**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 |  | Находько А.Ю. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2019

**Цель работы.**

Изучить работу умного указателя shared\_ptr языка C++, также написать свою реализацию умного указателя для разделяемого владения объектом.

**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.

**Требования к реализации**:

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.**

Для реализации класса умного указателя shared\_ptr были написаны следующие функции: конструктор копирования, деструктор, оператор «=» для присваивания объектов друг другу; оператор «==» для сравнения двух полиморфных объектов; оператор «bool», метод get(), который возвращает указатель на объект, use\_count, возвращающая количество указателей, владеющих объектом, операторы «» и «», методы swap и reset.

Исходный код представлен в приложении А.

**Выводы.**

В результате выполнения лабораторной работы ознакомился с работой умного указателя shared\_ptr языка C++, также написал свою реализацию умного указателя для разделяемого владения объектом.

Приложение А

Код программы

**shared\_ptr.cpp**

namespace stepik

{

template <typename T>

class shared\_ptr

{

template <typename T2> friend class shared\_ptr;

public:

explicit shared\_ptr(T \*ptr = 0) : m\_ptr(ptr), m\_count(new long(1)){}

~shared\_ptr()

{

delete\_ptr();

}

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

{

if(m\_count)

(\*m\_count)++;

}

template<typename T2> shared\_ptr(const shared\_ptr<T2>& other) : m\_ptr(other.m\_ptr), m\_count(other.m\_count)

{

if(m\_count)

(\*m\_count)++;

}

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

{

if(m\_ptr != other.m\_ptr)

{

delete\_ptr();

m\_ptr = other.m\_ptr;

m\_count = other.m\_count;

(\*m\_count)++;

}

return \*this;

}

template<typename T2> shared\_ptr& operator=(const shared\_ptr<T2>& other)

{

if(m\_ptr!=other.m\_ptr)

{

delete\_ptr();

m\_ptr=other.m\_ptr;

m\_count=other.m\_count;

(\*m\_count)++;

}

return \*this;

}

template <typename T2> bool operator==(const shared\_ptr<T2>& other) const

{

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

}

explicit operator bool() const

{

return m\_ptr != nullptr;

}

T\* get() const

{

return m\_ptr;

}

long use\_count() const

{

return (m\_ptr) ? (\*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 = 0)

{

shared\_ptr tmp(ptr);

swap(tmp);

}

private:

T\* m\_ptr;

long\* m\_count;

void delete\_ptr()

{

if((\*m\_count) > 0)

(\*m\_count)--;

if((\*m\_count) == 0)

{

delete m\_ptr;

delete m\_count;

}

}

};

} // namespace stepik