***Шаблонный класс стек.***

1. Постановка задачи.

Реализация шаблонного класса, реализующего стек и поддерживающего операции помещения объекта в стек, извлечения объекта из стека и получение размерности стека.

В случае попытки вызова операции извлечения 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.

1. Предлагаемое решение.

Стек описывается как шаблонный класс на основе односвязного списка. Узел данного списка описывается структурой, состоящей из значения и указателя на предыдущее звено стека. Структуру определяем в приватном доступе вместе с указателем на последний элемент стека и целочисленной переменной, позже используемой для получения размера стека.

Реализуем метод Push, кладущий элемент в стек. Данный метод с типом void, то есть не возвращающий ничего после исполнения; на вход принимает значение по ссылке. Сперва создается новое звено списка, затем проверяется создался ли объект. Если после создания указатель на звено остается пустым, то выбрасываться исключение. Далее новое звено делается последним, и увеличивается размер стека на единицу.

Следом реализуем метод Pop, позволяющий извлекать последний объект из стека. Возвращает константное значение, для невозможности это значение, зачем-то, изменить. Сперва проверяется пустота стека, т.е. не указывает ли последний элемент стека на ничто, если это так, то выбрасывается исключение. Далее последнее значение сохраняется в константную переменную, а конец стека переносится на предпоследний элемент, счётчик элементов в стеке уменьшается. Наконец, возвращается сохранённое значение.

Реализуем функцию Clear, очищающую стек. Суть заключается в том, что функция проходит по всем элементам стека, каждый из них помещается во временную переменную, которая после сдвига последнего элемента на предыдущий, удаляется.

1. Коды программы.

К данной реализации относятся файлы:

EStackEmpty.h

EStackException.h

Stack.h

1. Инструкция пользователя.

Так как стек шаблонный, у пользователя есть выбор значения элементов стека. Предположим, что пользователь выбрал простое - целочисленное значение.

Объявление стека будет выглядеть так:

Stack<int> Stack;

Далее, пользователь захочет стек чем-нибудь заполнить:

Stack.Push(2);

Stack.Push(3);

Stack.Push(4);

Stack.Push(5);

Пользователь может убедиться, что ему это удалось, проверив размер стека:

std::cout << Stack.Size() << std::endl;

Также, он может извлечь последний элемент стека:

int lastElement = Stack.Pop();

std::cout << lastElement << std::endl;

1. Тестирование.

Протестировать данную программу можно с помощью попытки извлечения значения с ещё не заполненного стека:

Объявим стек и вызовем метод Pop:

Stack<int> Stack;

Stack.Pop();

Программа выбросит исключение о пустом стеке, что и предполагалось при формулировке задания.

***Класс PersonKeeper.***

1. Постановка задачи.

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. Предлагаемое решение.   
   Сперва, нужно прояснить, что есть Singleton. Singleton (рус. Одиночка) – паттерн программирования, гарантирующий, что у класса есть только один экземпляр, и что он предоставляет к нему глобальную точку доступа. Реализуется это помещением конструкторов в приватную область, чтобы у пользователя не было такой возможности создать экземпляр объекта. Вместо этого, класс Одиночка предоставляет метод GetInstance, который ведет себя как альтернативный конструктор и позволяет пользователю получать один и тот же экземпляр класса при каждом вызове.

Класс Person хранит член-данные: Фамилию Имя Отчество, и соответствующие им get- и set-методы.   
  
Помимо этого, класс предоставляет ещё два метода: ReadPersons и WritePerson.   
  
ReadPerson получает на вход полный путь до текстового файла, в котором хранятся член-данные, на выходе передаёт стек с типом данных Person. Для открытия файла и чтения из него используется библиотека fstream. Сперва проверяется корректное открытие файла, в случае ошибки, выбрасывается сообщение о ней. Затем, пока не конец файла, член-данные помещаются в стек. Для этого используется потоковое чтение из файла. Наконец, после закрытия файла, результатом метода возвращается стек.  
  
WritePerson получает на вход стек с типом данных Person. Первым делом, с помощью ofstream, в переменную file передаётся путь до файла. Затем, создаётся копия исходного стека, так как по условию задачи исходный стек изменяется не должен. Далее, открывается (или создаётся и затем открывается) файл на запись, путь до которого был закреплен ранее, и в него записываются данные со стека с помощью метода getFullName.

1. Коды программы.  
   К данной реализации относятся файлы:  
   Person.h  
   PersonKeeper.h
2. Инструкция пользователя.

Для получения данных класса Person используются get-методы член-данных (геттеры), для записи данных – set-методы (сеттеры).   
Для обращения к Синглтону используется статистический метод getInstance().

Если пользователь решил записать в стек данные из файла, ему необходимо вызвать метод ReadPerson и указать полный путь до файла, в котором они хранятся.   
  
Если пользователь решил записать в файл данные из стека, ему необходимо вызвать метод WritePerson и указать стек. Данные запишутся в файл writeFile.txt.

1. Тестирование.   
   Протестируем получение данных из файла в стек.

Файл file.txt:

*Pelevin Victor Olegovich*

*Alehin Evgeni Igorevich*

*Veniamonovich Eduard Limonov*

Проверим корректность данной операции помещением в переменную типа Person последних данных из исходного файла, используя для этого метод Pop().   
  
Протестируем помещение данных в файл. Так как, метод Pop, извлекает последний элемент стека, удаляя его оттуда, содержимое файла writeFile.txt будет таким:

*Pelevin Victor Olegovich*

*Alehin Evgeni Igorevich,*

что и предполагалось при формулировке задания.