Лабораторная работа № 8. Указатели. Динамическая память.

Указания:

В качестве решения лабораторной работы требуется сдать файл **.zip**, созданный на основе **lab08-memory_template.zip (файл прикреплен к заданию по лабораторной работе)**, состоящий из следующих файлов:

1. containers.h -- *Решение задач № 1-2*
2. containers_tests.cpp
3. shared_ptr.h -- *Решение задачи № 3*
4. shared_ptr_tests.cpp
5. main.cpp -- *Решение задачи № 4*

В архиве-шаблоне в файлах *container_tests.cpp* и *shared_ptr_tests.cpp* уже содержится некоторый набор тестов, возможно, поясняющий некоторые моменты из текстового условия (ведь сэмплы - также часть условия). Рекомендуется добавлять и свои тесты в эти файлы для самостоятельного локального тестирования решений.

Важно!!! Не изменяйте *enum class Variant* из первого файла. Таким образом, там оставляете раскомментированными оба варианта, а меняете лишь строчку в *GetVariant()* и две строчки с *using*.

- * Проверьте качество вашего кода с помощью cpplint. Убедитесь, что вы следуете требованию неконстантные ссылки запрещены.
- При объявлении классов, необходимо сопроводить комментариями как сами классы, так и объявление всех методов.
- Старайтесь избавиться от дублирования кода при помощи создания приватных вспомогательных методов, в которые он будет вынесен.
- Рекомендуется сначала объявить все методы, а потом начать писать их реализации (возможно, где-то выражая одни методы через другие).

Задание 1. Двусвязный список

Создать шаблонный класс двусвязного списка BiDirectionalList<T> и публичную вложенную структуру Node.

Структура Node должна содержать:

- Публичное поле value типа T со значением, хранимым в данном элементе списка.
- Приватные поля для осуществления связи элементов в списке.
- Приватный конструктор (ничто кроме BiDirectionalList не должно иметь возможности создания элементов типа Node).

Класс BiDirectionalList<T> должен иметь следующий публичный интерфейс:

- Конструктор по умолчанию.
- Конструктор от std::initializer_list<T>.

- Методы Size() и IsEmpty().

- Методы PushFront(const T& value) и PushBack(const T& value), осуществляющие вставку элемента в начало и конец списка.

- Методы Node* Front() и Node* Back(), позволяющие получить первый/последний элемент в списке.
- Методы PopFront() и PopBack(), осуществляющие удаление первого/последнего элемента из списка.

- Метод ToVector(), возвращающий представление в виде вектора.

- Метод int Find(const T& value), который возвращает позицию (индекс) первого вхождения элемента value в список. Если элемент отсутствует в списке, то вернуть -1.
- Метод std::vector<int> FindAll(const T& value), который возвращает индексы всех вхождений элемента value в список в порядке возрастания.

- Оператор обращения по индексу ([]), который позволяет обратиться к элементу списка по его индексу, возвращающий Node*.

- Методы InsertBefore(Node* element, const T& value) и InsertAfter(Node* element, const T& value), которые позволяют добавить элемент на произвольную позицию в список.
- Метод Erase(Node* element), который позволяет удалить произвольный элемент из списка.

- Проверить корректность вызовов и передаваемых параметров с помощью assertов.

- Создать константные аналоги для методов Back, Front и оператора [], которые будут возвращать const Node*.

- Обеспечить возможность проверки на равенство/различие списков при помощи соответствующих операторов.

- Реализовать конструктор копирования и копирующий оператор присваивания.
- Реализовать конструктор перемещения и перемещающий оператор присваивания.

Подумайте, какая проблема существует в описанном интерфейсе, объясните её и придумайте, как можно было бы её избежать.

Задание 2. Обёртка над двусвязным списком

Создать класс Queue<T> / Stack<T> (по вариантам), единственным полем которого будет приватное поле типа BiDirectionalList<T>**.** Класс должен предоставлять следующие возможности:

- Конструктор по умолчанию.
- Конструктор от std::initializer_list<T>.

- Методы Push(const T& value) и Pop().
- Метод const T& Get().
- Методы Size() и IsEmpty().

- Обеспечить возможность проверки на равенство/различие при помощи соответствующих операторов.

Если Ваш номер в журнале чётный, то реализуйте класс Queue<T>, если нечётный - Stack<T>.

Задание 3. SharedPtr

Создайте класс SharedPtr<T>, аналогичный по поведению стандартному std::shared_ptr<T>, и реализуйте для него следующую функциональность:

- Конструктор по умолчанию, который бы инициализировал поле нулевым указателем.
- Конструктор, принимающий указатель на объект.
- Деструктор, автоматически удаляющий хранимый по указателю объект, если на него больше не ссылается ни один указатель.

- Возможность вызова метода Get(), который бы возвращал указатель на хранимый объект.

- Возможность вызова метода Release(), который бы "отвязывал" хранимый по указателю объект от объекта SharedPtr<T>, так что даже при удалении всех ссылающихся на него SharedPtr<T>, объект по-прежнему остается валидным.
  Upd. После выполнения данной операции указатель должен становиться равным nullptr.

- Возможность вызова операторов '*' и '->', которые бы возвращали ссылку и указатель на объект соответственно.

- Возможность вызова операторов выше и метода Get() для объектов типа const SharedPtr<T>. Разумеется, возвращаемые значения должны также иметь модификатор неизменяемости.

- Обеспечить возможность проверки на равенство/различие для двух SharedPtr<T> при помощи соответствующих операторов, а также возможность аналогичного сравнения с T*.

Задача 4. Putting all together...

В файле main.cpp:

- Создайте функцию, которая считывает с клавиатуры последовательность целых чисел и создаёт из них объект типа Container<int> (задача 2). Функция не должна иметь возвращаемого значения, для передачи результата используйте передачу по указателю. При этом, Container<int> должен быть создан в динамической памяти, а возвращать следует SharedPtr на него.