**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. 7304 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Юруть Е.А. |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Размочаева Н.В. |

Санкт-Петербург

2019

**Цель работы.**

Необходимо реализовать умный указатель разделяемого владения объектом (shared\_ptr).

**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\_counter для инкриментирования счётчика умных указателей и deg\_counter 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 deg\_counter.

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. Файл shared\_ptr.h.**

#include <algorithm> // std::swap

#include <cstddef> // size\_t

namespace stepik

{

template <typename T>

class shared\_ptr

{

template <typename U>

friend class shared\_ptr;

public:

explicit shared\_ptr(T \*ptr = nullptr) :

m\_ptr(ptr),

m\_count(ptr ? new size\_t(1) : nullptr)

{

// implement this

}

~shared\_ptr()

{

// implement this

release();

}

template <typename U>

shared\_ptr(const shared\_ptr<U> & other) :

m\_ptr(other.m\_ptr),

m\_count(other.m\_count)

{

// implement this

if (m\_count)

(\*m\_count)++;

}

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

m\_ptr(other.m\_ptr),

m\_count(other.m\_count)

{

// implement this

if (m\_count)

(\*m\_count)++;

}

template <typename U>

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

{

// implement this

shared\_ptr p(other);

swap(p);

return \*this;

}

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

{

// implement this

shared\_ptr p(other);

swap(p);

return \*this;

}

explicit operator bool() const

{

// implement this

return (m\_ptr != nullptr);

}

T\* get() const

{

// implement this

return m\_ptr;

}

long use\_count() const

{

// implement this

return (m\_count ? \*m\_count : 0);

}

T& operator\*() const

{

// implement this

return (\*m\_ptr);

}

T\* operator->() const

{

// implement this

return m\_ptr;

}

template <typename U>

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

{

return (m\_ptr == other.m\_ptr);

}

void swap(shared\_ptr& x) noexcept

{

// implement this

std::swap(m\_ptr, x.m\_ptr);

std::swap(m\_count, x.m\_count);

}

void reset(T \*ptr = nullptr)

{

// implement this

shared\_ptr p(ptr);

swap(p);

}

private:

// data members

T \*m\_ptr;

size\_t \*m\_count;

void release()

{

if (m\_ptr)

{

(\*m\_count)--;

if (\*m\_count == 0)

{

delete m\_ptr;

delete m\_count;

}

}

}

};

} // namespace stepik