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.7304 _____ Давыдов А.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2019

1. Постановка задачи

1.1. Цель работы

Исследование реализацию умного указателя разделяемого владения объектом в языке программирования c++;

1.2. Формулировка задачи

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.

2. Ход работы

2.1. В классе shared_ptr созданы 2 приватных поля T* pointer – указатель на объект и size_t* count_link – указатель на счетчик, который показывает количество shared_ptr, указывающих на данный объект.

2.1.1. Реализован основной конструктор, который принимает указатель на объект, который по умолчанию равен 0. А также записывает в счетчик 1;

2.1.2. Реализованы конструкторы копирования для обычного объекта и для реализации полиморфизма. Перемещают данный с объекта other и увеличивают счетчик на 1.

2.1.3. Реализованы операторы присваивания для обычного объекта и для реализации полиморфизма. Удаляет старый объект, то есть уменьшает счетчик, если счетчик равен 0, то удаляем указатель. Далее перемещает данные с other и увеличивает счетчик на 1;

2.1.4. Реализован деструктор, который уменьшает счетчик на 1, если он равен 0, то удаляем указатель.

2.1.5. Реализованы операторы сравнения для хранимых указателей.

2.1.6. Реализован оператор bool(), который выводит true, если указатель не нуль, иначе false.

2.1.7. Реализован метод get(), который возвращает указатель на объект.

2.1.8. Реализован метод use_count(), который возвращает число указателей на объект.

2.1.9. Реализован метод swap, который меняет 2 shared_ptr местами.

2.1.10. Реализован метод reset(), который удаляет старый указатель и создает новый.

3. Вывод

В результате работы были изучены способы реализации умного указателя shared_ptr разделяемого владения объектом. Также были реализованы основные функции, поведение которых полностью аналогично функциям из стандартной библиотеки. Преимущество умных указателей в том, что они сами очищают память.

Приложение А:

Исходный код

Файл lr4.cpp

```
#include <iostream>

using namespace std;

namespace stepik
{
  template <typename T>
  class shared_ptr
  {
  public:
    //implementation step 1
    explicit shared_ptr(T *ptr = nullptr) : pointer(ptr), count_link(new
size_t(1))
    {}
```

```
~shared_ptr()
{
    delete_shared_ptr();
}

//Copy constructor
shared_ptr(const shared_ptr & other)
{
    pointer = other.pointer;
    count_link = other.count_link;
    ++*count_link;
}

//Assingment operator
shared_ptr& operator=(const shared_ptr & other)
{
    shared_ptr<T>(other).swap(*this);

    return *this;
}

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

T* get() const
{
    return pointer;
}

long use_count() const
{
   if(pointer)
       return *count_link;
   else
       return 0;
}

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

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

void swap(shared_ptr& x) noexcept
{
    std::swap(pointer, x.pointer);
    std::swap(count_link, x.count_link);
}

void reset(T *ptr = 0)
{
    shared_ptr<T>(ptr).swap(*this);
}

//implementation step 2
template<class Y> friend class shared_ptr;
```

```
    template<class Y>
    friend bool operator ==(shared_ptr<T> const &shr1, shared_ptr<Y> const
&shr2)
    {
        return shr1.get() == shr2.get();
    }

    template <class Y>
    shared_ptr(const shared_ptr<Y> & other) : pointer(other.pointer),
count_link(other.count_link)
    {
        ++*count_link;
    }

    template <class Y>
    shared_ptr & operator = (shared_ptr<Y> const &other)
    {
        shared_ptr<T>(other).swap(*this);

        return *this;
    }

  private:
    T* pointer;
    size_t *count_link;

    void delete_shared_ptr()
    {
        if(*count_link > 0)
            --*count_link;

        if(*count_link == 0)
        {
            delete pointer;
            delete count_link;
        }
    }
  };
} // namespace stepik
```