Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення



**ЗВІТ**

**Про виконання лабораторної роботи № 7**

*«*Робота з динамічною пам’яттю*»*

**з дисципліни «***Об’єктно–орієнтоване програмування***»**

**Лектор:**

ст. викладач кафедри ПЗ

Коротєєва Т.О.

**Виконав:**

студ. групи ПЗ-12

Артех О.Ю.

**Прийняла:**

асистент кафедри ПЗ

Терендій О.В.

«\_\_\_» \_\_\_\_\_\_ 2020 р.

∑ = \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Львів – 2020

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

**Мета роботи:** Навчитися виділяти місце під об’єкти динамічно. Навчитися створювати та використовувати конструктор копіювання, перевантажувати оператор присвоєння. Ознайомитися з принципами створення та функціонування деструкторів.

**Теоретичні відомості**

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 С++ виділяється за допомогою оператора new і звільняється за допомогою оператора delete. Можна також використовувати c-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, як і змінні будь-якого стандартного типу, можна виділяти в усіх трьох видах пам’яті.

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.

**Індивідуальне завдання**

1. Переглянути лістинг коду в прикладі. Пояснити вивід програми.  
2. Створити клас відповідно до завдання   
3. Розробити для класу конструктор за замовчуванням та декілька звичайних конструкторів. Реалізувати функції-члени відповідно до завдання   
4. Створити конструктор копіювання.  
5. Перевантажити операцію присвоєння.  
6. Створити деструктор для вивільнення динамічно виділеної пам’яті.  
7. Продемонструвати розроблені можливості класу завдяки створеному віконному застосуванню.   
8. Оформити звіт до лабораторної роботи.

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

Реалізувати такі функції члени:

Знаходження максимального значення.

Знаходження мінімального значення.

Знаходження середнього арифметичного значення масиву.

Сортування елементів масиву методом вибірки за спаданням.

Сортування елементів масиву методом бульбашки за зростанням.

Зміна розмірів масиву.

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

Додавання (почленне додавання елементів масиву)

Віднімання (почленне віднімання елементів масиву)

Множення на скаляр.

Введення масиву з StringGrid (>>)

Виведення масиву у StringGrid (<<)

Введення масиву з ListBox (>>)

Виведення масиву у ListBox (<<)

Виведення масиву у Memo (<<)

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

**Хід роботи**

Код програми:

**array.h**

#ifndef ARRAY\_H

#define ARRAY\_H

#include <QMainWindow>

#include <QObject>

#include <QWidget>

#include <QListWidget>

#include <QTableWidget>

#include <QPlainTextEdit>

*extern* int arSize;

*extern* int arScalar;

*extern* int arChangeSize;

*class* **Array**

{

int m\_iSizeOfArray;

double \*m\_dblArray;

*public*:

double m\_dblResult;

*//--------------------------------------------------------------------------------*

*//Конструктор*

**Array**();

*//--------------------------------------------------------------------------------*

*//Конструктор* *з* *1* *параметром*

**Array**(int Size);

*//--------------------------------------------------------------------------------*

*//Конструктор* *з* *2* *параметрами* *(занулення)*

**Array**(double \*elArray, int Size);

*//--------------------------------------------------------------------------------*

*//Конструктор* *копіювання*

**Array**(*const* Array &other);

*//--------------------------------------------------------------------------------*

*//Деструктор*

~**Array**();

*//--------------------------------------------------------------------------------*

*//Перегрузка* *оператора* *присвоєння*

Array& *operator*=(*const* Array &other);

*//--------------------------------------------------------------------------------*

*//Функції:*

double **dblMaxEl**();

double **dblMinEl**();

double **dblFindAver**();

*//Cпадання* *вибіркою*

void **m\_DecreasingSort**();

*//Зростання* *бульбшкою*

void **m\_IncreasingSort**();

void **m\_ChangeSize**(int iN);

*//--------------------------------------------------------------------------------*

*//Перегрузки:*

void *operator*\*(int iScalar);

void *operator*+();

void *operator*-();

double& *operator*[](int iN);

*//--------------------------------------------------------------------------------*

*friend* void *operator*<<(QListWidget \*widget,Array &el){

widget->clear();

*for*(int i=0; i<arSize;i++){

QString item=QString::number(el[i]);

widget->addItem(item);

}

}

*//--------------------------------------------------------------------------------*

*friend* void *operator*<<(QTableWidget \*widget, Array &el){

widget->setRowCount(1);

widget->setColumnCount(arSize);

*for*(int i=0;i<widget->rowCount();++i){

*for*(int j=0;j<widget->columnCount();++j){

QString item=QString::number(el[j]);

QTableWidgetItem \*isItem=*new* QTableWidgetItem(item);

widget->setItem(i,j,isItem);

}

}

}

*//--------------------------------------------------------------------------------*

*friend* void *operator*<<(QPlainTextEdit\*edit, Array &el){

QString str;

*for*(int i=0; i<arSize; ++i){

str+=QString::number(el[i]);

str+=" ";

}

edit->setPlainText(str);

}

*//--------------------------------------------------------------------------------*

void *operator*>> (QTableWidget \*widget);

void *operator*>> (QListWidget \*widget);

*//Геттери/Сеттери:*

double\* **dblGetArray**();

void **setArray**(double\*arr, int Size);

int **sGetSizeOfArray**();

};

#endif *//* *ARRAY\_H*

**mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include "array.h"

#include <QMainWindow>

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **MainWindow**; }

QT\_END\_NAMESPACE

*class* **MainWindow** : *public* QMainWindow

{

Q\_OBJECT

*public*:

**MainWindow**(QWidget \*parent = *nullptr*);

~***MainWindow***();

*private* slots:

void **on\_sizeOK\_clicked**();

void **on\_scalarOK\_clicked**();

void **on\_changingOK\_clicked**();

void **on\_elementsOK\_clicked**();

void **on\_pushButton\_2\_clicked**();

void **on\_pushButton\_clicked**();

void **on\_accessOK\_clicked**();

void **on\_btn\_Max\_clicked**();

void **on\_btn\_Min\_clicked**();

void **on\_btn\_Average\_clicked**();

void **on\_btn\_Decrease\_clicked**();

void **on\_btn\_Increase\_clicked**();

void **on\_btn\_Multiply\_clicked**();

void **on\_btn\_ChangeSize\_clicked**();

void **on\_plus\_clicked**();

void **on\_minusSubstract\_clicked**();

*private*:

Ui::MainWindow \*ui;

};

#endif *//* *MAINWINDOW\_H*

**array.cpp**

#include "array.h"

int arSize;

int arScalar;

int arChangeSize;

*//--------------------------------------------------------------------------------*

Array::**Array**()

{

m\_dblArray=0;

}

*//--------------------------------------------------------------------------------*

Array::**Array**(int Size){

m\_iSizeOfArray = Size;

m\_dblArray = *new* double [m\_iSizeOfArray];

*for*(int i=0; i<m\_iSizeOfArray;i++){

m\_dblArray[i]=0;

}

}

*//--------------------------------------------------------------------------------*

Array::**Array**(double \*elArray, int Size){

m\_iSizeOfArray = Size;

m\_dblArray = *new* double [m\_iSizeOfArray];

*for*(int i=0; i<m\_iSizeOfArray;i++){

m\_dblArray[i]=elArray[i];

}

}

*//--------------------------------------------------------------------------------*

*//Конструктор* *копіювання*

Array::**Array**(*const* Array &other){

*this*->m\_iSizeOfArray=other.m\_iSizeOfArray;

*this*->m\_dblArray=*new* double [other.m\_iSizeOfArray];

*for* (int i=0;i< m\_iSizeOfArray;i++){

*this*->m\_dblArray[i]=other.m\_dblArray[i];

}

}

*//--------------------------------------------------------------------------------*

*//Деструктор*

Array::~**Array**(){

*delete* [] m\_dblArray;

}

*//--------------------------------------------------------------------------------*

*//Перегрузка* *оператора* *присвоєння*

Array& Array::*operator*=(*const* Array &other){

*if*(*this*->m\_dblArray!= *nullptr*){

*delete* [] *this*->m\_dblArray;

}

*this*->m\_iSizeOfArray=other.m\_iSizeOfArray;

m\_dblArray=*new* double[other.m\_iSizeOfArray];

*for* (int i=0;i<m\_iSizeOfArray;i++){

*this*->m\_dblArray[i]=other.m\_dblArray[i];

}

*return* \**this*;

}

*//--------------------------------------------------------------------------------*

void Array:: *operator*\*(int iScalar){

*for*(int i=0; i<arSize;i++){

*this*->m\_dblArray[i]=*this*->m\_dblArray[i]\*iScalar;

}

}

*//--------------------------------------------------------------------------------*

void Array:: *operator*+(){

*this*->m\_dblResult=0;

*for*(int i=0; i<arSize; i++){

*this*->m\_dblResult=*this*->m\_dblResult+*this*->m\_dblArray[i];

}

}

*//--------------------------------------------------------------------------------*

void Array:: *operator*-(){

*this*->m\_dblResult=*this*->m\_dblArray[0];

*for*(int i=1; i< arSize; ++i){

*this*->m\_dblResult=*this*->m\_dblResult-*this*->m\_dblArray[i];

}

}

*//--------------------------------------------------------------------------------*

double& Array::*operator*[](int iN){

*return* m\_dblArray[iN];

}

*//--------------------------------------------------------------------------------*

void Array:: *operator*>>(QTableWidget\*widget){

*for*(int i=0; i<widget->rowCount();++i){

*for*(int j=0; j<widget->columnCount();++j){

*this*->m\_dblArray[j]=widget->item(i,j)->text().toDouble();

}

}

}

*//--------------------------------------------------------------------------------*

void Array:: *operator*>>(QListWidget\*widget){

*for*(int i=0; i<*this*->m\_iSizeOfArray;++i){

*this*->m\_dblArray[i]=widget->item(i)->text().toDouble();

}

}

*//--------------------------------------------------------------------------------*

double\* Array:: **dblGetArray**(){

double\*arr=*this*->m\_dblArray;

*return* arr;

}

*//--------------------------------------------------------------------------------*

int Array:: **sGetSizeOfArray**(){

*return* *this*->m\_iSizeOfArray;

}

*//--------------------------------------------------------------------------------*

void Array:: **setArray**(double \*arr, int Size){

*for*(int i=0; i<Size; ++i){

m\_dblArray[i]=arr[i];

}

}

*//--------------------------------------------------------------------------------*

double Array:: **dblMaxEl**(){

double max = *this*->m\_dblArray[0];

*for*(int i = 1; i < *this*->m\_iSizeOfArray; i++)

{

*if*(max < *this*->m\_dblArray[i])

{

max = *this*->m\_dblArray[i];

}

}

*return* max;

}

*//--------------------------------------------------------------------------------*

double Array:: **dblMinEl**(){

double min = *this*->m\_dblArray[0];

*for*(int i = 1; i < *this*->m\_iSizeOfArray; i++)

{

*if*(min > *this*->m\_dblArray[i])

{

min = *this*->m\_dblArray[i];

}

}

*return* min;

}

*//--------------------------------------------------------------------------------*

double Array:: **dblFindAver**(){

double avrg = 0 ;

*for*(int i = 0; i < *this*->m\_iSizeOfArray; ++i)

{

avrg += *this*->m\_dblArray[i];

}

*return* avrg/*this*->m\_iSizeOfArray;

}

*//--------------------------------------------------------------------------------*

void Array:: **m\_DecreasingSort**(){

*for*(int i = 0; i < (m\_iSizeOfArray - 1); i++)

{

*//min* *=* *i*

int min = i;

*//порівнюємо* *мінімальний* *елемент* *зі* *всіма* *елементами* *непосортованого* *масиву*

*for*(int j = (i + 1); j < m\_iSizeOfArray; j++)

{

*//якщо* *находимо* *менший* *елемент* *за* *зарішній* *він* *стає* *min*

*if*(m\_dblArray[j] > m\_dblArray[min])

{

min = j;

}

}

*//міняємо* *щоб* *поставити* *в* *коректне* *місце*

double temp = m\_dblArray[min];

m\_dblArray[min] = m\_dblArray[i];

m\_dblArray[i] = temp;

}

}

*//--------------------------------------------------------------------------------*

void Array:: **m\_IncreasingSort**(){

int swaps;

*//swap>0*

*do*

{

swaps = 0;

*//ітерація* *через* *весь* *масив*

*for*(int i = 0; i < (m\_iSizeOfArray-1); i++)

{

*//якщо* *array[i]* *>* *array[i+1]*

*if*(m\_dblArray[i] > m\_dblArray[i + 1])

{

*//міняємо* *місцями*

int temp = m\_dblArray[i];

m\_dblArray[i] = m\_dblArray[i + 1];

m\_dblArray[i + 1] = temp;

*//міняєм* *інкримент*

swaps++;

}

}

}

*while*(swaps != 0);

}

*//--------------------------------------------------------------------------------*

void Array:: **m\_ChangeSize**(int iN){

*for*(int i = 0 ; i < iN; ++i)

{

m\_dblArray[i] = *this*->m\_dblArray[i];

}

}

**main.cpp**

#include "mainwindow.h"

#include "array.h"

#include <QApplication>

int **main**(int argc, char \*argv[])

{

QApplication a(*argc*, argv);

MainWindow w;

w.show();

*return* a.exec();

}

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "array.h"

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

}

MainWindow::~***MainWindow***()

{

*delete* ui;

}

Array \*arrElements;

void MainWindow::**on\_sizeOK\_clicked**()

{

int s;

s = ui->enterSizeOfArray->text().toInt();

ui->tableInputeWidget->setRowCount(1);

ui->tableInputeWidget->setColumnCount(s);

arSize = s;

*for* (int i = 0 ; i < ui->tableInputeWidget->rowCount(); i++)

{

*for*(int j = 0; j < ui->tableInputeWidget->columnCount(); j++)

{

QTableWidgetItem \*itm = *new* QTableWidgetItem(tr("%1").arg(0));

ui->tableInputeWidget->setItem(i, j, itm);

}

}

arrElements = *new* Array(arSize);

ui->listInputWidget->clear();

*for*(int i = 0; i < arSize; ++i)

{

QListWidgetItem \*itm = *new* QListWidgetItem(tr("%1").arg(0));

ui->listInputWidget->addItem(itm);

}

}

void MainWindow::**on\_scalarOK\_clicked**()

{

arScalar = ui->enterScalar->text().toInt();

}

void MainWindow::**on\_changingOK\_clicked**()

{

arChangeSize = ui->enterChangingSize->text().toInt();

}

void MainWindow::**on\_elementsOK\_clicked**()

{

double num;

num = ui->enterElementsOfArray->text().toDouble();

*if*(ui->listInputWidget->currentItem()){

QListWidgetItem \*itm = ui->listInputWidget->currentItem();

itm->setText(QString::number(num));

}

}

void MainWindow::**on\_pushButton\_2\_clicked**()

{

Array \*res = *new* Array(arSize);

\*res >> ui->listInputWidget;

arrElements = *new* Array(res->dblGetArray(), arSize);

ui->tableInputeWidget << \*res;

}

void MainWindow::**on\_pushButton\_clicked**()

{

Array \*res = *new* Array(arSize);

\*res >> ui->tableInputeWidget;

arrElements = *new* Array(res->dblGetArray(), arSize);

ui->listInputWidget << \*res;

}

void MainWindow::**on\_accessOK\_clicked**()

{

Array \*access = *new* Array(arrElements->dblGetArray(), arSize);

int i = ui->accessLineEdit->text().toInt();

double \*array = access->dblGetArray();

double number = array[i];

int max = arSize;

*if*(i < max && i >= 0)

{

ui->acessLabel->setText(QString::number(number));

} *else*

{

ui->acessLabel->setText(QString("FALSE"));

}

}

void MainWindow::**on\_btn\_Max\_clicked**()

{

Array \*max\_Res = *new* Array(arrElements->dblGetArray(), arSize);

ui->MainResultLabel->setText(QString::number(max\_Res->dblMaxEl()));

}

void MainWindow::**on\_btn\_Min\_clicked**()

{

Array \*min\_Res = *new* Array(arrElements->dblGetArray(), arSize);

ui->MainResultLabel->setText(QString::number(min\_Res->dblMinEl()));

}

void MainWindow::**on\_btn\_Average\_clicked**()

{

Array \*average\_Res = *new* Array(arrElements->dblGetArray(), arSize);

ui->MainResultLabel->setText(QString::number(average\_Res->dblFindAver()));

}

void MainWindow::**on\_btn\_Decrease\_clicked**()

{

Array \*decreas\_Res = *new* Array(\*arrElements);

decreas\_Res->m\_DecreasingSort();

ui->tableOutput << \*decreas\_Res;

ui->listOutput << \*decreas\_Res;

ui->memoOutput << \*decreas\_Res;

}

void MainWindow::**on\_btn\_Increase\_clicked**()

{

Array \*icrease\_Res = *new* Array(\*arrElements);

icrease\_Res->m\_IncreasingSort();

ui->tableOutput << \*icrease\_Res;

ui->listOutput << \*icrease\_Res;

ui->memoOutput << \*icrease\_Res;

}

void MainWindow::**on\_btn\_Multiply\_clicked**()

{

Array \*scalar\_Res = *new* Array(arrElements->dblGetArray(), arSize);

\*scalar\_Res \* arScalar;

ui->tableOutput << \*scalar\_Res;

ui->listOutput << \*scalar\_Res;

ui->memoOutput << \*scalar\_Res;

}

void MainWindow::**on\_btn\_ChangeSize\_clicked**()

{

Array \*change\_Res = *new* Array(\*arrElements);

change\_Res->m\_ChangeSize(arChangeSize);

double \*arr = change\_Res->dblGetArray();

ui->tableOutput->setRowCount(1);

ui->tableOutput->setColumnCount(arChangeSize);

ui->tableInputeWidget->setRowCount(1);

ui->tableInputeWidget->setColumnCount(arChangeSize);

*for* (int i = 0 ; i < ui->tableOutput->rowCount(); i++)

{

*for*(int j = 0; j < arChangeSize; j++)

{

*if*(arSize > j)

{

QTableWidgetItem \*itm = *new* QTableWidgetItem(tr("%1").arg(arr[j]));

ui->tableOutput->setItem(i, j, itm);

ui->tableInputeWidget->setItem(i, j, itm);

} *else*

{

QTableWidgetItem \*itm = *new* QTableWidgetItem(tr("%1").arg(0));

ui->tableOutput->setItem(i, j, itm);

ui->tableInputeWidget->setItem(i, j, itm);

}

}

}

ui->listOutput->clear();

*for*(int i = 0; i < arChangeSize; ++i)

{

*if*(arSize > i)

{

QListWidgetItem \*itm = *new* QListWidgetItem(tr("%1").arg(arr[i]));

ui->listOutput->addItem(itm);

} *else*

{

QListWidgetItem \*itm = *new* QListWidgetItem(tr("%1").arg(0));

ui->listOutput->addItem(itm);

}

}

QString str;

}

void MainWindow::**on\_plus\_clicked**()

{

Array \*adding\_Res = *new* Array(arrElements->dblGetArray(), arSize);

+\*adding\_Res;

ui->MainResultLabel->setText(QString::number(adding\_Res->m\_dblResult));

}

void MainWindow::**on\_minusSubstract\_clicked**()

{

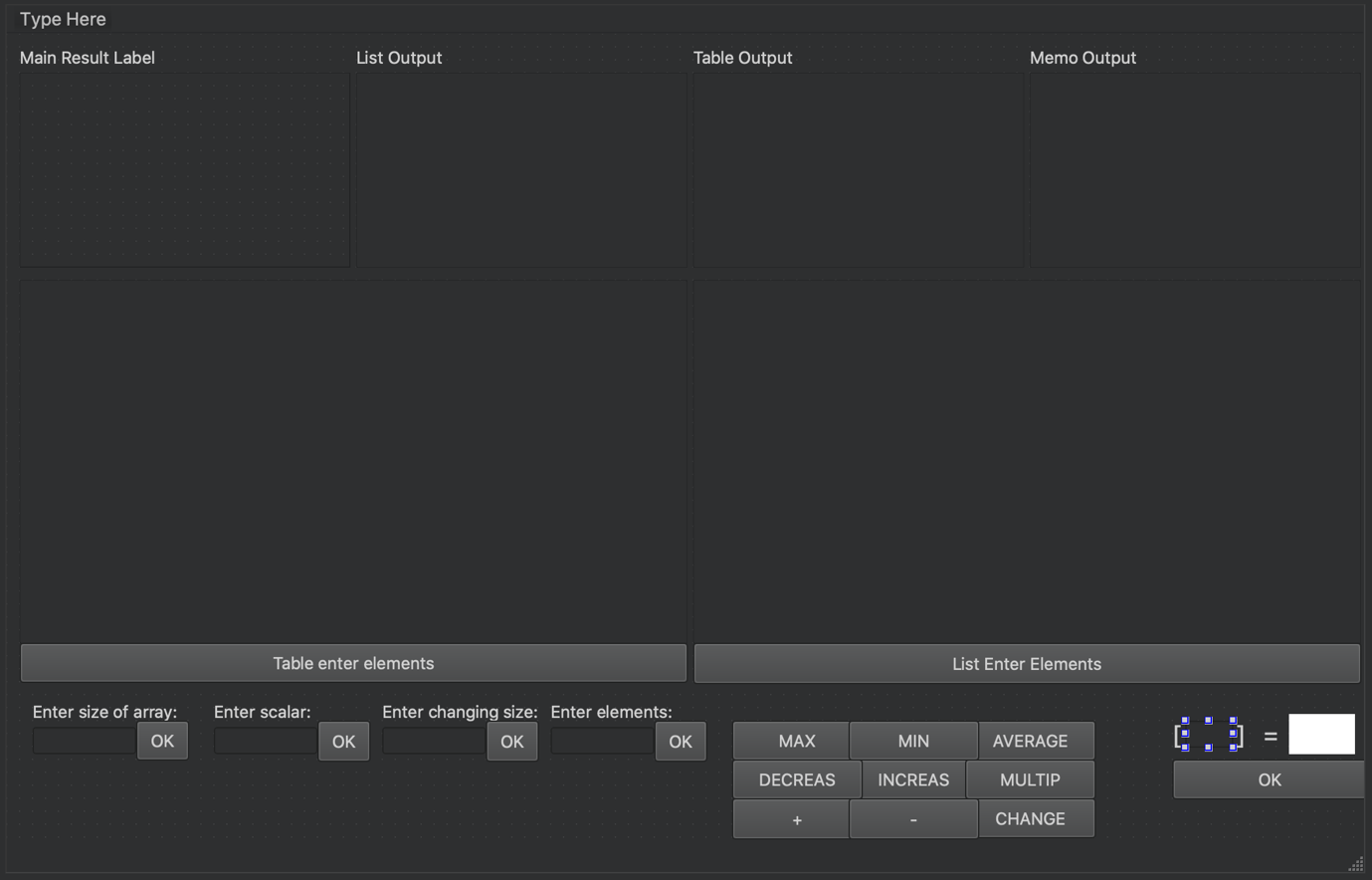
Array \*difference\_Res = *new* Array(arrElements->dblGetArray(), arSize);

-\*difference\_Res;

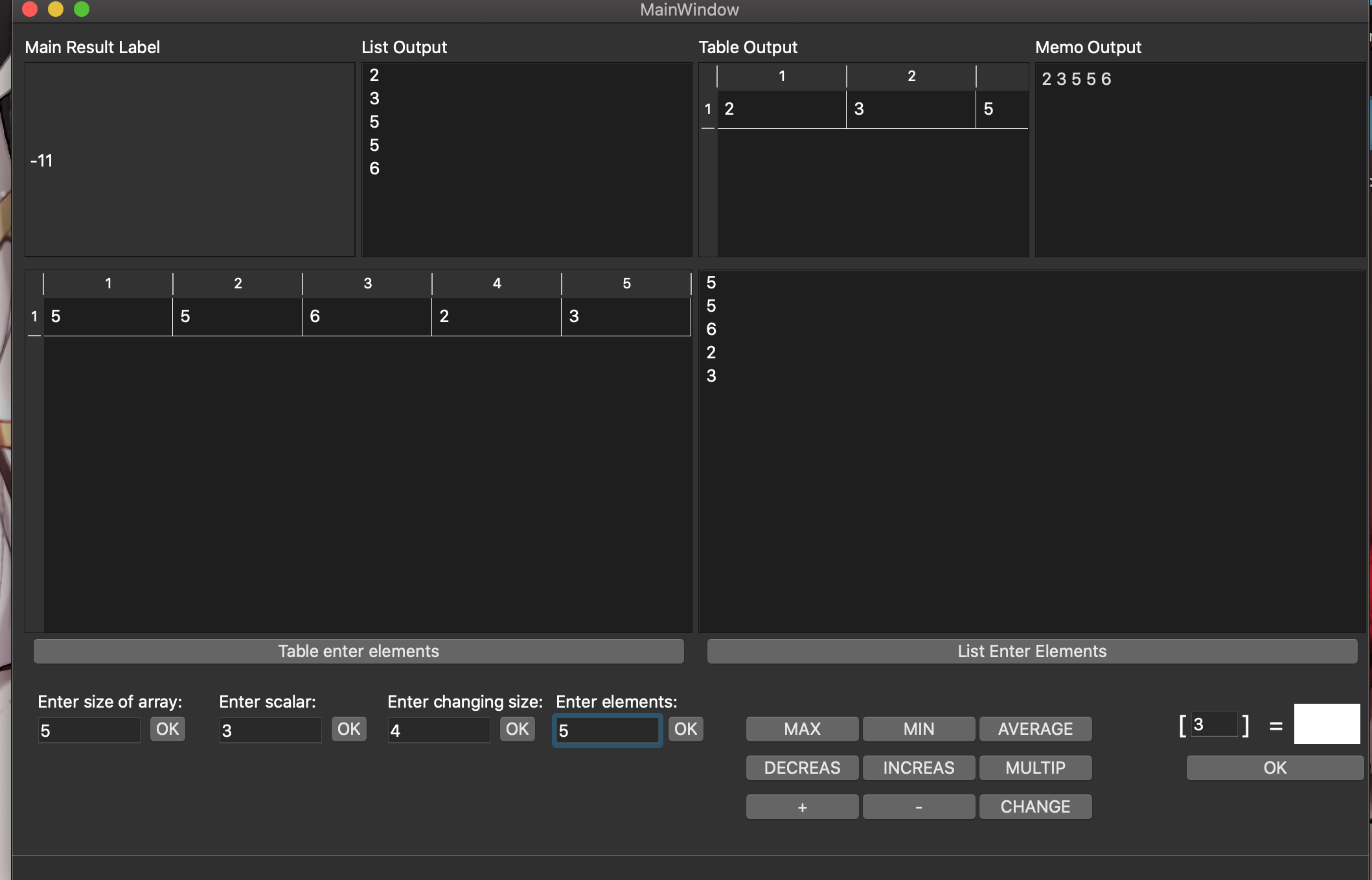
ui->MainResultLabel->setText(QString::number(difference\_Res->m\_dblResult));

}

Форма :



Результат виконання:



**Висновки**

Під час виконання лабораторної роботи я навчився працювати з динамічною пам’яттю, використовуючи засоби мови C++, розробив динамічний контейнер та графічний застосунок який надає можливість повноцінного використання створеного класу.