**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

|  |  |  |
| --- | --- | --- |
| Студентка гр. 7382 |  | Головина Е.С. |
| Преподаватель |  | Жангиров Т. Р. |

Санкт-Петербург

2019

**Цель работы.**

Реализация умного указателя для разделяемого владения объектом.

**Задание.**

Необходимо реализовать умный указатель разделяемого владения объектом (shared\_ptr), именно его конструкторы, деструктор, операторы =, bool(), \*, ->, функции get, use\_count, swap, reset.

Также нужно предусмотреть, чтобы shared\_ptr 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.

Поведение реализованных функций должно быть аналогично функциям std::shared\_ptr﻿.

При выполнении этого задания можно 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 – один из умных указателей, суть которого заключается в том, что он хранит в себе обычный указатель, а так же счетчик указателей shared\_ptr, владеющих управляемым объектом. 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.

Таким образом, для реализации данного умного указателя необходимы 2 поля класса: указатель на управляемый объект (ptr) и счетчик владельцев (counter).

Была реализована вспомогательная функция: destroying() – если счетчик равен 1, то освобождается память, выделенная под счетчик и управляемый объект, иначе значение счетчика уменьшается на единицу.

Конструктор, принимающий обычный указатель ptr, инициализирует значения членов ptr указателем ptr и counter значением nullptr. Затем выделяет память под счетчик и инициализирует его единицей, если ptr не равен nullptr.

Деструктор класса вызывает метод destroying().

Конструктор, принимающий другой shared\_ptr, копирует поля переданного указателя в текущий и, если ptr не равен nullptr, увеличивает счетчик на 1.

Конструктор, принимающий другой shared\_ptr с произвольным хранимым типом указателя реализуется аналогично предыдущему указателю с тем лишь изменением, что shared\_ptr для любого класса объявлен дружественным классом к данному, поскольку ptr и counter находятся в private области класса и доступа к ним извне нет.

Операторы присваивания реализованы таким образом, что если происходит попытка присвоения аналогичного shared\_ptr, изменений не происходит, иначе, если до этого ptr не был равен nullptr, то выполнится функция destroying(), а затем произойдет присвоение значений ptr и counter из other в текущий shared\_ptr. Значение счётчика увеличится на 1, если полученный ptr не равен nullptr.

Оператор приведения shared\_ptr к типу bool возвращается результат сравнения хранимого указателя с nullptr. Если в ptr nullptr функция возвращает false, иначе true.

Метод get() возвращает указатель ptr.

Метод use\_count() возвращает значение счетчика, если он существует, иначе 0. Счетчика не существует в том случае, если в ptr хранится nullptr.

Оператор разыменования указателя (оператор \*) возвращает разыменованный указатель ptr.

Оператор -> возвращает указатель ptr.

Функция обмена указателей swap обменивает поля данных между данным объектом и переданным.

Функция reset() работает аналогично конструктору, принимающему обычный указатель ptr, с тем лишь исключением, что если изначально ptr не был равен nullptr, то в начале выполнится метод destroying().

Для сравнения указателей перегружаем операторы == и !=, которые возвращают результат сравнения указателя ptr для текущего и для переданного shared\_ptr.

**Выводы.**

В ходе выполнения лабораторной работы были реализован умный указатель разделяемого владения. Поведение реализованных функций аналогично std::shared\_ptr.

**ПРИЛОЖЕНИЕ А**

**SHARED\_PTR**

namespace stepik

{

template <typename T>

class shared\_ptr

{

public:

template <typename U>

friend class shared\_ptr;

explicit shared\_ptr(T \*ptr = 0):ptr(ptr),counter(nullptr)

{

if (this->ptr) counter = new long(1);

}

~shared\_ptr()

{

if (ptr) destroying();

}

shared\_ptr(const shared\_ptr & other):ptr(other.ptr),counter(other.counter)

{

if (ptr) (\*counter)++;

}

template <typename U>

shared\_ptr(const shared\_ptr<U> & other):ptr(other.ptr),counter(other.counter)

{

if (ptr) (\*counter)++;

}

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

{

if (ptr != other.ptr){

if (ptr) destroying();

ptr=other.ptr;

counter=other.counter;

if (ptr!= nullptr) (\*counter)++;

}

return \*this;

}

template <typename U>

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

{

if (ptr != other.ptr){

if (ptr) destroying();

ptr=other.ptr;

counter=other.counter;

if (ptr!= nullptr) (\*counter)++;

}

return \*this;

}

template <typename U>

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

return ptr == other.ptr;

}

template <typename U>

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

return ptr != other.ptr;

}

explicit operator bool() const

{

if (ptr == nullptr) return false;

else return true;

}

T\* get() const

{

return ptr;

}

long use\_count() const

{

if (counter) return \*counter;

else return 0;

}

T& operator\*() const

{

return \*ptr;

}

T\* operator->() const

{

return ptr;

}

void swap(shared\_ptr& x) noexcept

{

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

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

}

void reset(T \*ptr = 0)

{

if (this->ptr) destroying();

this->ptr = ptr;

if (this->ptr) counter = new long(1);

else counter = nullptr;

}

private:

void destroying(){

if (\*counter==1){

delete ptr;

delete counter;

}

else (\*counter)--;

}

T \* ptr;

long \* counter;

};

} // namespace stepik