**1. Постановка задачи Описание постановки задачи.**

1.1

Запрограммируйте шаблонный класс, реализующий стек. Класс должен поддерживать следующие операции: помещение объекта в стек, извлечение объекта из стека, получение размерности стека. В случае попытки вызова операции извлечение объекта из стека при условии, что стек пуст, должно генерироваться исключение класса EStackEmpty. Данный класс должен содержать публичный метод char\* what(), возвращающий диагностическое сообщение.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1.2

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. 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)**

2.1

Как было сказано в задании, данный класс нужно было реализовать в виде шаблона.

Шаблонный класс, позволяет задать тип для объектов, используемых в классе.

Для реализации стека я использовал однонаправленный список. Узел однонаправленного списка состоит двух полей, ключ и указатель на предыдущий узел.

Обработка исключений, реализован в двух классах. Обработка исключений являются локальные стеки функции, локальный стек - это область памяти, в которой функция хранит аргументы, также хранится адрес возврата.

Метод помещения объекта в стек **void Push(const T &k)**; на входе значение, которое будем добавлять в стек, в начале метода создается новое звено, далее идет проверка создался ли новый узел, если указатель пустой, то вызываем исключение. Новое звено создаётся со значением и для которого предыдущий узел, является последний элемент стека и после создания данного узла, делаем его последний. И для подсчета размера стека увеличиваем счетчик (**size**) на +1.

Метод извлечение объекта из стека **Pop();** возвращаемым значением является константное значение, чтобы мы его не смогли изменять (например st.Pop()=10) и так как метод должен удалить узел, поэтому возвращаем не ссылкой. Если стек оказался пустым, вызываем исключение. Также для подсчета размера стека, уменьшаем счетчик size на -1.

Метод перебора значений **Enum(std::function<void(const T &k)> f)** я использовал для вывода стека на консоль, аргументом для данной функции является аргумент функции обратной связи. Внутри метода идет проход от последнего до первого элемента, для которых вызывается функция обратной связи.

Метод удаления из стека всех узлов, используется для деструктора, для начала идет сохранения последнего элемента, потом идет сдвиг и удаление, данный метод похож на метод извлечения объекта из стека.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2.2

Как было сказано в задание, для решения данной задаче нужно использовать **Singlеton**,

он заключается в том, что класс состоит только в одном экземпляре и его методы

такие как конструктора по умолчанию, конструктора копирования и оператора присвоения находятся в приватной части класса.

Для получения доступа к экземпляру класса, используется метод **Instance**, который объявлен статической переменной с единственным экземпляром.

Диаграмма **Singleton**



Для данного класса, нужно реализовать класс **Person**, который будет хранить полное имя(ФИО), для получение и установки имени используются специальные методы геттеры и сеттер, также в классе, используется конструктор, который на входе получает полное ФИО, то есть строкой, чтобы разделить данную строку на фамилию, имя, отчество, я использую специальную функцию **split(QRegularExpression("\\s+"), Qt::SkipEmptyParts)**, первое значение - это использование пробелов и табуляции, второе значение, чтобы функция не возвращала пустую строку.

Также проверяем корректно разбилась ли данная строка, иначе выбрасываем исключение, также допускается использование только фамилии и имени.

Данные личности будут хранится в стеке.

Для класса **PersonKeeper**, используется два основных метода - это чтение файла и запись в файл.

Метод чтения файла, на вход метода задается путь, где находится файл, делаем проверку открылся ли файл, в противном случае выбрасываем исключение, для считывания данных из файла использую функцию **readLineInto**, которая будет считывать имена построчно, это упрощает решение задачи, так как в классе **Person**, я использую конструктор, который на входе идет полное ФИО.

Метод записи в файл, также как и в методе чтения файла, задается путь, проверяем открылся ли файл на запись, в противном случае также выбрасываем исключение, так как имя хранится в стеке, я просто перебираю значения из стека, методом **Enum** и записываю их в файл, через пробел, как было сказано ранее данный метод вызывается для каждого значения.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**3. Коды программ. полностью прокомментированы, с включением объяснений решения (фактически это ваш устный ответ).**

3.1

Для реализации данного задания были использованы следующие файлы с комментариями:

Empty.h

Exceptions.h

Stack.h

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_3.2

Для решения данной задачи используются следующие файлы с комментариями:

person\_keeper.h

Person.h

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**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.).**

4.1

Объявляем объект класса:

Stack<float> st;

Далее, что можно сделать это добавить элементы в стек

st.Push(1.2398);

st.Push(2.38);

st.Push(9.048);

Также можно удалить последний элемент

St.Pop();

Для просмотра вывода всех элементов на консоле, мы используем метод Enum

st.Enum([&](const float &k)

{

cout<<k<<" ";

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4.2

Первое, что может сделать пользователь - это считать имена из файла в стек, использую функцию **ReadPersons,** при этом задать путь до файла.

Следующее, что он может сделать - это записать данные из стека в файл, методом **WritePersons**, при этом также указать путь до файла.

Также он может посмотреть количество личностей методом **Size()**;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**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.**

5.1

Для начала я пробовал вызвать исключение, для этого я создал объект класса и попытался извлечь элемент из стека, таким образом программа вызвала исключение.

Stack<float> st;

St.Pop();

Дальше я проверил правильную работу стека, то есть для начала я добавил несколько элементов в стек и удалил последний, все сработало правильно. Исключение не вызвало, элементы добавились, последний удалился.

Stack<float> st;

st.Push(1.2398);

st.Push(2.38);

st.Push(9.048);

St.Pop();

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5.2

Для начала я попробовал создать правильный файл, то есть указав в нем полное ФИО, программа сработала правильно файл создался аналогичный.

Ardakov Igor Gerasimovich

Donchenko Ivan Nikolaevich

Biryukov Evgeny Evgenievich

Потом я удалил у некоторых личностей Отчество, программа также сработала без ошибок.

Ardakov Igor Gerasimovich

Donchenko Ivan

Biryukov Evgeny

Следующее, что я решил проверить, это вызвать некорректную работу программы, для этого я оставил только имена в файле, программа вызвала исключение.

Ardakov

Donchenko

Biryukov

Также я попробовал к ФИО добавить еще какое то имя, например Ardakov Igor Gerasimovich Aleks программа вызвало исключение.