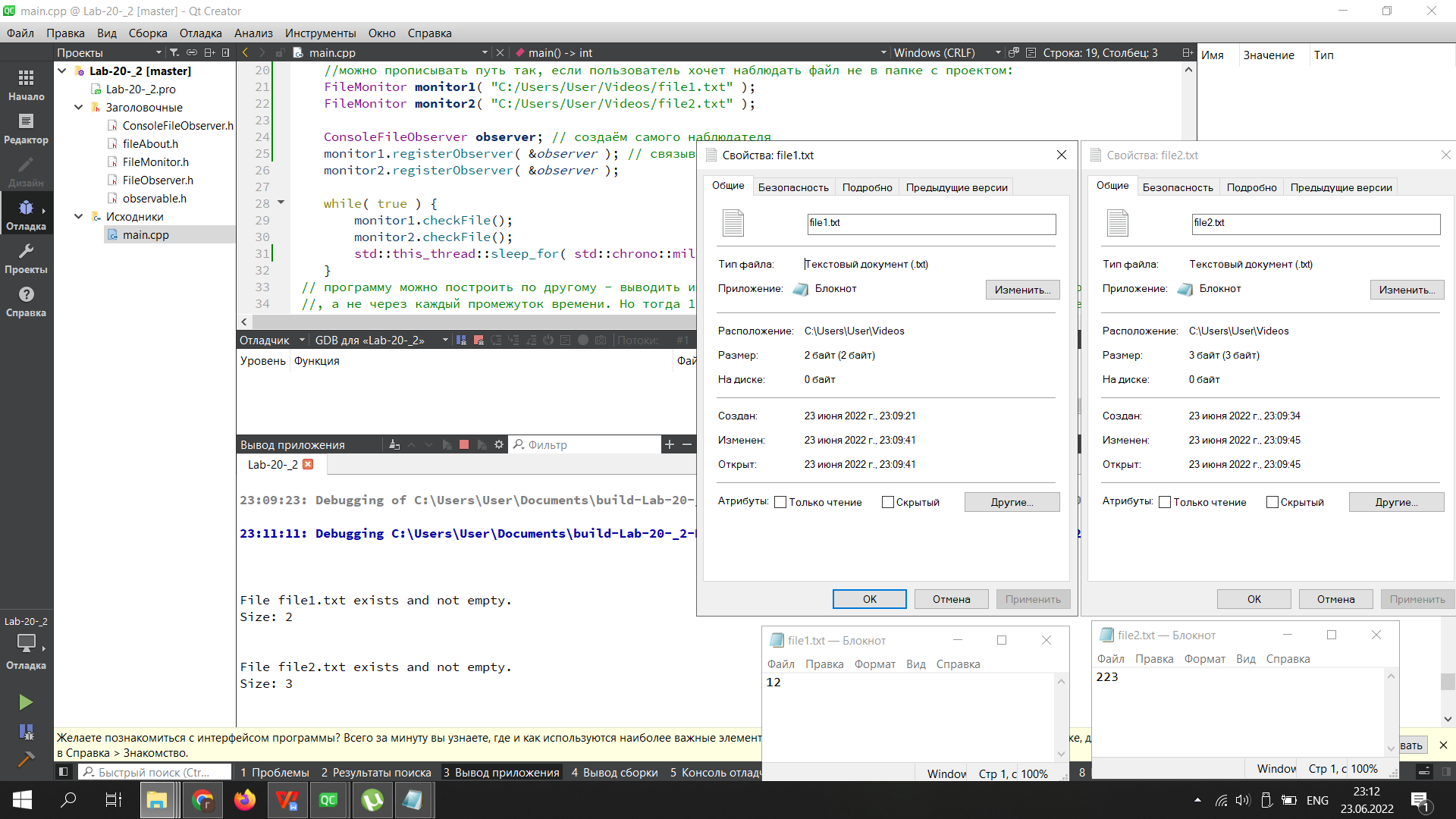


Вывелось корректное уведомление об изменении 2-го файла (теперь он существует и длина 8 символов), затем циклически выводится корректная информация об обоих файлах.

**Case 6**: Удалим оба файла.



**Case 7**: Проверим работу программы при записи путей к файлу 2-м способом, т.е. если они находятся вне папки с проектом. Прописанный путь в main, реальный путь в проводнике, и результаты программы - на скриншоте:



Таким образом были рассмотрены все возможные случаи работы наблюдателя из условия задания:

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.