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. 7381 _____ Дорох С.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2019

Цель работы

Исследование умных указателе в C++. Необходимо реализовать умный указатель разделяемого владения объектом (shared_ptr).

Задание

Необходимо реализовать умный указатель разделяемого владения объектом (shared_ptr). Поведение реализованных функций должно быть аналогично функциям std::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.

Ход работы

shared_ptr – один из видов умных указателей, который помимо указателя на объект, хранит и счетчик умных указателей, ссылающихся на один указатель.

В классе имеется 2 поля: указатель на тип T и указатель на тип long в качестве счетчика.

Были добавлены вспомогательные методы увеличения счетчика, который увеличивает счетчик умных указателей, ссылающихся на один указатель, при создании нового умного указателя, и разрушения умного указателя, который уменьшает счетчик, и удаляет указатель, если счетчик стал равен 0.

Созданы конструкторы аналогичные конструкторам std::shared_ptr, получения значения указателя, получения количества умных указателей,

ссылающихся на такой же указатель, swap – метод меняющий содержимое двух умных указателей и метод заменяющий указатель в shared_ptr.

Исходный код каждого написанного класса представлен в приложении А.

Выводы

В ходе написания лабораторной работы были изучены умные указатели и реализован класс shared_ptr, аналогичный классу std::shared_ptr из стандартной библиотеки.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД КЛАССА shared_ptr

```
namespace stepik
{
  template <typename T>
  class shared_ptr
  {
      public:
      template <typename S>
      friend class shared_ptr;
      explicit shared_ptr(T *ptr = nullptr) : m_ptr(ptr), m_count(ptr
? new size_t(1) : nullptr) {}

      ~shared_ptr()
      {
          destroy();
      }

      shared_ptr(const shared_ptr & other) : m_ptr(other.m_ptr),
m_count(other.m_count)
      {
          if(m_count)
              (*m_count)++;
      }

      template <typename S>
      shared_ptr(const shared_ptr<S> & other) : m_ptr(other.m_ptr),
m_count(other.m_count)
      {
          if(m_count)
              (*m_count)++;
      }

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

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

      explicit operator bool() const
      {
          return (m_ptr != nullptr);
```

```
}

T* get() const
{
    return m_ptr;
}

long use_count() const
{
    return (m_count ? *m_count : 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_count, x.m_count);
}

void reset(T *ptr = nullptr)
{
    shared_ptr tmp(ptr);
    swap(tmp);
}

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


private:
T *m_ptr;
size_t *m_count;
void destroy() {
    if (m_count) {
        (*m_count)--;
        if (*m_count == 0)
        {
            delete m_ptr;
            delete m_count;
        }
```

```
            }
        }
    };
}
```