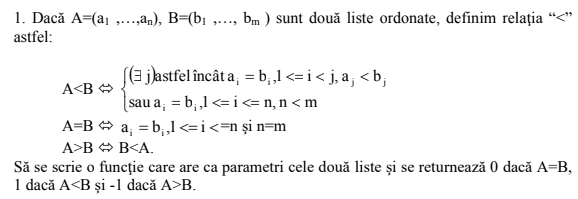
Structuri de Date

Laboratorul 2 – Tema 2 - Nicoleta Radu

# Exercitiul 1



int exercitiul\_1(int a[], int b[])

{

unsigned int size\_a = sizeof(a) / sizeof(a[0]);

unsigned int size\_b = sizeof(b) / sizeof(b[0]);

int result{ 0 };

if (size\_a == size\_b)

{

// cazul 1: a si b sunt egale

for (size\_t i = 0; i < size\_a; i++)

{

if (a[i] == b[i])

{

result = 0;

}

else

{

break;

}

}

}

else if (size\_a < size\_b)

{

// cazul 2: a este mai mic decat b

// nu inteleg cerinta (ai = bi, 1 <= i < j, aj < bj)

result = 1;

}

else

{

// cazul 3: b este mai mic decat a

// nu inteleg cerinta

result = -1;

}

return result;

}

# Exercitiul 2



#include <iostream>

#include "SimpleList.h"

typedef int\*\* Matrix;

Matrix InstantiateMatrix(unsigned int Rows, unsigned int Columns)

{

Matrix TEMP;

TEMP = new int\* [Rows];

for (size\_t i = 0; i < Rows; i++)

{

TEMP[i] = new int[Columns];

}

return TEMP;

}

double determinant(Matrix rareMatrix, unsigned int rows, unsigned int columns) {

if (rows != columns) {

throw std::runtime\_error("Matricea nu este patratica");

}

int dimension = rows;

if (dimension == 0) {

return 1;

}

if (dimension == 1) {

return rareMatrix[0][0];

}

// rows/columns = 2

if (dimension == 2) {

return rareMatrix[0][0] \* rareMatrix[1][1] - rareMatrix[0][1] \* rareMatrix[1][0];

}

double result = 0;

int sign = 1;

unsigned int subMatrix\_rows = dimension - 1;

unsigned int subMatrix\_columns = dimension - 1;

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

//Submatrix

Matrix subMatrix = InstantiateMatrix(subMatrix\_rows, subMatrix\_columns);

for (int m = 1; m < dimension - 1; m++) {

int z = 0;

for (int n = 0; n < dimension - 1; n++) {

if (n != i) {

subMatrix[m - 1][z] = subMatrix[m][n];

z++;

}

}

}

//recursive call

result = result + sign \* subMatrix[0][i] \* determinant(subMatrix, subMatrix\_rows, subMatrix\_columns);

sign = -sign;

}

return result;

}

Matrix Transpusa(const Matrix matrix1, unsigned int rows, unsigned int columns ) {

//Transpose-matrix: height = width(matrix), width = height(matrix)

unsigned int solution\_rows = rows;

unsigned int solution\_columns = columns;

Matrix solution = InstantiateMatrix(solution\_rows, solution\_columns);

// Umplere solution-matrix

for (size\_t i = 0; i < solution\_rows; i++) {

for (size\_t j = 0; j < solution\_columns; j++) {

solution[j][i] = matrix1[i][j];

}

}

return solution;

}

Matrix Cofactor(Matrix vect, unsigned int rows, unsigned int columns) {

if (rows != columns) {

throw std::runtime\_error("Matricea nu este patratica");

}

unsigned int solution\_rows = rows;

unsigned int solution\_columns = columns;

Matrix solution = InstantiateMatrix(solution\_rows, solution\_columns);

unsigned int subMatrix\_rows = solution\_rows - 1;

unsigned int subMatrix\_columns = solution\_columns - 1;

//Submatrix

Matrix subMatrix = InstantiateMatrix(subMatrix\_rows, subMatrix\_columns);

for (std::size\_t i = 0; i < rows; i++) {

for (std::size\_t j = 0; j < columns; j++) {

int p = 0;

for (size\_t x = 0; x < rows; x++) {

if (x == i) {

continue;

}

int q = 0;

for (size\_t y = 0; y < rows; y++) {

if (y == j) {

continue;

}

subMatrix[p][q] = vect[x][y];

q++;

}

p++;

}

solution[i][j] = pow(-1, i + j) \* determinant(subMatrix, subMatrix\_rows, subMatrix\_columns);

}

}

return solution;

}

Matrix exercitiul\_2(const Matrix vect, unsigned int rows, unsigned int columns) {

if (determinant(vect, rows, columns) == 0) {

throw std::runtime\_error("Determinantul este 0");

}

double d = determinant(vect, rows, columns);

d = 1.0 / d;

unsigned int solution\_rows = rows;

unsigned int solution\_columns = columns;

Matrix solution = InstantiateMatrix(solution\_rows, solution\_columns);

for (size\_t i = 0; i < rows; i++) {

for (size\_t j = 0; j < columns; j++) {

solution[i][j] = vect[i][j];

}

}

solution = Transpusa(Cofactor(solution, rows, columns), rows, columns);

for (size\_t i = 0; i < rows; i++) {

for (size\_t j = 0; j < columns; j++) {

solution[i][j] \*= d;

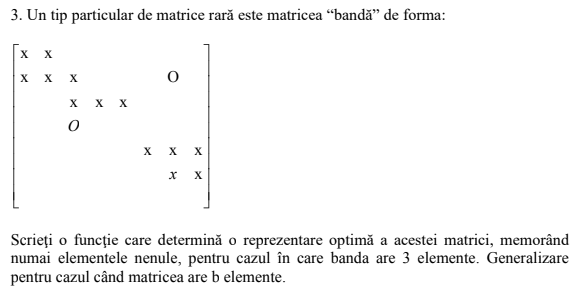
}

}

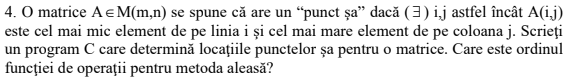
return solution;

}

# Exercitiul 3



# Exercitiul 4



typedef int\*\* Matrix;

Matrix InstantiateMatrix(unsigned int Rows, unsigned int Columns)

{

Matrix TEMP;

TEMP = new int\* [Rows];

for (size\_t i = 0; i < Rows; i++)

{

TEMP[i] = new int[Columns];

}

return TEMP;

}

void ManualMatrixGeneration(Matrix yourMatrix, unsigned int Rows, unsigned int Columns)

{

for (size\_t i = 0; i < Rows; i++)

{

for (size\_t j = 0; j < Columns; j++)

{

// user input

std::cout << "[" << i << "," << j << "] = ";

std::cin >> yourMatrix[i][j];

}

}

}

class Sa

{

public:

unsigned int linie;

unsigned int coloana;

Sa()

{

linie = 0;

coloana = 0;

}

};

int celMaiMic(Matrix matrice, unsigned int index\_linie, unsigned int coloane, int& coloana)

{

int rezultat = matrice[index\_linie][0];

for (size\_t i = 1; i < coloane; i++)

{

if (matrice[index\_linie][i] < rezultat)

{

rezultat = matrice[index\_linie][i];

coloana = i;

}

}

return rezultat;

}

bool esteCelMaiMarePeColoana(Matrix matrice,unsigned int index\_coloana ,int valoare, unsigned int linii, unsigned int coloane)

{

bool result = true;

for (size\_t i = 0; i < linii; i++)

{

if (matrice[i][index\_coloana] > valoare)

{

return result = false;

break;

}

}

return result;

}

void exercitiul4(Sa temporar[], Matrix matrice, unsigned int linii, unsigned int coloane)

{

int celMaiMicPeLinie{};

int indexColoana{};

for (unsigned int i = 0; i < linii; i++)

{

celMaiMicPeLinie = celMaiMic(matrice, i, coloane, indexColoana);

//std::cout << "cel mai mic pe linie: " << celMaiMicPeLinie << std::endl;

if (esteCelMaiMarePeColoana(matrice, indexColoana, celMaiMicPeLinie, linii, coloane))

{

//std::cout << "cel mai mare pe coloana: " << matrice[i][indexColoana] << std::endl;

temporar[i].linie = i;

temporar[i].coloana = indexColoana;

}

}

}

int main()

{

unsigned int rows = 3;

unsigned int columns = 3;

Matrix test = InstantiateMatrix(rows,columns);

ManualMatrixGeneration(test, rows, columns);

PrintMatrix(test, rows, columns);

Sa testSa[3];

exercitiul4(testSa, test, rows, columns);

for (size\_t i = 0; i < 3; i++)

{

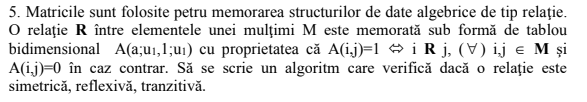
std::cout << "Sa[" << i << "] = " << "[l: " << testSa[i].linie << " c: " << testSa[i].coloana << "]" << std::endl;

}

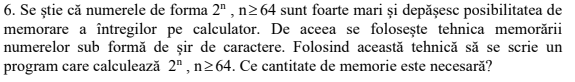
return 0;

}

# Exercitiul 5



# Exercitiul 6



# Exercitiul 7



Polinom Polinom::Impartire(Polinom P, Polinom Q, Polinom H, Polinom R)

{

float u;

int x;

Polinom nr(P.getSize());

nr = P;

Polinom nh(P.getSize() - Q.getSize());

for (size\_t i = nh.getSize(); i >= 0; i--)

{

nh.polinom[i].coeficient = nr.polinom[nh.getSize()].coeficient / Q.polinom[Q.getSize()].coeficient;

for (size\_t j = i; j <= nh.getSize(); j++)

{

u = nh.polinom[i].coeficient \* Q.polinom[j - i].coeficient;

nr.polinom[j].coeficient = nr.polinom[j].coeficient - u;

}

}

H = nh;

R = nr;

}

# Exercitiul 8



void Polinom::Impartire(Polinom P, Polinom Q, Polinom& H, Polinom& R)

{

float u;

int x;

Polinom nr(P.getSize());

nr = P;

Polinom nh(P.getSize() - Q.getSize());

for (size\_t i = nh.getSize(); i >= 0; i--)

{

nh.polinom[i].coeficient = nr.polinom[nh.getSize()].coeficient / Q.polinom[Q.getSize()].coeficient;

for (size\_t j = i; j <= nh.getSize(); j++)

{

u = nh.polinom[i].coeficient \* Q.polinom[j - i].coeficient;

nr.polinom[j].coeficient = nr.polinom[j].coeficient - u;

}

}

H = nh;

R = nr;

}

void Polinom::Cmmdc(Polinom P, Polinom& Q, Polinom& R)

{

Polinom v(1);

while (1)

{

Impartire(P, Q, v, R);

if (R.getSize() == 0 && R.polinom[0].coeficient == 0)

{

break;

}

else

{

P = Q;

}

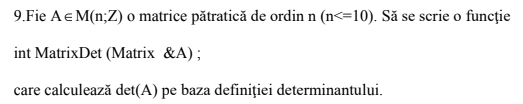
Q = R;

}

R = Q;

}

# Exercitiul 9



int MatrixDet(Matrix rareMatrix, unsigned int rows, unsigned int columns) {

if (rows != columns) {

throw std::runtime\_error("Matricea nu este patratica");

}

int dimension = rows;

if (dimension == 0) {

return 1;

}

if (dimension == 1) {

return rareMatrix[0][0];

}

// rows/columns = 2

if (dimension == 2) {

return rareMatrix[0][0] \* rareMatrix[1][1] - rareMatrix[0][1] \* rareMatrix[1][0];

}

Int result = 0;

int sign = 1;

unsigned int subMatrix\_rows = dimension - 1;

unsigned int subMatrix\_columns = dimension - 1;

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

//Submatrix

Matrix subMatrix = InstantiateMatrix(subMatrix\_rows, subMatrix\_columns);

for (int m = 1; m < dimension - 1; m++) {

int z = 0;

for (int n = 0; n < dimension - 1; n++) {

if (n != i) {

subMatrix[m - 1][z] = subMatrix[m][n];

z++;

}

}

}

//recursive call

result = result + sign \* subMatrix[0][i] \* MatrixDet (subMatrix, subMatrix\_rows, subMatrix\_columns);

sign = -sign;

}

return result;

}