МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Обьектно-ориентированное программирование»

Тема: Умные указатели

Студентка гр. 7304 _____ Нгуен Т.Т. Зуен

Преподаватель _____ Расмочаева Н.В.

Санкт-Петербург

2019

1. Цель работы:

Изучить умные указатели (пример shared_ptr) и реализовать их в языке программирования C++ .

2. Задание:

Основные теоретические положения:

std::shared_ptr – умный указатель, с разделяемым владением объектом через его указатель. Несколько указателей shared_ptr могут владеть одним и тем же объектом; объект будет уничтожен, когда последний shared_ptr, указывающий на него, будет уничтожен или сброшен. Объект уничтожается с использованием delete-expression или с использованием пользовательской функции удаления объекта, переданной в конструктор shared_ptr.

shared_ptr может не владеть ни одним объектом, в этом случае он называется пустым.

Задание:

Необходимо реализовать умный указатель разделяемого владения объектом (shared_ptr).

Для того, чтобы shared_ptr можно было использовать везде, где раньше использовались обычные указатели, он должен полностью поддерживать их семантику. Модифицируйте созданный на предыдущем шаге shared_ptr, чтобы он был пригоден для полиморфного использования. Должны быть обеспечены следующие возможности:

- копирование указателей на полиморфные объекты

```
stepik::shared_ptr<Derived> derivedPtr(new Derived);
stepik::shared_ptr<Base> basePtr = derivedPtr;
```

- сравнение shared_ptr как указателей на хранимые объекты.

Поведение реализованных функций должно быть аналогично функциям.

Требования к реализации: при выполнении этого задания вы можете определять любые вспомогательные функции. Вводить или выводить что-либо не нужно. Реализовывать функцию main не нужно. Не используйте функции из cstdlib (malloc, calloc, realloc и free).

3. Ход работы:

Класс shared_ptr:

Переменые: T* m_ptr; (T – тип указателя)

long* m_counter;

В классе реализованы следующийе методы:

- Конструктур класса от указателя;
- Деструктор;
- Конструктур копирования;
- Оператор копирования;
- Оператор сравнения;
- Функция bool() для проверки хранения элементов;
- Функция get() для возвращения хранимый указетель на объект;
- Функция use_count() для возврощения количество объектов;
- Оператор * для возврощения ссылки на управляемый объект;
- Оператор -> для возврощения указатели на управляемый объект;
- Функция swap(shared_ptr&) для обмена между двумями уквзателями;
- Функция reset() для замещения указатели на другой.

4. Выводы:

В ходе данной лабораторной работе была изучена реализацию контейнеры vector и list. Поведение реализованных функций должно быть таким же, как у класса std::vector и std::list из стандартной библиотеки C++ и полученные результаты.