

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ**  
**по практической работе №4**  
**по дисциплине «Объектно-ориентированное программирование»**  
**Тема: Умные указатели**

Студент гр. 7382

\_\_\_\_\_

Токарев А.П.

Преподаватель

\_\_\_\_\_

Жангиров Т.Р.

Санкт-Петербург

2019

### **Цель работы.**

Необходимо реализовать умный указатель разделяемого владения объектом (`shared_ptr`).

### **Постановка задачи.**

Необходимо реализовать умный указатель разделяемого владения объектом (`shared_ptr`). 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 `std::shared_ptr`.

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.**

`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. Реализуемый класс имеет два поля: указатель на объект и указатель на счётчик указателей на этот объект.

Были реализованы две вспомогательные функции: `inc_counter` для инкрементирования счётчика умных указателей и `deg_counter` для декрементирования счётчика и удаления объекта, если счётчик достигает нуля. Конструктор, принимающий C-указатель на объект, для которого инициализируется новый счётчик, или ссылку на другой `shared_ptr`, копирую

его поля и увеличиваю счётчик на единицу. Деструктор вызывает функцию `deg_counter`.

Также были реализованы функции `get` (возвращающая указатель на объект), `use_count` (возвращающая значение счётчика), `swap` (обменивающая поля двух умных указателей), `reset` (заменяющая объект, которым владеет указатель) и перегружены операторы `=`, `==`, `!=`, `<`, `>`, `<=`, `>=`, `*`, `->` и `bool` аналогично обычным указателям.

Реализация класса представлена в приложении А.

### **Выводы.**

В ходе выполнения данной работы был реализован класс `shared_ptr`, аналогичный классу `std::shared_ptr` из стандартной библиотеки.

## ПРИЛОЖЕНИЕ А

### Исходный код oop4.cpp

```
#include <iostream>
namespace stepik
{
    template <typename T>
    class shared_ptr
    {
    public:

        template <typename V> friend class shared_ptr;

        explicit shared_ptr(T *ptr = 0) :
            m_ptr(ptr), m_counter(new size_t(1))
        {}

        ~shared_ptr()
        {
            dec_counter();
        }

        template<typename V>
        shared_ptr(const shared_ptr<V> &other) :
            m_ptr(other.m_ptr), m_counter(other.m_counter)
        {
            inc_counter();
        }

        shared_ptr(const shared_ptr &other) :
            m_ptr(other.m_ptr), m_counter(other.m_counter)
        {
            inc_counter();
        }

        template<typename V>
        shared_ptr &operator=(const shared_ptr<V> &other)
        {
            shared_ptr(other).swap(*this);
            return *this;
        }

        shared_ptr &operator=(const shared_ptr &other)
        {
            shared_ptr(other).swap(*this);
            return *this;
        }

        template<typename V>
        bool operator == (const shared_ptr<V> &other) const
        {
            return m_ptr == other.m_ptr;
        }

        template<typename V>
        bool operator != (const shared_ptr<V> &other) const
        {
            return m_ptr != other.m_ptr;
        }
    };
}
```

```

    }

    explicit operator bool() const
    {
        return m_ptr != 0;
    }

    T *get() const
    {
        return m_ptr;
    }

    long use_count() const
    {
        return m_ptr ? *m_counter : 0;
    }

    T &operator*() const
    {
        return *m_ptr;
    }

    T *operator->() const
    {
        return m_ptr;
    }

    void swap(shared_ptr &x) noexcept
    {
        std::swap(m_ptr, x.m_ptr);
        std::swap(m_counter, x.m_counter);
    }

    void reset(T *ptr = 0)
    {
        shared_ptr new_ptr(ptr);
        swap(new_ptr);
    }

private:
    // data members
    void dec_counter()
    {
        if(--(*m_counter) == 0)
        {
            delete m_ptr;
            delete m_counter;
        }
    }

    void inc_counter()
    {
        (*m_counter)++;
    }

    T *m_ptr;
    size_t *m_counter;
};
} // namespace stepik

```