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: shared_ptr

Студент гр. 7304 _____ Овчинников Н.В.

Преподаватель _____ Размочаева Н.В

Санкт-Петербург

2019

Цель работы

Изучить класс shared_ptr – умного указателя с разделяемым владением объектом и реализовать его на языке программирования C++.

Задание

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

Ход работы

Был реализован класс shared_ptr. Данный класс содержит в себе два приватных поля: T* m_ptr – указатель на объект типа Ти long *count – счетчик, который хранит в себе количество указателей, указывающих на данный объект. Для полного функционирования класса shared_ptr были реализованы следующие методы:

- Конструктор explicit shared_ptr (T* ptr = 0), который принимает указатель на объект и записывает в поле count значение равное 1.
- Были реализованы два конструктора копирования, один из которых для поддержания полиморфизма. Реализованные конструкторы копируют данные объекта other и увеличивают счетчик на 1.
- Были реализованы два оператора присваивания, один из которых для поддержания полиморфизма. Реализованные операторы перемещают данные объекта other и увеличивают счетчик на 1.
- Были реализован деструктор, который уменьшает значение счетчика на 1. Если значение счетчика стало равным 0, то объект удаляется.
- Были реализованы операторы == для сравнения указателей.
- Были реализован метод bool, который возвращает true, если указатель не равен nullptr.
- Был реализован метод use_count, который возвращает количество указателей для данного объекта.
- Был реализован метод swap, который обеспечивает обмен содержимым.
- Были реализован метод reset, который заменяет управляемый объект объектом, на который указывает ptr.

Вывод

В ходе выполнения лабораторной работы были изучены умные указатели и реализован указатель shared_ptr – умный указатель с разделяемым владением объектом. Для заданного указателя были реализованы основные функции для работы с ним. Поведение реализованных функций соответствуют классу std::shared_ptr.

Приложение №1. Класс shared_ptr.

```
#include <iostream>

namespace stepik
{
  template <typename T>
  class shared_ptr
  {
      template <class A> friend class shared_ptr;

  public:
    explicit shared_ptr(T *ptr = 0)
    {
        m_ptr = ptr;
        if(ptr)
            count = new long(1);
        else
            count = nullptr;
    }

    ~shared_ptr()
    {
        if(use_count() > 1)
        {
            (*count) -= 1;
        }
        else
        {
            delete count;
            delete m_ptr;
            m_ptr = nullptr;
            count = nullptr;
        }
    }

    shared_ptr(const shared_ptr & other)
    {
        m_ptr = other.m_ptr;
        count = other.count;
        if(use_count())
            (*count)++;
    }

      template <typename A>
      shared_ptr(const shared_ptr<A> & other)
      {
        m_ptr = other.m_ptr;
        count = other.count;
        if(use_count())
            (*count)++;
      }

    shared_ptr& operator=(const shared_ptr & other)
    {
        if(this != &other)
        {
            this->~shared_ptr();
            m_ptr = other.m_ptr;
            count = other.count;
            if(use_count())
                (*count)++;
        }
        return *this;
    }

      template <typename A>
      shared_ptr& operator=(const shared_ptr<A> & other)
```

```
      {
          if(m_ptr != other.get())
          {
            this->~shared_ptr();
            m_ptr = other.m_ptr;
            count = other.count;
            if(use_count())
                (*count)++;
          }
          return *this;
      }

    explicit operator bool() const
    {
        return get() != nullptr;
    }

    T* get() const
    {
        return m_ptr;
    }

    long use_count() const
    {
        return (m_ptr) ? *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(count, x.count);
    }

    void reset(T *ptr = 0)
    {
        this->~shared_ptr();
        m_ptr = ptr;
        count = (ptr) ? new long(1) : nullptr;
    }

      bool operator==(const shared_ptr & other) const
      {
          return get() == other.get();
      }

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

  private:
    T* m_ptr;
    long *count;
  };
}
```