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. 7382 Ленковский В.В.

Преподаватель Жангиров Т.М..

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.

1. Реализуется основной конструктор, который принимает указатель на объект, по умолчанию равен 0. А так же записывает в счетчик 1.
2. Реализуются конструкторы копирования, один обычный, другой для поддержания полиморфизма. Перемещают данные с объекта other. И увеличиваю счетчик на 1.
3. 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.
4. Реализуется деструктор, который уменьшает счетчик на 1, если счетчик будет равен 0. Удаляет объект.
5. Реализуются операторы сравнения для хранимых указателей.

6. Реализуется оператор bool(), который выводит true,если указатель не NULL.
7. Реализуется метода get(), который возвращает указатель на объект.
8. Реализуется метод use_count, который возвращает число указатель на объект.
9. Реализуется метод swap, который меняет два shared_ptr местами.
10. Реализуется метод reset, который удаляет старый указатель и создает новый.

Результат работы.

Входные данные:

```
int first=10;
int second=40;
int *res = new int(30);
stepik::shared_ptr<int> ptr1(&first);
stepik::shared_ptr<int> ptr2(&second);
cout<<"First:  "<<*ptr1<<" "<<"First count:  "<<ptr1.use_count()<<endl;
cout<<"Second: "<<*ptr2<<" "<<"Second count: "<<ptr2.use_count()<<endl;
if(ptr1 == ptr2)
        cout<<"First = Second"<<endl;
else
        cout<<"First != Second"<<endl;
cout<<"Swap:"<<endl;
ptr1.swap(ptr2);
cout<<"First:  "<<*ptr1<<" "<<"First count:  "<<ptr1.use_count()<<endl;
cout<<"Second: "<<*ptr2<<" "<<"Second count: "<<ptr2.use_count()<<endl;
cout<<"Reset:"<<endl;
ptr1.reset(res);
cout<<"First:  "<<*ptr1<<" "<<"First count:  "<<ptr1.use_count()<<endl;
cout<<"Second: "<<*ptr2<<" "<<"Second count: "<<ptr2.use_count()<<endl;
```

Результат:

```
First:  10 First count:  1
Second: 40 Second count: 1
First != Second
Swap:
First:  40 First count:  1
Second: 10 Second count: 1
Reset:
First:  30 First count:  1
Second: 10 Second count: 1
```

Вывод.

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 lr4.cpp

```
#include <algorithm>
#include <iostream>
using namespace std;

namespace stepik
{
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()
    {
        if(use_count()>0)
            minusCount();
        else{
            delete count;
            delete pointer;
            pointer=nullptr;
            count=nullptr;
        }
    }

    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){
          this->~shared_ptr();
          addPointer(other);
      }
      return *this;
    }

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

    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)
  {
    this->~shared_ptr();
      pointer=ptr;
      count=new unsigned(1);
  }

private:
  T* pointer;
  unsigned *count;

  void plusCount(){(*count)++;}

  void minusCount(){(*count)--;}

  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();
    }
  };
}; // namespace stepik

int main(){
int first=10;
int second=40;
int *res = new int(30);
stepik::shared_ptr<int> ptr1(&first);
stepik::shared_ptr<int> ptr2(&second);
cout<<"First:  "<<*ptr1<<" "<<"First count:  "<<ptr1.use_count()<<endl;
cout<<"Second: "<<*ptr2<<" "<<"Second count: "<<ptr2.use_count()<<endl;
if(ptr1 == ptr2)
     cout<<"First = Second"<<endl;
else
     cout<<"First != Second"<<endl;
cout<<"Swap:"<<endl;
ptr1.swap(ptr2);
cout<<"First:  "<<*ptr1<<" "<<"First count:  "<<ptr1.use_count()<<endl;
cout<<"Second: "<<*ptr2<<" "<<"Second count: "<<ptr2.use_count()<<endl;
cout<<"Reset:"<<endl;
ptr1.reset(res);
cout<<"First:  "<<*ptr1<<" "<<"First count:  "<<ptr1.use_count()<<endl;
cout<<"Second: "<<*ptr2<<" "<<"Second count: "<<ptr2.use_count()<<endl;
return 0;
}
```