НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ ІНФОРМАТИКИ І ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

**Лабораторна робота №3**

з дисципліни **«**Паралельні та розподілені обчислення**»**

Виконав:

студент 3 курсу гр. ІО-31

Долинний Олександр

№ ЗК 3110

Перевірив:

Корочкін О. В.

Київ 2015 р.

***Завдання:***

1.1 3.21 3.11

F1: A = sort(B) \* (MB \* MC)

F2: W = sort(R \* MT) \* (MX \* MS)

F3: R = sort(S + T) \* trans(MS \* MR)

***Лістинг програми:***

**package** pro.lab3;  
  
*/\*\*  
Parallel and distributed computing.  
Labwork 2. Threads in Java  
 Dolinniy Alexandr  
IO-31  
19.10.2015  
  
Func1: A = sort(B)\*(MB\*MC)  
Func2: W = sort(R\*MT)\*(MX\*MS)  
Func3: R = sort(S+T)\*trans(MS\*MR)  
\*/***public class** Lab3 {  
  
 **public static void** main(String[] args) {  
 **int** n = 1000;  
 Data data = **new** Data(n);  
 T1 t1 = **new** T1(data);  
 t1.setPriority(Thread.***MAX\_PRIORITY***);  
 T2 t2 = **new** T2(data);  
 t2.setPriority(6);  
 T3 t3 = **new** T3(data);  
 t3.setPriority(Thread.***MIN\_PRIORITY***);  
 t1.start();  
 t2.start();  
 t3.start();  
 **try** {  
 t1.join();  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 **try** {  
 t2.join();  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 **try** {  
 t3.join();  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 System.***out***.println(**"Lab3 finished"**);  
 }  
}

**package** pro.lab3;  
  
**import** java.util.Arrays;  
**import** java.util.Scanner;  
  
*/\*\*  
 \* Parallel and distributed computing.  
 \* Labwork 2. Threads in Java  
 \* Dolinniy Alexandr  
 \* IO-31  
 \* 19.10.2015  
 \*   
 \* Func1: A = sort(B)\*(MB\*MC)  
 \* Func2: W = sort(R\*MT)\*(MX\*MS)  
 \* Func3: R = sort(S+T)\*trans(MS\*MR)  
 \*/***public class** Data {  
  
 **private int n**;  
  
 **public int** getN() {  
 **return n**;  
 }  
  
 **public** Data(**int** n) {  
 **this**.**n** = n;  
 }  
  
 **public int**[] vectorInput() {  
 **int**[] vector = **new int**[**n**];  
 Scanner sc = **new** Scanner(System.***in***);  
 **for** (**int** i = 0; i < **n**; i++) {  
 vector[i] = sc.nextInt();  
 }  
 **return** vector;  
 }  
  
 **public int**[][] matrixInput() {  
 **int**[][] matrix = **new int**[**n**][**n**];  
 Scanner sc = **new** Scanner(System.***in***);  
 **for** (**int** i = 0; i < **n**; i++) {  
 **for** (**int** j = 0; j < **n**; j++) {  
 matrix[i][j] = sc.nextInt();  
 }  
 }  
 **return** matrix;  
 }  
  
 **public int**[] func1(**int**[] b, **int**[][] mb, **int**[][] mc) {  
  
 **return** vectorMatrixMult(b, matrixMult(mb, mc));  
 }  
  
 **public int**[] func2(**int**[] r, **int**[][] mt, **int**[][] mx, **int**[][] ms) {  
  
 **return** vectorMatrixMult(vectorSort(vectorMatrixMult(r, mt)), matrixMult(mx, ms));  
 }  
  
 **public int**[] func3(**int**[] s, **int**[] t, **int**[][] ms, **int**[][] mr) {  
  
 **return** vectorMatrixMult(vectorAdd(s, t), matrixTrans(matrixMult(ms, mr)));  
 }  
  
 **private int**[] vectorAdd(**int**[] a, **int**[] b) {  
 **if** (a.**length** != **n** || b.**length** != **n**) {  
 **return null**;  
 }  
 **int**[] c = **new int**[**n**];  
 **for** (**int** i = 0; i < **n**; i++) {  
 c[i] = a[i] + b[i];  
 }  
 **return** c;  
 }  
  
 **private int**[][] matrixTrans(**int**[][] ma) {  
 **int** buf;  
 **for** (**int** i = 0; i < ma.**length**; i++) {  
 **for** (**int** j = 0; j <= i; j++) {  
 buf = ma[i][j];  
 ma[i][j] = ma[j][i];  
 ma[j][i] = buf;  
 }  
 }  
 **return** ma;  
 }  
  
 **private int**[] vectorMatrixMult(**int**[] a, **int**[][] ma) {  
 **if** (a.**length** != **n** || ma.**length** != **n**) {  
 **return null**;  
 }  
 **int**[] c = **new int**[**n**];  
 **for** (**int** i = 0; i < **n**; i++) {  
 **for** (**int** j = 0; j < **n**; j++) {  
 c[i] += a[j] \* ma[j][i];  
 }  
 }  
 **return** c;  
 }  
  
 **private int**[][] matrixMult(**int**[][] ma, **int**[][] mb) {  
 **if** (ma.**length** != **n** || mb.**length** != **n**) {  
 **return null**;  
 }  
 **int**[][] c = **new int**[**n**][**n**];  
 **for** (**int** i = 0; i < **n**; i++) {  
 **for** (**int** j = 0; j < **n**; j++) {  
 **for** (**int** k = 0; k < **n**; k++) {  
 c[i][j] += ma[i][k] \* mb[k][j];  
 }  
 }  
 }  
 **return** c;  
 }  
  
 **private int**[] vectorSort(**int**[] a) {  
 **if** (a.**length** != **n**) {  
 **return null**;  
 }  
 **int**[] c = Arrays.*copyOf*(a, **n**);  
 Arrays.*sort*(c);  
 **return** c;  
 }  
  
}

**package** pro.lab3;  
  
**import** java.util.Arrays;  
  
*/\*\*  
 \* Parallel and distributed computing.  
 \* Labwork 2. Threads in Java  
 \* Dolinniy Alexandr  
 \* IO-31  
 \* 19.10.2015  
 \*   
 \* Func1: A = sort(B)\*(MB\*MC)  
 \* Func2: W = sort(R\*MT)\*(MX\*MS)  
 \* Func3: R = sort(S+T)\*trans(MS\*MR)  
 \*/***class** T1 **extends** Thread {  
  
 **int**[] **a**, **b**;  
 **int**[][] **mb**, **mc**;  
 Data **d**;  
  
  
 **public** T1(Data d) {  
 **this**.**d** = d;  
 }  
  
 **public void** run() {  
 System.***out***.println(**"Task 1 started"**);  
  
 **int** n = **d**.getN();  
  
 **b** = **new int**[n];  
 Arrays.*fill*(**b**, 1);  
 **mb** = **new int**[n][n];  
 **mc** = **new int**[n][n];  
 **for** (**int** i = 0; i < n; i++) {  
 Arrays.*fill*(**mb**[i], 1);  
 Arrays.*fill*(**mc**[i], 1);  
 }  
 **try** {  
 *sleep*(1000);  
 } **catch** (InterruptedException e1) {  
 e1.printStackTrace();  
 }  
 **a** = **d**.func1(**b**, **mb**, **mc**);  
 System.***out***.println(**"F1:"**);  
 **if**(n<=7)  
 System.***out***.println(Arrays.*toString*(**a**));  
 System.***out***.println(**"Task 1 finished"**);  
 }  
}

**package** pro.lab3;  
  
  
**import** java.util.Arrays;  
  
*/\*\*  
 \* Parallel and distributed computing.  
 \* Labwork 2. Threads in Java  
 \* Dolinniy Alexandr  
 \* IO-31  
 \* 19.10.2015  
 \*   
 \* Func1: A = sort(B)\*(MB\*MC)  
 \* Func2: W = sort(R\*MT)\*(MX\*MS)  
 \* Func3: R = sort(S+T)\*trans(MS\*MR)  
 \*/***class** T2 **extends** Thread {  
  
 **int**[] **w**, **r**;  
 **int**[][] **mt**, **mx**, **ms**;  
 Data **d**;  
  
  
 **public** T2(Data d) {  
 **this**.**d** = d;  
 }  
  
 **public void** run() {  
 System.***out***.println(**"Task 2 started"**);  
  
 **int** n = **d**.getN();  
 **r** = **new int**[n];  
 Arrays.*fill*(**r**, 1);  
 **mt** = **new int**[n][n];  
 **mx** = **new int**[n][n];  
 **ms** = **new int**[n][n];  
 **for** (**int** i = 0; i < n; i++) {  
 Arrays.*fill*(**mt**[i], 1);  
 Arrays.*fill*(**mx**[i], 1);  
 Arrays.*fill*(**ms**[i], 1);  
 }  
 **try** {  
 *sleep*(1000);  
 } **catch** (InterruptedException e1) {  
 e1.printStackTrace();  
 }  
 **w** = **d**.func2(**r**, **mt**, **mx**, **ms**);  
 System.***out***.println(**"F2:"**);  
 **if**(n<=7)  
 System.***out***.println(Arrays.*toString*(**w**));  
 System.***out***.println(**"Task 2 finished"**);  
 }  
}

**package** pro.lab3;  
  
**import** java.util.Arrays;  
  
*/\*\*  
 \* Parallel and distributed computing.  
 \* Labwork 2. Threads in Java  
 \* Dolinniy Alexandr  
 \* IO-31  
 \* 19.10.2015  
 \*   
 \* Func1: A = sort(B)\*(MB\*MC)  
 \* Func2: W = sort(R\*MT)\*(MX\*MS)  
 \* Func3: R = sort(S+T)\*trans(MS\*MR)  
 \*/***class** T3 **extends** Thread {  
 **int**[] **r**, **s**, **t**;  
 **int**[][] **ms**, **mr**;  
 Data **d**;  
  
  
 **public** T3(Data d) {  
 **this**.**d** = d;  
 }  
  
 **public void** run() {  
 System.***out***.println(**"Task 3 started"**);  
  
 **int** n = **d**.getN();  
 **s** = **new int**[n];  
 Arrays.*fill*(**s**, 1);  
 **t** = **new int**[n];  
 Arrays.*fill*(**t**, 1);  
 **ms** = **new int**[n][n];  
 **mr** = **new int**[n][n];  
 **for** (**int** i = 0; i < n; i++) {  
 Arrays.*fill*(**ms**[i], 1);  
 Arrays.*fill*(**mr**[i], 1);  
 }  
 **try** {  
 *sleep*(1000);  
 } **catch** (InterruptedException e1) {  
 e1.printStackTrace();  
 }  
 **r** = **d**.func3(**s**, **t**, **ms**, **mr**);  
 System.***out***.println(**"F3:"**);  
 **if**(n<=7)  
 System.***out***.println(Arrays.*toString*(**r**));  
 System.***out***.println(**"Task 3 finished"**);  
 }  
}