



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, искусственный интеллект и системы управления» (ИУ)  
КАФЕДРА «Информационная безопасность» (ИУ8)

Лабораторная работа №8  
ПО КУРСУ  
«Алгоритмические языки»

Студент

ИУ8-21  
(Группа)

Г. А. Карев  
(И. О. Фамилия)

Преподаватель:

В. В. Соборова  
(И.О. Фамилия)

## Цель работы:

Изучить умные указатели и прописать их самому.

## Задача:

### 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) с 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.

**Код:**

## ЧАСТЬ 1

*MyUnique.cpp*

---

```
#include <iostream>
using namespace std;

template<class T>
class MyUnique {
    T *p = nullptr;

public:
    MyUnique(T *ptr) : p(ptr) {}

    ~MyUnique() { delete p; }

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

    MyUnique(MyUnique&& move) : p(move.p) {
        move.p = nullptr;
    }

    MyUnique& operator=(MyUnique&& g_m) {
        if (this != &g_m) {
            delete p;
            p = g_m.p;
            g_m.p = nullptr;
        }
        return *this;
    }

    T* get() const { return p; } //сырой указатель
    T& operator*() { return *p; } //разыменовывание
    T* operator->() { return p; } //для доступа к полям класса
};

template<typename T, typename... Args>
MyUnique<T> Make_MyUnique(Args&&... args) {
    return MyUnique<T>(new T(forward<Args>(args)...));
}
```

## *main.cpp*

---

```
#include <iostream>
#include "MyUnique.cpp"
using namespace std;

class MyPoint {
    double x, y;

public:
    MyPoint(double x, double y) : x(x), y(y) {}
    void print() const {
        cout << "Точка ( " << x << ", " << y << " )" << endl;
    }
};

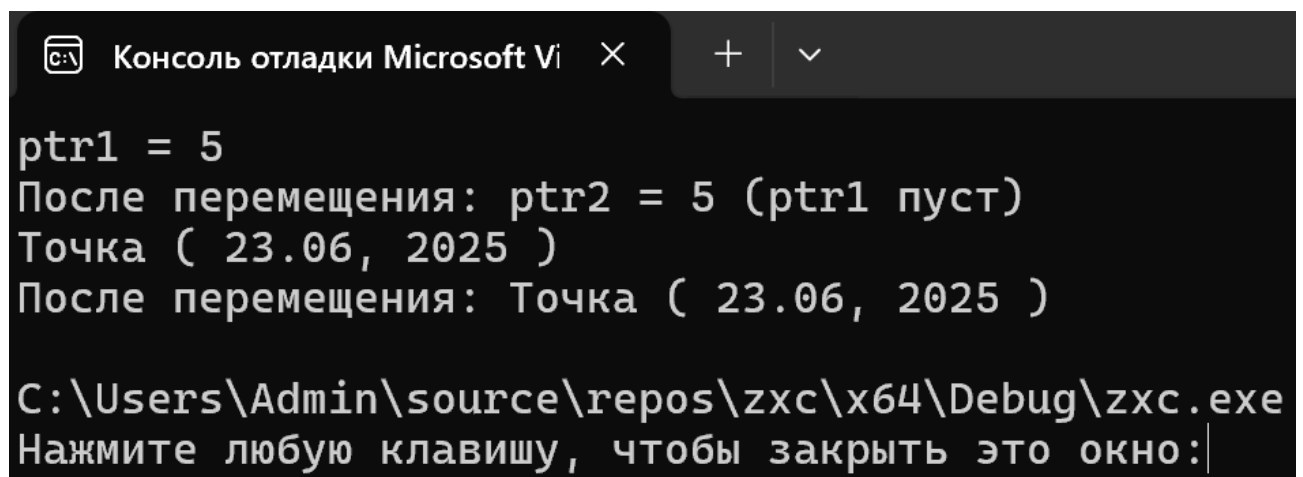
int main() {
    setlocale(LC_ALL, "RUS");

    MyUnique<int> ptr1(new int(52));
    cout << "ptr1 = " << *ptr1 << endl;
    MyUnique<int> ptr2 = move(ptr1);
    cout << "После перемещения: ptr2 = " << *ptr2 << " (ptr1 пуст)" << endl;

    auto pointPtr = Make_MyUnique<MyPoint>(228, 20.25);
    pointPtr->print();
    MyUnique<MyPoint> movedpointPtr = move(pointPtr);
    cout << "После перемещения: ";
    movedpointPtr -> print();
    return 0;
}
```

## *Вывод программы*

---



```
Консоль отладки Microsoft Vi  X  +  v

ptr1 = 5
После перемещения: ptr2 = 5 (ptr1 пуст)
Точка ( 23.06, 2025 )
После перемещения: Точка ( 23.06, 2025 )

C:\Users\Admin\source\repos\zxc\x64\Debug\zxc.exe
Нажмите любую клавишу, чтобы закрыть это окно:
```

## ЧАСТЬ 2

*MyShared.cpp*

---

```
#include <iostream>
using namespace std;

template<class T>
class MyShared {
    T* p = nullptr;
    int* count = nullptr;

    void clear() {
        if (count && --(*count) == 0) {
            delete p;
            delete count;
        }
        p = nullptr;
        count = nullptr;
    }

public:
    MyShared(T* ptr = nullptr) : p(ptr), count(ptr ? new int(1) : nullptr) {}

    ~MyShared() { clear(); }

    MyShared(const MyShared& copy) : p(copy.p), count(copy.count) {
        if (count) {
            ++(*count);
        }
    }

    MyShared& operator=(const MyShared& c_m) {
        if (this != &c_m) {
            clear();
            p = c_m.p;
            count = c_m.count;
            if (count) {
                ++(*count);
            }
        }
        return *this;
    }

    MyShared(MyShared&& move) : p(move.p), count(move.count) {
        move.p = nullptr;
        move.count = nullptr;
    }

    MyShared& operator=(MyShared&& g_m) {
        if (this != &g_m) {
            clear();
            p = g_m.p;
            count = g_m.count;
            g_m.p = nullptr;
            g_m.count = nullptr;
        }
        return *this;
    }

    T* get() const { return p; } //сырой указатель
    T& operator*() { return *p; } //разыменовывание
    T* operator->() { return p; } //для доступа к полям класса

    int use_count() const {
        return count ? *count : 0;
    }
};
```

```

template<typename T, typename... Args>
MyShared<T> Make_MyShared(Args&&... args) {
    return MyShared<T>(new T(forward<Args>(args)...));
}

```

### *main.cpp*

---

```

#include <iostream>
#include "MyShared.cpp"
using namespace std;

class Territory {
    int x, y;

public:
    Territory(int x, int y) : x(x), y(y) {}

    void print() const {
        cout << "Территория: " << x << " на " << y << " метров" << endl;
    }

    int area() const {
        return x * y;
    }
};

int main() {
    setlocale(LC_ALL, "RUS");

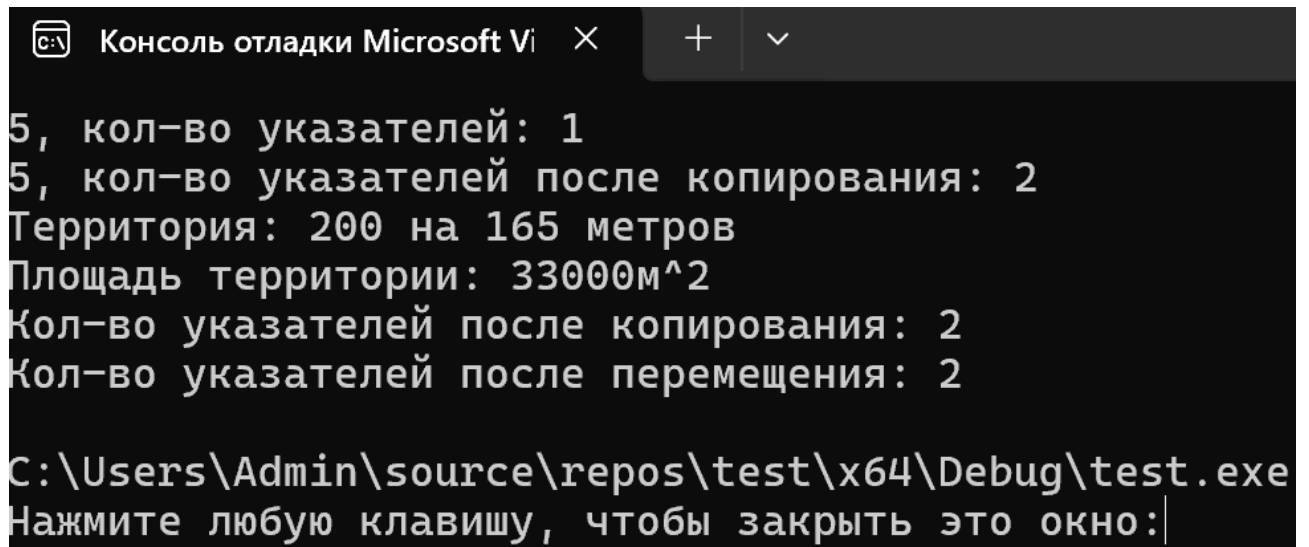
    MyShared<int> shared1(new int(5));
    cout << *shared1 << ", кол-во указателей: " << shared1.use_count() << endl;
    MyShared<int> shared2 = shared1;
    cout << *shared2 << ", кол-во указателей после копирования: " <<
shared1.use_count() << endl;

    auto terPtr = Make_MyShared<Territory>(200, 165);
    terPtr->print();
    cout << "Площадь территории: " << terPtr->area() << "м^2 " << endl;

    auto terPtr2 = terPtr;
    cout << "Кол-во указателей после копирования: " << terPtr.use_count() << endl;
    auto terPtr3 = move(terPtr2);
    cout << "Кол-во указателей после перемещения: " << terPtr.use_count() << endl;

    return 0;
}

```



```
5, кол-во указателей: 1
5, кол-во указателей после копирования: 2
Территория: 200 на 165 метров
Площадь территории: 33000м^2
Кол-во указателей после копирования: 2
Кол-во указателей после перемещения: 2

C:\Users\Admin\source\repos\test\x64\Debug\test.exe
Нажмите любую клавишу, чтобы закрыть это окно:
```

**Вывод:**

Научился использовать умные указатели, создал аналоги `unique_ptr` и `shared_ptr`. Написал работоспособные программы.