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

Звіт
Про виконання лабораторних робіт №10

На тему:

«Шаблони класів»

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: Шаблони класів.
Мета: Навчитись створювати шаблони класу та екземпляри шаблонів.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Шаблон функції або класу є заготовкою яку можна використовувати для різних типів даних.

У найпростішому випадку загальна форма оголошення шаблону класу без аргументів має такий вигляд:

```
template <class T>
class ClassName
{
        //тіло шаблону класу
}
```

де

template - ключове слово, що відзначає створення шаблону;

T – узагальнене ім'я типу, що використовується методами класу;

ClassName – ім'я класу, що містить методи оперування узагальненим типом класу.

Коли в шаблоні потрібно використати аргументи, загальна форма набуває вигляду

```
template <class T, type arg>
class ClassName
{
        //тіло шаблону класу
}
```

де

template - ключове слово, що відзначає створення шаблону;

T – узагальнене ім'я типу, що використовується методами класу;

type - конкретний тип аргументу використаного в шаблоні класу;

arg - ім'я аргументу використаного в шаблоні класу

ClassName – ім'я класу, що містить методи оперування узагальненим типом класу.

Оголошення шаблону класу дає такі переваги:

- уникнення повторюваності написання програмного коду для різних типів даних. Програмний код (методи, функції) пишеться для деякого узагальненого типу T;
- зменшення текстової частини програмного коду, і, як наслідок, підвищення читабельності програм;
- забезпечення зручного механізму передачі аргументів у шаблон класу з метою їх обробки методами класу.

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

Створити шаблон класу та продемонструвати його роботу за індивідуальним варіантом. Оформити звіт до лабораторної роботи. Звіт має містити варіант завдання, код розробленої програми, результати роботи програми (скріншоти), висновок.

Завдання згідно варіанту

Варіант 2

Створити шаблон класу Array, який містить однотипні елементи. Шаблон класу повинен давати можливість вивести всі елементи на екран, відсортувати всі елементи в порядку зростання та спадання, а також мінімальний з елементів. Продемонструвати функціонал шаблону на створеному користувацькому типі String – символьна стрічка. Для порівняння стрічок використовувати алфавітний порядок.

Код програми

array.h

```
#ifndef ARRAY_H
#define ARRAY_H

#include <cassert>
#include <QLabel>
#include <QString>
#include <iostream>

template <class Type>
class Array {
private:
 int m_arrayLength;
 Type *m_arrayData;
public:

 Array() {
   m_arrayLength = 0;
   m_arrayData = nullptr;
 }
```

```
        ~Array() {
          delete[] m_arrayData;
        }

        void ClearArray() {
          delete[] m_arrayData;
          m_arrayData = nullptr;
          m_arrayLength = 0;
        }

        void addElement(Type data) {
          ResizeArray(m_arrayLength+1);
          m_arrayData[m_arrayLength-1] = data;
        }

        void ResizeArray(int new_length) {
          Type *new_arr = new Type[new_length];

          if(m_arrayData) {
          m_arrayLength = (new_length <= m_arrayLength) ? new_length:
m_arrayLength;
          for(int i = 0; i < m_arrayLength; i++){
              new_arr[i] = m_arrayData[i];
            }
          }
          m_arrayData = new_arr;
          m_arrayLength = new_length;
        }

        void ShowArray(QLabel *lbl) {
          if(m_arrayLength == 0){
              std::cout << "Array has not have element." << std::endl;
              return;
            }
            QString str;
              for(int current = 0; current < m_arrayLength; current++) {
                str += QString::number(m_arrayData[current]) += " ";
              }
            lbl->setText(str);
        }

        void SortToMin() {
          if(m_arrayLength == 0){
              std::cout << "Array has not have element." << std::endl;
              return;
            }

              for(int step = 0, j; step < m_arrayLength; step++) {
                  Type key = m_arrayData[step];
                  for(j = step - 1; j >= 0 && key > m_arrayData[j]; j--){
                      m_arrayData[j+1] = m_arrayData[j];
```

```
                }
                m_arrayData[j+1] = key;
            }
        }

        void SortToMax() {
          if(m_arrayLength == 0){
              std::cout << "Array has not have element." << std::endl;
              return;
            }
              for(int step = 0, j; step < m_arrayLength; step++) {
                  Type key = m_arrayData[step];
                  for(j = step - 1; j >= 0 && key < m_arrayData[j]; j--){
                      m_arrayData[j+1] = m_arrayData[j];
                  }
                m_arrayData[j+1] = key;
              }
        }

        Type getMin() {
          if(m_arrayLength == 0){ throw ("Incorrect index."); }

          Type* minNum = new Type;
          *minNum = m_arrayData[0];

          for(int current; current < m_arrayLength; current++) {
              (*minNum) = ((*minNum) > m_arrayData[current]) ?
m_arrayData[current] : (*minNum);
            }
          return *minNum;
        }

        Array& operator=(Array* a) {
          this->ClearArray();
          this->m_arrayData = a->m_arrayData;
          this->m_arrayLength = a->m_arrayLength;
          return this;
        }

        Array operator=(Type data) {
          this->addElement(data);
          return *this;
        }
       };

       #endif // TEMPLATEARRAY_H
```

mystring.h

```
       #ifndef MYSTRING_H
       #define MYSTRING_H
```

```
#include "array.h"
#include <cstring>

class MyString {
private:
 char* m_string;
 int m_size;
public:

 MyString() {
   m_string = nullptr;
   m_size = 0;
 }

 MyString(const MyString &a) {
   m_size = a.m_size;
   qDebug() << m_size << "class array size";
   m_string = new char[m_size];
   for(int i = 0; i < m_size; i++){
       m_string[i] = a.m_string[i];
     }
 }

 ~MyString(){
   delete[] m_string;
 }

 char* getString() const{
   return m_string;
 }

 int getSize(){
   return m_size;
 }

 friend bool operator>(const MyString a, const MyString b){
   if(strcmp(b.m_string, a.m_string))
     return true;
   else
     return false;
 }

 friend bool operator<(const MyString a, const MyString b){
   if(strcmp(a.m_string, b.m_string))
     return true;
   else
     return false;
 }

 MyString& operator=(const MyString &str) {
```

```
    if(m_string){
        delete[] m_string;
        m_string = nullptr;
      }

    m_size = str.m_size;
    m_string = new char[m_size];

    for(int i = 0; i < m_size; i++){
        m_string[i] = str.m_string[i];
      }

    return *this;
  }

  MyString& operator=(const QString &qstr){
    if(m_string){
        delete[] m_string;
        m_string = nullptr;
      }

    qDebug() << qstr.size() << "size from qstr";

    m_size = qstr.size();
    m_string = new char[m_size];

    for(int i = 0; i < m_size; i++){
        m_string[i] = qstr[i].toLatin1();
      }

    return *this;
  }

  QString toQString(){
    qDebug() << m_size << "size to qstr";
    qDebug() << m_string << "m_string";
    return QString::fromLatin1(m_string, m_size);
  }

};

template<>
class Array<MyString> {
private:
  int m_length;
  MyString *m_data;
public:

  Array() {
    m_length = 0;
    m_data = nullptr;
```

```
}

~Array() {
  delete[] m_data;
}

void ClearArray() {
  delete[] m_data;
  m_data = nullptr;
  m_length = 0;
}

void addElement(QString data) {
  resize(m_length+1);
  m_data[m_length-1] = data;
}

void resize(int newlength){
  MyString *newarray = new MyString[newlength];

  if(newlength < m_length) m_length = newlength;

  for(int i = 0; i < m_length; i++)
    newarray[i] = m_data[i];

  m_data = newarray;
  m_length = newlength;
}

void ShowArray(QLabel *lbl) {
  QString str;
    for(int current = 0; current < m_length; current++) {
      str += m_data[current].toQString() + '\n';
    }
    lbl->setText(str);
}

void SortToMin() {
      for(int step = 0, j; step < m_length; step++) {
          MyString key = m_data[step];
          for(j = step - 1; j >= 0 && key < m_data[j]; j--){
              m_data[j+1] = m_data[j];
            }
          m_data[j+1] = key;
        }
}

void SortToMax() {
      for(int step = 0, j; step < m_length; step++) {
          MyString key = m_data[step];
          for(j = step - 1; j >= 0 && key > m_data[j]; j--){
```

```
                m_data[j+1] = m_data[j];
              }
            m_data[j+1] = key;
          }
 }

 MyString getMin() {
   MyString minNum = m_data[0];

   for(int current = 1; current < m_length; current++) {
       minNum = (minNum > m_data[current]) ? m_data[current] : minNum;
     }
   return minNum;
 }

 MyString operator[](int index){
   return m_data[index];
 }
};

#endif // MYSTRING_H
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QLabel>
#include <QLineEdit>
#include <QValidator>
#include "mystring.h"

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_pushButton_input_clicked();

 void on_pushButton_output_clicked();
```

```
 void on_pushButton_sort_to_larger_clicked();

 void on_pushButton_sort_to_lower_clicked();

 void on_pushButton_min_clicked();

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

mainwindow.cpp

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

Array<MyString> stringArray;

MainWindow::MainWindow(QWidget *parent)
 : QMainWindow(parent)
 , ui(new Ui::MainWindow)
 ,m_stringValidator(QRegularExpression(".{512}"))
{
 ui->setupUi(this);
 ui->lineEdit_input->setValidator(&m_stringValidator);
 ui->lineEdit_input->setPlaceholderText("Enter string here");
}

MainWindow::~MainWindow() {
 delete ui;
}

void MainWindow::on_pushButton_input_clicked()
{
 stringArray.addElement(ui->lineEdit_input->text());
 stringArray.ShowArray(ui->label_output);
}


void MainWindow::on_pushButton_output_clicked()
{
 stringArray.ShowArray(ui->label_output);
}


void MainWindow::on_pushButton_sort_to_larger_clicked()
{
   stringArray.SortToMax();
   stringArray.ShowArray(ui->label_output);
}
```

```
void MainWindow::on_pushButton_sort_to_lower_clicked()
{
 stringArray.SortToMin();
 stringArray.ShowArray(ui->label_output);
}


void MainWindow::on_pushButton_min_clicked()
{
 MyString minstr = stringArray.getMin();
 ui->label_output->setText(minstr.toQString());
}
```

window2.h

```
#ifndef WINDOW2_H
#define WINDOW2_H

#include <QWidget>

namespace Ui {
 class window2;
}

class window2 : public QWidget
{
 Q_OBJECT

public:
 explicit window2(QWidget *parent = nullptr);
 ~window2();

private slots:
 void on_pushButton_input_clicked();

 void on_pushButton_sort_to_larger_clicked();

 void on_pushButton_sort_to_lower_clicked();

 void on_pushButton_min_clicked();

 void on_pushButton_output_clicked();

private:
 Ui::window2 *ui;
};

#endif // WINDOW2_H
```

window2.cpp

```
#include "window2.h"
#include "ui_window2.h"
#include "array.h"

Array<int> intArray;

window2::window2(QWidget *parent) :
 QWidget(parent),
 ui(new Ui::window2)
{
 ui->setupUi(this);
 ui->lineEdit_input->setPlaceholderText("Enter array here");
}

window2::~window2()
{
 delete ui;
}


void window2::on_pushButton_input_clicked()
{
 intArray.addElement(ui->lineEdit_input->text().toInt());
 intArray.ShowArray(ui->label_output);
}


void window2::on_pushButton_sort_to_larger_clicked()
{
 intArray.SortToMax();
 intArray.ShowArray(ui->label_output);
}


void window2::on_pushButton_sort_to_lower_clicked()
{
 intArray.SortToMin();
 intArray.ShowArray(ui->label_output);
}


void window2::on_pushButton_min_clicked()
{
 ui->label_output->setText(QString::number(intArray.getMin()));
}


void window2::on_pushButton_output_clicked()
```

```
{
 intArray.ShowArray(ui->label_output);
}
```

main.cpp

```
#include "mainwindow.h"
#include "window2.h"

#include <QApplication>

int main(int argc, char *argv[])
{
 QApplication a(argc, argv);

 MainWindow w;
 w.show();
 w.setWindowTitle("Template Array<String>");
 w.setWindowIcon(QIcon(":/icon//baseline_data_array_black_48dp.png"));

 window2 w2;
 w2.show();
 w2.setWindowTitle("Template Array<Int>");
 w2.setWindowIcon(QIcon(":/icon//baseline_data_array_black_48dp.png"));

 return a.exec();
}
```

Вигляд програми

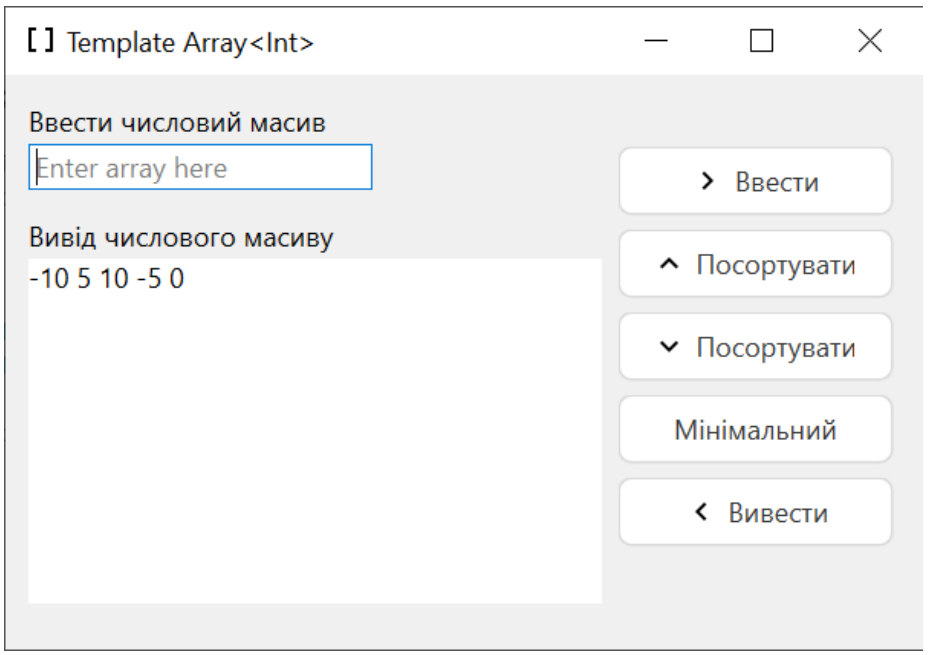

Рис 1. приклад масиву типу int

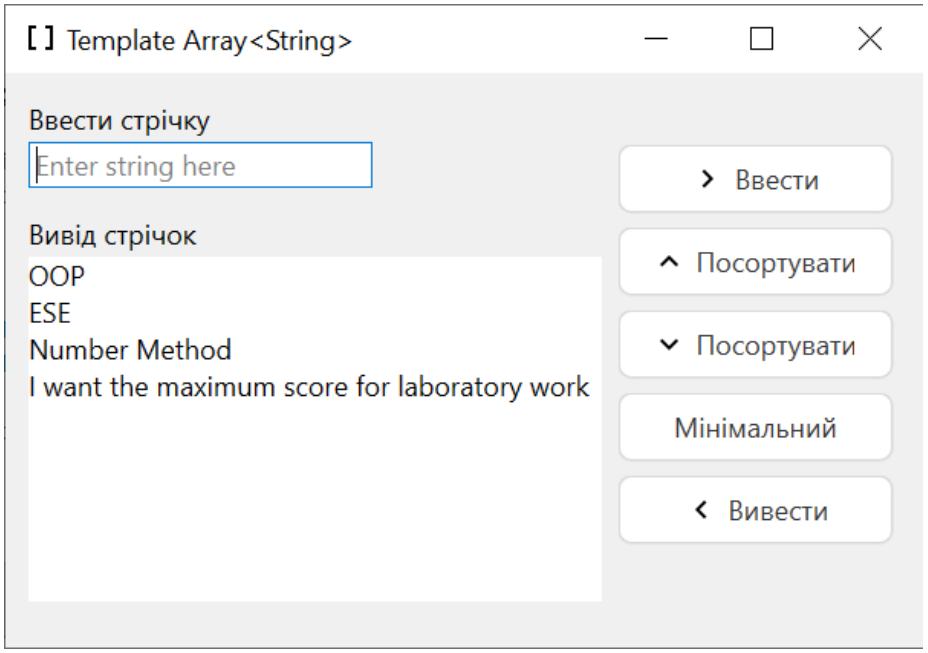

Рис 2. приклад масиву з типом MyString

ВИСНОВКИ

Під час виконання лабораторної роботи №10 я навчився створювати шаблони класу та екземпляри шаблонів. Та закріпив знання створивши шаблон класу масив, спеціалізував його для користувацького класу MyString, та продемонстрував роботу шаблону на 2 типах даних int та MyString.