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 _____ Сергеев И.Д.

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* my_ptr – указатель на объект и unsigned* counter – 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. Экспериментальные результаты

3.1. Shared_ptr(2)

Код программы

```
stepik::shared_ptr<number> ptr1(new number());
stepik::shared_ptr<test> ptr2(ptr1);
cout << ptr1.use_count() << endl;
if (ptr1 == ptr2)
   cout << "SAME" << endl;
```

Вывод

```
2
SAME
```

3.2. Shared_ptr(1)

Код программы

```
int a = 5,b = 6;
stepik::shared_ptr<int> ptr1(&a);
stepik::shared_ptr<int> ptr2(ptr1);
cout << ptr1.use_count() << endl;
int c = 9;
stepik::shared_ptr<int> ptr3(&c);
cout << *ptr3 << " " << *ptr1 << endl;
```

Вывод

```
2
9 5
```

4. 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 <algorithm>
#include <iostream>
using namespace std;
namespace stepik
{
  template <typename T>
  class shared_ptr
  {
      template <typename next_obj>
      friend class shared_ptr;
  public:
    explicit shared_ptr(T *ptr = 0)
    {
      this->my_ptr = ptr;
      this->count = new unsigned;
      *count = 1;
    }

    ~shared_ptr()
    {
      remove_shared();
    }

    shared_ptr(const shared_ptr & other)
    {
      my_ptr = other.my_ptr;
      count = other.count;
      ++(*count);
    }

    template <typename next_obj>
    shared_ptr(const shared_ptr<next_obj> & other)
    {
      this->my_ptr = other.my_ptr;
      this->count = other.count;
      ++(*count);
    }

    shared_ptr& operator=(const shared_ptr & other)
    {
      if (this != &other){
          remove_shared();
          my_ptr = other.my_ptr;
          count = other.count;
          ++(*count);
      }
      return *this;
    }

    template <typename next_obj>
    shared_ptr& operator=(const shared_ptr<next_obj> & other)
    {
      if (my_ptr != other.my_ptr){
          remove_shared();
```

```
        my_ptr = other.my_ptr;
        count = other.count;
        ++(*count);
      }
      return *this;
    }

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


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

    T* get() const
    {
      return my_ptr;
    }

    long use_count() const
    {
      return my_ptr == nullptr ? 0 : (*count);
    }

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

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

    void swap(shared_ptr& x) noexcept
    {
      std::swap(my_ptr,x.my_ptr);
      std::swap(count,x.count);
    }

    void reset(T *ptr = 0)
    {
      remove_shared();
      my_ptr = ptr;
      count = new unsigned(1);
    }

  private:
     void remove_shared(){
        if ((*count) > 0)
            --(*count);
        if ((*count) == 0){
            delete my_ptr;
            delete count;
        }
      }
      T* my_ptr;
      unsigned* count;
```

```
    // data members
  };
} // namespace stepik

class test{
    int val;
};
class number : public test{
    int value;
};

int main(){
    stepik::shared_ptr<number> ptr1(new number());
    stepik::shared_ptr<test> ptr2(ptr1);
    cout << ptr1.use_count() << endl;
    if (ptr1 == ptr2)
        cout << "SAME" << endl;
}
```