Міністерство освіти і науки України
Національний університет “Львівська політехніка”
Інститут комп’ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

Звіт
Про виконання лабораторних робіт №7

На тему:

« Робота з динамічною пам’яттю»

Лектор:
доцент каф. ПЗ
Коротєєва Т.О.
Виконав:
ст. гр. ПЗ-11
Морозов О. Р.
Прийняла:
доцент каф. ПЗ
Коротєєва Т.О.

« __ » _____ 2022 р.

∑ = _____ _____ .

Львів – 2022

Тема: Робота з динамічною пам’яттю
Мета: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

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.
Приклади роботи з об’єктами в різних видах пам’яті наведено в лістингу:
#include <iostream>

```
#include <string>

using namespace std;

class MyClass
{
private:
   static int m_allocated_objects_number;
   int m_object_id;

public:
   MyClass()
   {
      m_object_id = m_allocated_objects_number++;
      cout << "MyClass instance created with default constructor. Id = " <<
m_object_id << endl;
   }

   MyClass(const MyClass& another)
   {
      m_object_id = m_allocated_objects_number++;
      cout << "MyClass instance created as copy of instance (" <<
         another.m_object_id << "). Id = " << m_object_id << endl;
   }

   MyClass(int p)
   {
      m_object_id = m_allocated_objects_number++;
      cout << "MyClass instance created with int parameter " << p <<
         ". Id = " << m_object_id << endl;
   }

   ~MyClass()
   {
      cout << "MyClass instance destroyed. Id = " << m_object_id << endl;
   }
};

int MyClass::m_allocated_objects_number = 0;

MyClass g_my_object_1;
MyClass g_my_object_2(1);
static MyClass g_my_object_3(2);




int main(int argc, const char * argv[])
{
   cout << "Stage 1: " << endl;

   MyClass my_object_4(4);
```

```
    MyClass my_objects_1[3];

    {
        cout << "Stage 2: " << endl;

        MyClass my_object_5(5);
        MyClass my_objects_2[4];

        cout << "Stage 3: " << endl;
    }

    cout << "Stage 4: " << endl;
    MyClass* my_object_6 = new MyClass(6);
    MyClass* my_object_7;
    MyClass* my_objects_3 = new MyClass[3];
    MyClass  my_object_8 = my_object_4;
    MyClass  my_object_9 = *my_object_6;
    my_object_7 = new MyClass(3);

    {
        cout << "Stage 5: " << endl;
        MyClass  my_object_10 = my_objects_3[2];
        MyClass* my_object_11 = new MyClass(my_object_9);
        MyClass* my_object_12 = new MyClass(*my_object_7);

        cout << "Stage 6: " << endl;
        delete my_object_11;
    }

    cout << "Stage 7: " << endl;
    delete my_object_6;
    delete [] my_objects_3;

    cout << "Stage 8: " << endl;

    return 0;
}
```

В результаті виконання програми матимемо наступний вивід на консоль:

```
MyClass instance created with default constructor. Id = 0
MyClass instance created with int parameter 1. Id = 1
MyClass instance created with int parameter 2. Id = 2
Stage 1:
MyClass instance created with int parameter 4. Id = 3
MyClass instance created with default constructor. Id = 4
MyClass instance created with default constructor. Id = 5
MyClass instance created with default constructor. Id = 6
Stage 2:
MyClass instance created with int parameter 5. Id = 7
MyClass instance created with default constructor. Id = 8
MyClass instance created with default constructor. Id = 9
MyClass instance created with default constructor. Id = 10
```

MyClass instance created with default constructor. Id = 11
Stage 3:
MyClass instance destroyed. Id = 11
MyClass instance destroyed. Id = 10
MyClass instance destroyed. Id = 9
MyClass instance destroyed. Id = 8
MyClass instance destroyed. Id = 7
Stage 4:
MyClass instance created with int parameter 6. Id = 12
MyClass instance created with default constructor. Id = 13
MyClass instance created with default constructor. Id = 14
MyClass instance created with default constructor. Id = 15
MyClass instance created as copy of instance (3). Id = 16
MyClass instance created as copy of instance (12). Id = 17
MyClass instance created with int parameter 3. Id = 18
Stage 5:
MyClass instance created as copy of instance (15). Id = 19
MyClass instance created as copy of instance (17). Id = 20
MyClass instance created as copy of instance (18). Id = 21
Stage 6:
MyClass instance destroyed. Id = 20
MyClass instance destroyed. Id = 19
Stage 7:
MyClass instance destroyed. Id = 12
MyClass instance destroyed. Id = 15
MyClass instance destroyed. Id = 14
MyClass instance destroyed. Id = 13
Stage 8:
MyClass instance destroyed. Id = 17
MyClass instance destroyed. Id = 16
MyClass instance destroyed. Id = 6
MyClass instance destroyed. Id = 5
MyClass instance destroyed. Id = 4
MyClass instance destroyed. Id = 3
MyClass instance destroyed. Id = 2
MyClass instance destroyed. Id = 1
MyClass instance destroyed. Id = 0

Варто звернути увагу на те, що при роботі із об'єктами my_object_7 та my_object_12 були допущені помилки, оскільки об'єкти не були видалені після використання, про що свідчить вивід програми.

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).

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

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.

Завдання згідно варіанту

Варіант 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.

Код програми
array.h

```
#ifndef ARRAY_H
#define ARRAY_H

#ifndef SET_H
#define SET_H
#include <iostream>

#include <QTableWidget>
#include <QListWidget>
#include <QTextEdit>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QTextEdit>
#include <QMessageBox>
#endif // SET_H

class Array
{
public:
 //-------------------------------------------------------
 double *array;
 int size;
 //-------------------------------------------------------
 Array(){
  array = new double[0];
  size = 0;
 }
 //-------------------------------------------------------
 Array(const Array &ar){
  array = ar.array;
  size = ar.size;
 }
 //-------------------------------------------------------
 Array(int n){
  array = new double[n];
  size = n;
 }
 //-------------------------------------------------------
```

```
 Array(double *ar, int n){
  array = ar;
  size = n;
 }
 //-------------------------------------------------------
 ~Array(){clearArray();}
 //-------------------------------------------------------
 void changeSize(int n);
 void addElement(double data, int n);
 void clearArray();
 //-------------------------------------------------------
 double getMaxValue();
 double getMinValue();
 double averageValue();
 double getElementValue(int num);
 //-------------------------------------------------------
 void selectionSortToLower();
 void bubbleSortToLarger();
 //-------------------------------------------------------
 friend void operator>>(QListWidget *in, Array &ar);
 friend void operator<<(QListWidget *out, Array &ar);
 friend void operator<<(QLabel *out, Array &ar);
 friend void operator>>(QTableWidget *in, Array &ar);
 friend void operator<<(QTableWidget *out, Array &ar);
 //-------------------------------------------------------
 Array operator+(Array &ar);
 Array operator-(Array &ar);
 //-------------------------------------------------------
 friend void operator*(Array &, double n);
 //-------------------------------------------------------
};

#endif // ARRAY_H
```

complex.cpp

```
#include "array.h"


void Array::addElement(double data, int n) {
 if(n >= size || n < 0){
     std::cout << "Error: overwrite " << std::endl;
  } else {
     array[n] = data;
  }
}

void Array::clearArray() {
 delete[] array;
```

```
 array = nullptr;
 size = 0;
}

void Array::changeSize(int n) {
 double *ar = new double[n];
 for(int i = 0; i < size; i++){
   ar[i] = array[i];
 }
 array = ar;
 size = n;
}

double Array::getMaxValue() {
 double max = array[0];
 for(int i = 0; i < size; i++){
    max = (array[i] > max) ? array[i] : max;
  }
 return max;
}

double Array::getMinValue() {
 double max = array[0];
 for(int i = 0; i < size; i++){
    max = (array[i] < max) ? array[i] : max;
  }
 return max;
}

double Array::averageValue() {
 double ave = 0;
 for(int i = 0; i < size; i++){
    ave += array[0];
  }
 return ave/size;
}

double Array::getElementValue(int num) {
 return array[num];
}

void Array::selectionSortToLower() {
 if(!array){
    std::cout << "Error: array is NULL " << std::endl;
    return;
 }

 for(int step = 0; step < size - 1; step++){
    int min_idx = step;
```

```
    for(int i = step + 1; i < size-step; i++){
       min_idx = (array[i] > array[min_idx]) ? i : min_idx;
    }//for(i)
    if(min_idx != step){
      double temp = array[step];
      array[step] = array[min_idx];
      array[min_idx] = temp;
    }//if(swap)
 }//for(step)
 std::cout << "Selection sort done" << std::endl;
}

void Array::bubbleSortToLarger() {

 if(!array){
    std::cout << "Error: array is NULL " << std::endl;
    return;
 }

 for(int step = 0; step < size; step++){
    bool swapped = false;
    for(int i = 0; i < size-step-1; i++){
       if(array[i] > array[i+1]){
          double temp = array[i];
          array[i] = array[i+1];
          array[i+1] = temp;
          swapped  = true;
        }//if
      }//for(i)
    if(!swapped)
      break;
  }//for(step)

 std::cout << "Bubble sort done" << std::endl;
}

void operator>>(QListWidget *in, Array &ar) {
 ar.clearArray();

 ar.changeSize(in->count());

 for(int i = 0; i < in->count(); i++) {
    in->setCurrentRow(i);
    ar.addElement(in->currentItem()->text().toDouble(), i);
  }
}

void operator<<(QListWidget *out, Array &ar) {
 out->clear();
```

```
 for(int i = 0; i < ar.size; i++){
    out->addItem(QString::number(ar.array[i]));
  }
}

void operator>>(QTableWidget *in, Array &ar) {
 ar.clearArray();

 ar.changeSize(in->rowCount());

 QString str;
 for(int i = 0; i < in->rowCount(); i++) {
    str = in->item(i, 0)->text();
    if(str != ""){
      ar.addElement(str.toDouble(), i);
      }
  }
}

void operator<<(QTableWidget *out, Array &ar) {
 out->clearContents();

 out->setRowCount(ar.size);
 out->setColumnCount(1);

 for(int i = 0; i < ar.size; i++) {
    QTableWidgetItem *item = new QTableWidgetItem();
    item->setText(QString::number(ar.array[i]));
    out->setItem(i, 0, item);
  }
}

void operator<<(QLabel *out, Array &ar) {
 out->clear();

 QString str = "";

 for(int i = 0; i < ar.size; i++) {
    str += QString::number(ar.array[i]) += " ";
  }
 out->setText(str);
}

Array Array::operator+(Array &ar){
 if(size > ar.size){ar.changeSize(size);}
 else if(size < ar.size) {changeSize(ar.size);}

 double *res_arr = new double[size];
```

```
 for(int i = 0; i < size; i++){
    res_arr[i] = array[i] + ar.array[i];
  }
 return Array(res_arr, size);
}

Array Array::operator-(Array &ar){
 if(size > ar.size){ar.changeSize(size);}
 else if(size < ar.size) {changeSize(ar.size);}

 double *res_arr = new double[size];

 for(int i = 0; i < size; i++){
    res_arr[i] = array[i] - ar.array[i];
  }
 return Array(res_arr, size);
}

void operator*(Array &ar, double n){
 for(int i = 0; i < ar.size; i++){
   ar.array[i] = ar.array[i] * n;
  }
}
```

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();

 int setArray();

private slots:
```

```
 void on_pushButton_input_table_widget_clicked();

 void on_pushButton_output_table_widget_clicked();

 void on_pushButton_iput_list_widget_clicked();

 void on_pushButton_output_list_widget_clicked();

 void on_pushButton_min_clicked();

 void on_pushButton_max_clicked();

 void on_pushButton_ave_clicked();

 void on_pushButton_sort_to_higher_clicked();

 void on_pushButton_sort_lower_clicked();

 void on_pushButton_n_element_2_clicked();

 void on_pushButton_mult_n_clicked();

 void on_pushButton_n_element_clicked();

 void on_pushButton_plus_clicked();

 void on_pushButton_minus_clicked();

private:
 Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

Array ar1(10), ar2(10), ar3(10);

MainWindow::MainWindow(QWidget *parent)
 : QMainWindow(parent)
 , ui(new Ui::MainWindow)
{
 ui->setupUi(this);
}

MainWindow::~MainWindow()
{
```

```
 delete ui;
 ar1.clearArray();
 ar2.clearArray();
 ar3.clearArray();
}

int MainWindow::setArray() {
 QMessageBox msgBox;
 msgBox.setText("Chose what array should use");
 msgBox.setWindowIcon(QIcon(":/icons/icons/info.png"));
 QAbstractButton* pButtonAr1 = msgBox.addButton(tr("Array 1"),
QMessageBox::YesRole);
 QAbstractButton* pButtonAr2 = msgBox.addButton(tr("Array 2"),
QMessageBox::AcceptRole);
 QAbstractButton* pButtonAr3 = msgBox.addButton(tr("Array 3"),
QMessageBox::NoRole);
 msgBox.exec();

 if (msgBox.clickedButton() == pButtonAr1) {
     return 0;
 } else if (msgBox.clickedButton() == pButtonAr2) {
     return 1;
  } else if (msgBox.clickedButton() == pButtonAr3) {
     return 2;
 }
 return 0;
}

void MainWindow::on_pushButton_input_table_widget_clicked() {
 int check = setArray();
 switch(check){
   case(0):{
      ui->tableWidget >> ar1;
      ui->label_array << ar1;
      break;
    }
   case(1):{
      ui->tableWidget >> ar2;
      ui->label_array << ar2;
      break;
    }
   case(2):{
      ui->tableWidget >> ar3;
      ui->label_array << ar3;
      break;
    }
  }
}
```

```
void MainWindow::on_pushButton_output_table_widget_clicked() {
 int check = setArray();
 switch(check){
  case(0):{
     ui->tableWidget << ar1;
     break;
    }
  case(1):{
     ui->tableWidget << ar2;
     break;
    }
  case(2):{
     ui->tableWidget << ar3;
     break;
    }
  }
}


void MainWindow::on_pushButton_iput_list_widget_clicked() {
 int check = setArray();
 switch(check){
  case(0):{
     ui->listWidget >> ar1;
     ui->label_array << ar1;
     break;
    }
  case(1):{
     ui->listWidget >> ar2;
     ui->label_array << ar2;
     break;
    }
  case(2):{
     ui->listWidget >> ar3;
     ui->label_array << ar3;
     break;
    }
  }
}


void MainWindow::on_pushButton_output_list_widget_clicked() {
 int check = setArray();
 switch(check){
  case(0):{
     ui->listWidget << ar1;
     break;
    }
```

```
  case(1):{
     ui->listWidget << ar2;
     break;
    }
  case(2):{
     ui->listWidget << ar3;
     break;
    }
  }
}


void MainWindow::on_pushButton_min_clicked() {
 int check = setArray();
 switch(check){
  case(0):{
     ui->label_Result->setText(QString::number(ar1.getMinValue()));
     break;
    }
  case(1):{
     ui->label_Result->setText(QString::number(ar2.getMinValue()));
     break;
    }
  case(2):{
     ui->label_Result->setText(QString::number(ar3.getMinValue()));
     break;
    }
  }
}


void MainWindow::on_pushButton_max_clicked() {
 int check = setArray();
 switch(check){
  case(0):{
     ui->label_Result->setText(QString::number(ar1.getMaxValue()));
     break;
    }
  case(1):{
     ui->label_Result->setText(QString::number(ar2.getMaxValue()));
     break;
    }
  case(2):{
     ui->label_Result->setText(QString::number(ar3.getMaxValue()));
     break;
    }
  }
}
```

```
void MainWindow::on_pushButton_ave_clicked() {
 int check = setArray();
 switch(check){
  case(0):{
     ui->label_Result->setText(QString::number(ar1.averageValue()));
     break;
   }
  case(1):{
     ui->label_Result->setText(QString::number(ar2.averageValue()));
     break;
   }
  case(2):{
     ui->label_Result->setText(QString::number(ar3.averageValue()));
     break;
   }
  }
}


void MainWindow::on_pushButton_sort_to_higher_clicked() {
 int check = setArray();
 switch(check){
  case(0):{
     ar1.bubbleSortToLarger();
     ui->label_array << ar1;
     break;
   }
  case(1):{
     ar2.bubbleSortToLarger();
     ui->label_array << ar2;
     break;
   }
  case(2):{
     ar3.bubbleSortToLarger();
     ui->label_array << ar3;
     break;
   }
  }
}


void MainWindow::on_pushButton_sort_lower_clicked() {
 int check = setArray();
 switch(check){
  case(0):{
     ar1.selectionSortToLower();
     ui->label_array << ar1;
     break;
```

```
    }
   case(1):{
      ar2.selectionSortToLower();
      ui->label_array << ar2;
      break;
    }
   case(2):{
      ar3.selectionSortToLower();
      ui->label_array << ar3;
      break;
    }
  }
}


void MainWindow::on_pushButton_n_element_2_clicked(){
 int check = setArray();
 switch(check){
  case(0):{
     ar1.changeSize(ui->label_N->text().toInt());
     break;
   }
  case(1):{
     ar2.changeSize(ui->label_N->text().toInt());
     break;
   }
  case(2):{
     ar3.changeSize(ui->label_N->text().toInt());
     break;
   }
  }
}


void MainWindow::on_pushButton_mult_n_clicked() {
 int check = setArray();
 switch(check){
  case(0):{
     double num = ui->lineEdit->text().toDouble();
     ar1*num;
     ui->label_array << ar1;
     ui->label_Result->setText(QString::number(num));
     break;
   }
  case(1):{
     double num = ui->lineEdit->text().toDouble();
     ar2*num;
     ui->label_array << ar2;
     ui->label_Result->setText(QString::number(num));
```

```
      break;
    }
   case(2):{
      double num = ui->lineEdit->text().toDouble();
      ar3*num;
      ui->label_array << ar3;
      ui->label_Result->setText(QString::number(num));
      break;
    }
  }
}


void MainWindow::on_pushButton_n_element_clicked() {
 int check = setArray();
 switch(check){
  case(0):{

ui->label_Result->setText(QString::number(ar1.array[ui->label_N->text().toInt()])
);
      break;
    }
   case(1):{

ui->label_Result->setText(QString::number(ar2.array[ui->label_N->text().toInt()])
);
      break;
    }
   case(2):{

ui->label_Result->setText(QString::number(ar3.array[ui->label_N->text().toInt()])
);
      break;
    }
  }
}


void MainWindow::on_pushButton_plus_clicked() {
 ar3 = ar1 + ar2;
 ui->label_array << ar3;
}


void MainWindow::on_pushButton_minus_clicked() {
 ar3 = ar1 - ar2;
 ui->label_array << ar3;
}
```

main.cpp

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
 QApplication a(argc, argv);
 MainWindow w;
 w.setWindowTitle("Array editor");
 w.show();
 return a.exec();
}
```

Вигляд програми

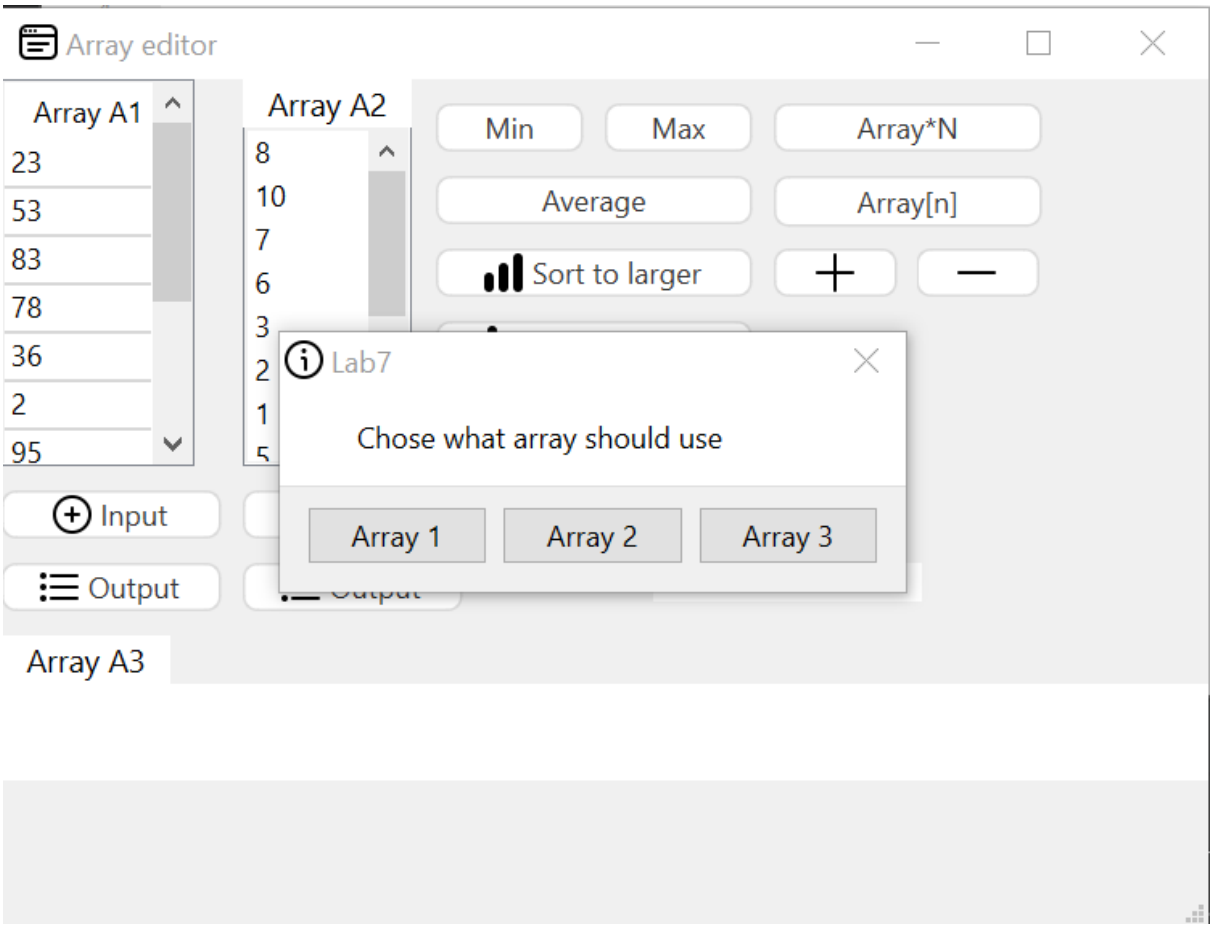

рис 1. Вибір в який масив зберегти дані

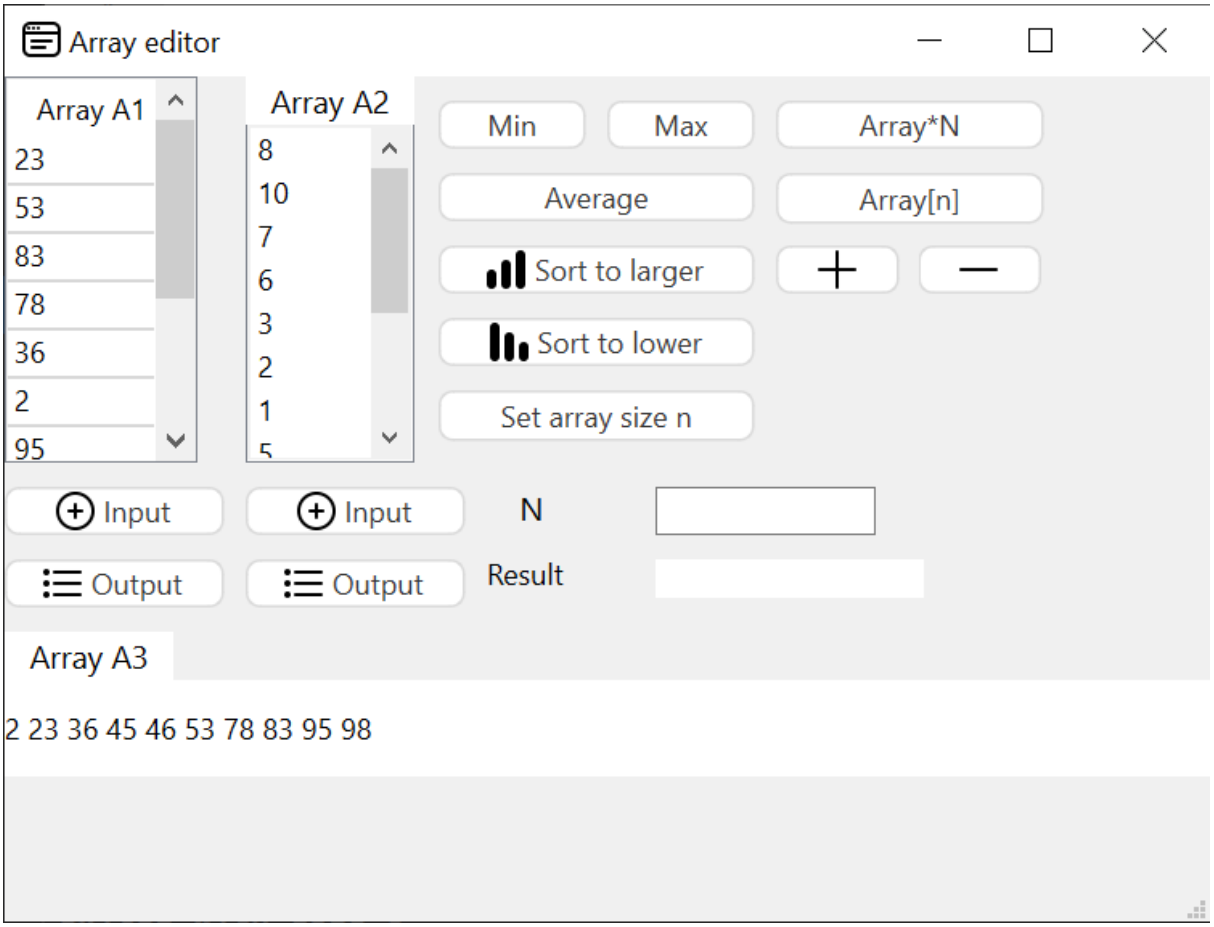

рис 2. Вивід посортованого масиву в QLabel знизу програми

ВИСНОВКИ

Виконавши лабораторну роботу №7 я навчився виділяти місце під об’єкти динамічно. Навчився створювати та використовувати конструктор копіювання, перевантажувати оператор присвоєння. Ознайомився з принципами створення та функціонування деструкторів.