**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 |  | Ильясов А.В. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2019

**Цель работы**

Исследование умных указателе в С++.

**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 реализация умных указателей и реализована собственная реализация.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД КЛАССА shared\_ptr**

namespace stepik

{

template <typename T>

class shared\_ptr

{

public:

template<class Y>

friend class shared\_ptr;

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

: ptr(ptr), ptr\_count(ptr ? new long(1) : nullptr)

{}

~shared\_ptr()

{

destroy();

}

shared\_ptr(const shared\_ptr<T> &other)

: ptr(other.ptr), ptr\_count(other.ptr\_count)

{

increase\_ptr\_count();

}

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

{

if (ptr != other.get()) {

destroy();

shared\_ptr tmp(other);

tmp.swap(\*this);

increase\_ptr\_count();

}

return \*this;

}

template <typename Y>

shared\_ptr(const shared\_ptr<Y> &other)

: ptr(other.ptr), ptr\_count(other.ptr\_count)

{

increase\_ptr\_count();

}

template <typename Y>

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

{

if (ptr != other.get()) {

destroy();

shared\_ptr tmp(other);

tmp.swap(\*this);

//increase\_ptr\_count();

}

return \*this;

}

explicit operator bool() const

{

return ptr != nullptr;

}

T \* get() const

{

return ptr;

}

long use\_count() const

{

return ptr\_count ? \*ptr\_count : 0;

}

T & operator\*() const

{

return \*ptr;

}

T \* operator->() const

{

return ptr;

}

void swap(shared\_ptr &x) noexcept

{

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

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

}

void reset(T \*new\_ptr = nullptr)

{

destroy();

ptr = new\_ptr;

ptr\_count = ptr ? new long(1) : nullptr;

}

private:

T \*ptr;

long \*ptr\_count;

void increase\_ptr\_count() {

if (ptr\_count)

(\*ptr\_count)++;

}

void destroy() {

if (ptr\_count)

(\*ptr\_count)--;

if (!use\_count()) {

delete ptr;

delete ptr\_count;

}

}

};

template<typename T, typename Y>

bool operator ==(const shared\_ptr<T> &left, const shared\_ptr<Y> &right) {

return left.get() == right.get();

}

} // namespace stepik