Д.З. №2

1) Написати простий клас та простий параметризований клас(C++) відповідно до власного варіанту з лабораторної роботи №4.
2) Написати простий клас та простий узагальнений клас(Java) відповідно до власного варіанту з лабораторної роботи №4.
2) Написати простий клас та простий узагальнений клас(C#) відповідно до власного варіанту з лабораторної роботи №4.

** 4 лабораторна робота має бути виконана у повному обсязі, а для здачі Д.З. №2 достатньо реалізувати поля класу та методи доступу до них*

```cpp
#include <iostream>
#include <algorithm>
#include <vector>

//#define USE_MOVE_SEMANTICS
class MemoryBlock{
private:
        size_t size;
        int* data;

public:
        explicit MemoryBlock(size_t size_) : size(size_), data(new int[size_]){
                std::cout << "MemoryBlock(size_t length): size = " << size << "." << std::endl;
        }

        MemoryBlock(const MemoryBlock & other) : size(other.size), data(new int[other.size])
        {
                std::cout << "MemoryBlock(const MemoryBlock & other): size = " << other.size << std::endl;
                std::copy(other.data, other.data + size, data);
        }

#ifdef USE_MOVE_SEMANTICS
        MemoryBlock(MemoryBlock&& other) : data(nullptr), size(0){
                std::cout << "MemoryBlock(MemoryBlock&&): size = " << other.size << std::endl;

                data = other.data;
                size = other.size;

                other.data = nullptr;
                other.size = 0;
        }
#endif
        ~MemoryBlock(){
                std::cout << "~MemoryBlock(): length = " << size;

                if (data != nullptr){
                        delete[] data;
                        std::cout << " (resource deleted)";
                }
                std::cout << std::endl;
        }
};

int main()
{
        std::vector<MemoryBlock> v;
        v.push_back(MemoryBlock(25));
}
```

```cpp
//#pragma once
#include <iostream>
#include <algorithm>
#include <vector>

//using namespace std;
#define USE_MOVE_SEMANTICS
class MemoryBlock{
private:
        size_t _length; // The length of the resource.
        int* _data; // The resource.

public:
        // Simple constructor that initializes the resource.
        explicit MemoryBlock(size_t length) : _length(length), _data(new int[length]){
                std::cout << "In MemoryBlock(size_t). length = "
                        << _length << "." << std::endl;
        }

        // Copy constructor.
        MemoryBlock(const MemoryBlock& other)
                : _length(other._length)
```

```
                , _data(new int[other._length])
        {
                std::cout << "In MemoryBlock(const MemoryBlock&). length = "
                        << other._length << ". Copying resource." << std::endl;

                std::copy(other._data, other._data + _length, _data);
        }

        // Copy assignment operator.
        MemoryBlock& operator=(const MemoryBlock& other)
        {
                std::cout << "In operator=(const MemoryBlock&). length = "
                        << other._length << ". Copying resource." << std::endl;

                if (this != &other)
                {
                        // Free the existing resource.
                        delete[] _data;

                        _length = other._length;
                        _data = new int[_length];
                        std::copy(other._data, other._data + _length, _data);
                }
                return *this;
        }

        // Retrieves the length of the data resource.
        size_t Length() const
        {
                return _length;
        }

#ifdef USE_MOVE_SEMANTICS
        // Move constructor.
        MemoryBlock(MemoryBlock&& other) : _data(nullptr), _length(0){
                std::cout << "In MemoryBlock(MemoryBlock&&). length = "
                        << other._length << ". Moving resource." << std::endl;

                // Copy the data pointer and its length from the
                // source object.
                _data = other._data;
                _length = other._length;

                // Release the data pointer from the source object so that
                // the destructor does not free the memory multiple times.
                other._data = nullptr;
                other._length = 0;
        }
        // Move assignment operator.
        MemoryBlock& operator=(MemoryBlock&& other){
                std::cout << "In operator=(MemoryBlock&&). length = "
                        << other._length << "." << std::endl;

                if (this != &other)
                {
                        // Free the existing resource.
                        delete[] _data;

                        // Copy the data pointer and its length from the
                        // source object.
                        _data = other._data;
                        _length = other._length;

                        // Release the data pointer from the source object so that
                        // the destructor does not free the memory multiple times.
                        other._data = nullptr;
                        other._length = 0;
                }
                return *this;
        }
#endif
        // Destructor.
        ~MemoryBlock()
        {
                std::cout << "In ~MemoryBlock(). length = "
                        << _length << ".";

                if (_data != nullptr)
                {
                        std::cout << " Deleting resource.";
                        // Delete the resource.
                        delete[] _data;
                }

                std::cout << std::endl;
        }
};
```

```
int main(){
        // Create a vector object and add a few elements to it.
        std::vector<MemoryBlock> v;
        v.push_back(MemoryBlock(25));
        //v.push_back(MemoryBlock(75));

        // Insert a new element into the second position of the vector.
        //v.insert(v.begin() + 1, MemoryBlock(50));
}
```