**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине «Объектно-ориентированное программирование»**

**Тема: «Умные указатели»**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент гр. 7381 | |  | Вологдин М.Д. |
| Преподаватель |  |  | Жангиров Т. Р. |

Санкт-Петербург

2019

**Цель работы:**

Реализовать умный указатель разделяемого владения объектом (shared\_ptr) с функциями, аналогичными﻿ std::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 – 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.

Класс shared\_ptr содержит 2 приватных поля – указатель на объект и на счетчик количества умных указателей на этот объект.

Реализованы следующие методы класса:

1. Конструктор, принимающий стандартный указатель, деструктор и конструктор копирования.
2. Копирующий оператор присваивания, оператор приведения типа bool, оператор \* и ->.
3. Clear – освобождает указатель и уменьшает счетчик.
4. use\_count – возвращает количество указателей на объект.
5. swap – меняет содержимое заданного и текущего указателя.
6. reset – заменяет текущий указатель на объект заданным.

Для поддержки полиморфизма был добавлен дружеский класс shared\_ptr с другим типом указателя.

Код представлен в приложении А.

**Выводы:**

В ходе выполнения лабораторной работы были изучены и реализован умный указатель разделяемого владения объектом (shared\_ptr) с поддержкой полиморфного использования.

**Приложение А**

**Исходный код**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

#include <iostream>

#include <memory>

namespace stepik

{

template <typename T>

class shared\_ptr

{

public:

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

: ptr(ptr), count( new int(1))

{

}

~shared\_ptr()

{

clear();

}

shared\_ptr(const shared\_ptr & other)

: ptr(other.ptr), count(other.count)

{

++(\*count);

}

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

{

if (ptr != other.ptr)

{

clear();

ptr = other.ptr;

count = other.count;

++(\*count);

}

return \*this;

}

explicit operator bool() const

{

return ptr != nullptr;

}

T\* get() const

{

return ptr;

}

int use\_count() const

{

return ptr == nullptr ? 0 : \*count;

}

T& operator\*() const

{

return \*ptr;

}

T\* operator->() const

{

return ptr;

}

void swap(shared\_ptr& x) noexcept

{

std::swap(ptr, x.ptr);

std::swap(count, x.count);

}

void reset(T \*ptr = 0)

{

shared\_ptr temp(ptr);

swap(temp);

}

void clear()

{

if (--(\*count) <= 0)

{

delete ptr;

delete count;

}

ptr = nullptr;

count = nullptr;

}

template <typename D>

friend class shared\_ptr;

template <class D>

shared\_ptr(const shared\_ptr<D> & other) : ptr(other.ptr), count(other.count)

{

++ (\*count);

}

template <typename D>

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

{

if (ptr != other.ptr)

{

clear();

ptr = other.ptr;

count = other.count;

++ (\*count);

}

return \*this;

}

private:

T\* ptr;

int \* count;

};

// namespace stepik

template <typename T, typename D>

bool operator==(const shared\_ptr<T>& lhs, const shared\_ptr<D>& rhs) {

return lhs.get() == rhs.get();

}

template<typename T, typename... \_Args>

stepik::shared\_ptr<T>

my\_make\_shared(\_Args&&... \_\_args)

{

return stepik::shared\_ptr<T>(new T(\_\_args...));

}

}