**Рынк Даниил Витальевич группа 931921**

**Часть 1.**

1. Постановка задачи

Необходимо запрограммировать шаблонный класс, реализующий 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 , пример приведен в документе ForTask\_1 ). Данный класс должен содержать публичный метод char\* what(), возвращающий диагностическое сообщение.

1. Предполагаемое решение

Стек это структура хранения данных, представляющий собой упорядоченный (по времени помещения в стек) набор элементов .

Связаны элементы так: каждый элемент указывает на тот, который нужно использовать следующим. Это линейная связь — данные идут друг за другом и нужно брать их по очереди. Из середины стека брать нельзя.

Главный принцип работы стека — данные, которые попали в стек недавно, используются первыми. Чем раньше попал — тем позже используется. После использования элемент стека исчезает, и верхним становится следующий элемент.

В качестве внутренний реализации стека я выбрал односвязный список состоящий из узлов (Node). Также в нем у меня присутствует 2 поля: Значение и указатель на следующий узел.

Класс Stack хранит список как указатель на верхний элемент списка(который является верхним элементом стека) и размер самого стека.

Рассмотрим реализации функций Stack:

1. Два конструктора. Один по умолчанию. Второй конструктор копирования
2. Функция помещения объекта в Стек – push.
3. Функция удаления верхнего объекта из Стека – pop. \*
4. Функция извлечения верхнего объекта из Стека без удаления – topFunc. \*
5. Функция для проверки на пустоту Стека – empty.
6. Функция получения размерности Стека – getSize.

\* - на этих функциях есть проверка, если стек будет пустой, то тогда вызовется исключение.

1. Коды программ

Все файлы находятся в репозитории: <https://github.com/rynkdesign/TRPO_lab1>

А именно:

Node.h  
Stack.h

Exception.cpp

1. Инструкция пользователя

Данный класс представляет из себя шаблон, значит для его использования нам необходимо указать данные которые он хранит.   
  
Пример создания экземпляра класса Stack:

type - любой тип данных

Stack<type> test(value);

конструктор с одним аргументом. Значение value сразу помещается в контейнер.

Следующее что мы можем сделать с нашим стеком это добавить туда элемент.

Пример добавления элемента в стек:

test.push(value);

Также мы можем получить элемент из стека:

test.topFunc();

Можем узнать количество элементов стека:

test.getSize();

Можем также удалить верхний элемент:

test.pop();

Или проверить его на пустоту:

test.empty();   
  
Таким образом мы описали весь функционал этого класса.

1. Тестирование

Для тестирования в функции main реализован простой выбор типа теста.

Виды тестирований:

// Тестирование 1 - получим размер Стека (Функция не зависит от того пустой стек или нет)

std::cout<<"Test 1"<< std::endl;

std::cout<<"Razmer: "<<massiv.getSize()<< std::endl;

std::cout<<"-----------------------------------"<< std::endl;

break;

// Тестирование 2 - вызовем функцию Top в пустом стеке

std::cout<<"Test 2"<< std::endl;

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

std::cout<<"-----------------------------------"<< std::endl;

break;

// Тестирование 3 - вызовем функцию pop в пустом стеке

std::cout<<"Test 3"<< std::endl;

massiv.pop();

std::cout<<"-----------------------------------"<< std::endl;

break;

// Тестирование 4 - Добавим элемент в Стек и покажем, что он добавился и размер увеличился

std::cout<<"Test 4"<< std::endl;

massiv.push(3);

std::cout<<"Razmer: "<<massiv.getSize()<< std::endl;

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

std::cout<<"-----------------------------------"<< std::endl;

break;

// Тестирование 5 - Добавим ещё один элемент в Стек и покажем, что функция Top работает корректно

std::cout<<"Test 5"<< std::endl;

massiv.push(3);

massiv.push(5);

std::cout<<"Razmer: "<<massiv.getSize()<< std::endl;

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

std::cout<<"-----------------------------------"<< std::endl;

break;

На этом все возможные варианты отработки класса Стек закончены.

**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.

1. Предполагаемое решение

В качестве решения я реализовал класс **PersonKeeper** , который отвечает за чтение либо запись Стека по информации о людях из входного файла, на основе этих данных он создаёт объекты класс **Person,** в которых хранятся ФИО людей и помещает их в стек.

Класс **PersonKeeper** я реализовал используя шаблон Singleton (Одиночка). **Singleton –** это порождающий паттерн, который гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа. То есть Одиночка скрывает все способы создания нового объекта, кроме **специального метода - Instance.** Она возвращает ссылку на этот единственный статический объект. Т.к. объект у нас единственный и он же создаётся в Instance, то привычным методом через Объект его вызвать не получится, поэтому Instance – статический метод и мы можем его реализовывать вне объекта.

В нашем случае этот паттерн реализован для того чтобы у нас был только единственный экземпляр класса PersonKeeper. Он создаётся только 1 раз в методе Instance.  
  
Класс **PersonKeeper** имеет следующие методы:

**Instance** – статический метод, возвращает ссылку на единственный экземпляр PersonKeeper;

**readPersons** – метод, получающий на вход ссылку на входной поток. Он считывает из потока информацию, и на ее основе создает и возвращает стек Person.

**writePersons** – метод, принимающий стек и выходной поток. В методе считывается информация из стека(стек не изменяется), которая записывается в выходной поток.

А также конструктор по умолчанию и конструктор копирования.

Также для удобства перемещения пришлось добавить в класс Stack – operator[], с помощью которого мы можем передвигаться по элементам Стека для чтения из файла.

1. Коды программ  
   Все файлы находятся в репозитории: https://github.com/rynkdesign/TRPO\_lab1

А именно:

person.h  
person.cpp

PersonKeeper.h  
PersonKeeper.cpp  
файлы для тестирования: main.cpp и текстовые файлы: input.txt и output.txt

1. Инструкция пользователя  
   Для работы данной программы на вход мы подаём файл в форме “.txt” в котором у нас содержатся ФИО и получаем на выходе такой же файл формата “.txt” в котором эти ФИО выписываются пройдя через Стек.

Создаем входной поток, который считывает наш файл на чтение (Допустим у нас там хранятся нужные нам ФИО):  
std::fstream File ("input.txt", std:: ios\_base::in);   
  
Главной идей заключается создание Стека типа Person – который будет хранить ФИО и считать благодаря классу PersonKeeper необходимую нам информацию из нашего Файла:  
Stack<Person> Persons= PersonKeeper::instance().readPersons(File);  
  
Теперь мы можем создать выходной поток, то есть файл, в который мы будем выводить:

std::fstream File1 ("output.txt", std:: ios\_base::out);  
  
И наконец записать все наши ФИО в выходной файл:

PersonKeeper::instance().writePersons(Persons, File1);

Теперь все наши ФИО из файла “input.txt” прошли через Стек и поместились в файл “output.txt”.

1. Тестирование  
   Тестирование реализованно в файле main.cpp  
   Оно представляет собой попытки передать разные входные данные и посмотреть, как на это отреагирует наш класс.  
     
   В файле input.txt у нас находятся ФИО и символы. Мы сначала считываем данные из файла, записываем их в Стек, а потом записываем их из Стека в файл output.txt.  
   И по ходу этого дела у нас возникают разные ситуации.  
     
   Виды тестов:  
   // Тестирование 1. Мы не можем открыть файл на чтение (Его нет)

std::fstream File; // Открываем файл

File.open("0input.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;

// Тестирование 2. Всё впорядке, мы вводим данные разных форматов.

std::fstream File; // Открываем файл

File.open("input.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;

Наши ФИО записанные в разных форматах, не полностью и даже через табы, всё равно корректно переносятся в выходной файл.