

Часть 1.

1. Постановка задачи

Была поставлена задача запрограммировать шаблонный класс, реализующий стек. Данный класс должен поддерживать следующие операции:

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

В случае попытки вызова операции извлечения объекта из стека при условии, что стек пуст, должно генерироваться исключение класса `EStackEmpty` (который в свою очередь является наследником класса `EStackException`). Данный класс должен содержать публичный метод `char* what()`, возвращающий диагностическое сообщение.

2. Предполагаемое решение

По своей сути, стек это структура данных, которая представляет из себя упорядоченный по времени добавления в стек набор элементов. Эти элементы связаны следующим образом: каждый из них указывает на тот, который идет после него, элементы идут друг за другом и брать их можно строго по очереди. Взять из середины стека не получится. Это и является главным принципом работы стека, «последним пришёл — первым ушёл». После того как верхний элемент стека использован/удаляется/исчезает/испаряется, верхним становится следующий элемент.

Для реализации стека мною был выбран односвязный список, состоящих из узлов. В каждом из них есть два (а больше нам и не надо) поля: в одном из них хранится его значение, в другом - указатель на следующий узел. В самом классе `Stack` хранится список в следующем виде: есть указатель на верхний элемент списка (который и является верхним элементом стека) и размер стека.

Шаблонный класс - это класс, использующий определенные типы, но неизвестно какие они будут, шаблонные классы позволяют описать конструкции, которые не зависят от определенного типа.

Ниже приведены реализации функций класса Stack:

- Конструктор по умолчанию и конструктор копирования
- Функция проверки стека на пустоту (empty())
- Функция получения размера стека (getSize())
- Функция помещения объекта в стек (push(T value_);)
- Функция удаления верхнего узла стека (pop();)
- Функция, позволяющая извлечь верхний элемент без его удаления (topFunc();)

В функциях, которых необходимо, есть проверка стека на пустоту, и в случае таковой вызывается исключение

3. Коды программ

Файлы с кодом программы:

1. Node.h
2. Stack.h
3. Exception.cpp

Находятся на репозитории github.com/timuraknazarov/trpo_lab_1

4. Инструкция пользователя

В первую очередь, так как класс является шаблонным, необходимо указать какой тип данных он будет использовать. В этом примере я буду использовать данные типа int. Выглядит это следующим образом:

```
Stack<int> test;
```

Для помещения чего-го либо в стек необходимо воспользоваться функцией push();, в качестве примера добавим в стек единицу :

```
Stack<int> test;
```

```
test.push(1);
```

Для получения размера стека необходимо вызвать функцию getSize(); Пример:

```
Stack<int> test;

test.push(1);

std::cout<<"stack size: "<<test.getSize()<< std::endl;
```

Для получения значения верхнего элемента стека необходимо вызвать функцию topFunc(); Пример:

```
Stack<int> test;

test.push(1);

std::cout<<"Top: "<<test.topFunc()<< std::endl;
```

Для удаления верхнего элемента стека необходимо вызвать функцию pop();

```
test.pop();|
```

5. Тестирование

Добавим несколько элементов в стек и посмотрим его размер:

```
Stack<int> test;
test.push(10);
test.push(11);
test.push(12);
std::cout<<"stack size: "<<test.getSize()<< std::endl;
```

Вывод приложения:

```
stack size: 3
```

Теперь посмотрим значение верхнего элемента и размер стека, после чего добавим в стек новый элемент и еще раз посмотрим эти значения:

```
Stack<int> test;
test.push(10);
test.push(11);
test.push(12);
std::cout<<"top: "<<test.topFunc()<< " " <<"stack size: "<< test.getSize()<< std::endl;
test.push(13);
std::cout<<"top: "<<test.topFunc()<< " | " <<"stack size: "<< test.getSize()<< std::endl;
```

Вывод приложения:

```
top: 12 stack size: 3
top: 13 stack size: 4
```

На этот раз попробуем узнать размер стека, не добавляя в него ничего:

```
Stack<int> test;
std::cout<< "stack size: "<< test.getSize()<< std::endl;
```

Вывод приложения:

```
stack size: 0
```

Так как в стек мы ничего не добавляли, его размер равен нулю.

Попробуем вызвать функцию pop(); в пустом стеке:

```
Stack<int> test;
try {
    test.pop();
} catch (const excep_stack::EStackEmpty & exception) {
    std::cout << exception.what() << std::endl;
}
```

Вывод приложения:

```
Runtime error: stack is empty. Called function pop()
```

Мы получаем ошибку, так как пытаемся удалить элемент из заведомо пустого стека.

Попробуем вызвать функцию topFunc(); в пустом стеке:

```
Stack<int> test;
try {
    test.topFunc();
} catch (const excep_stack::EStackEmpty & exception) {
    std::cout << exception.what() << std::endl;
}
```

Вывод приложения:

```
Runtime error: stack is empty. Called function top()
```

Мы получаем ошибку, так как пытаемся узнать значение верхнего элемента из заведомо пустого стека.

Это все возможные варианты отработки класса стек. Программа выводит корректные значения.

Программа работает корректно.

Часть 2.

1. Постановка задачи

Необходимо реализовать класс PersonKeeper с методами readPersons и writePersons.

Метод `readPersons` должен считывать информацию о людях из входного потока (файла), создавать на основе этой информации объекты класса `Person`, и помещать их в стек. Формат входного файла должен быть такой:

Фамилия Имя Отчество

В качестве разделителей могут выступать пробелы, табуляции, переводы строки. **Пример файла:**

Иванов Василий Иванович

Сидоров Александр Михайлович

...

Метод `readPersons` должен возвращать стек.

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. Класс `PersonKeeper` должен быть реализован в соответствии с шаблоном `Singleton`.

2. Предполагаемое решение

В качестве решения мною был реализован класс `PersonKeeper`, который отвечает за чтение либо запись стека по информации о людях из входного файла, на основе этих данных создается объект класса `Person`, в котором находятся ФИО людей, а также помещает их в стек. Сам класс `PersonKeeper` был реализован в соответствии с шаблоном `Singleton`, давайте поподробнее поговорим о нем.

`Singleton` это такой порождающий шаблон проекта, который может гарантировать что у класса есть лишь один экземпляр, и предоставляет к нему глобальную точку доступа. Этот шаблон скрывает всевозможные способы создания нового объекта, кроме метода `Instance`, который возвращает ссылку на этот единственный статический объект. Поскольку объект у нас единственный (он же создается в `Instance`), то привычным всем методом вызвать его через объект не получится. Именно поэтому `Instance` - статический метод и мы можем его реализовать вне нашего объекта.

Класс `PersonKeeper` имеет следующие методы:

- instance - статический метод, который возвращает ссылку на единственный экземпляр PersonKeeper
- readPersons - метод, который получает на вход ссылку на входной поток, считывает из потока необходимую информацию и на ее основе создает и возвращает стек Person.
- writePersons - метод, который принимает стек и выходной поток. В нем считывается информация из стека, не изменяя его и записывается в выходной поток.
- Конструктор по умолчанию и конструктор копирования

Для удобства в класс Stack был добавлен оператор [], для удобства перемещения по элементам стека.

3. Коды программ

Файлы с кодом программы:

1. PersonKeeper.h
2. PersonKeeper.cpp
3. person.h
4. person.cpp

Находятся на репозитории github.com/timuraknazarov/trpo_lab_1

4. Инструкция пользователя

При запуске программы появляется меню, предлагающее выбрать номер теста:

```
Test 1. No file to read
Test 2. File is empty.
Test 3. Data in file is correct
Test 4. In file there is more data than 3
Test 5. Some data in file with comma
Test 6. Some characters in the file data are incorrect
input test number:
```

Для выбора необходимо ввести номер теста и нажать клавишу enter. Далее программа в зависимости от номера тестирования будет работать с конкретными файлами формата .txt. Эти файлы созданы специально для проверки всевозможных сценариев работы программы. Сценарии не ограничиваются набором файлов с корректными данными, большая часть

файлов создана чтобы проверить, как поведет себя приложение в случаях, когда данные некорректны или отсутствуют.

5. Тестирование

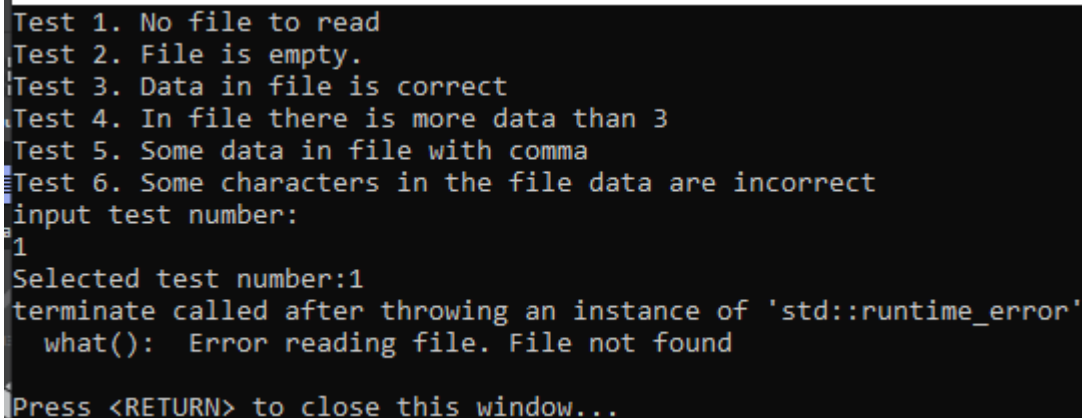
Все тестирование реализовано в файле `main.cpp`, оно заключается в том, что в зависимости от выбранного номера теста используются разные файлы, необходимые для проверки поведения программы в разных ситуациях.

Код для всех случаев выглядит аналогично, за исключением имени открываемых файлов:

```
std::fstream File; // Открываем файл
File.open("filename.txt", std::ios_base::in); // Открываем его на чтение
Stack<Person> Persons = PersonKeeper::instance().readPersons(File); // Считываем
std::fstream File1; // Открываем файл1
File1.open("output.txt", std::ios_base::out); // Открываем его на запись
PersonKeeper::instance().writePersons(Persons, File1); // Записываем
break;
```

Давайте посмотрим, как приложение ведет себя во всех случаях:

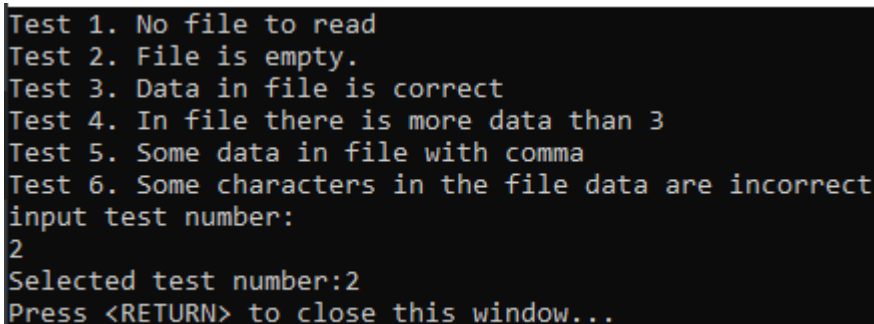
- **Тест 1: Не можем открыть файл на чтение т.к. его нет**



```
Test 1. No file to read
Test 2. File is empty.
Test 3. Data in file is correct
Test 4. In file there is more data than 3
Test 5. Some data in file with comma
Test 6. Some characters in the file data are incorrect
input test number:
1
Selected test number:1
terminate called after throwing an instance of 'std::runtime_error'
  what():  Error reading file. File not found
Press <RETURN> to close this window...
```

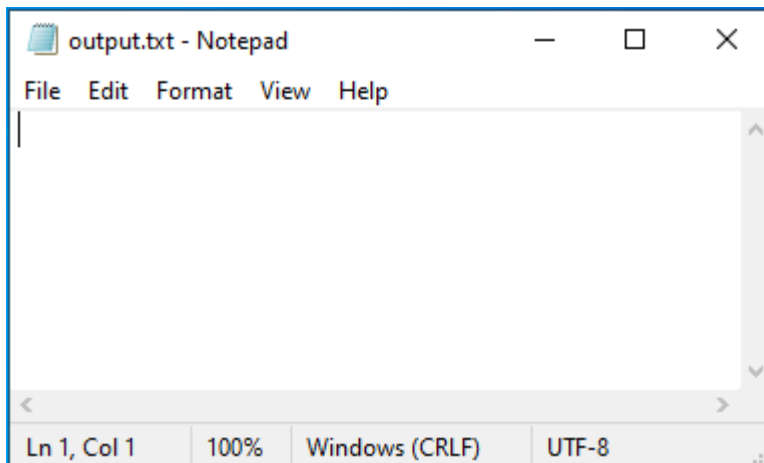
Так как файла нет, появляется сообщение о соответствующей ошибке.

- **Тест 2: Файл пустой**



```
Test 1. No file to read
Test 2. File is empty.
Test 3. Data in file is correct
Test 4. In file there is more data than 3
Test 5. Some data in file with comma
Test 6. Some characters in the file data are incorrect
input test number:
2
Selected test number:2
Press <RETURN> to close this window...
```

Так как файл существует, но пустой, на выходе получаем пустой файл output.txt.

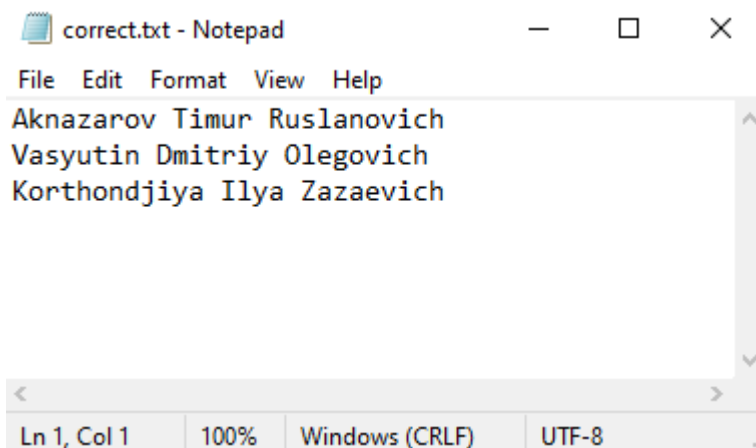


Программа работает корректно.

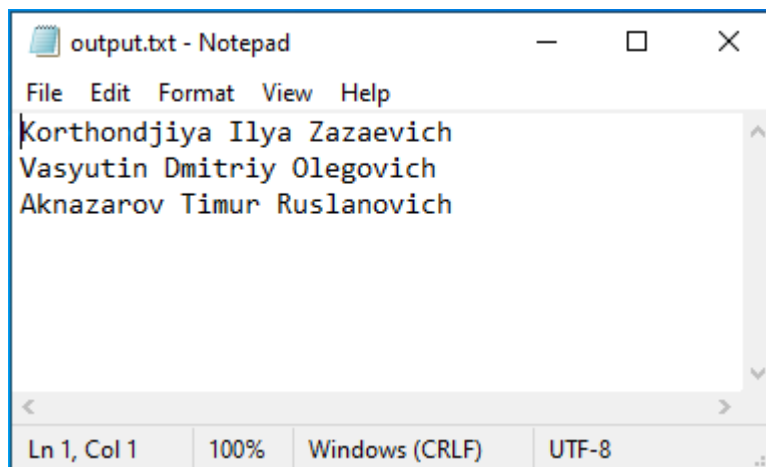
- **Тест 3: Данные в файле корректны.**

```
Test 1. No file to read
Test 2. File is empty.
Test 3. Data in file is correct
Test 4. In file there is more data than 3
Test 5. Some data in file with comma
Test 6. Some characters in the file data are incorrect
input test number:
3
Selected test number:3
Press <RETURN> to close this window...
```

Входной файл:



Так как данные в файле корректны, на выходе получаем файл output.txt с корректно переписанными ФИО:

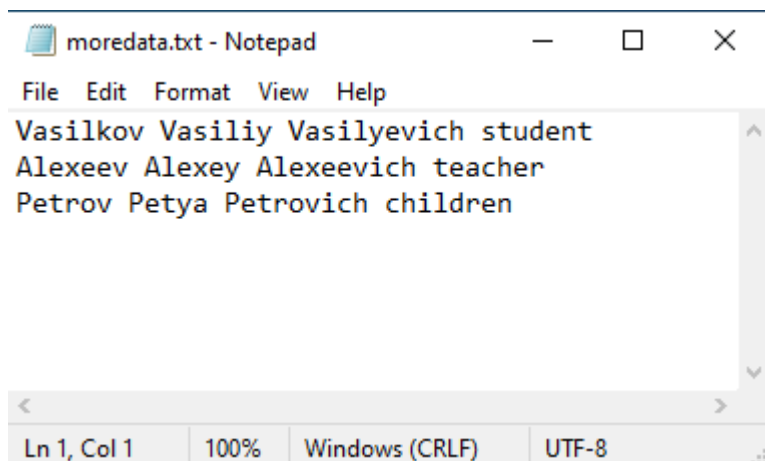


Программа работает корректно.

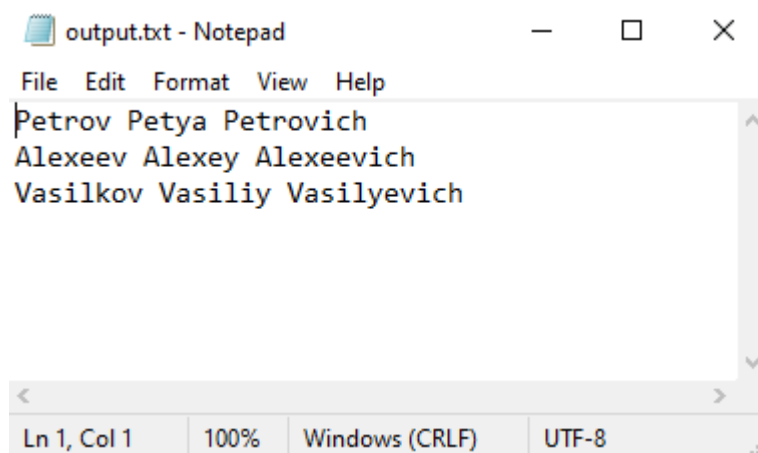
- **Тест 4: В файле больше данных, чем 3**

```
Test 1. No file to read
Test 2. File is empty.
Test 3. Data in file is correct
Test 4. In file there is more data than 3
Test 5. Some data in file with comma
Test 6. Some characters in the file data are incorrect
input test number:
4
Selected test number:4
Press <RETURN> to close this window...
```

Входной файл:



Во входном файле находится больше данных, чем нужно (ФИО состоит из 3 слов), на выходе получаем файл output.txt с корректно записанными фео, лишние данные отбрасываются.



output.txt - Notepad

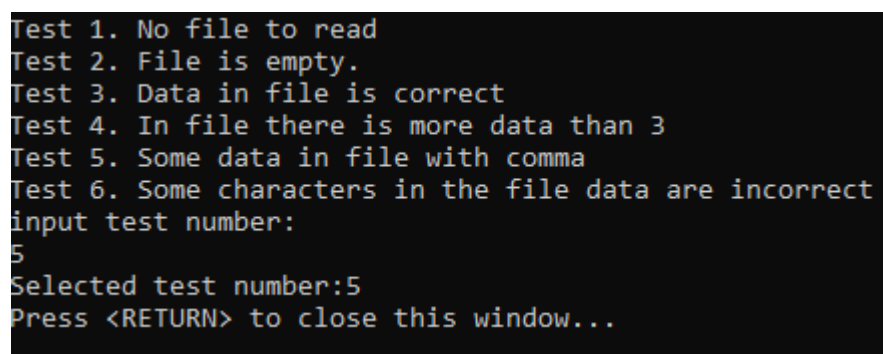
File Edit Format View Help

```
Petrov Petya Petrovich  
Alexeev Alexey Alexeevich  
Vasilkov Vasiliy Vasilyevich
```

Ln 1, Col 1 100% Windows (CRLF) UTF-8

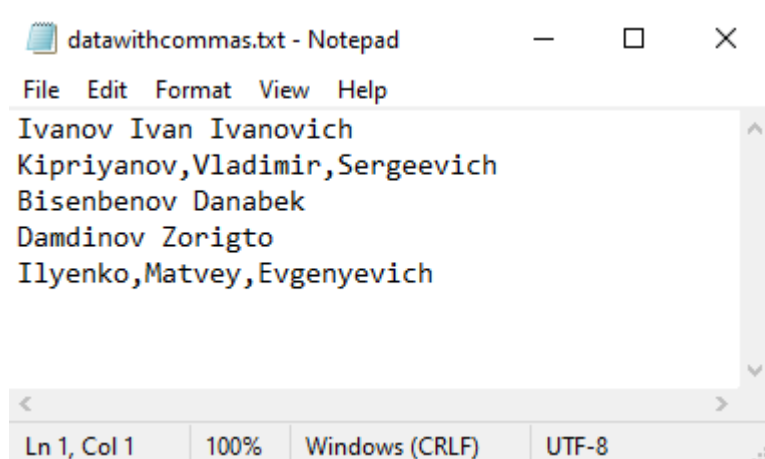
Программа работает корректно.

- **Тест 5: Некоторые данные файлы записаны через запятую**



```
Test 1. No file to read  
Test 2. File is empty.  
Test 3. Data in file is correct  
Test 4. In file there is more data than 3  
Test 5. Some data in file with comma  
Test 6. Some characters in the file data are incorrect  
input test number:  
5  
Selected test number:5  
Press <RETURN> to close this window...
```

Входной файл:



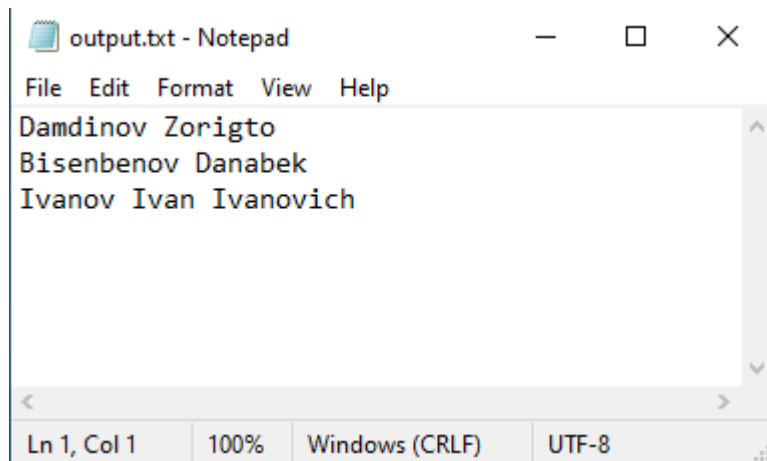
datawithcommas.txt - Notepad

File Edit Format View Help

```
Ivanov Ivan Ivanovich  
Kipriyanov,Vladimir,Sergeevich  
Bisenbenov Danabek  
Damdinov Zorigto  
Ilyenko,Matvey,Evgenyevich
```

Ln 1, Col 1 100% Windows (CRLF) UTF-8

Так как во втором файле некоторые данные записаны через запятую, на выходе получаем файл output.txt с корректно записанными ФИО, данные указанные через запятую отбрасываются.

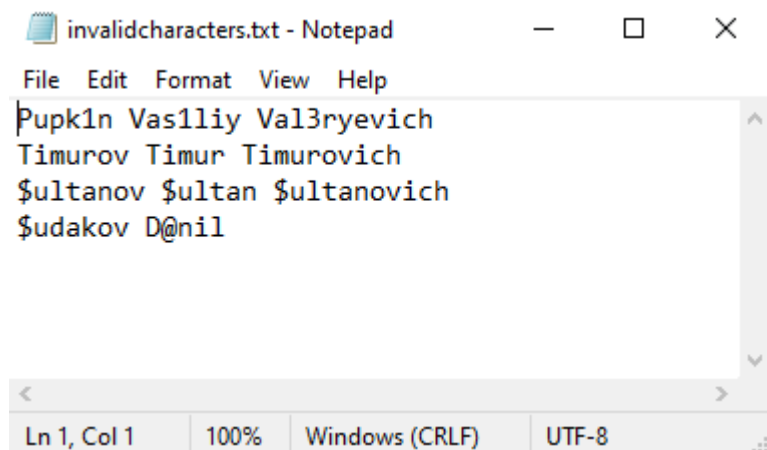


Программа работает корректно.

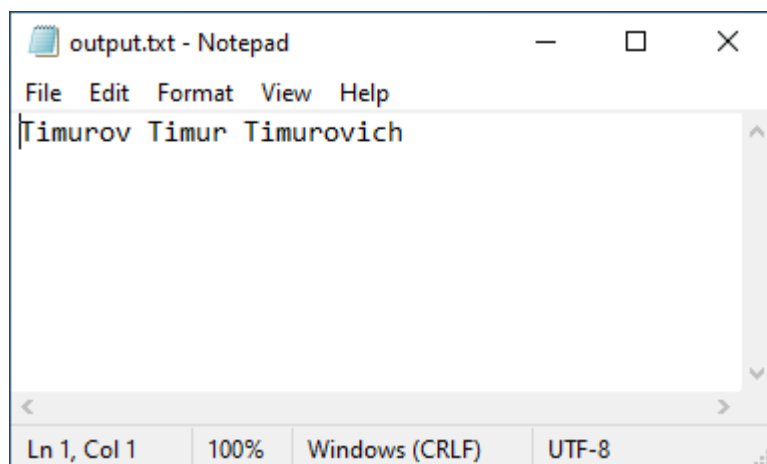
- **Тест 6:**

```
Test 1. No file to read
Test 2. File is empty.
Test 3. Data in file is correct
Test 4. In file there is more data than 3
Test 5. Some data in file with comma
Test 6. Some characters in the file data are incorrect
input test number:
6
Selected test number:6
Press <RETURN> to close this window...
```

Входной файл:



Так как во входном файле некоторые данные записаны с некорректными символами, на выходе получаем файл output.txt с корректно записанными ФИО, данные с некорректными символами отбрасываются.



Программа работает корректно.