**Шаблонный класс стек**

**1. Постановка задачи**

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.

**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)

Теоретическая часть:

Шаблонный класс — это класс, который использует определенные типы. Шаблонный класс позволяет описывать конструкции, которые не зависят от определенного типа. Шаблонный класс нам позволит задавать тип для объектов.

Стек — это структура данных, представляющая определенный набор элементов, в которой добавление новых элементов и удаление существующих производится с одного конца, называемого вершиной стека. Это область памяти, в которой функция хранит аргументы и адрес.

Для того, чтобы реализовать стек, я использовала однонаправленный список. Узел списка состоит из двух полей: ключ и указатель на предыдущее звено(узел).

Методы стека:

Метод Print — метод перебора значений. С его помощью я осуществляю вывод стека. Аргумент у этой функции — аргумент функции обратной связи.

Метод Push — метод помещения объекта в стек. Этот метод ничего не возвращает. Мы получаем значение, которое добавим в стек, создаем пустое звено. Новое звено ссылается на вершину стека, а после этого новой вершиной стека станет новый созданный узел. Делаем счетчик, то есть увеличиваем размер стека.

Метод Pop — метод извлечения объекта из стека. Этот метод нужен, чтобы удалить из стека узел и вернуть его константой. В этом методе я предусматриваю ситуацию с пустым стеком. Поэтому проверяю перед использованием, есть ли в стеке элементы, если нет, то вызываю исключение. Удаляемая вершина в этом методе сохраняется, после этого мы сдвигаем вершину стека на предыдущий элемент списка и удаляем предыдущую вершину, далее возвращаем значение удаленного элемента.

У исключений тоже есть свои классы:

class **Exception** — общий класс.

class **EStackEmpty** — наследник класса **Exception.** Из конструктора он передает в конструктор общего класса строку.

**3. Коды программ. Полностью прокомментированы, с включением объяснений решения.**

estackexception.h

estackempty.h

stack.h

main.cpp

**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.).**

Так как наш стек — шаблонный класс, то мы обязательно укажем с каким типом мы будем работать. В программе я использую три типа (int, float и string).

Возьмем для примера тип int:

Stack<int> stacktest2;

Пользователь может добавить элемент, для этого необходимо вызвать метод Push(добавлять элемент можно как созданием отдельной переменной, так и внесением нужного числа в скобки):

int num = 2;

stacktest2.Push(num);

stacktest2.Push(10);

Далее нам нужно получить данные из стека, поэтому вызываем метод Pop:

cout << stacktest2.Pop() << ' ' << stacktest2.Pop() << endl;

А если бы во время извлечения из стека элемента стек оказался пуст, то у нас вызовется исключение, которое об этом сообщит. В консоли появится сообщение: “This stack is empty”.

Также мы можем получить размер стека:

cout << "Stack size: " << stacktest2.Size() << endl;

И можем напечатать значения какого-либо типа, если они в стеке имеются:

Stacktest2.Print([&](const int &k){cout<<k<<" ";});

**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.**

Создаём пустой стек и получаем его размер:





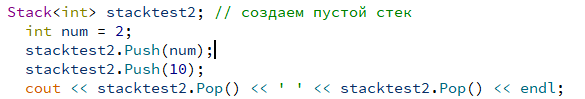
Добавляем в стек числа и проверяем и получаем его размер:

Изображение выглядит как текст

Автоматически созданное описание



Создаём пустой стек, добавляем два числа и выводим их





Также можно проверить размер стека после добавления каждого из чисел:

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

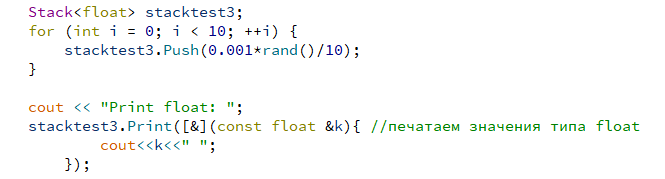
Далее пытаемся получить последнее значение и встречаем исключение:

Изображение выглядит как текст

Автоматически созданное описание



Создаём пустой стек со значениями типа float и заполняем его, потом печатаем:





Создаём пустой стек со значениями типа string и заполняем его, потом печатаем:

Изображение выглядит как текст

Автоматически созданное описание



**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*.**

**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)

Паттерн Singleton позволяет создавать класс в одном экземляре, предоставляет простой доступ к нему и его методы находятся в приватной части (чтобы не было возможности создания снаружи). Для того, чтобы обратиться к классу с конструкторами в приватной области, нужно воспользоваться статичным методом Intance, он предоставит доступ к статичной переменной(создаётся она всего один раз), при дальнейших обращениях к методу, будет возвращаться только ссылка на созданный экземпляр.

Нужно реализовать класс Person, хранящий в себе ФИО. Для этого я использовала конструктор, который получает ФИО в виде строки и разбивает строку на 3 член данных. Для этого использовалась функция split(QRegularExpression("\\s+"), Qt::SkipEmptyParts); Методы: геттеры и сеттеры для каждого член данного. Также есть проверка, корректное ли разбиение.

Для класса PersonKeeper я использовала два метода: **ReadPersons** и **WritePersons.**

Метод **ReadPersons** — метод чтения. Метод получает на вход путь до файла, в котором хранятся ФИО, а на выход метод передаёт стек типом данных Person. Для считывания данных из файла я использовала readLineInfo(функция считывает имена построчно). До тех пор, пока не считали из файла все данные, записываем в стек. Когда файл закончился, то возвращаем его.

Метод **WritePersons** — метод записи. Метод получает на вход стек и путь до файла, на выход ничего не передаётся. Мы не должны изменять исходный стек (по заданию) и поэтому создаём его копию и работаем только с ней. В файл WritePerson будем записывать все наши данные со стека. Записываем в файл ФИО пока стек не опустеет. Для этого я перебираю все значения методом **Print**() с записью через пробел.

**3. Коды программ. Полностью прокомментированы, с включением объяснений решения.**

Person.h

PersonKeeper.h

main.cpp

**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.).**

Для обращения к классу Singleton, используется статический метод Intance().



Пользователь может считать данные из файла ReadPerson.txt в стек, для этого необходимо вызвать ReadPersons и указать путь до файла с данными(ФИО):





А потом записать данные из стека в файл с помощью WritePersons и данные запишутся в файл WritePerson.txt:





**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.**

Создаём файл с ФИО и проверяем получим ли мы данные из файла в стек. В файле ReadPerson.txt будет храниться:



Изображение выглядит как текст

Автоматически созданное описание

Получаем:

Изображение выглядит как текст, человек, документ

Автоматически созданное описание

Проверяем работу с лишними пробелами:



Не повлияло на результат:

Изображение выглядит как текст

Автоматически созданное описание

Также можно работать с именами без отчества:



Работает корректно:

Изображение выглядит как текст

Автоматически созданное описание