Пиманкин Данил, студент группы 931921

Отчет по лабораторной работе №1

Часть 1. Шаблонный класс, реализующий стек

1. Постановка задачи

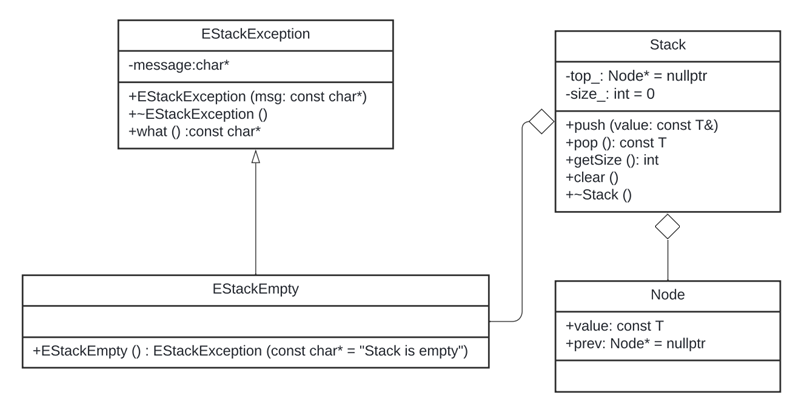
Задача состоит в создании шаблонного класса, реализующего стек. Класс должен поддерживать следующие операции:

1. Помещение объекта в стек;
2. Извлечение объекта из стека;
3. Получение размерности стека.

В случае попытки вызова операции извлечение объекта из стека при условии, что стек пуст, должно генерироваться исключение класса **EStackEmpty (**наследник класса EStackException**)**. Данный класс должен содержать публичный метод char\* what(), возвращающий диагностическое сообщение.

2. Предлагаемое решение

Архитектура предлагаемого решения представлена на UML-диаграмме.



Шаблонный класс Stack

Стек реализован на основе однонаправленного списка. Каждый узел списка – структура Node, хранящая значение и указатель на предыдущую вершину списка. Таким образом, член-данными класса Stack выступает указатель на вершину стека (агрегация), и размерность стека.

Метод push(), помещающий элемент в стек, ставит новый элемент над вершиной стека и смещает указатель на вершину к нему.

Метод pop(), извлекающий элемент из вершины стека, помимо считывания содержимого вершины удаляет её из стека. Если стек пуст, создается ошибка класса EStackEmpty (агрегация).

Метод clear(), очищающий стек, проходит стек с вершины, удаляя последовательно каждый элемент. Используется в деструкторе стека.

Исключения, связанные с работой стека

Исключения стека реализованы в классе EStackExeption и его наследнике EStackEmpty (наследование). Член-данным базового класса выступает строка с диагностическое сообщением, записывающаяся конструктором. Получить диагностическое сообщение позволяет метод what().

Класс EStackEmpty отвечает за ошибку, возникающую при попытке извлечь вершину пустого стека. Конструктор EStackEmpty вызывает конструктор базового класса, передавая аргументом сообщение “Stack is empty”.

3. Коды программ

Коды программ размещены на удаленном репозитории GitHub: https://github.com/danilpimankin/TRPO/tree/main/Lab\_1-develop

Объявления и определения классов реализованы в следующих файлах:

* EStackEmty.h – класс EStackEmty
* EStackExeption.h – класс EStackExeption
* Stack.h – класс Stack

4. Инструкция пользователя

Пользователем выступает программист, использующий класс Stack.

Использование класса-шаблона подразумевает указания типа хранимого данного с стеке при объявлении стека:

Stack <тип\_данных> имя\_стека;

Например, стек имен names, хранящий строковые данные в формате std::string объявляется следующим образом:

Stack <std::string> names;

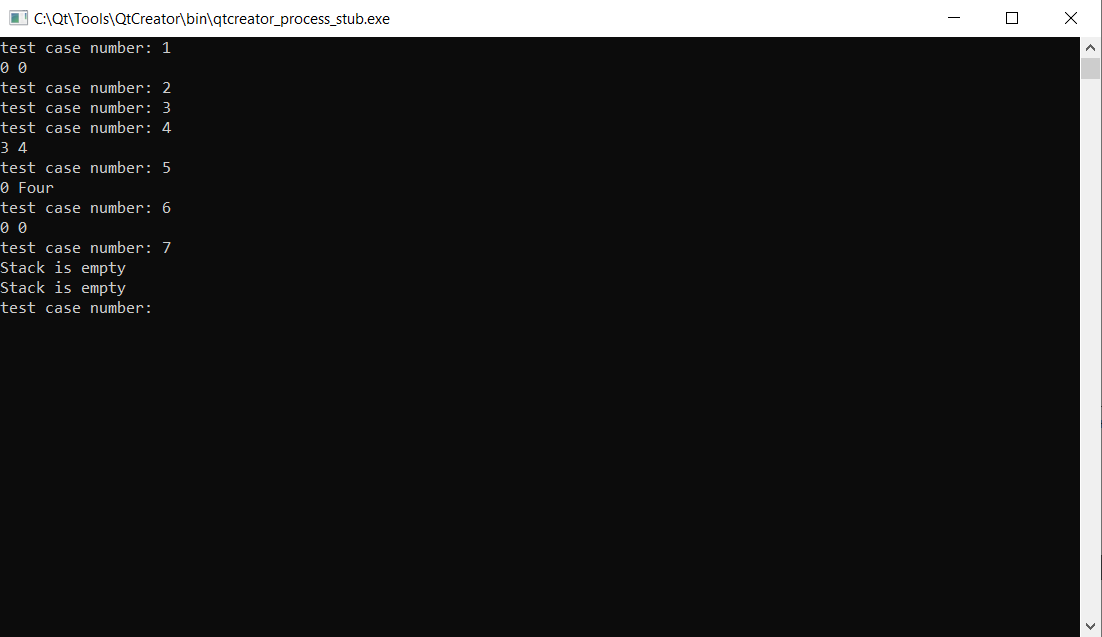
Для добавления элемента в стек используется метод push(), аргументом которого выступает переменная с типом данных, хранящихся в стеке.

Организация стека подразумевает, что в его вершине хранится последнее занесенное значение.

Извлечение из стека производится с его вершины, т.е. реализуется правило «последним пришел, первым ушел». В случае попытки применения метода pop() к пустому стеку будет выброшено исключение класса EStackEmpty.

5. Тестирование

Тестирование проводится на стеках с int- и std::string- данными. Описание тестирования находится на первом листе приложенного файла “Тестирование лаб\_1.xlsx”. Результаты тестирования представлены на скриншоте.



Код программы тестирования:

while (1) {

int case\_number;

cout << "test case number: ";

cin >> case\_number;

switch (case\_number) {

case 1: {

Stack <int> intstack;

Stack <string> stringstack;

cout << intstack.getSize() << " " << stringstack.getSize() << endl;

break;

}

case 2: {

Stack <int> intstack;

Stack <string> stringstack;

intstack.clear();

stringstack.clear();

break;

}

case 3: {

Stack <int> intstack;

Stack <string> stringstack;

intstack.push(1);

intstack.push(-1);

intstack.push(0);

stringstack.push("One");

stringstack.push("Two");

stringstack.push("Three");

stringstack.push("Four");

break;

}

case 4: {

Stack <int> intstack;

Stack <string> stringstack;

intstack.push(1);

intstack.push(-1);

intstack.push(0);

stringstack.push("One");

stringstack.push("Two");

stringstack.push("Three");

stringstack.push("Four");

cout << intstack.getSize() << " " << stringstack.getSize() << endl;

break;

}

case 5: {

Stack <int> intstack;

Stack <string> stringstack;

intstack.push(1);

intstack.push(-1);

intstack.push(0);

stringstack.push("One");

stringstack.push("Two");

stringstack.push("Three");

stringstack.push("Four");

cout << intstack.pop() << " " << stringstack.pop() << endl;

break;

}

case 6: {

Stack <int> intstack;

Stack <string> stringstack;

intstack.push(1);

intstack.push(-1);

stringstack.push("One");

stringstack.push("Two");

stringstack.push("Three");

intstack.clear();

stringstack.clear();

cout << intstack.getSize() << " " << stringstack.getSize() << endl;

break;

}

case 7: {

Stack <int> intstack;

Stack <string> stringstack;

try {

stringstack.pop();

}

catch (EStackException& e) {

cout << e.what() << endl;

}

try {

intstack.pop();

}

catch (EStackException& e) {

cout << e.what() << endl;

}

break;

}

default: cout << "no such testcase" << endl;

}

}

Часть 2. Класс, хранящий личности

1. Постановка задачи

Реализовать класс PersonKeeper, имеющий методы readPersons и writePersons:

1. Метод readPersons должен считывать информацию о людях из входного потока (файла), создавать на основе этой информации объекты класса Person, и помещать их в стек.
2. Метод writePersons должен записывать в поток из стека (стек передается аргументом) информацию о людях в соответствии с вышеописанным форматом. Передаваемый методу writePersons стек не должен изменяться.

Формат входного файла: Фамилия Имя Отчество.

Пример файла:

Иванов Василий Иванович

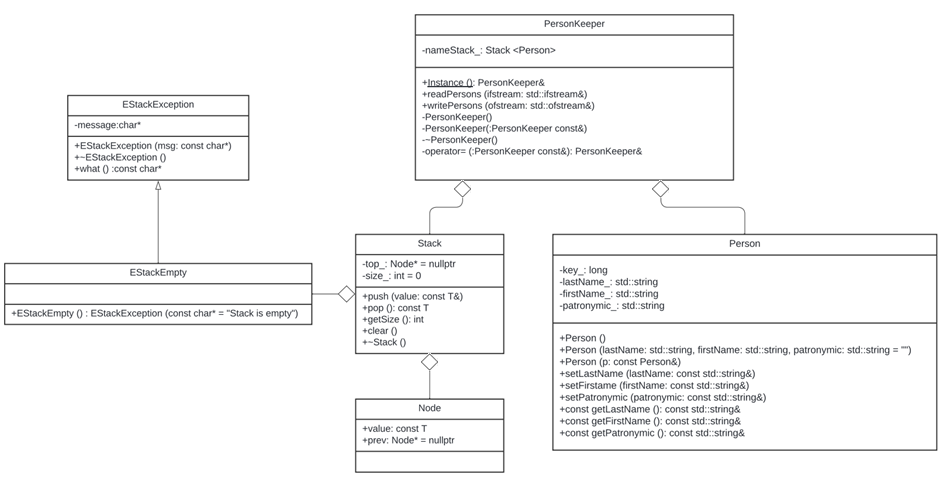
Сидоров Александр Михайлович

…

Класс PersonKeeper должен быть реализован в соответствии с шаблоном Singleton.

2. Предлагаемое решение

Полная архитектура предлагаемого решения представлена на UML-диаграмме. Класс Stack и связанные с ним элементы описаны в первой части лабораторной работы.



Класс Person, хранящий данные о личности

Член-данными класса выступает фамилия, имя и отчество персоны. Они являются приватными, и для работы с ними извне реализованы сеттеры и геттеры: методы setLastName() и getLastName(), setFirstName() и getFirstName(), setPatronymic() и getPatronymic() для работы с фамилией, именем, отчеством соответственно.

Предусмотрен также конструктор со строковыми аргументами (фамилия, имя, отчество) и конструктор копирования.

Класс-одиночка PersonKeeper

Паттерн Singleton (одиночка) - шаблон проектирования, гарантирующий, что в приложении будет единственный экземпляр класса, и предоставляющий глобальную точку доступа к этому экземпляру.

При попытке создания данного объекта он создаётся только в том случае, если ещё не существует, в противном случае возвращается ссылка на уже существующий экземпляр и нового выделения памяти не происходит. За создание или выдачу экземпляра класса в PersonKeeper отвечает статический метод Instance().

Конструктор, деструктор, конструктор копирования и оператор присвоения при таком методе проектирования объявляются в приватной части класса.

Член-данным класса выступает класс-шаблон Stack (агрегация), хранящий объекты класса Person (агрегация).

Метод readPersons(), считывающий личности из файла стек, принимает аргументом адрес файлового потока, открытого для чтения. Его работа основана на построчном чтении файла и разделении каждой строки на отдельные слова.

Метод writePersons(), записывающий личности из стека в файл, также принимает аргументом адрес файлового потока, но открытого для записи. Он последовательно извлекает из стека персон, записывая их фамилию, имя и отчество в файл.

3. Коды программ

Коды программ размещены на удаленном репозитории GitHub: https://github.com/danilpimankin/TRPO/tree/main/Lab\_1-develop

Объявления классов находятся в следующих файлах:

* Person.h – класс Person с реализацией в Person.cpp
* PersonKeeper.h - класс PersonKeeper с реализацией в PersonKeeper.cpp

Файлы для тестирования: NameSource\_1.txt, NameSource\_2.txt, EmptyNameSource.txt

4. Инструкция пользователя

Инструкция для пользователя-программиста, использующего класс PersonKeeper:

Класс PersonKeeper основан на паттерне singleton, поэтому для создания и обращения к единственному объекту класса используется метод Instance().

Для чтения файла из ifstream-потока используется метод readPersons, принимающий адрес потока аргументом. Допускается последовательное чтение из нескольких потоков, содержимое файлов будет дополнять существующий стек.

Для записи стека в файловый ofstream-поток используется метод writePersons, принимающий адрес потока аргументом. В процессе записи стек опустошается.

Формат файлов, с которыми работает класс PersonKeeper описан в инструкции конечного пользователя.

Инструкция для конечного пользователя, взаимодействующего с консольным приложением:

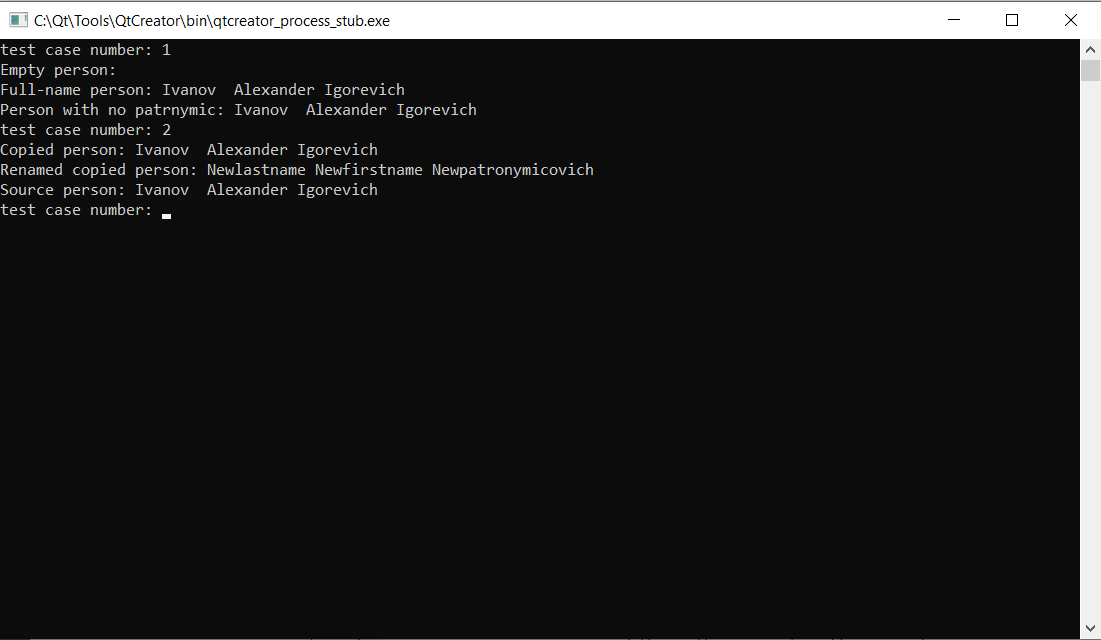
Текстовые файлы, с которыми взаимодействует программа должны иметь расширение “.txt” и содержать список имен в формате “Фамилия Имя Отчество” отделенных друг от друга переносом строки. Частями одного полного имени разделяются пробелами или табуляциями. Допускается отсутствие отчества в полном имени, а также использование многократного пробела или табуляции в качестве разделителя, однако использование других разделителей не допускается. Порядок следования слов строки фиксирован, программа всегда воспринимает первое – как фамилию, второе – как имя, третье – как отчество (при наличии).

Порядок использования программы:

1. Чтобы считать данные из файла, нужно ввести в консоль символ “r”, а затем полный путь к файлу (с указанием расширения), либо название файла с расширением, если он находится в папке проекта.
2. Программа позволяет последовательно считать несколько файлов, в том числе допускается повторное чтение файлов. Для этого повторяются действия пункта 1. Содержимое нескольких считанных файлов объединяется.
3. Чтобы записать считанные данные в файл, в консоль вводится “w” и путь сохранения, оканчивающийся названием файла. Если такого файла нет по указанному пути, он создастся автоматически. Если файл существует – его содержимое будет перезаписано. В процессе записи в файл стек имен опустошается.

5. Тестирование

Тестирование класса Person проведено отдельно от основной программы чтения/записи файлов. Описание тестирования класса Person приведено на втором листе файла “ Тестирование лаб\_1.xlsx”. Результаты тестирование представлены на скриншоте.



Код программы тестирования класса Person:

while (1) {

int case\_number;

cout << "test case number: ";

cin >> case\_number;

switch (case\_number) {

case 1: {

Person emptypers;

Person pers ("Ivanov", " Alexander", "Igorevich");

Person nonpatrinymicpers ("Pavlov", " Konstantin");

cout << "Empty person: ";

cout << emptypers.getLastName() << " ";

cout << emptypers.getFirstName()<< " ";

cout << emptypers.getPatronymic()<< endl;

cout << "Full-name person: ";

cout << pers.getLastName() << " ";

cout << pers.getFirstName()<< " ";

cout << pers.getPatronymic()<< endl;

cout << "Person with no patrnymic: ";

cout << pers.getLastName() << " ";

cout << pers.getFirstName()<< " ";

cout << pers.getPatronymic()<< endl;

break;

}

case 2: {

Person pers ("Ivanov", " Alexander", "Igorevich");

Person copypers (pers);

cout << "Copied person: ";

cout << copypers.getLastName() << " ";

cout << copypers.getFirstName()<< " ";

cout << copypers.getPatronymic()<< endl;

copypers.setFirstame("Newfirstname");

copypers.setLastName("Newlastname");

copypers.setPatronymic("Newpatronymicovich");

cout << "Renamed copied person: ";

cout << copypers.getLastName() << " ";

cout << copypers.getFirstName()<< " ";

cout << copypers.getPatronymic()<< endl;

cout << "Source person: ";

cout << pers.getLastName() << " ";

cout << pers.getFirstName()<< " ";

cout << pers.getPatronymic()<< endl;

break;

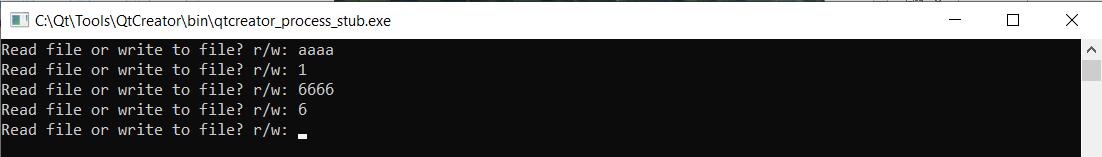
}

default: cout << "no such testcase" << endl;

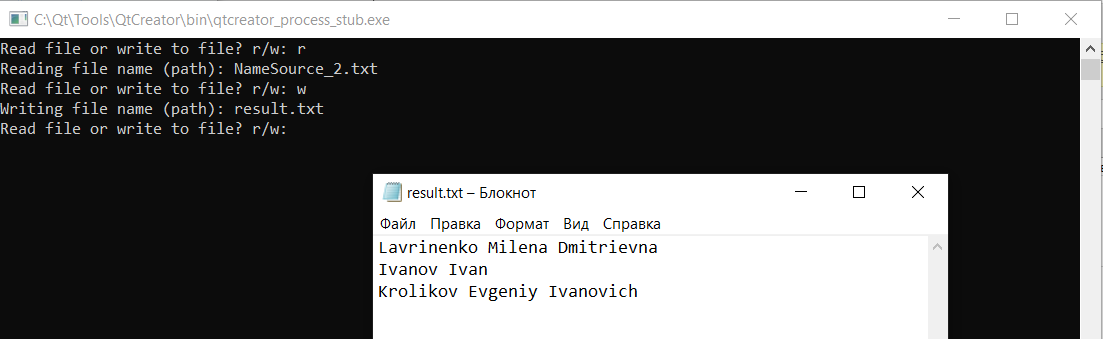
}

}

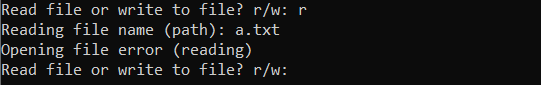
Тестирование класса PersonKeeper проведено на основной программе. Описание тестирования приведено на третьем листе файла “Тестирование лаб\_1.xlsx”. Результаты тестирования представлены на скриншотах.



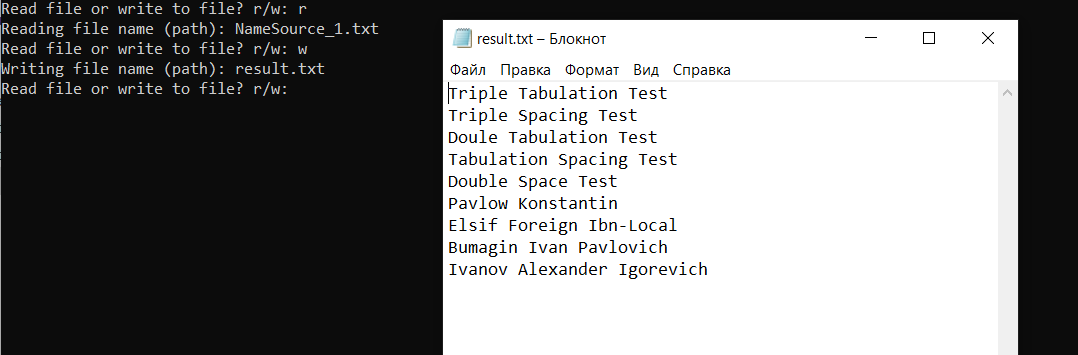
Тест кейс 1



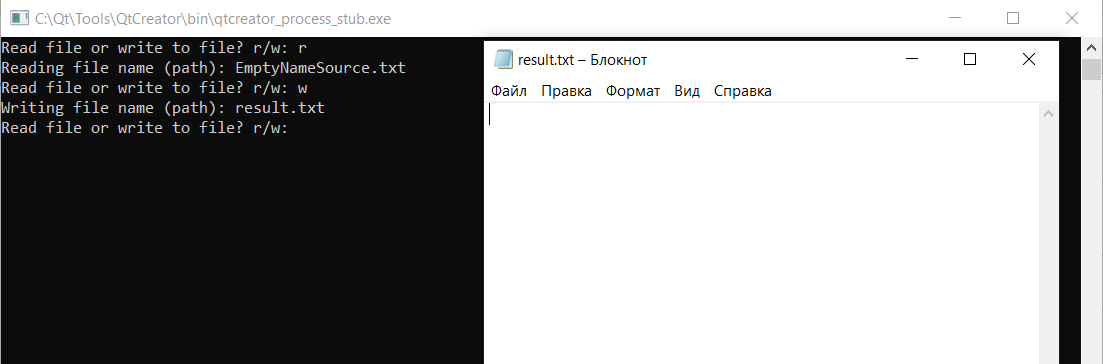
Тест кейс 2



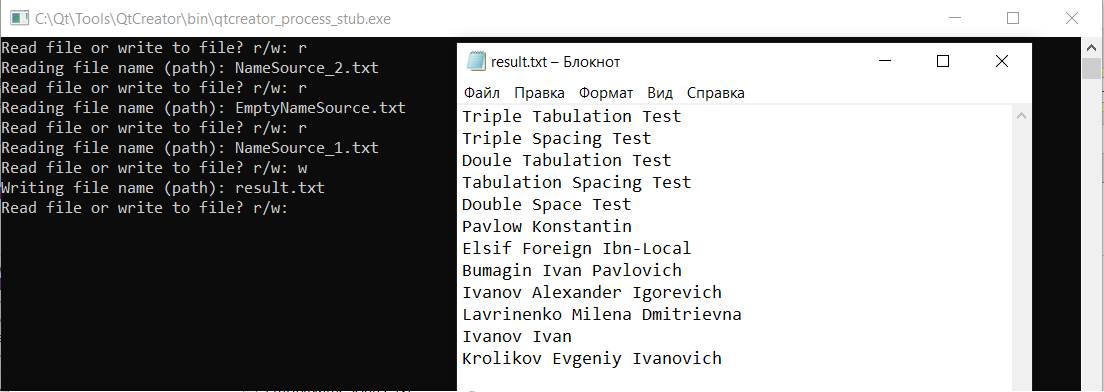
Тест кейс 3



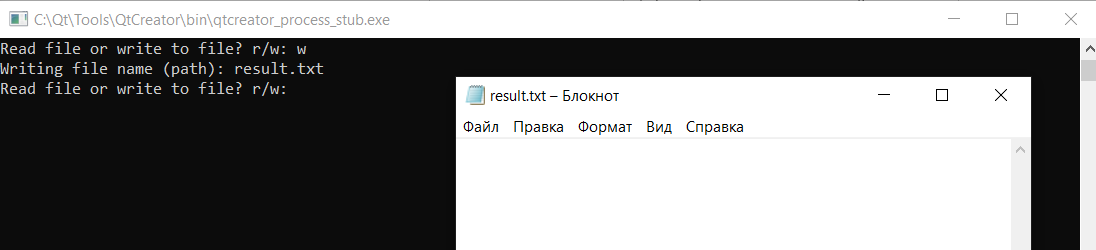
Тест кейс 4



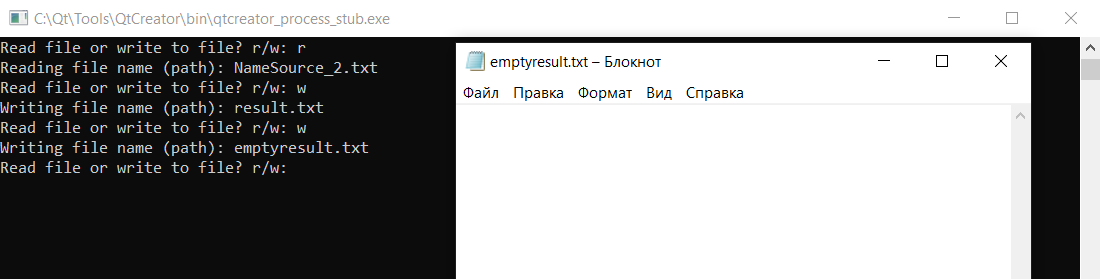
Тест кейс 5



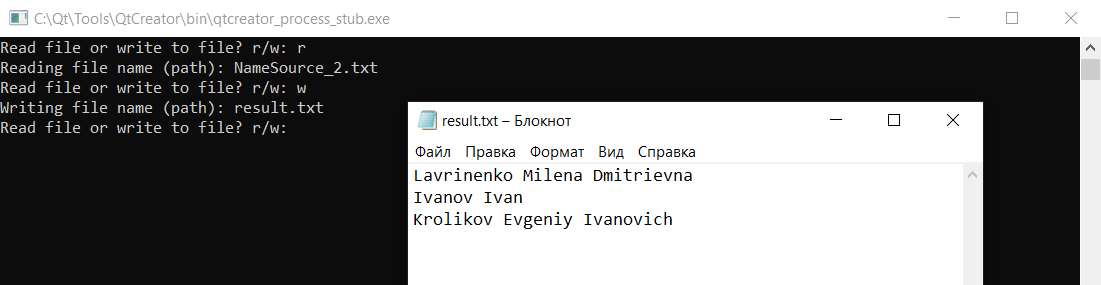
Тест кейс 6



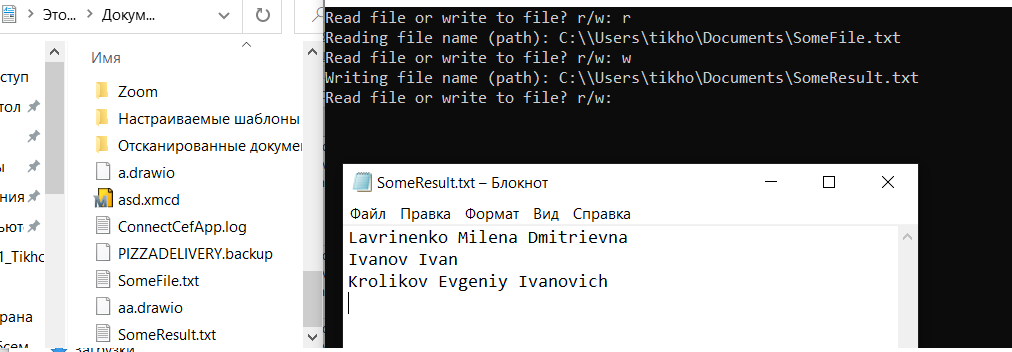
Тест кейс 7



Тест кейс 8



Тест кейс 9



Тест кейс 10