**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине «Объектно-ориентированное программирование»**

**Тема: «Умные указатели»**

Студент гр. 7303 Швец А. А.

Преподаватель Размочаева Н.А.

Санкт-Петербург

2019

**Цель работы.**

Ознакомиться с идиомой косвенного обращения к памяти, основной целью которой является инкапсуляция работы с динамической памятью таким образом, чтобы свойства и поведение умных указателей имитировали свойства

* поведение обычных указателей. При этом на них возлагается обязанность своевременного и аккуратного высвобождения выделенных ресурсов, что упрощает разработку кода и процесс отладки, исключая утечки памяти и возникновения висячих ссылок.

**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:

1. Копирование указателей на полиморфные объекты;
2. Сравнение shared\_ptr как указателей на хранимые объекты. Поведение реализованных функций должно быть аналогично функциям

std::shared\_ptr

**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).

**Ход работы.**

1. shared\_ptr(const shared\_ptr & other) – создает shared\_ptr которая разделяет право собственности на объект, управляемый other (Шаблонный перегрузки не участвует в разрешении перегрузки, если Y\* не неявно преобразуются в T\*);
2. explicit shared\_ptr(T \*ptr = 0) – конструктор (берёт неуправляемый указатель ptr под автоматическое управление.);
3. ~shared\_ptr() – деструктор;
4. shared\_ptr& operator=(const shared\_ptr & other) – оператор присваивания;
5. explicit operator bool() const – проверяет управляет ли \*this объектом;
6. T\* get() const –возвращает указатель на управляемый объект;
7. Void swap(shared\_ptr& x) noexcept – обмен содержимым \*this и x;
8. void reset(T \*ptr = 0) – заменяет управляемого объекта с объектом, на который указывает ptr;
9. bool operator==(const shared\_ptr<Y> & other) const – сравнивает два объекта shared\_ptr<T> (шаблонный перегрузки не участвует в разрешении перегрузки, если Y\* не неявно преобразуются в T\*).

**Выводы.**

* ходе выполнения лабораторной работы был реализован класс, аналогичный классу std::shared\_ptr и стандартной библиотеки. Данный умный указатель с разделяемым владением позволяет не заботиться об освобождении памяти для объекта, доступ к которому прекращён, поскольку это происходит автоматически.