Задорожнов 931921

1. Реализация шаблонного класс стек

1. Постановка задачи

Требуется запрограммировать 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

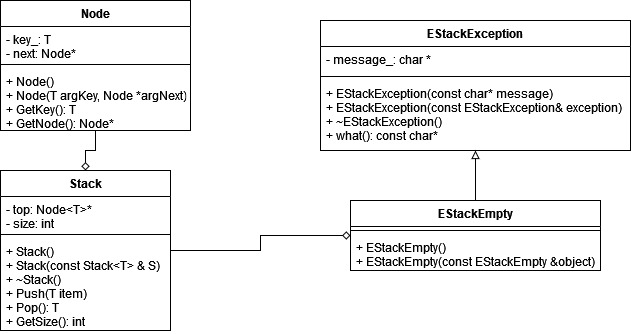
Для решения данной задачи сначала познакомимся с классом шаблоном. Класс шаблон – это такой класс, который использует определенные типы, но неизвестно какие будут эти типы; классы шаблоны позволяют описать конструкции, которые не зависят от определенного типа.

Для реализации стека я использовал однонаправленный список. У класса списка два поля: значения и указателем на предыдущее звено.

Метод Push. Метод получает значение элемента, которое будет записано в стек и ничего не возвращает. В начале создаем новый элемент списка, который ссылается на вершину стека. После этого новой вершиной стека окажется только что созданный узел и увеличиваем размер стека.

Метод Pop. Метод в аргументы ничего не получает, его назначение — это удалить из стека элемент и вернуть его. В этом методе необходимо предусмотреть ситуацию, когда в стеке нет элементов. Поэтому перед всей работой проверяю есть ли в стеке хоть какие-либо элементы если нет, то вызываем исключение. Смысл данного метода заключается в сохранении удаляемой вершины, затем мы сдвигаем вершину стека на предыдущий элемент списка и удаляем предыдущую вершину, после этих действий возвращаем значение удаленного элемента.

Исключения реализованы в виде двух классов: общий класс EStackException и его наследник EStackEmpty. В конструкторе класс наследник предает в конструктор класса EStackException вручную прописанную строку.

UML

3. Коды программ

EStackException.h

EStackEmpty.h

Stack.h

Stack.cpp

4. Инструкция пользователя

Так как класс стек является шаблоном то для него необходимо указать какие данные он будет использовать. Для примера буду использовать данные типа int. В коде это будет выглядеть так:

Stack<int> stackName;

Для того что бы записать в класс что-либо необходимо вызвать метод Push:

stackName(5);

stackName(10);

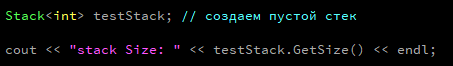
Для того что бы получить данные из стека вызывается метод Pop:

int lastNumber = stackName.Pop();

Если во время извлечения из стека стек оказался пуст, то выведется исключение, сообщающее об этом;

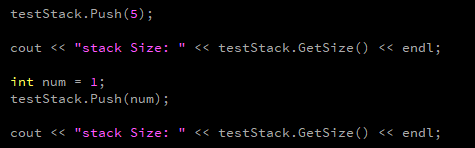
5. Тестирование

Создадим пустой стек типа int и получим его размер



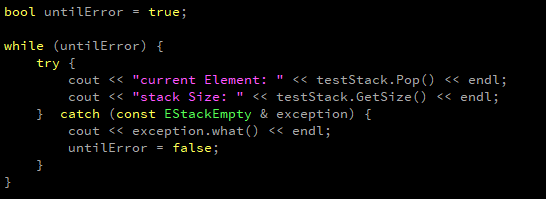


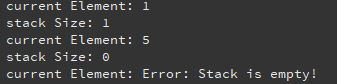
Добавим в стек два числа и проверим размер стека





Теперь в цикле будем получать все элементы пока не встретим исключение





2. Реализация класса 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.

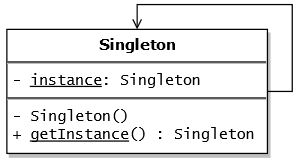
Метод *writePersons* должен записывать в поток из стека (стек передается аргументом) информацию о людях в соответствии с вышеописанным форматом. Передаваемый методу *writePersons* стек не должен изменяться.

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

2. Предлагаемое решение

Для начала проясним что является паттерном Singleton. Singleton позволяет создавать единственный экземпляр класса и обычно предоставляет простой доступ к нему. В С++ реализация, следующая: в классе Одиночке конструкторы помещаются в приватную область видимости, что бы снаружи не было возможности создать его. Для того что бы обратиться к нему создается специальный статичный метод Instance, внутри которого предоставляется доступ к статичной переменной с единственным экземпляром. Так как переменная является статичной то создастся она всего 1 раз, при следующих вызовах метода будет возвращаться только ссылка на созданный экземпляр.

UML пример:



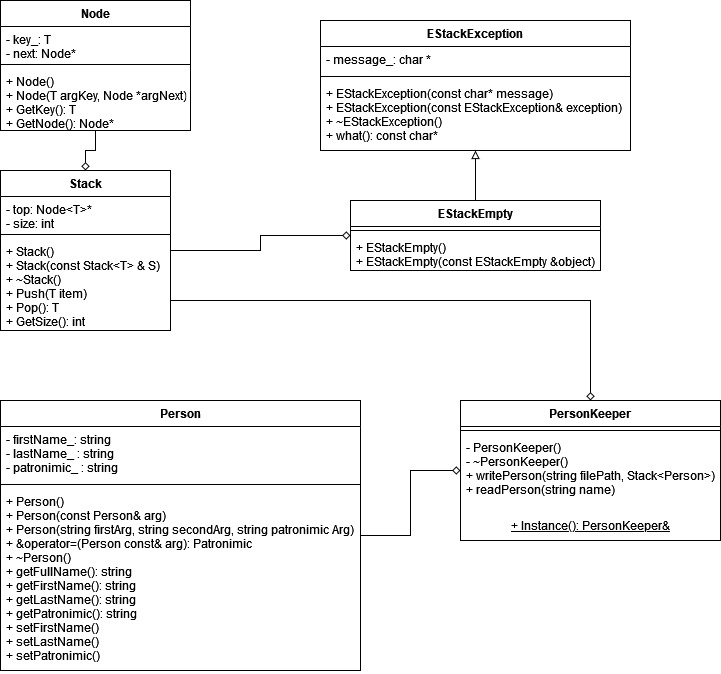
В нашем классе одиночке присутствуют два метода: ReadPersons и WrtiePersons, но перед этим ознакомимся с классом Person.

Класс Person содержит 3 член данных: ФИО. Из методов можно перечислить: геттеры и сеттеры для каждого член данного и метод для получения полного имени.

Метод ReadPersons. Метод получает на вход путь до файла, в котором хранятся данные ФИО, и на выход передает стек типом данных Person. Для открытия и чтения из файла я использовал библиотеку fstream. Сначала необходимо проверить что файл открылся, если такого файла не существует либо произошла какая-либо иная ошибка сообщаем что файл не может быть открыт. Для записи в переменные из файла используем потоковое чтение из файла. Затем до тех пор, пока не считали с файла все данные записываем в стек, и когда файл закончился возвращаем его.

Метод WritePersons. Метод на вход получает стек с типом данных Person и путь до файла, на выход ничего не передаем. По условию задания мы не должны изменять исходный стек поэтому перед началом каких-либо действия создаем его копию и уже будем работать с этой копией. В этом методе мы открываем либо создаем файл на запись, переданный на вход, и в него будем записывать наши данные со стека. Записываем в этот файл полные ФИО используя метод getFullName класса Person до тех пор, пока стек не опустеет.

UML диаграмма:



3. Коды программ

Person.h

Person.cpp

PersonKeeper.h

PersonKeeper.cpp

4. Инструкция пользователя

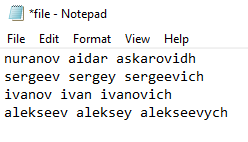
Для класса Person что бы получить из него данные необходимо вызывать методы геттеров член данных, а для записи сеттеры

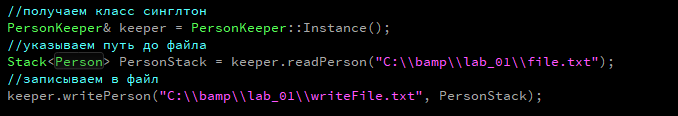
Что бы обратиться к классу Одиночке обычный вызов конструктора не получится, что бы его получить необходимо вызывать статический метод PersonKeeper::Instance().

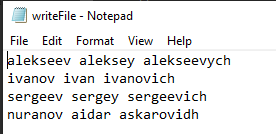
Что бы считать из файла в стек пользователю необходимо вызвать метод ReadPersons и указать путь до файла с именами. Он сохранится в стек. Чтобы записать стек с ФИО пользователю нужно вызвать метод WritePersons и передать стек, который необходимо записать, запишется в файл с именем “writeFile.txt”

5. Тестирование

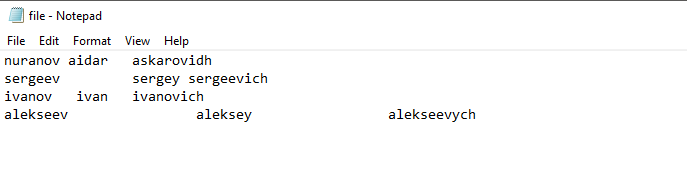
Для начала протестируем получим ли мы данные из файла в стек для примера в файле будет хранится:

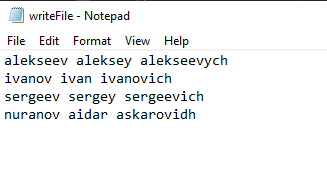






Количество пробелов не будет влиять на получение стека





Проверим исключение что файла не существует

try {

Stack<Person> PersonTest = keeper.readPerson("C:\\no\\existing\\path\\file.txt");

} catch (const char\* exc) {

cout << exc << endl;

}

Получаем ошибку что не удалось открыть файл.

