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 _____ Абдульманов Э.М

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2019

Цель работы.

Изучить реализацию умного указателя разделяемого владения объектом в языке программирования c++.

Задача.

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.

Ход работы.

В классе shared_ptr создаются два приватных поля. T* pointer – указатель на объект T. И unsigned * counter – счетчик, который показывает сколько shared_prt указывают на данный объект.

- Реализуется основной конструктор, который принимает указатель на объект, по умолчанию равен 0. А так же записывает в счетчик 1.
- Реализуются конструкторы копирования, один обычный, другой для поддержания полиморфизма. Перемещают данные с объекта other. И увеличиваю счетчик на 1.
- Реализуются операторы перемещения, один обычный, другой для поддержания полиморфизма. Удаляет старый объект, то есть уменьшает счетчик, если он станет равный 0, то удаляет указатель. И дальше перемещает данные с объекта other и увеличивает счетчик на 1.
- Реализуется деструктор, который уменьшает счетчик на 1, если счетчик будет равен 0. Удаляет объект.
- Реализуются операторы сравнения для хранимых указателей.
- Реализуется оператор bool(), который выводит true,если указатель не NULL.
- Реализуется метода get(), который возвращает указатель на объект.

- Реализуется метод use_count, который возвращает число указатель на объект.
- Реализуется метод swap, который меняет два shared_ptr местами.
- Реализуется метод reset, который удаляет старый указатель и создает новый.

Результат работы.

1. Входные данные

```
void example1(){
    int a=5;
    int b=6;
    shared_ptr<int> ptr_a(&a);
    shared_ptr<int> ptr_b(ptr_a);
    cout<<ptr_a.use_count()<<endl;
}
```

Выходные данные

```
C:\Users\1\CLionProjects\Shared\cmake-build-debug\Shared.exe
2

Process finished with exit code -1073740940 (0xC0000374)
```

2. Входные данные

```
void example2(){
    int a=5;
    int b=6;
    shared_ptr<int> ptr_a(&a);
    shared_ptr<int> ptr_b(&b);
    cout<<*ptr_a<<" "<<*ptr_b<<endl;
}
```

Выходные данные

```
C:\Users\1\CLionProjects\Shared\cmake-build-debug\Shared.exe
5 6

Process finished with exit code -1073740940 (0xC0000374)
```

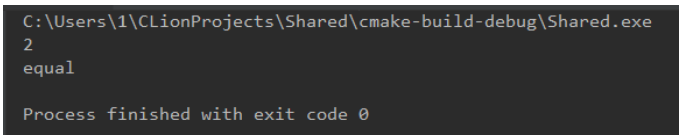
3. Входные данные

```
class A{
    int a;
};

class B:public A{
    int b;
};

void example3(){
    shared_ptr<B> ptr_B(new B());
    shared_ptr<A> ptr_A(ptr_B);
    cout<<ptr_A.use_count()<<endl;
    if(ptr_A==ptr_B)
        cout<<"equal"<<endl;
}
```

Выходные данные



Выводы.

В ходе выполнения данной лабораторной работы была изучена реализация умного указателя shared_ptr разделяемого владения объектом. И были реализованы основные функции, поведение которых полностью аналогично функциям из стандартной библиотеки данных. Умные указатели очень облегчают жизнь, ведь они сами очищают память.

Приложение А.

Исходный код.

Файл Shared.h

```
#ifndef SHARED_SHARED_H
#define SHARED_SHARED_H

#include <algorithm>
#include <iostream>
using namespace std;
template <typename T>
  class shared_ptr
  {
      template <typename object>
      friend class shared_ptr;
  public:
    explicit shared_ptr(T *ptr = 0):pointer(ptr),count(new unsigned(1)){}

    ~shared_ptr()
    {
        deleteSharedPtr();
    }

    shared_ptr(const shared_ptr & other){
        addPointer(other);
    }

    template <typename object>
    shared_ptr(const shared_ptr<object>& other){
        addPointer(other);
    }

    shared_ptr& operator=(const shared_ptr & other)
    {
      if(pointer!=other.pointer){
          deleteSharedPtr();
          addPointer(other);
      }
      return *this;
    }

    template <typename object>
    shared_ptr& operator=(const shared_ptr<object>& other){
        if(pointer!=other.pointer){
            deleteSharedPtr();
            addPointer(other);
        }
        return *this;
    }
    template <typename object>
    bool operator==(const shared_ptr<object>& other) const{
        return pointer == other.pointer;
    }
    template <typename object>
    bool operator!=(const shared_ptr<object>& other) const{
        return pointer != other.pointer;
    }
    template <typename object>
    bool operator<(const shared_ptr<object>& other)const {
```

```
        return pointer < other.pointer;
    }
    template <typename object>
    bool operator>(const shared_ptr<object>& other) const {
        return pointer > other.pointer;
    }

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

    T* get() const
    {
        return pointer;
    }

    long use_count() const
    {
        return pointer== NULL?0:(*count);
    }

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

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

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

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

  private:
    T* pointer;
    unsigned *count;
    void plusCount(){(*count)++;}
    void minusCount(){(*count)--;}
    void deleteSharedPtr(){
        if((*count)>0)
            minusCount();
        if((*count)==0){
            delete count;
            delete pointer;
        }
    }
    void addPointer(const shared_ptr & other){
        pointer=other.pointer;
        count=other.count;
```

```
        plusCount();
    }
    template <typename object>
    void addPointer(const shared_ptr<object> & other){
        this->pointer=other.pointer;
        this->count=other.count;
        plusCount();
    }
  };
#endif //SHARED_SHARED_H
```