Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №3**

З дисципліни «Паралельне програмування»

**Потоки в WinAPI**

ВИКОНАВ:

Студент ІІІ курсу ФІОТ

Групи ІО-91

Діденко В.В.

ПЕРЕВІРИВ:

Корочкін О.В.

Київ 2021 р.

**Варіант завдання:**

F1: A = B + C + D\*(MD\*ME)

F2: MG = SORT(MF - MH \* MK)

F3: S = SORT(O\*MO)\*(MS \*MT)

**Виконання роботи**

**Лістинг програми:**

**Lab4.cpp**

#include <iostream>

#include "F1.h"

#include "F2.h"

#include "F3.h"

#include <omp.h>

/\*\*

\* Parallel computing.

\* Labwork 4. OpenMP.

\*

\* F1 1.12 A = B + C + D \* (MD \* ME)

\* F2 2.24 MG = sort(MF - MH \* MK)

\* F3 3.21 S = sort(O \* MO) \* (MS \* MT)

\*

\* Didenko Vladyslav

\* IO-91

\* 10.11.2021

\*/

int N = 5;

int main()

{

std::cout << "Lab 4 started!\n";

F1\* T1 = new F1(N);

F2\* T2 = new F2(N);

F3\* T3 = new F3(N);

int tid;

#pragma omp parallel num\_threads(3)

{

tid = omp\_get\_thread\_num();

switch (tid) {

case 0:

T1->run();

case 1:

T2->run();

case 2:

T3->run();

}

}

std::cout << "Lab 4 end\n" << "Press Enter...";

}

**F1.cpp**

#include "F1.h"

F1::F1(int N) {

this->N = N;

}

Vector\* F1::getResult() {

return result;

}

// F1 1.12 A = B + C + D \* (MD \* ME)

void F1::run() {

cout << "Task 1 start\n";

Vector \*B = new Vector(N), \*C = new Vector(N), \*D = new Vector(N) ;

Matrix \*MD = new Matrix(N), \*ME = new Matrix(N);

result = B->sum(C->sum(MD->multiply(ME)->multiply(D)));

cout << "Thread 1 result:\n" << endl << result->toString() << endl;

cout << "Task 1 end\n";

delete B;

delete C;

delete D;

delete MD;

delete ME;

}

**F2.cpp**

#include "F2.h"

F2::F2(int N) {

this->N = N;

}

Matrix\* F2::getResult() {

return result;

}

// F2 2.24 MG = sort(MF - MH \* MK)

void F2::run() {

cout << "Task 2 start\n";

Matrix \*MF = new Matrix(N), \*MH = new Matrix(N), \*MK = new Matrix(N);

result = MF->sub(MH->multiply(MK));

result = result->sort();

cout << "Thread 2 result:\n" << endl << result->toString() << endl;

cout << "Task 2 end\n";

delete MF;

delete MH;

delete MK;

}

**F3.cpp**

#include "F3.h"

F3::F3(int N) {

this->N = N;

}

Vector\* F3::getResult() {

return result;

}

// F3 3.21 S = sort(O \* MO) \* (MS \* MT)

void F3::run() {

cout << "Task 3 start\n";

Vector \*O = new Vector(N);

Matrix \*MO = new Matrix(N), \*MS = new Matrix(N), \*MT = new Matrix(N);

result = MS->multiply(MT)->multiply(MO->multiply(O)->sort());

cout << "Thread 3 result:\n" << endl << result->toString() << endl;

cout << "Task 3 end\n";

delete O;

delete MO;

delete MS;

delete MT;

}

**Matrix.cpp**

#include "Matrix.h"

#include <iostream>

Matrix::Matrix(int N) {

this->N = N;

srand(time(NULL));

grid = new long\*[N];

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

grid[i] = new long[N];

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

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

grid[i][k] = 1;//rand() % 20;

}

Matrix::Matrix(long\*\* grid, int N) {

this->N = N;

this->grid = new long\*[N];

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

this->grid[i] = new long[N];

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

this->grid[i][k] = grid[i][k];

}

}

Matrix::~Matrix() {

int N = getSize();

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

delete[] grid[i];

delete[] grid;

}

long Matrix::get(int i, int k) {

return grid[i][k];

}

int Matrix::getSize() {

return N;

}

Matrix\* Matrix::multiply(Matrix\* m) {

int N = getSize();

long\*\* newGrid = new long\*[N];

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

newGrid[i] = new long[N];

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

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

newGrid[i][k] = 0;

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

newGrid[i][k] += grid[i][j] \* m->get(j, k);

}

}

}

Matrix\* newMatrix = new Matrix(newGrid, N);

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

delete[] newGrid[i];

delete[] newGrid;

return newMatrix;

}

Vector\* Matrix::multiply(Vector\* v) {

int N = getSize();

long\* newGrid = new long[N];

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

newGrid[i] = 0;

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

newGrid[i] += v->get(k) \* grid[i][k];

}

}

Vector\* newVector = new Vector(newGrid, N);

delete[] newGrid;

return newVector;

}

Matrix\* Matrix::multiply(long \*a) {

int N = getSize();

long\*\* newGrid = new long\*[N];

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

newGrid[i] = new long[N];

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

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

newGrid[i][k] = grid[i][k] \* \*a;

}

}

Matrix\* newMatrix = new Matrix(newGrid, N);

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

delete[] newGrid[i];

delete[] newGrid;

return newMatrix;

}

Matrix\* Matrix::sum(Matrix\* m) {

int N = getSize();

long\*\* newGrid = new long\*[N];

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

newGrid[i] = new long[N];

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

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

newGrid[i][k] = grid[i][k] + m->get(i, k);

}

}

Matrix\* newMatrix = new Matrix(newGrid, N);

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

delete[] newGrid[i];

delete[] newGrid;

return newMatrix;

}

Matrix\* Matrix::sub(Matrix\* m) {

int N = getSize();

long\*\* newGrid = new long\*[N];

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

newGrid[i] = new long[N];

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

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

newGrid[i][k] = grid[i][k] - m->get(i, k);

}

}

Matrix\* newMatrix = new Matrix(newGrid, N);

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

delete[] newGrid[i];

delete[] newGrid;

return newMatrix;

}

Matrix\* Matrix::sort() {

int N = getSize();

long\*\* newGrid = new long\*[N];

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

newGrid[i] = grid[i];

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

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

for (int k = 0; k < N - i - 1; ++k) {

if (newGrid[j][k] > newGrid[j][k + 1]) {

long t = newGrid[j][k];

newGrid[j][k] = newGrid[j][k + 1];

newGrid[j][k + 1] = t;

}

}

}

}

Matrix\* newMatrix = new Matrix(newGrid, N);

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

delete[] newGrid[i];

delete[] newGrid;

return newMatrix;

}

long Matrix::get\_min() {

long res = grid[0][0];

int N = getSize();

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

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

if (res < grid[i][k])

res = grid[i][k];

}

}

return res;

}

long Matrix::get\_max() {

long res = grid[0][0];

int N = getSize();

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

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

if (res > grid[i][k])

res = grid[i][k];

}

}

return res;

}

string Matrix::toString() {

string res = "";

int N = getSize();

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

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

res += std::to\_string(grid[i][k]) + "\t";

}

res += "\n";

}

return res;

}

**Vector.cpp**

#include "Vector.h"

#include <iostream>

Vector::Vector() {}

Vector::Vector(int N) {

this->N = N;

srand(time(NULL));

grid = new long[N];

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

grid[i] = 1;//rand() % 20;

}

Vector::Vector(long\* grid, int N) {

this->N = N;

this->grid = new long[N];

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

{

this->grid[i] = grid[i];

//cout << grid[i] << endl;

}

}

Vector::~Vector() {

delete[] grid;

}

int Vector::getSize() {

return N;

}

long Vector::get(int i) {

return grid[i];

}

Vector\* Vector::sum(Vector\* v) {

int N = getSize();

long\* newGrid = new long[N];

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

newGrid[i] = grid[i] + v->get(i);

Vector\* newVector = new Vector(newGrid, N);

delete[] newGrid;

return newVector;

}

Vector\* Vector::sub(Vector\* v) {

int N = getSize();

long\* newGrid = new long[N];

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

newGrid[i] = grid[i] - v->get(i);

Vector\* newVector = new Vector(newGrid, N);

delete[] newGrid;

return newVector;

}

Vector\* Vector::multiply(long\* a) {

int N = getSize();

long\* newGrid = new long[N];

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

newGrid[i] = \*a \* grid[i];

}

Vector\* newVector = new Vector(newGrid, N);

delete[] newGrid;

return newVector;

}

long Vector::multiply(Vector\* v) {

int N = getSize();

long result = 0L;

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

result += v->get(i) \* grid[i];

}

return result;

}

Vector\* Vector::sort() {

int N = getSize();

long\* newGrid = new long[N];

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

newGrid[i] = grid[i];

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

for (int k = 0; k < N - i - 1; ++k) {

if (newGrid[k] > newGrid[k + 1]) {

long t = newGrid[k];

newGrid[k] = newGrid[k + 1];

newGrid[k + 1] = t;

}

}

}

Vector\* newVector = new Vector(newGrid, N);

delete[] newGrid;

return newVector;

}

string Vector::toString() {

string res = "";

int N = getSize();

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

res += std::to\_string(grid[i]) + " ";

return res;

}

**F1.h**

#pragma once

#include <iostream>

#include <windows.h>

#include "Matrix.h"

#include "Vector.h"

class F1 {

private:

Vector\* result;

int N;

public:

F1(int N);

Vector\* getResult();

void run();

};

**F2.h**

#pragma once

#include <iostream>

#include <windows.h>

#include "Matrix.h"

#include "Vector.h"

class F2 {

private:

Matrix\* result;

int N;

public:

F2(int N);

Matrix\* getResult();

void run();

};

**F3.h**

#pragma once

#include <iostream>

#include <windows.h>

#include "Matrix.h"

#include "Vector.h"

class F3 {

private:

Vector\* result;

int N;

public:

F3(int N);

Vector\* getResult();

void run();

};

**Matrix.h**

#pragma once

#include <random>

#include <ctime>

#include "Vector.h"

class Matrix {

private:

long\*\* grid;

int N;

public:

Matrix(int N);

Matrix(long\*\* grid, int N);

~Matrix();

long get(int i, int k);

int getSize();

Matrix\* multiply(Matrix\* m);

Vector\* multiply(Vector\* v);

Matrix\* multiply(long\* a);

Matrix\* sum(Matrix\* m);

Matrix\* sub(Matrix\* m);

Matrix\* sort();

long get\_min();

long get\_max();

string toString();

};

**Vector.h**

#pragma once

#include <random>

#include <ctime>

#include <string>

using namespace std;

class Vector {

private:

long\* grid;

int N;

public:

Vector();

Vector(int N);

Vector(long\* grid, int N);

~Vector();

int getSize();

long get(int i);

Vector\* sum(Vector\* v);

Vector\* sub(Vector\* v);

Vector\* multiply(long\* a);

long multiply(Vector\* v);

Vector\* sort();

string toString();

};

**Результат виконання програми:**

