|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА ИУ8 “КОМПЬЮТЕРНАЯ БЕЗОПАСНОСТЬ”

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №8**

Студент Лустин Павел Евгеньевич

*(*Фамилия И.О)

Группа\_\_ИУ8-24\_\_\_\_\_\_\_\_

Студент  **\_\_\_\_\_\_15.05\_\_\_\_\_\_\_\_\_**  Лустин \_\_ \_

*дата (*Фамилия И.О)

Преподаватель **\_\_\_\_\_\_15.05\_\_\_\_\_\_\_**  Барыкин Дмитрий

*дата (*Фамилия И.О)*.*

*2024 г.*

Цель работы

Цель работы состоит в изучении использования “умных” указателей

Условие задачи

**Часть 1**

Реализовать шаблон для задания «умного» указателя по аналогии шаблона **std::unique\_ptr.** В шаблоне предусмотреть конструктор, который получает «сырой» указатель, деструктор, конструкторы копирования и перемещения (что-то удаляется), операторы присваивания с копированием и перемещением (что-то удаляется), метод get, возвращающий «сырой указатель», выполнить перегрузку операций \* и ->, запретить создание копий объектов, реализовать перемещение. Возможные заголовки методов шаблона кроме конструкторов и операторов присваивания копирования и перемещения приведены ниже:

template<class T>

class **MyUnique**

{

T \* p=nullptr;

public:

**MyUnique**(T \*p);

~**MyUnique**();

T \* **get**() const;

T & operator\*();

T \* operator->();

};

Для создания объекта типа **MyUnique** разработать глобальную шаблонную функцию Make\_MyUnique с переменным числом параметров, которая получает параметры как у конструктора объекта, на который указывает указатель. Теория по таким функциям приведена ниже в подразделе **Шаблоны с переменным числом параметров (*variadic template*).**

В функции main продемонстрировать все заданные возможности, в том числе, создать указатель на объект своего класса, имеющего конструктор с параметрами (например, класса MyPoint – точка на плоскости) c помощью функции Make\_MyUnique, получающей параметры как конструктора своего класса (MyPoint).

**Часть 2**

Аналогично разработать шаблон для задания «умного» указателя по аналогии шаблона **std::shared\_ptr,** который отличается от шаблона части 1 тем, что разрешает копирование. Назвать его можно MyShared. Разработать также глобальную шаблонную функцию Make\_MyShared, которая получает параметры как у конструктора объекта, на который указывает указатель. Продемонстрировать все возможности в функции main по аналогии с частью 1.

**Часть 1**

Текст программы с комментариями

#include <iostream>

using namespace std;

template<class T>

class MyUnique //Шаблонный класс "умного" указателя

{

private:

T\* ptr;

public:

MyUnique() //Пустой конструктор

{

ptr = nullptr;

}

MyUnique(T\*ptr) //Конструктор с начальным значением

{

this->ptr = ptr;

}

MyUnique(const MyUnique& Unique) = delete; //Конструктор копирования (его не может быть, тк запрещено создание копий)

MyUnique(MyUnique&& Unique) //Конструктор перемещения

{

this->ptr = Unique.ptr;

Unique.ptr = nullptr;

delete Unique.ptr;

}

T& operator=(const MyUnique& Unique) = delete; //Оператор присваивания копирования (его не может быть, тк запрещено создание копий)

T& operator=(MyUnique&& Unique) //Оператор перемещения

{

this->ptr = Unique.ptr;

Unique.ptr = nullptr;

delete Unique.ptr;

return \*this;

}

T \* operator->() //Перегрузка оператора ->

{

return ptr;

}

T& operator\* () //Перегрузка оператора \*

{

return \*ptr;

}

T \* get() const //Метод get, который возвращает "сырой указатель"

{

return ptr;

}

~MyUnique() //Деструктор

{

delete ptr;

}

};

template<typename T, typename...Args>

MyUnique<T> Make\_MyUnique(Args&&... args) { //Шаблонная функция с переменным числом параметров

return MyUnique<T>(new T(forward<Args>(args)...));

}

class MyPoint //Класс - точка на плоскости

{

public:

int x, y;

MyPoint() //Пустой конструктор

{

x = 0;

y = 0;

}

MyPoint(int x, int y) //Конструктор с начальным значением

{

this->x = x;

this->y = y;

}

void print() //Метод печати координат точки на плоскости

{

cout << "MyPoint(int x, int y): x=" << x << " y=" << y << endl;

}

~MyPoint() {}; //Деструктор

};

int main()

{

setlocale(LC\_ALL, "Rus");

MyUnique<MyPoint> mp1 = Make\_MyUnique<MyPoint>(3, 4); //Создание указателя на объект своего класса, имеющего конструктор с параметрами с помощью функции Make\_MyUnique, получающей параметры как конструктора своего класс (MyPoint)

mp1->print(); //Печать координат точки, на которую указывает умный указатель

MyUnique<MyPoint> mp2 = move(mp1); //Демонстрация оператора перемещения

mp2->print();

return 0;

}

**Часть 2**

Текст программы с комментариями

#include <iostream>

using namespace std;

template<class T>

class MyShared //Шаблонный класс "умного" указателя с возможностью копирования

{

private:

T\* ptr;

public:

int\* count; //Счётчик для ссылок

MyShared() //Пустой конструктор

{

ptr = nullptr;

count = new int(0);

}

MyShared(T\* ptr) //Конструктор с начальным значением

{

this->ptr = ptr;

this->count = new int(1);

}

MyShared(const MyShared& Shared) //Конструктор копирования

{

this->ptr = Shared.ptr;

this->count = Shared.count;

++(\*count);

}

MyShared(MyShared&& Shared) //Конструктор перемещения

{

this->ptr = Shared.ptr;

this->count = Shared.count;

Shared.ptr = nullptr;

Shared.count = new int(0);

}

T& operator=(const MyShared& Shared) //Оператор присваивания копирования

{

~MyShared();

this->ptr = Shared.ptr;

this->count = Shared.count;

if (this->count != nullptr)

{

(\*this->count)++;

}

return \*this;

}

T& operator=(MyShared&& Shared) //Оператор присваивания перемещения

{

~MyShared();

this->ptr = Shared.ptr;

this->count = Shared.count;

Shared.ptr = nullptr;

Shared.count = new int(0);

return \*this;

}

T\* operator->() //Перегрузка оператора ->

{

return ptr;

}

T& operator\* () //Перегрузка оператора \*

{

return \*ptr;

}

T\* get() const //Метод get, который возвращает "сырой указатель"

{

return ptr;

}

~MyShared() //Деструктор

{

--(\*count);

if (\*count == 0)

{

delete ptr;

delete count;

count = nullptr;

}

}

};

template<typename T, typename...Args>

MyShared<T> Make\_MyShared(Args&&... args) { //Шаблонная функция с переменным числом параметров

return MyShared<T>(new T(forward<Args>(args)...));

}

class MyPoint //Класс - точка на плоскости

{

public:

int x, y;

MyPoint() //Пустой конструктор

{

x = 0;

y = 0;

}

MyPoint(int x, int y) //Конструктор с начальным значением

{

this->x = x;

this->y = y;

}

void print() //Метод печати координат точки на плоскости

{

cout << "MyPoint(int x, int y): x=" << x << " y=" << y << endl;

}

~MyPoint() {}; //Деструктор

};

int main()

{

setlocale(LC\_ALL, "Rus");

MyShared<MyPoint> ms1 = Make\_MyShared<MyPoint>(5, 6); //Создание указателя на объект своего класса, имеющего конструктор с параметрами с помощью функции Make\_MyUnique, получающей параметры как конструктора своего класс (MyPoint)

ms1->print(); //Печать координат точки, на которую указывает умный указатель

cout << "Кол-во ссылок в ms1:" << \*ms1.count << endl;

MyShared<MyPoint> ms2 = ms1;

ms2->print();

cout << "Кол-во ссылок в ms2:" << \*ms2.count << endl;

return 0;

}

Вывод:

Я научился использовать “умные” указатели