

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
На тему:
«Робота з динамічною пам'яттю»
з дисципліни
«Об'єктно-орієнтоване програмування»

Лектор:

Доцент каф. ПЗ
Коротєєва Т. О.

Виконав:

ст. гр. ПЗ-11
Ясногородський Н. В.

Прийняла:

Доцент каф. ПЗ
Коротєєва Т. О.

« __ » _____ 2022 р.

Σ = _____ .

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

В мові C++ існує декілька основних **типів 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 знаходитися 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.

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. 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**. 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.

Завдання. Варіант №15

Клас `Matrix` – матриця. Пам'ять під елементи масиву повинна виділятися динамічно. Елементи матриці повинні зберігатися у двовимірному масиві.

Реалізувати такі функції члени:

- ☐ Знаходження максимального від'ємного значення матриці.
- ☐ Знаходження мінімального додатного значення матриці.
- ☐ Знаходження максимального за модулем значення матриці.
- ☐ Зміна розмірів матриці.
- ☐ Транспонування матриці.
- ☐ Обертання матриці.

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

- ☐ Додавання матриць.
- ☐ Віднімання матриць.
- ☐ Множення матриць.
- ☐ Додавання до матриці одиничної (++)
- ☐ Введення матриці з `StringGrid` (>>)
- ☐ Виведення матриці у `StringGrid` (<<)

Хід роботи

Код програми:

`main.cpp`:

```
#include <QApplication>

#include "widget.h"

int main(int argc, char *argv[]) {
    QApplication a(argc, argv);
    Widget w;
    w.show();
    return a.exec();
}
```

`uimatrix.cpp`:

```
#include "uimatrix.h"

UIMatrix::UIMatrix(unsigned _rows, unsigned _cols, const double &_initial) {
    this->mtrx = new Matrix<double>(_rows, _cols, _initial);
}

UIMatrix::~~UIMatrix() { delete this->mtrx; }

void operator>>(QTableWidget *in, UIMatrix &matrix) {
    auto mtrx = matrix.mtrx;
    for (auto i = 0; i < mtrx->rows; i++)
        for (auto j = 0; j < mtrx->cols; j++) {
            (*mtrx)(i, j) = in->item(i, j)->text().toDouble();
        }
}
```

```

    }
}

void operator<<(QTableWidget *out, UIMatrix &matrix) {
    auto mtrx = matrix.mtrx;
    for (auto i = 0; i < mtrx->rows; i++)
        for (auto j = 0; j < mtrx->cols; j++) {
            out->setItem(i, j, new QTableWidgetItem(QString::number((*mtrx)(i, j))));
        }
}

double UIMatrix::get_max_negative() {
    auto mtrx = (*this->mtrx);
    auto val = mtrx(0, 0);

    for (auto i = 0; i < mtrx.rows; i++)
        for (auto j = 0; j < mtrx.cols; j++) {
            val = std::min(val, mtrx(i, j));
        }

    return val;
}

double UIMatrix::get_min_positive() {
    auto mtrx = (*this->mtrx);
    auto val = mtrx(0, 0);

    for (auto i = 0; i < mtrx.rows; i++)
        for (auto j = 0; j < mtrx.cols; j++) {
            if (mtrx(i, j) > 0) val = std::min(val, mtrx(i, j));
        }

    return val;
}

double UIMatrix::get_abs_max() {
    auto mtrx = (*this->mtrx);
    auto val = mtrx(0, 0);

    for (auto i = 0; i < mtrx.rows; i++)
        for (auto j = 0; j < mtrx.cols; j++) {
            val = std::max(val, mtrx(i, j), [](double a, double b) {
                return std::abs(a) < std::abs(b);
            });
        }

    return val;
}

```

matrix.cpp:

```

#ifndef __QS_MATRIX_CPP
#define __QS_MATRIX_CPP

#include "matrix.h"

template <typename T>
Matrix<T>::Matrix(unsigned _rows, unsigned _cols, const T& _initial) {
    mat.resize(_rows);
    for (unsigned i = 0; i < mat.size(); i++) {
        mat[i].resize(_cols, _initial);
    }
    rows = _rows;
    cols = _cols;
}

```

```

}

template <typename T>
Matrix<T>::Matrix(const Matrix<T>& rhs) {
    mat = rhs.mat;
    rows = rhs.get_rows();
    cols = rhs.get_cols();
}

template <typename T>
Matrix<T>::~Matrix() {}

template <typename T>
Matrix<T>& Matrix<T>::operator=(const Matrix<T>& rhs) {
    if (&rhs == this) return *this;

    auto new_rows = rhs.get_rows();
    auto new_cols = rhs.get_cols();

    mat.resize(new_rows);
    for (auto i = 0; i < mat.size(); i++) mat[i].resize(new_cols);

    for (auto i = 0; i < new_rows; i++)
        for (auto j = 0; j < new_cols; j++) mat[i][j] = rhs(i, j);

    rows = new_rows;
    cols = new_cols;

    return *this;
}

template <typename T>
Matrix<T> Matrix<T>::operator+(const Matrix<T>& rhs) {
    Matrix result(rows, cols, 0.0);

    for (auto i = 0; i < rows; i++)
        for (auto j = 0; j < cols; j++) result(i, j) = this->mat[i][j] + rhs(i, j);

    return result;
}

template <typename T>
Matrix<T> Matrix<T>::operator-(const Matrix<T>& rhs) {
    Matrix result(rows, cols, 0.0);

    for (auto i = 0; i < rows; i++)
        for (auto j = 0; j < cols; j++) result(i, j) = this->mat[i][j] - rhs(i, j);

    return result;
}

template <typename T>
Matrix<T>& Matrix<T>::operator++() {
    for (auto i = 0; i < rows; i++) this->mat[i][i]++;
    return *this;
}

template <typename T>
Matrix<T> Matrix<T>::operator*(const Matrix<T>& rhs) {
    auto rows = rhs.get_rows();
    auto cols = rhs.get_cols();
    if (this->cols != rows) throw "Cannot multiply matrices";

    Matrix result(this->rows, cols, 0.0);

    for (auto i = 0; i < this->rows; i++)

```

```

        for (auto j = 0; j < cols; j++)
            for (auto k = 0; k < rows; k++)
                result(i, j) += this->mat[i][k] * rhs(k, j);

    return result;
}

template <typename T>
Matrix<T> Matrix<T>::transpose() {
    Matrix result(rows, cols, 0.0);

    for (auto i = 0; i < rows; i++)
        for (auto j = 0; j < cols; j++) result(i, j) = this->mat[j][i];

    return result;
}

template <typename T>
T& Matrix<T>::operator()(const unsigned& row, const unsigned& col) {
    return this->mat[row][col];
}

template <typename T>
const T& Matrix<T>::operator()(const unsigned& row, const unsigned& col) const {
    return this->mat[row][col];
}

template <typename T>
unsigned Matrix<T>::get_rows() const {
    return this->rows;
}

template <typename T>
unsigned Matrix<T>::get_cols() const {
    return this->cols;
}

#endif

```

widget.cpp:

```

#include "widget.h"

#include <QGridLayout>
#include <iostream>

void Widget::on_add() {
    this->sync_with_table();
    *this->matrix_res->mtrx = (*this->matrix_a->mtrx + *this->matrix_b->mtrx);
    emit value_changed();
}

void Widget::on_multiply() {
    this->sync_with_table();
    *this->matrix_res->mtrx = (*this->matrix_a->mtrx * *this->matrix_b->mtrx);
    emit value_changed();
}

void Widget::on_subtract() {
    this->sync_with_table();
    *this->matrix_res->mtrx = (*this->matrix_a->mtrx - *this->matrix_b->mtrx);
    emit value_changed();
}

```

```

void Widget::on_transpose() {
    this->sync_with_table();
    *this->matrix_res->mtrx = this->matrix_res->mtrx->transpose();
    emit value_changed();
}

void Widget::sync_with_table() {
    this->ui_matrix_a >> *this->matrix_a;
    this->ui_matrix_b >> *this->matrix_b;
    this->ui_matrix_res >> *this->matrix_res;
}

void Widget::sync_with_model() {
    this->ui_matrix_a << *this->matrix_a;
    this->ui_matrix_b << *this->matrix_b;
    this->ui_matrix_res << *this->matrix_res;
}

void Widget::on_value_change() {
    auto matrix = this->matrix_res;
    this->max_abs_input->setText(QString::number(matrix->get_abs_max()));
    this->max_negative_input->setText(
        QString::number(matrix->get_max_negative()));
    this->min_positive_input->setText(
        QString::number(matrix->get_min_positive()));

    this->sync_with_model();
}

Widget::Widget(QWidget *parent) : QWidget(parent) {
    auto *mainLayout = new QGridLayout;
    auto *btns_layout = new QGridLayout;

    this->matrix_a = new UIMatrix(10, 10, 1.0);
    this->matrix_b = new UIMatrix(10, 10, 2.0);
    this->matrix_res = new UIMatrix(10, 10, 0.0);
    this->ui_matrix_a = new QTableWidgetItem(10, 10);
    this->ui_matrix_b = new QTableWidgetItem(10, 10);
    this->ui_matrix_res = new QTableWidgetItem(10, 10);

    this->add_matrices_btn = new QPushButton("Add");
    this->subtract_matrices_btn = new QPushButton("Substract");
    this->multiply_matrices_btn = new QPushButton("Multiply");
    this->transpose_matrix_btn = new QPushButton("Transpose");

    this->min_positive_input = new QLineEdit;
    this->max_negative_input = new QLineEdit;
    this->max_abs_input = new QLineEdit;
    min_positive_input->setReadOnly(true);
    max_negative_input->setReadOnly(true);
    max_abs_input->setReadOnly(true);

    btns_layout->addWidget(this->add_matrices_btn, 0, 0);
    btns_layout->addWidget(this->subtract_matrices_btn, 1, 0);

    btns_layout->addWidget(this->multiply_matrices_btn, 0, 2);
    btns_layout->addWidget(this->transpose_matrix_btn, 1, 2);

    mainLayout->addWidget(this->ui_matrix_a, 0, 0, 1, 1);
    mainLayout->addWidget(this->ui_matrix_b, 0, 1, 1, 1);
    mainLayout->addLayout(btns_layout, 1, 0, 1, 2);
    mainLayout->addWidget(this->ui_matrix_res, 2, 0, 1, 2);

    mainLayout->addWidget(new QLabel("Max negative:"), 3, 0);
    mainLayout->addWidget(this->max_negative_input, 3, 1);
}

```

```

mainLayout->addWidget(new QLabel("Min positive:"), 4, 0);
mainLayout->addWidget(this->min_positive_input, 4, 1);

mainLayout->addWidget(new QLabel("Max abs:"), 5, 0);
mainLayout->addWidget(this->max_abs_input, 5, 1);

connect(this->add_matrices_btn, &QPushButton::released, this,
        &Widget::on_add);
connect(this->subtract_matrices_btn, &QPushButton::released, this,
        &Widget::on_subtract);
connect(this->multiply_matrices_btn, &QPushButton::released, this,
        &Widget::on_multiply);
connect(this->transpose_matrix_btn, &QPushButton::released, this,
        &Widget::on_transpose);

connect(this, &Widget::value_changed, &Widget::on_value_change);

setLayout(mainLayout);

this->sync_with_model();
}

```


Результати виконання програми

	1	2	3	4	5	6	7	8		1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	1		1	2	2	2	2	2	2	2
2	1	1	20	1	1	1	1	1		2	2	-999	2	2	2	2	2
3	1	1	1	1	1	1	1	1		3	2	2	2	2	2	2	2
4	1	1	1	1	1	1	1	1		4	2	30	2	2	2	2	2
5	1	1	1	1	1	1	1	-10	1	5	2	2	2	2	2	2	2
6	1	1	1	1	1	1	1	1	1	6	2	2	2	2	2	2	2
7	1	1	1	1	100	1	1	1	1	7	2	2	2	2	2	2	2
8	1	1	1	1	1	1	1	1	1	8	2	2	2	2	2	2	2
9	1	1	1	1	1	1	1	1	1	9	2	2	2	2	30	2	2
10	1	1	1	1	1	1	1	1	1	10	2	2	2	2	2	2	2
Add									Multiply								
Subtract									Transpose								
	1	2	3	4	5	6	7	8	9	10							
1	20	-953	20	20	20	48	20	20	20	20							
2	58	-915	58	58	58	86	58	58	58	58							
3	20	-953	20	20	20	48	20	20	20	20							
4	20	-953	20	20	20	48	20	20	20	20							
5	-2	-975	-2	-2	-2	26	-2	-2	-2	-2							
6	20	-953	20	20	20	48	20	20	20	20							
7	218	-755	218	218	218	246	218	218	218	218							
8	20	-953	20	20	20	48	20	20	20	20							
9	20	-953	20	20	20	48	20	20	20	20							
10	20	-953	20	20	20	48	20	20	20	20							
Max negative:											-975						
Min positive:											20						
Max abs:											-975						

Рис. 1. Результати обчислень програми

	1	2	3	4	5	6	7	8		1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	234	1		1	2	2	2	2	2	2	2
2	1	1	1	1	1	1	1	1		2	2	2	2	2	2	2	2
3	1	1	1	1	1	1	1	1		3	2	234	2	2	2	2	2
4	1	1	1	1	1	1	1	1		4	2	2	2	2000	2	2	2
5	1	1	33	1	1	1	1	1		5	2	2	2	2	2	2	2
6	1	1	1	1	1	99	1	1		6	2	2	2	2	2	2	2
7	1	1	1	1	1	1	1	1		7	234	2	2	2	2	2	2
8	1	1	1	1	1	1	1	1		8	2	2	2	2	2	2	2
9	1	1	1	1	1	19	1	1		9	2	2	-2923	2	2	2	2
10	1	3095	1	1	1	1	1	1		10	2	2	2	2	2	2	2
Add									Multiply								
Subtract									Transpose								

	1	2	3	4	5	6	7	8	9	10	
1	54774	718	-2439	2484	495	486	486	486	495	486	
2	252	252	-2906	2018	20	20	20	20	20	20	
3	252	252	-2906	2018	20	20	20	20	20	20	
4	252	252	-2906	2018	20	20	20	20	20	20	
5	316	7740	-2841	2082	84	84	84	84	84	84	
6	448	448	-2709	2214	216	216	216	216	216	216	
7	252	252	-2906	2018	20	20	20	20	20	20	
8	252	252	-2906	2018	20	20	20	20	20	20	
9	288	288	-2669	2054	56	56	56	56	56	56	
10	6440	6440	3283	8206	6208	6208	6208	6208	6208	6208	
Max negative: -2905											
Min positive: 20											
Max abs: 54774											

Рис. 2. Результати обчислень програми

Висновок

Виконуючи лабораторну роботу №7 я навчився працювати з динамічною пам'ятю в мові C++, створив власний клас Matrix та продемонстрував його можливості на віконному застосуванні.