# МИНОБРНАУКИ РОССИИ

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МОЭВМ**

**ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине «Объектно-ориентированное программирование» Тема: «Умные указатели»**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 7382 |  | Давкаева В.С. |
| Преподаватель |  | Жангиров Т.М. |

Санкт-Петербург

# Цель работы.

Изучить и реализовать shared\_ptr языка С++.

# Задание.

Необходимо реализовать умный указатель разделяемого владения объектом (shared\_ptr). Должны быть обеспечены следующие возможности:

* копирование указателей на полиморфные объекты stepik::shared\_ptr<Derived> derivedPtr(new Derived); stepik::shared\_ptr<Base> basePtr = derivedPtr;
* сравнение shared\_ptr, как указателей на хранимые объекты.

Поведение реализованных функций должно быть аналогично функциям std::shared\_ptr.

При выполнении этого задания вы можете определять любые вспомогательные функции. Вводить или выводить что-либо не нужно. Реализовывать функцию main не нужно. Не используйте функции из cstdlib (malloc, calloc, realloc и free).

# Ход работы.

Shared\_ptr – умный указатель, с разделяемым владением объектом через его указатель. Несколько указателей shared\_ptr могут владеть одним и тем же объектом; объект будет уничтожен, когда последний shared\_ptr, указывающий на него, будет уничтожен или сброшен. Реализуемый класс имеет два поля: указатель на объект и указатель на счётчик указателей на этот объект.

Были реализованы две вспомогательные функции: inc для инкриментирования счётчика умных указателей и dec для декрементирование счётчика и удаления объекта, если счётчик достигает нуля. Конструктор, принимающий C-указатель на объект, для которого инициализируется новый счётчик, или ссылку на другой shared\_ptr, копирую его поля и увеличиваю счётчик на единицу. Деструктор вызывает функцию dec.

Также были реализованы функции get (возвращающая указатель на объект), use\_count (возвращающая значение счётчика), swap (обменивающая поля двух умных указателей), reset (заменяющая объект, которым владеет указатель) и перегружены операторы =, ==, !=, <, >, <=, >=, \*, -> и bool аналогично обычным указателям.

Реализация класса представлена в приложении А.

# Вывод.

В ходе выполнения данной работы был реализован класс shared\_ptr, аналогичный классу std::shared\_ptr из стандартной библиотеки.

# Приложение А.

#include "pch.h"

#include <iostream>

namespace stepik {

template<typename T>

class shared\_ptr {

template<typename TT>

friend class shared\_ptr;

public:

explicit shared\_ptr(T \*ptr = 0) {

ref = ptr;

count = new unsigned int(1);

}

~shared\_ptr() {

dec();

}

shared\_ptr(const shared\_ptr &other) {

ref = other.ref;

count = other.count;

inc();

}

template<typename TT>

shared\_ptr(const shared\_ptr<TT> &other) {

ref = other.ref;

count = other.count;

inc();

}

shared\_ptr &operator=(const shared\_ptr &other) {

if ((void\*)this == (void\*)&other)return \*this;

dec();

ref = (T\*)other.ref;

count = other.count;

inc();

return \*this;

}

template<typename TT>

shared\_ptr &operator=(const shared\_ptr<TT> &other) {

if ((void\*)this == (void\*)&other)return \*this;

dec();

ref = (T\*)other.ref;

count = other.count;

inc();

return \*this;

}

bool operator==(const shared\_ptr &other)const {

return (void\*)ref == (void\*)other.ref;

}

template<typename TT>

bool operator==(const shared\_ptr<TT> &other)const {

return (void\*)ref == (void\*)other.ref;

}

explicit operator bool() const {

return ref != nullptr;

}

T \*get() const {

return ref;

}

long use\_count() const {

return (ref == nullptr) ? 0 : \*count;

}

T &operator\*() const {

return \*ref;

}

T \*operator->() const {

return ref;

}

void swap(shared\_ptr &x) noexcept {

auto t = ref;

auto c = count;

ref = x.ref;

x.ref = t;

count = x.count;

x.count = c;

}

void reset(T \*ptr = 0) {

dec();

count = new unsigned int(1);

ref = ptr;

}

private:

T \*ref;

unsigned int\* count;

void inc() {

(\*count)++;

}

void dec() {

if ((\*count == 0))throw " ";

(\*count)--;

if ((\*count) == 0) {

delete ref;

delete count;

}

}

};

}