Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра інформаційних систем та мереж

ЗВІТ

про виконання лабораторних робіт №1

**“ Mоделювання представлення в памяті векторів і таблиць**.**”**

з дисципліни "Алгоритми та структури даних ”

В-15

Виконала студентка групи СА-32

Катеринчук Тетяна Андріївна

Прийняв доцент

Щербак С.С.

*2017*

**Мета роботи:** Набуття навичок розміщення в памяті векторів і таблиць

**Завдання:**

Ознайомитися з основними типами даних вибраної мови програмування та реалізувати калькулятор простих чисел з підтримкою зворотного польського запису. Додати можливість поелементого +,-,/,\* векторів та таблиць. Розробити спосіб економного зберігання в памяті розріджених матриць (таблиць). Розробити процедури і функції для забезпечення доступу (читання-запис) до елементів матриці. В контрольному прикладі забезпечити читання і запис всіх елементів матриці. Оцінити час виконання операцій.

**Індивідуальне завдання:**

Всі нульові елементи розміщені нижче головної діагоналі на непарних стрічках і вище головної діагоналі - на парних.

**Текст програми**

Main.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "Calculator.h"

#include "Matrix.h"

#include "Interface.h"

#include "ArchivingMatrix.h"

#include <iostream>

#include <clocale>

using namespace std;

void main() {

setlocale(LC\_CTYPE, "ukr");

Interface expression = "";

int choice = 1;

bool b = true;

while (choice != 0) {

cout << "[1] - Обчислення виразу з простих числел\n[2] - Дiї з матрицями\n[3] - Архiвування матрицi\n[0] - Вихiд з програми\nВведiть дiю яку ви хочете обрати: ";

cin >> choice;

if (choice == 1) {

while (b) {

expression.InputExpression();

Calculate calculate = expression.OutputExpression();

if (calculate.RightExpression()) {

calculate.TransferPolishForm();

calculate.SteckMoveExit();

calculate.CalculationsPolishForm();

b = false;

}

else b = true;

cout << endl;

}

}

else if (choice == 2) {

Interface matrix;

int nMatrix = 0;

cout << "Введiть кiлькiсть стовпцiв i рядкiв матрицi(квадратної): ";

cin >> nMatrix;

double \*\*mainMatrix1 = (double\*\*)malloc(nMatrix \* sizeof(double));

cout << "Введiть елементи 1 матрицi:" << endl;

for (int i = 0; i < nMatrix; i++)

mainMatrix1[i] = (double\*)malloc(nMatrix \* sizeof(double));

matrix.InputMatrix(mainMatrix1, nMatrix);

cout << "Введiть елементи 2 матрицi:" << endl;

double \*\*mainMatrix2 = (double\*\*)malloc(nMatrix \* sizeof(double));

for (int i = 0; i < nMatrix; i++)

mainMatrix2[i] = (double\*)malloc(nMatrix \* sizeof(double));

matrix.InputMatrix(mainMatrix2, nMatrix);

cout << "A=\t";

matrix.OutputMatrix(mainMatrix1, nMatrix);

cout << "B=\t";

matrix.OutputMatrix(mainMatrix2, nMatrix);

int actionMatrix = 1;

while (actionMatrix != 0) {

cout << "Введiть номер дiї яку ви хочете зробити з матрицями:\n[1] - Додавання\n[2] - Вiднiмання\n[3] - Множення\n[4] - Дiлення\n[0] - Вихiд в меню\n";

cin >> actionMatrix;

double \*\*mainMatrix = (double\*\*)malloc(nMatrix \* sizeof(double));

for (int i = 0; i < nMatrix; i++)

mainMatrix[i] = (double\*)malloc(nMatrix \* sizeof(double));

if (actionMatrix == 1) {

PlusMatrix(mainMatrix, mainMatrix1, mainMatrix2, nMatrix);

cout << "A+B=\t";

matrix.OutputMatrix(mainMatrix, nMatrix);

}

if (actionMatrix == 2) {

MinusMatrix(mainMatrix, mainMatrix1, mainMatrix2, nMatrix);

cout << "A-B=\t";

matrix.OutputMatrix(mainMatrix, nMatrix);

}

if (actionMatrix == 3) {

MultiplyMatrix(mainMatrix, mainMatrix1, mainMatrix2, nMatrix);

cout << "A\*B=\t";

matrix.OutputMatrix(mainMatrix, nMatrix);

}

if (actionMatrix == 4) {

DivisionMatrix(mainMatrix, mainMatrix1, mainMatrix2, nMatrix);

cout << "A/B=\t";

matrix.OutputMatrix(mainMatrix, nMatrix);

}

for (int i = 0; i < nMatrix; i++)

free(mainMatrix[i]);

free(mainMatrix);

}

for (int i = 0; i < nMatrix; i++)

free(mainMatrix1[i]);

free(mainMatrix1);

for (int i = 0; i < nMatrix; i++)

free(mainMatrix2[i]);

free(mainMatrix2);

}

else if (choice == 3) {

Interface arhivMatrix = "";

int nArchivingMatrix = 6;

cout << "Введiть кількість стовпцiв i рядкiв матрицi: "; cin >> nArchivingMatrix;

int \*\*mainArchivingMatrix = (int\*\*)malloc(nArchivingMatrix \* sizeof(int));

for (int i = 0; i < nArchivingMatrix; i++)

mainArchivingMatrix[i] = (int\*)malloc(nArchivingMatrix \* sizeof(int));

int nLowerPartMatrix = 0, nUpperPartMatrix = 0;

cout << "Матриця до архiвацiї: \n";

InputArchivingMatrix(mainArchivingMatrix, nArchivingMatrix, nLowerPartMatrix, nUpperPartMatrix);

arhivMatrix.OutputArchivingMatrix(mainArchivingMatrix, nArchivingMatrix);

int \*lowerPartMatrix = (int\*)malloc(nLowerPartMatrix \* sizeof(int));

int \*upperPartMatrix = (int\*)malloc(nUpperPartMatrix \* sizeof(int));

int \*diagonalMatrix = (int\*)malloc(nArchivingMatrix \* sizeof(int));

TransferMatrix(mainArchivingMatrix, diagonalMatrix, lowerPartMatrix, upperPartMatrix, nArchivingMatrix);

cout << "Дiагональ матрицi: ";

for (int i = 0; i < nArchivingMatrix; i++) {

cout << diagonalMatrix[i] << " ";

}

cout << endl<<"Нижня частина матрицi(ненульовi елементи): ";

for (int i = 0; i < nLowerPartMatrix;i++) {

cout << lowerPartMatrix[i] << " ";

}

cout << endl<<"Верхня частина матрицi(ненульовi елементи):: ";

for (int i = 0; i < nUpperPartMatrix; i++) {

cout << upperPartMatrix[i] << " ";

}

cout << endl;

for (int i = 0; i < nArchivingMatrix; i++)

free(mainArchivingMatrix[i]);

free(mainArchivingMatrix);

int \*\*createdMatrix = (int\*\*)malloc(nArchivingMatrix \* sizeof(int));

for (int i = 0; i < nArchivingMatrix; i++)

createdMatrix[i] = (int\*)malloc(nArchivingMatrix \* sizeof(int));

UnzippingMatrix(createdMatrix, diagonalMatrix, lowerPartMatrix, upperPartMatrix, nArchivingMatrix);

cout << "Розархiвована матриця: \n";

arhivMatrix.OutputArchivingMatrix(createdMatrix, nArchivingMatrix);

cout << endl;

free(diagonalMatrix);

free(lowerPartMatrix);

free(upperPartMatrix);

for (int i = 0; i < nArchivingMatrix; i++)

free(createdMatrix[i]);

free(createdMatrix);

}

else if(choice!=0)cout << "Некоректне введення!!!\n";

}

}

Interface.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Interface {

protected:

string initialExpression;

public:

Interface() {}

Interface(string f1);

string InputExpression();

string OutputExpression();

void InputMatrix(double \*\*matrix, int n);

void OutputMatrix(double \*\*matrix, int n);

void OutputArchivingMatrix(int \*\*matrix, int n);

};

Interface::Interface(string f1) {

initialExpression = f1;

}

//введення виразу

string Interface::InputExpression() {

cout << "Введiть вираз: ";

cin >> initialExpression;

return initialExpression;

}

//виведення виразу

string Interface::OutputExpression() {

return initialExpression;

}

//введення матриці

void Interface::InputMatrix(double \*\*matrix, int n) {

for (int i = 0; i<n; i++) {

for (int j = 0; j<n; j++)

{

cout << "[" << i << "][" << j << "]:"; cin >> matrix[i][j];

}

}

}

//виведення матриці

void Interface::OutputMatrix(double \*\*matrix, int n) {

for (int i = 0; i < n; i++) {

cout << "|";

for (int j = 0; j < n; j++) {

cout << matrix[i][j] << " ";

}

cout << "|" << endl << "\t";

}

cout << endl;

}

//виведення матриці архівування

void Interface::OutputArchivingMatrix(int \*\*matrix, int n) {

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

cout << matrix[i][j] << " ";

}

cout << endl;

}

}

Calculator.h

#pragma once

#include "Interface.h"

#include <stack>

#include <queue>

class Calculate:public Interface {

protected:

string initialExpression;

queue <string> exit;

queue <string> expression;

stack <string> steck;

stack <double> number;

string auxiliary;

int numberMulDiv[100];

int numberPow[100];

int numberPlMin[100];

int numberBrackets = 0;

public:

Calculate() {}

Calculate(string f1);

bool RightExpression();

bool ProstoNumber(int n);

void TransferPolishForm();

void VerifyNumber(string element);

void VerifyPlusMinus(string element);

void VerifyMultiplicationDivision(string element);

void VerifyPower(string element);

void VerifyBrackets(string element);

void SteckMoveExit();

void CalculationsPolishForm();

};

Calculate::Calculate(string f1) {

initialExpression = f1;

}

//розділення виразу на числа і знаки, запис кожного у чергу

bool Calculate::RightExpression() {

for (int i = 0; i < initialExpression.size(); i++) {

if (initialExpression[i] >= 48 && initialExpression[i] <= 57) {

if (initialExpression[i + 1] >= 48 && initialExpression[i + 1] <= 57) {

auxiliary.append(initialExpression, i, 1);

}

else {

expression.push(auxiliary.append(initialExpression, i, 1));

if (!ProstoNumber(atoi(auxiliary.c\_str()))) {

cout << "Помилка! Введіть вираз який буде складатися тільки з простих чисел:" << endl;

while (!expression.empty()) {

expression.pop();

}

auxiliary.clear();

return false;

}

auxiliary.clear();

}

}

else if (initialExpression[i] == '+' || initialExpression[i] == '-' || initialExpression[i] == '\*' || initialExpression[i] == '/' || initialExpression[i] == '^' || initialExpression[i] == '(' || initialExpression[i] == ')' || initialExpression[i] == '=') {

expression.push(auxiliary.append(initialExpression, i, 1));

auxiliary.clear();

}

else {

cout << "Помилка! Некоректне значення. ";

while (!expression.empty()) {

expression.pop();

}

InputExpression();

RightExpression();

}

}

return true;

}

//перевірка чи число просте

bool Calculate::ProstoNumber(int n) {

for (int i = 2; i <= sqrt(n); i++)

if (n % i == 0)

return false;

return true;

}

//перевід виразу в польський зворотній запис

void Calculate::TransferPolishForm() {

for (int i = 0; i < 100; i++) {

numberMulDiv[i] = 0;

numberPow[i] = 0;

numberPlMin[i] = 0;

}

while (!expression.empty()) {

VerifyNumber(expression.front());

VerifyPlusMinus(expression.front());

VerifyMultiplicationDivision(expression.front());

VerifyPower(expression.front());

VerifyBrackets(expression.front());

if (!expression.empty())

expression.pop();

}

}

//перевірка чи елемент черги число

void Calculate::VerifyNumber(string element) {

if (element[0] >= 48 && element[0] <= 57) {

exit.push(element);

if (numberPow[numberBrackets] > 0) {

exit.push(steck.top());

steck.pop();

if (numberMulDiv[numberBrackets] > 0) {

exit.push(steck.top());

steck.pop();

numberMulDiv[numberBrackets] = 0;

}

numberPow[numberBrackets] = 0;

}

}

}

//перевірка чи елемент черги знак плюс чи мінус

void Calculate::VerifyPlusMinus(string element) {

if (element == "+" || element == "-") {

if (numberMulDiv[numberBrackets] > 0) {

exit.push(steck.top());

steck.pop();

numberMulDiv[numberBrackets] = 0;

}

if (numberPlMin[numberBrackets] > 0) {

exit.push(steck.top());

steck.pop();

numberPlMin[numberBrackets] = 0;

}

else

numberPlMin[numberBrackets]++;

steck.push(element);

}

}

//перевірка чи елемент черги знак множення або віднімання

void Calculate::VerifyMultiplicationDivision(string element) {

if (element == "\*" || element == "/") {

if (numberMulDiv[numberBrackets] > 0) {

exit.push(steck.top());

steck.pop();

numberMulDiv[numberBrackets] = 0;

}

steck.push(element);

numberMulDiv[numberBrackets]++;

}

}

//перевірка чи елемент черги дужки

void Calculate::VerifyBrackets(string element) {

if (element == "(") {

numberBrackets++;

steck.push(element);

expression.pop();

while (!expression.empty()) {

VerifyNumber(expression.front());

VerifyPlusMinus(expression.front());

VerifyMultiplicationDivision(expression.front());

VerifyPower(expression.front());

VerifyBrackets(expression.front());

if (expression.front() == ")") {

break;

numberBrackets--;

}

expression.pop();

}

while (steck.top() != "(") {

exit.push(steck.top());

steck.pop();

}

steck.pop();

}

}

//перевірка чи елемент черги знак степінь

void Calculate::VerifyPower(string element) {

if (element == "^") {

steck.push(element);

numberPow[numberBrackets]++;

}

}

//переміщення всіх елементів з стеку в чергу

void Calculate::SteckMoveExit() {

while (!steck.empty()) {

exit.push(steck.top());

steck.pop();

}

}

//обчислення польського виразу

void Calculate::CalculationsPolishForm() {

double calculations = 0;

cout << "Польський запис виразу: ";

while (!exit.empty()) {

auxiliary = exit.front();

if (auxiliary[0] >= 48 && auxiliary[0] <= 57) {

number.push(atoi(auxiliary.c\_str()));

}

else {

double right = number.top();

number.pop();

double left = number.top();

number.pop();

if (auxiliary == "+") {

number.push(left + right);

}

if (auxiliary == "-") {

number.push(left - right);

}

if (auxiliary == "\*") {

number.push(left \* right);

}

if (auxiliary == "/") {

number.push(left / right);

}

if (auxiliary == "^") {

number.push(pow(left, right));

}

}

calculations = number.top();

cout << auxiliary << " ";

exit.pop();

}

cout << endl;

cout << initialExpression << "=" << calculations << endl;

}

Matrix.h

#pragma once

//додавання матриць

void PlusMatrix(double \*\*matrix, double \*\*matrix1, double \*\*matrix2, int n) {

for (int i = 0; i<n; i++) {

for (int j = 0; j<n; j++)

{

matrix[i][j] = matrix1[i][j] + matrix2[i][j];

}

}

}

//віднімання матриць

void MinusMatrix(double \*\*matrix, double \*\*matrix1, double \*\*matrix2, int n) {

for (int i = 0; i<n; i++) {

for (int j = 0; j<n; j++)

{

matrix[i][j] = matrix1[i][j] - matrix2[i][j];

}

}

}

//множення матриць

void MultiplyMatrix(double \*\*matrix, double \*\*matrix1, double \*\*matrix2, int n) {

for (int i = 0; i<n; i++) {

for (int j = 0; j<n; j++)

{

matrix[i][j] = 0;

for (int k = 0; k < n; k++) {

matrix[i][j] += matrix1[i][k] \* matrix2[k][j];

}

}

}

}

//ділення матриць

void DivisionMatrix(double \*\*matrix, double \*\*matrix1, double \*\*matrix2, int n) {

double \*\*matrix3 = (double\*\*)malloc(n \* sizeof(double));

for (int i = 0; i < n; i++)

matrix3[i] = (double\*)malloc(n \* sizeof(double));

double element;

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

{

matrix[i][j] = 0;

if (i == j)

matrix3[i][j] = 1;

else

matrix3[i][j] = 0;

}

for (int i = 0; i < n; i++)

{

element = matrix2[i][i];

for (int j = 0; j < n; j++)

{

matrix2[i][j] /= element;

matrix3[i][j] /= element;

}

for (int k = i + 1; k < n; k++)

{

element = matrix2[k][i];

for (int j = 0; j < n; j++)

{

matrix2[k][j] -= matrix2[i][j] \* element;

matrix3[k][j] -= matrix3[i][j] \* element;

}

}

}

for (int i = n - 1; i > 0; i--)

{

for (int k = i - 1; k >= 0; k--)

{

element = matrix2[k][i];

for (int j = 0; j < n; j++)

{

matrix2[k][j] -= matrix2[i][j] \* element;

matrix3[k][j] -= matrix3[i][j] \* element;

}

}

}

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

for (int k = 0; k < n; k++)

matrix[i][j] += matrix1[i][k] \* matrix2[k][j];

for (int i = 0; i < n; i++)

free(matrix3[i]);

free(matrix3);

}

ArchivingMatrix.h

#pragma once

void InputArchivingMatrix(int\*\*, int, int &, int &);

//рандомне заповнення матриці, 0 стоять вище головної діагоналі на непарних рядкав і нище головної на парних

void InputArchivingMatrix(int \*\*matrix, int n, int &nLower, int &nUpper) {

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

if (i % 2 == 1 && i>j) {

matrix[i][j] = 0;

}

else if (i % 2 == 0 && i<j) {

matrix[i][j] = 0;

}

else {

matrix[i][j] = 1 + rand() % (10 - 1 + 1);

}

if (i % 2 == 0 && i>j) {

nLower++;

}

if (i % 2 == 1 && i<j) {

nUpper++;

}

}

}

}

//перепис двовимірного масива з нульовими елементами в три одновимірних масив без нульових елементів

void TransferMatrix(int \*\*matrix,int \*diagonalMatrix, int \*lowerMatrix, int \*upperMatrix, int n) {

int iUp = 0, iLow=0, iDiagonal=0;

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

if (i % 2 == 0 && i>j) {

lowerMatrix[iLow] = matrix[i][j];

iLow++;

}

else if (i % 2 == 1 && i<j) {

upperMatrix[iUp] = matrix[i][j];

iUp++;

}

else if (i == j) {

diagonalMatrix[iDiagonal] = matrix[i][j];

iDiagonal++;

}

}

}

}

//перепис три одновимірних масив без нульових елементів в двовимірний масив з нульовими елементами

void UnzippingMatrix(int \*\*matrix, int \*diagonalMatrix, int \*lowerMatrix, int \*upperMatrix, int n) {

int iUp = 0, iLow = 0, iDiagonal = 0;

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

if (i % 2 == 0 && i>j) {

matrix[i][j]=lowerMatrix[iLow] ;

iLow++;

}

else if (i % 2 == 1 && i<j) {

matrix[i][j]=upperMatrix[iUp];

iUp++;

}

else if (i == j) {

matrix[i][j]=diagonalMatrix[iDiagonal] ;

iDiagonal++;

}

else matrix[i][j] = 0;

}

}

}

**Результат програми**

Обчислення виразу з простих чисел

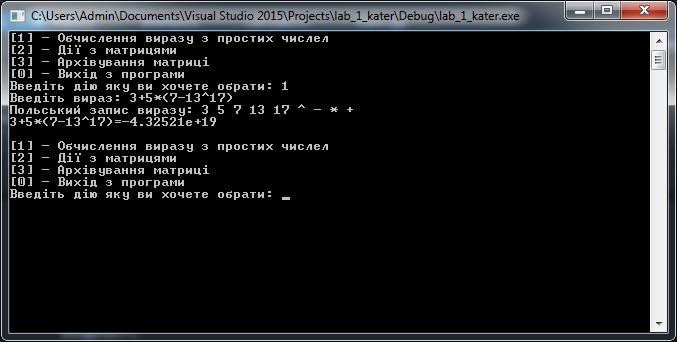


Рис.1 Обчислення виразу

Дії з матрицями:

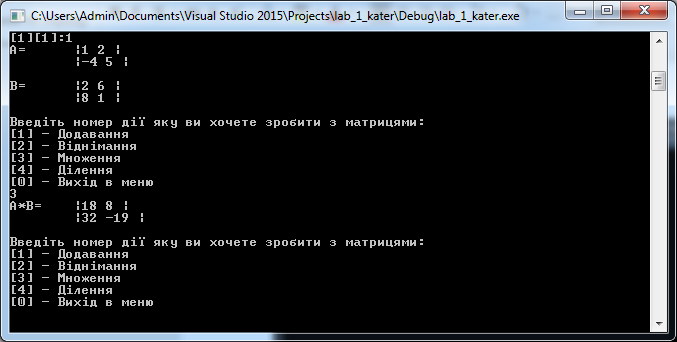


Рис. 2 Множення матриць

Економного зберігання в памяті розріджених матриць:

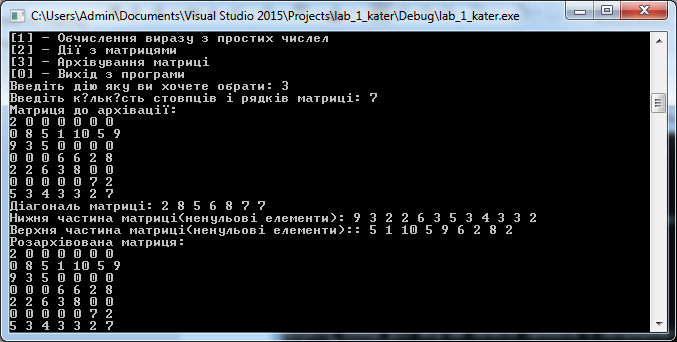


Рис. 3 Економного зберігання в памяті розріджених матриць

**Висновок:** При виконання даної лабараторної роботи я набула навичок розміщення в памяті векторів і таблиць. Створила програму яка складається з трьох частин. Перша частина це обрахунок вираза, попередньо конверсованого в польський зворотній вираз, даний вираз введенний з клавіатури, який повинен містити тільки прості числа. Друга частина це дії з матрицями (додавання, віднімання, множення, ділення), але є обмеження що матриці повинні бути квадратні, тому цей код потрібно вдосконалити. Третя частина реалізує спосіб економного зберігання в памяті розріджених матриць.