**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. 7381 |  | Дорох С.В. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2019

**Цель работы**

Исследование умных указателе в С++. Необходимо реализовать умный указатель разделяемого владения объектом (shared\_ptr).

**Задание**

Необходимо реализовать умный указатель разделяемого владения объектом (shared\_ptr). Поведение реализованных функций должно быть аналогично функциям ﻿ std::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. 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.

**Ход работы**

shared\_ptr – один из видов умных указателей, который помимо указателя на объект, хранит и счетчик умных указателей, ссылающихся на один указатель.

В классе имеется 2 поля: указатель на тип Т и указатель на тип long в качестве счетчика.

Были добавлены вспомогательные методы увеличения счетчика, который увеличивает счетчик умных указателей, ссылающихся на один указатель, при создании нового умного указателя, и разрушения умного указателя, который уменьшает счетчик, и удаляет указатель, если счетчик стал равен 0.

Созданы конструкторы аналогичные конструкторам std::shared\_ptr, получения значения указателя, получения количества умных указателей, ссылающихся на такой же указатель, swap – метод меняющий содержимое двух умных указателей и метод заменяющий указатель в shared\_ptr.

Исходный код каждого написанного класса представлен в приложении А.

**Выводы**

В ходе написания лабораторной работы были изучены умные указатели и реализован класс shared\_ptr, аналогичный классу std::shared\_ptr из стандартной библиотеки.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД КЛАССА shared\_ptr**

namespace stepik

{

template <typename T>

class shared\_ptr

{

public:

template <typename S>

friend class shared\_ptr;

explicit shared\_ptr(T \*ptr = nullptr) : m\_ptr(ptr), m\_count(ptr ? new size\_t(1) : nullptr) {}

~shared\_ptr()

{

destroy();

}

shared\_ptr(const shared\_ptr & other) : m\_ptr(other.m\_ptr), m\_count(other.m\_count)

{

if(m\_count)

(\*m\_count)++;

}

template <typename S>

shared\_ptr(const shared\_ptr<S> & other) : m\_ptr(other.m\_ptr), m\_count(other.m\_count)

{

if(m\_count)

(\*m\_count)++;

}

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

{

shared\_ptr tmp(other);

swap(tmp);

return \*this;

}

template <typename S>

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

{

shared\_ptr tmp(other);

swap(tmp);

return \*this;

}

explicit operator bool() const

{

return (m\_ptr != nullptr);

}

T\* get() const

{

return m\_ptr;

}

long use\_count() const

{

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

}

T& operator\*() const

{

return \*m\_ptr;

}

T\* operator->() const

{

return m\_ptr;

}

void swap(shared\_ptr& x) noexcept

{

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

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

}

void reset(T \*ptr = nullptr)

{

shared\_ptr tmp(ptr);

swap(tmp);

}

template <typename S>

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

{

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

}

private:

T \*m\_ptr;

size\_t \*m\_count;

void destroy() {

if (m\_count) {

(\*m\_count)--;

if (\*m\_count == 0)

{

delete m\_ptr;

delete m\_count;

}

}

}

};

}