НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
ФАКУЛЬТЕТ ІНФОРМАТИКИ І ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Лабораторна робота №5

з дисципліни «Паралельні та розподілені обчислення»

Виконав:
студент 3 курсу гр. ІО-42
Кочетов Данило
№ ЗК 4213

Перевірив:
Долголенко О. М.

Київ 2016 р.

Завдання:
1.13; 2.13; 3.13
F1: C = A*(MA*ME) + B + D
F2: ML = MIN(MF)*MG + MAX(MH) * (MK*MF)
F3: T = (MO*MP)*S + MR*SORT(S)

Лістинг програми:

// Lab5.cpp

```
#include <omp.h>
#include "F1.h"
#include "F2.h"
#include "F3.h"

const int N = 500;

int main() {
	cout << "Lab 5 start" << endl << endl;
	F1 f1 = F1(N);
	F2 f2 = F2(N);
	F3 f3 = F3(N);
	int tid;
	#pragma omp parallel num_threads(3)
	{
		tid = omp_get_thread_num();
		switch (tid) {
		case 0:
			f1.run();
			break;
		case 1:
			f2.run();
			break;
		case 2:
			f3.run();
			break;
		}
	}

	cout << endl << "Lab 5 end" << endl << endl;
	cout << "Press Enter...";
	string t;
	getline(cin, t);

}
```

// F1.h

```
#pragma once
#include <iostream>
#include "Matrix.h"

class F1 {
private:
	Vector* result;
	int N;
public:
	F1(int N);
	Vector* getResult();
	void run();
};
```

// F1.cpp

```
#include "F1.h"

F1::F1(int N) {
	this->N = N;
}

Vector* F1::getResult() {
	return result;
}

void F1::run() {
```

```
        cout << "Task 1 start\n";
        Vector *A = new Vector(N), *B = new Vector(N), *D = new Vector(N);
        Matrix *MA = new Matrix(N), *ME = new Matrix(N);
        result = MA->multiply(ME)->multiply(A)->sum(B)->sum(D);
        cout << "Task 1 end\n";
        delete A;
        delete B;
        delete D;
        delete MA;
        delete ME;
}
```

// F2.h

```
#pragma once
#include <iostream>
#include "Matrix.h"

class F2 {
private:
        Matrix* result;
        int N;
public:
        F2(int N);
        Matrix* getResult();
        void run();
};
```

// F2.cpp

```
#include "F2.h"

F2::F2(int N) {
        this->N = N;
}

Matrix* F2::getResult() {
        return result;
}

void F2::run() {
        cout << "Task 2 start\n";
        Matrix *MF = new Matrix(N), *MG = new Matrix(N), *MH = new Matrix(N), *MK = new Matrix(N);
        result = MG->multiply(MF->get_min())->sum(MK->multiply(MF)->multiply(MH->get_max()));
        cout << "Task 2 end\n";
        delete MF;
        delete MG;
        delete MH;
        delete MK;
}
```

// F3.h

```
#pragma once
#include <iostream>
#include "Matrix.h"

class F3 {
private:
        Vector* result;
        int N;
public:
        F3(int N);
        Vector* getResult();
        void run();
};
```

// F3.cpp

```
#include "F3.h"

F3::F3(int N) {
        this->N = N;
}

Vector* F3::getResult() {
        return result;
}

void F3::run() {
```

```
        cout << "Task 3 start\n";
        Vector* S = new Vector(N);
        Matrix *MO = new Matrix(N), *MP = new Matrix(N), *MR = new Matrix(N);
        result = MO->multiply(MP)->multiply(S)->sum(MR->multiply(S->sort()));
        cout << "Task 3 end\n";
        delete S;
        delete MO;
        delete MP;
        delete MR;
}
```

// 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* sort();
        string toString();
};
```

// Vector.cpp

```
#include "Vector.h"

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] = 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];
}

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::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 += grid[i] + " ";
        return res;
}
```

// 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);
        long get_min();
        long get_max();
        string toString();
};
```

// 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] = 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;
}
```

```
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 += grid[i][k] + "\t";
		}
		res += "\n";
	}
	return res;
}
```