**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

Інститут **ІКНІ**

Кафедра **ПЗ**

# ЗВІТ

До лабораторної роботи №7

На тему: «Робота з динамічною пам’яттю»

З дисципліни «Об’єктно-орієнтоване програмування»

**Лектор:** доцент каф. ПЗ

Коротєєва Т.О.

**Виконала:** ст.гр. ПЗ-23

Кохман О.В.

**Прийняла:** доцент каф. ПЗ

Коротєєва Т.О.

«_____»_____2022р.

∑ ____ _____.

Львів – 2022

**Тема:** Робота з динамічною пам’яттю.

**Мета:** Навчитись виділяти місце під обєкти динамічно. Навчитись створювати та використовувати конструктор копіювання, перевантажувати оператор присвоєння. Ознайомитися з принципами створення та функціонування деструкторів.

## Теоретичні відомості

ВИДИ ПАМ’ЯТІ

Кожна змінна чи константа програми розміщується в адресному просторі програми в одному з видів пам’яті: статичній, локальній (стек) чи динамічній.

В статичній пам’яті розміщуються глобальні змінні (оголошені поза всіма блоками – функцією, методом, класом) і статичні змінні (перед типом яких вказується ключове слово static, при цьому змінна може знаходиться де завгодно, в тому числі і в тілі функції, методу чи класу). Різниця між статичною та глобальною змінними проявляється, коли програма складається з декількох файлів: глобальні змінні доступні в будь-яких файлах вихідного коду, а статичні – тільки в тому файлі, де були оголошені. В статичній пам'яті не рекомендується тримати великі об'єкти (наприклад, масиви), а хорошим кодом з використанням ООП вважається програмний код, в якому використання глобальних і статичних змінних зведено до мінімуму.

Локальна пам’ять або стек – частина адресного простору програми, де розміщуються змінні функцій та методів. Пам’ять для них виділяється при вході в блок програми і вивільняється при виході з нього. Динамічна пам’ять – решта адресного простору програми, де можуть бути розміщені дані. Вона виділяється і вивільняється за допомогою спеціальних інструкцій, які може використовувати розробник ПЗ. Це дозволяє в ході виконання програми контролювати і коригувати об’єм використовуваної пам’яті і, відповідно, створювати програми, котрі можуть опрацьовувати великі об’єми даних, обходячи обмеженість розміру реально доступної фізичної пам’яті.

Доступ до динамічної пам’яті можливий тільки через вказівники, які програміст може зв’язувати з виділеною ділянкою пам’яті. Динамічна пам’ять в мові C++ виділяється за допомогою оператора new і звільняється за допомогою оператора delete. Можна також використовувати с-функції, такі як alloc, malloc, calloc, realloc для виділення/перевиділення пам’яті і відповідну їм функцію free для звільнення виділеної пам’яті. Проте

специфіка роботи оператора new і наведених вище функцій відрізняється. Тому не можна змішувати виклики оператора new і функції free, чи навпаки функції malloc, наприклад, і оператора delete. Якщо не звільняти виділену динамічну пам’ять, то вона буде зайнята до закінчення програми, що зменшує доступний обсяг вільної пам’яті і може призводити до некоректної роботи програми чи до її непередбачуваного завершення. Тому завжди, як тільки виділена пам’ять стає непотрібною, її необхідно звільняти.

## ОБ’ЄКТИ В СТАТИЧНІЙ, ЛОКАЛЬНІЙ ТА ДИНАМІЧНІЙ ПАМ’ЯТІ

Об’єкти, як і змінні будь-якого стандартного типу, можна виділяти в усіх трьох видах пам’яті.

## РОБОТА З ДИНАМІЧНОЮ ПАМ’ЯТТЮ В ОБ’ЄКТАХ

Якщо в об’єкті виділяється динамічна пам’ять, на яку вказує його поле-вказівник (об’єкт володіє виділеною пам’яттю), то ця пам’ять обов’язково має бути звільнена об’єктом або передана у володіння в інше місце програми. При цьому потрібно пам’ятати, що стандартні конструктор копіювання та оператор присвоєння не виконують “глибокого” копіювання (не копіюють виділену пам’ять, а копіюють тільки вказівники на неї). Тому при їх виконанні можливі ситуації, коли різні об’єкти міститимуть вказівники на одну і ту ж ділянку пам’яті. При цьому, якщо об’єкти не знатимуть про таку ситуацію, можливе повторне звільнення уже звільненої пам’яті, що приведе до фатальної помилки виконання програми. Тому, якщо екземпляри класів виділяють динамічну пам’ять і зберігають вказівники на неї у своїх полях, необхідно для них перевизначати конструктор копіювання та оператор присвоєння, а для звільнення пам’яті

при потребі – деструктор.

Альтернативою перевизначенню конструктора копіювання, оператора присвоєння та деструктора є використання спеціальних вказівників (std::auto_ptr для < C++11, std::shared_ptr, std::unique_ptr для C++11 і вище).

### Індивідуальне завдання

1. Переглянути лістинг коду в прикладі. Пояснити вивід програми.
2. Створити клас відповідно до завдання (див. Додаток).
3. Розробити для класу конструктор за замовчуванням та декілька звичайних конструкторів. Реалізувати функції-члени відповідно до завдання (див. Додаток).
4. Створити конструктор копіювання.

5. Перевантажити операцію присвоєння.
6. Створити деструктор для вивільнення динамічно виділеної пам’яті.
7. Об'єкти класу розмістити в динамічній пам'яті.
8. Продемонструвати розроблені можливості класу завдяки створеному віконному застосуванню.
9. Оформити звіт до лабораторної роботи.

Клас Array – одновимірний масив. Пам’ять під елементи масиву повинна виділятися динамічно.

Реалізувати такі функції члени:

1. Знаходження максимального значення.
2. Знаходження мінімального значення.
3. Знаходження середнього арифметичного значення масиву.
4. Сортування елементів масиву методом вибірки за спаданням.
5. Сортування елементів масиву методом бульбашки за зростанням.
6. Зміна розмірів масиву.

Перевантажити операції. При цьому вибір механізму перевантаження обрати самостійно (чи метод, чи дружня-функція):

1. Додавання (почленне додавання елементів масиву)
2. Віднімання (почленне віднімання елементів масиву
3. Множення на скаляр.
4. Введення масиву з StringGrid (>>)
5. Виведення масиву у StringGrid (<<)
6. Введення масиву з ListBox (>>)
7. Виведення масиву у ListBox (<<)
8. Виведення масиву у Memo (<<)

Забезпечити можливість отримання значення елементу [i] подібно до доступу до елементів звичайного одновимірного масиву.

## Код програми

Назва файлу: MyForm.h

```
#pragma once
#include "DynamicArray.h"
namespace Project {
	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;
```

```
	public ref class MyForm : public System::Windows::Forms::Form
	{
	public:
		MyForm(void)
		{
			InitializeComponent();
		}

	protected:
		~MyForm()
		{
			if (components)
			{
				delete components;
			}
		}
	private: System::Windows::Forms::DataGridView^ dataGridView1;
	private: System::Windows::Forms::ListBox^ listBox1;
	private: System::Windows::Forms::ListBox^ listBox2;
	private: System::Windows::Forms::RichTextBox^ richTextBox1;
	private: System::Windows::Forms::Button^ button1;
	private: System::Windows::Forms::Button^ button2;
	private: System::Windows::Forms::Button^ button3;
	private: System::Windows::Forms::Button^ button4;
	private: System::Windows::Forms::Button^ button5;
	private: System::Windows::Forms::Button^ button6;
	private: System::Windows::Forms::TextBox^ textBox1;
	private: System::Windows::Forms::TextBox^ textBox2;
	private: System::Windows::Forms::TextBox^ textBox3;
	private: System::Windows::Forms::DataGridViewTextBoxColumn^ Elements;
	private: System::Windows::Forms::Button^ button7;
	private: System::Windows::Forms::TextBox^ textBox4;
	private: System::Windows::Forms::TextBox^ textBox5;
	private: System::Windows::Forms::Button^ button8;
	private: System::Windows::Forms::TextBox^ textBox6;
	private: System::Windows::Forms::RichTextBox^ richTextBox2;
	protected:
	private:
		System::ComponentModel::Container ^components;
#pragma region Windows Form Designer generated code
                                // another code//

#pragma endregion
private: System::Void operations(System::Object^ sender, System::EventArgs^ e) {
    DynamicArray* object = new DynamicArray(10);
    dataGridView1 >> object;
      DynamicArray* secondObject =  new DynamicArray(*object);
      DynamicArray* thirdObject = new DynamicArray(10);
      thirdObject = object;
      System::Windows::Forms::Button^ button = (Button^)sender;
      System::String^ textOfButtons = button->Text;
      if (textOfButtons == "min") {
		textBox1->Text = System::Convert::ToString(object->minValue());
      }
      else if (textOfButtons == "max") {
		textBox2->Text = System::Convert::ToString(object->maxValue());
      }
      else if (textOfButtons == "average") {
		textBox3->Text = System::Convert::ToString(object->avgValue());
      }
      else if (textOfButtons == "to print") {
		richTextBox1 << object;
		listBox1 << secondObject;
```

```
			dataGridView1 << thirdObject;
		}
		else if (textOfButtons == "selection") {
			object->selection();
			richTextBox1 << object;
			listBox1 << object;
			dataGridView1 << object;
		}
		else if (textOfButtons == "bubble") {
			object->bubble();
			richTextBox1 << object;
			listBox1 << object;
			dataGridView1 << object;
		}
		else if (textOfButtons == "index") {
			const int index = System::Convert::ToInt64(textBox4->Text);
			int result = (*object)[index];
			textBox5->Text = System::Convert::ToString(result);
		}
		else if (textOfButtons == "*") {
			const int number = System::Convert::ToInt64(textBox6->Text);
			*object = (*object) * number;
			richTextBox2 << object;
		}
}
};
}
```

Назва файлу: MyForm.cpp

```
#include "MyForm.h"
using namespace Project;
int main() {
	Application::EnableVisualStyles();
	Application::SetCompatibleTextRenderingDefault(false);
	Application::Run(gcnew MyForm());
	return 0;
}
```

Назва файлу: DynamicArray.h

```
#include <random>
#include <cassert>
using namespace std;
class DynamicArray {
private:
	int length;
	int* array;
public:
	DynamicArray();
	DynamicArray(const int _length);
	~DynamicArray();
	DynamicArray(const DynamicArray& some);
	void randomNumbers();
	int maxValue() const;
	int minValue() const;
	double avgValue() const;
	void selection();
	void bubble();
	void changeSize(const int _length);
	DynamicArray operator*(const int n);
	DynamicArray& operator=(const DynamicArray& array);
	int& operator[](const int index);
```

```
	DynamicArray operator+(const DynamicArray& other);
	DynamicArray operator-(const DynamicArray& other);
	friend int operator+(const DynamicArray& some);
	friend int operator-(const DynamicArray& some);
	friend void operator<<(System::Windows::Forms::RichTextBox^ richTextBox,
const DynamicArray *array);
	friend void operator<<(System::Windows::Forms::ListBox^ listBox, const
DynamicArray *array);
	friend void operator<<(System::Windows::Forms::DataGridView^ dataGridView,
const DynamicArray *array);
	friend void operator>>(System::Windows::Forms::DataGridView^ dataGridView,
DynamicArray *array);
};
#endif
```

Назва файлу: DynamicArray.cpp

```
#include "DynamicArray.h"
DynamicArray::DynamicArray() {
	length = 0;
	array = new int[length];
}
DynamicArray::DynamicArray(const int _length) : length(_length) {
	array = new int[length];
}
DynamicArray::~DynamicArray() {
	delete[] array;
}
DynamicArray::DynamicArray(const DynamicArray& some) { // copy
	this->length = some.length;
	this->array = new int[this->length];
	for (int i = 0; i < length; i++) {
		this->array[i] = some.array[i];
	}
}
void DynamicArray::randomNumbers() {
	random_device random_device;
	mt19937 generator(random_device());
	uniform_int_distribution<int> distribution(0, 20);
	for (int i = 0; i < length; i++) {
		array[i] = distribution(generator);
	}
}
int DynamicArray::maxValue() const {
	int max = array[0];
	for (int i = 0; i < length; i++) {
		if (array[i] > max) {
			max = array[i];
		}
	}
	return max;
}
int DynamicArray::minValue() const {
	int min = array[0];
	for (int i = 0; i < length; i++) {
		if (array[i] < min) {
			min = array[i];
		}
	}
	return min;
}
double DynamicArray::avgValue() const {
	double avg = 0;
```

```
	for (int i = 0; i < length; i++) {
		avg += array[i];
	}
	return avg / length;
}
void DynamicArray::selection() { // метод вибірки за спаданням
	int max, count;
	int temp;
	for (int counter = 0; counter < length; counter++) {
		count = counter;
		max = array[counter];
		for (int i = counter; i < length; i++) {
			if (array[i] > max) {
				max = array[i];
				count = i;
			}
		}
		temp = array[counter];
		array[counter] = array[count];
		array[count] = temp;
	}
}
void DynamicArray::bubble() { // метод бульбашки за зростанням
	int size = length;
	int temp = 0;
	while (size >= 0) {
		for (int i = 1; i < size; i++) {
			if (array[i - 1] > array[i]) {
				temp = array[i - 1];
				array[i - 1] = array[i];
				array[i] = temp;
			}
		}
		size--;
	}
}
void DynamicArray::changeSize(const int _length) {
	length = _length;
	array = new int[length];
}
int operator+(const DynamicArray& some) {
	int sum = 0;
	for (int i = 0; i < some.length; i++) {
		sum += some.array[i];
	}
	return sum;
}
int operator-(const DynamicArray& some) {
	int sum = some.array[0];
	for (int i = 1; i < some.length; i++) {
		sum -= some.array[i];
	}
	return sum;
}
DynamicArray DynamicArray::operator+(const DynamicArray& other) {
	if (other.length >= this->length) {
		for (int i = 0; i < this->length; i++) {
			array[i] += other.array[i];
		}
	}
	return *this;
}
DynamicArray DynamicArray::operator-(const DynamicArray& other) {
```

```
	if (other.length >= this->length) {
		for (int i = 0; i < this->length; i++) {
			array[i] -= other.array[i];
		}
	}
	return *this;
}
DynamicArray DynamicArray::operator*(const int n) {
	for (int i = 0; i < this->length; i++) {
		this->array[i] *= n;
	}
	return *this;
}
void operator<<(System::Windows::Forms::RichTextBox^ richTextBox, const
DynamicArray *array) {
	richTextBox->Clear();
	for (int i = 0; i < array->length; i++) {
		richTextBox->Text += System::Convert::ToString(array->array[i]) + "
";
	}
}
void operator<<(System::Windows::Forms::ListBox^ listBox, const DynamicArray
*array) {
	listBox->Items->Clear();
	for (int i = 0; i < array->length; i++) {
		listBox->Items->Add(System::Convert::ToString(array->array[i]));
	}
}
void operator<<(System::Windows::Forms::DataGridView^ dataGridView,const
DynamicArray *array) {
	dataGridView->Rows->Clear();
	for (int i = 0; i < array->length; i++) {
		dataGridView->Rows->Add();
		dataGridView->Rows[i]->Cells[0]->Value =
System::Convert::ToString(array->array[i]);
	}
}
void operator>>(System::Windows::Forms::DataGridView^ dataGridView, DynamicArray
*some) {
	some->length = dataGridView->Rows->Count-1;
	for (int i = 0; i < some->length; i++) {
		some->array[i] = System::Convert::ToInt16(dataGridView->Rows[i]-
>Cells[0]->Value);
	}
}
int& DynamicArray::operator[](const int index) {
	assert(index >= 0 && index < length);
	return array[index];
}
DynamicArray& DynamicArray::operator=(const DynamicArray& some) {
	if (this == &some) {
		return *this;
	}
	length = some.length;
	for (int i = 0; i < length; i++) {
		array[i] = some.array[i];
	}
	return *this;
} #include "DynamicArray.h"
DynamicArray::DynamicArray() {
	length = 0;
	array = new int[length];
}
```

```cpp
DynamicArray::DynamicArray(const int _length) : length(_length) {
	array = new int[length];
}
DynamicArray::~DynamicArray() {
	delete[] array;
}
DynamicArray::DynamicArray(const DynamicArray& some) { // copy
	this->length = some.length;
	this->array = new int[this->length];
	for (int i = 0; i < length; i++) {
		this->array[i] = some.array[i];
	}
}
void DynamicArray::randomNumbers() {
	random_device random_device;
	mt19937 generator(random_device());
	uniform_int_distribution<int> distribution(0, 20);
	for (int i = 0; i < length; i++) {
		array[i] = distribution(generator);
	}
}
int DynamicArray::maxValue() const {
	int max = array[0];
	for (int i = 0; i < length; i++) {
		if (array[i] > max) {
			max = array[i];
		}
	}
	return max;
}
int DynamicArray::minValue() const {
	int min = array[0];
	for (int i = 0; i < length; i++) {
		if (array[i] < min) {
			min = array[i];
		}
	}
	return min;
}
double DynamicArray::avgValue() const {
	double avg = 0;
	for (int i = 0; i < length; i++) {
		avg += array[i];
	}
	return avg / length;
}
void DynamicArray::selection() { // метод вибірки за спаданням
	int max, count;
	int temp;
	for (int counter = 0; counter < length; counter++) {
		count = counter;
		max = array[counter];
		for (int i = counter; i < length; i++) {
			if (array[i] > max) {
				max = array[i];
				count = i;
			}
		}
		temp = array[counter];
		array[counter] = array[count];
		array[count] = temp;
	}
}
```

```
void DynamicArray::bubble() { // метод бульбашки за зростанням
	int size = length;
	int temp = 0;
	while (size >= 0) {
		for (int i = 1; i < size; i++) {
			if (array[i - 1] > array[i]) {
				temp = array[i - 1];
				array[i - 1] = array[i];
				array[i] = temp;
			}
		}
		size--;
	}
}
void DynamicArray::changeSize(const int _length) {
	length = _length;
	array = new int[length];
}
int operator+(const DynamicArray& some) {
	int sum = 0;
	for (int i = 0; i < some.length; i++) {
		sum += some.array[i];
	}
	return sum;
}
int operator-(const DynamicArray& some) {
	int sum = some.array[0];
	for (int i = 1; i < some.length; i++) {
		sum -= some.array[i];
	}
	return sum;
}
DynamicArray DynamicArray::operator+(const DynamicArray& other) {
	if (other.length >= this->length) {
		for (int i = 0; i < this->length; i++) {
			array[i] += other.array[i];
		}
	}
	return *this;
}
DynamicArray DynamicArray::operator-(const DynamicArray& other) {
	if (other.length >= this->length) {
		for (int i = 0; i < this->length; i++) {
			array[i] -= other.array[i];
		}
	}
	return *this;
}
DynamicArray DynamicArray::operator*(const int n) {
	for (int i = 0; i < this->length; i++) {
		this->array[i] *= n;
	}
	return *this;
}
void operator<<(System::Windows::Forms::RichTextBox^ richTextBox, const
DynamicArray *array) {
	richTextBox->Clear();
	for (int i = 0; i < array->length; i++) {
		richTextBox->Text += System::Convert::ToString(array->array[i]) + "
";
	}
}
```

```
void operator<<(System::Windows::Forms::ListBox^ listBox, const DynamicArray
*array) {
	listBox->Items->Clear();
	for (int i = 0; i < array->length; i++) {
		listBox->Items->Add(System::Convert::ToString(array->array[i]));
	}
}
void operator<<(System::Windows::Forms::DataGridView^ dataGridView,const
DynamicArray *array) {
	dataGridView->Rows->Clear();
	for (int i = 0; i < array->length; i++) {
		dataGridView->Rows->Add();
		dataGridView->Rows[i]->Cells[0]->Value =
System::Convert::ToString(array->array[i]);
	}
}
void operator>>(System::Windows::Forms::DataGridView^ dataGridView, DynamicArray
*some) {
	some->length = dataGridView->Rows->Count-1;
	for (int i = 0; i < some->length; i++) {
		some->array[i] = System::Convert::ToInt16(dataGridView->Rows[i]-
>Cells[0]->Value);
	}
}
int& DynamicArray::operator[](const int index) {
	assert(index >= 0 && index < length);
	return array[index];
}
DynamicArray& DynamicArray::operator=(const DynamicArray& some) {
	if (this == &some) {
		return *this;
	}
	length = some.length;
	for (int i = 0; i < length; i++) {
		array[i] = some.array[i];
	}
	return *this;
}
```

## Протокол роботи

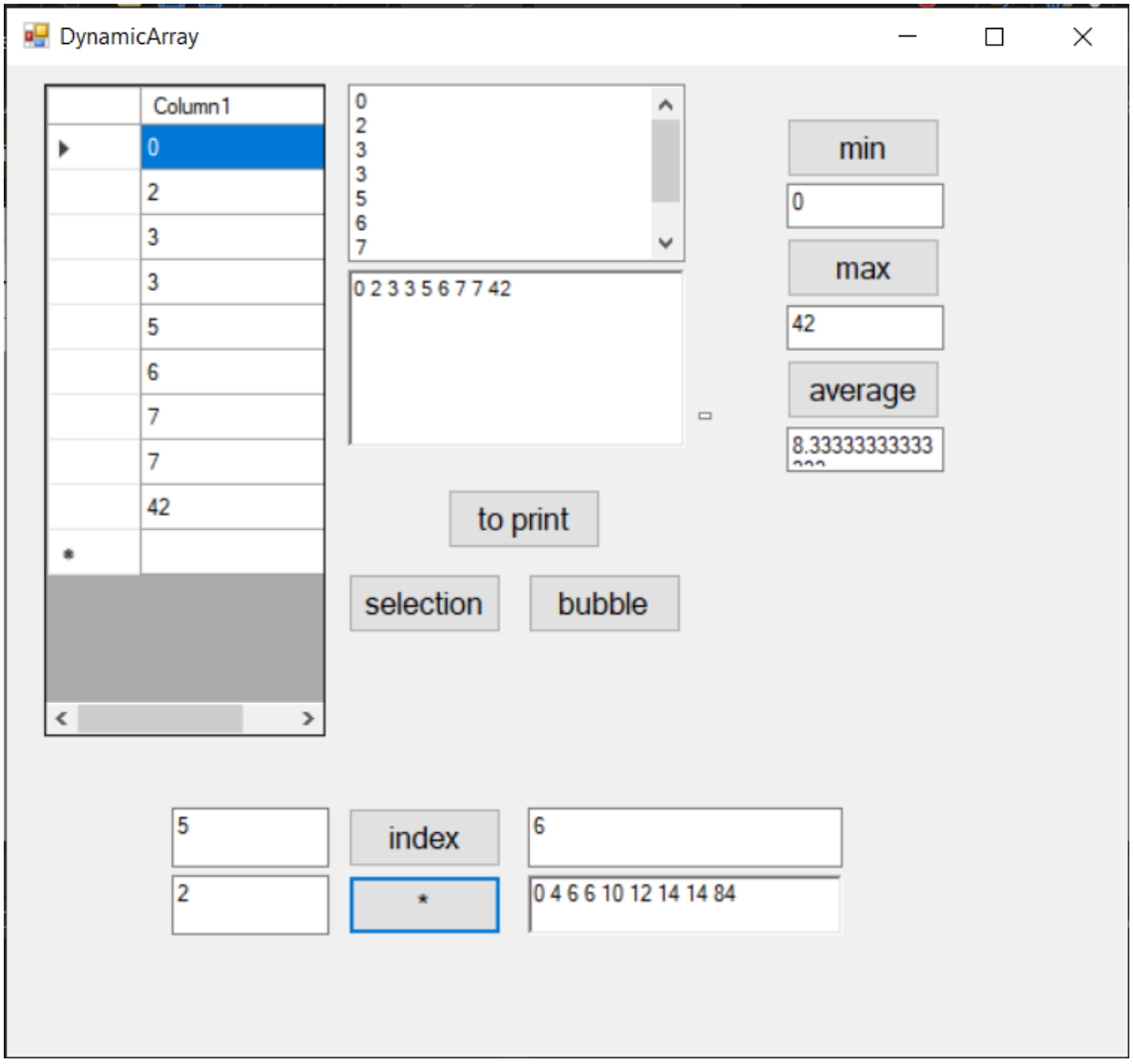


Рис.1 Форма програми з результатами

## Висновок

На цій лабораторній роботі я навчилась працювати з динамічною пам’яттю, а саме – створювати об’єкти динамічно, також працювала з перевантаженням операторів через методи класу та дружні функції, реалізувала конструктор копіювання, перевантажила оператор присвоєння, створила деструктор, де звільняється уся динамічно виділена пам’ять, та продемонструвала усе це за допомогою форми у Visual Studio 2022.