ФАКУЛЬТЕТ «Информатика и системы управления»  
КАФЕДРА «Информационная безопасность»

ОТЧЕТ  
по лабораторной работе № 8  
по учебной дисциплине «Алгоритмические языки»  
на тему: «Изучение «умных» указателей»

Вариант 23

Выполнил:   
Студент 1 курса, гр. ИУ8-24  
Спиридонов Олег

**Цель работы:**

Изучить принципы работы умных указателей и шаблонов с переменным числом параметров (variadic template).

**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.

**Выполнение работы:**

Файл **AL Laba 8 (smart pointers - unique).cpp:**

#include <iostream>

#include <utility>

using namespace std;

template<class T>

class MyUnique {

T \*p = nullptr;

public:

explicit MyUnique(T\* p) {

this->p = p;

};

MyUnique(MyUnique&& T2) noexcept: p(T2.p) {

T2.p = nullptr;

}

MyUnique(const T&) = delete;

~MyUnique() {

delete p;

};

T\* get() const {

return p;

};

T& operator\*() const {

return \*p;

};

T\* operator->() const {

return p;

};

T& operator=(T&& T2) noexcept {

if (this != &T2) {

delete p;

p = T2.p;

T2.p = nullptr;

}

return \*this;

}

T& operator=(const T&) = delete;

};

template<typename T, typename... Args>

MyUnique<T> Make\_MyUnique(Args&&... args) {

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

}

class MyPoint {

int x, y;

public:

MyPoint(int x, int y) : x(x), y(y) {

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

}

void print() const {

cout << "Point(" << x << ", " << y << ")" << std::endl;

}

};

int main()

{

MyUnique<MyPoint> unique\_pointer = Make\_MyUnique<MyPoint>(2, 3);

unique\_pointer->print();

cout << unique\_pointer.get() << endl;

MyUnique<MyPoint> unique\_pointer2 = move(unique\_pointer);

unique\_pointer2->print();

}

**Результат работы программы:**

MyPoint(int x, int y): x=2 y=3

Point(2, 3)

0000022C69DE8020

Point(2, 3)

Файл **AL Laba 8 (smart pointers - shared).cpp:**

#include <iostream>

#include <utility>

using namespace std;

template<class T>

class MyShared {

T\* p = nullptr;

int\* count = nullptr;

void deleter() {

if (\*count > 0) {

--(\*count);

}

if (\*count == 0) {

delete p;

delete count;

}

}

public:

MyShared(T\* p) {

this->p = p;

this->count = new int(1);

};

MyShared(const MyShared& T2){

this->p = T2.p;

this->count = T2.count;

++(\*count);

};

MyShared(MyShared&& T2) noexcept : p(T2.p), count(T2.count) {

T2.p = nullptr;

T2.count = nullptr;

};

~MyShared() {

deleter();

};

T\* get() const {

return p;

};

T& operator\*() const {

return \*p;

};

T\* operator->() const {

return p;

};

T& operator=(const T& T2) {

if (this != &T2) {

deleter();

p = T2.p;

count = T2.count;

\*count++;

}

return \*this;

}

T& operator=(T&& T2) noexcept {

if (this != &T2) {

deleter();

p = T2.p;

count = T2.count;

T2.p = nullptr;

T2.count = nullptr;

}

return \*this;

}

};

template<typename T, typename... Args>

MyShared<T> Make\_MyShared(Args&&... args) {

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

}

class MyPoint {

int x, y;

public:

MyPoint(int x, int y) : x(x), y(y) {

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

}

void print() const {

cout << "Point(" << x << ", " << y << ")" << std::endl;

}

};

int main()

{

MyShared<MyPoint> shared\_pointer1 = Make\_MyShared<MyPoint>(2, 3);

shared\_pointer1->print();

cout << shared\_pointer1.get() << endl;

MyShared<MyPoint> shared\_pointer2 = shared\_pointer1;

shared\_pointer2->print();

cout << shared\_pointer2.get() << endl;

if (shared\_pointer1.get() == shared\_pointer2.get()) {

cout << "shared\_pointer1 and shared\_pointer2 link to the same thing" << endl;

}

MyShared<MyPoint> shared\_pointer3 = move(shared\_pointer1);

shared\_pointer3->print();

cout << shared\_pointer3.get() << endl;

if (shared\_pointer1.get() == nullptr) {

cout << "shared\_pointer1 is gone" << endl;

}

}

**Результат работы программы:**

MyPoint(int x, int y): x=2 y=3

Point(2, 3)

0000022740473910

Point(2, 3)

0000022740473910

shared\_pointer1 and shared\_pointer2 link to the same thing

Point(2, 3)

0000022740473910

shared\_pointer1 is gone

**Вывод:**

В ходе выполнения лабораторной работы были изучены принципы работы умных указателей и шаблонов с переменным числом параметров (variadic template). Для выполнения задачи были реализованы 2 варианта программы: с запретом на копирование и без запрета.