Lab 2(Operating systems)

Multithreaded matrix multiplication

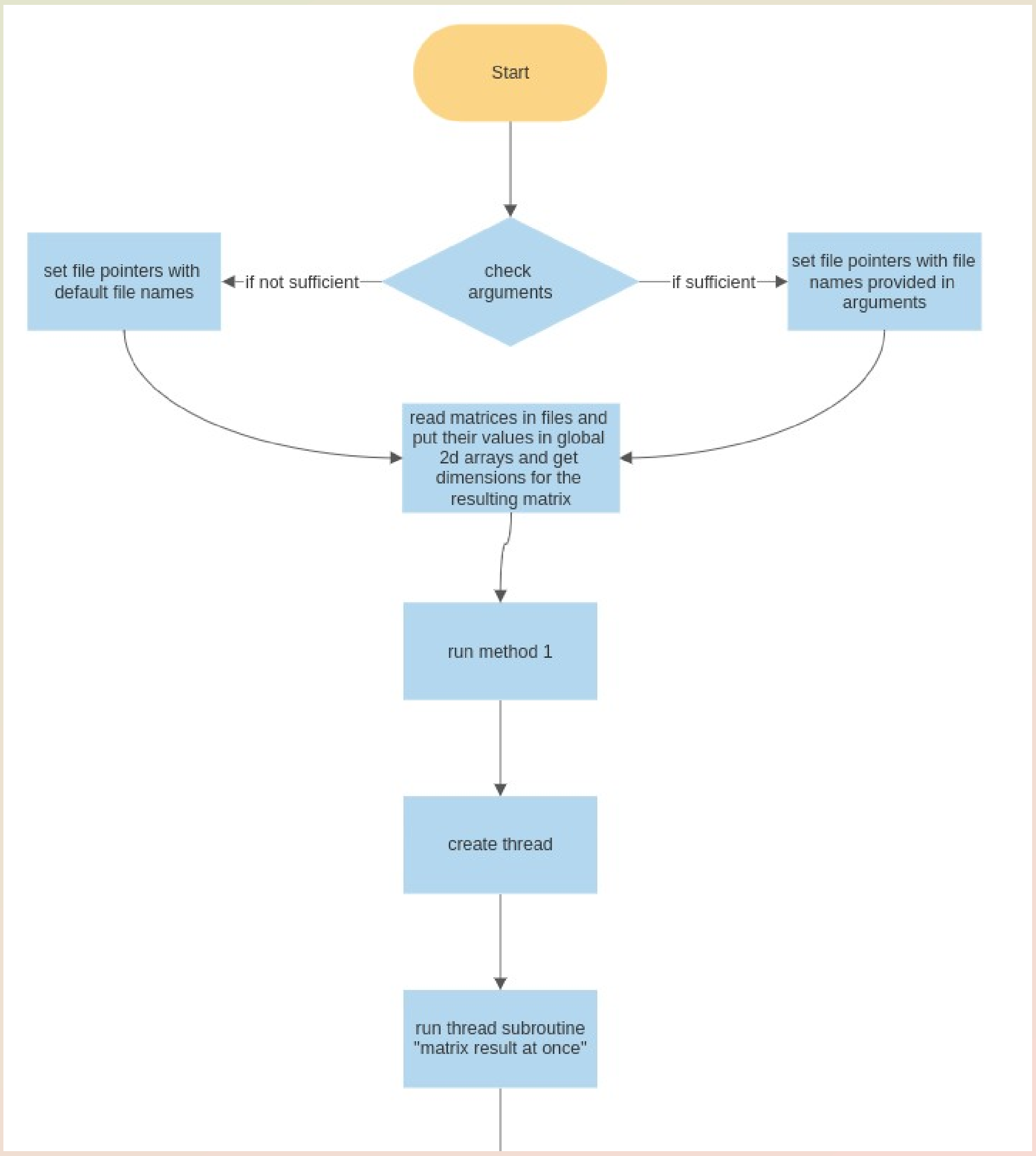
Student name & ID:

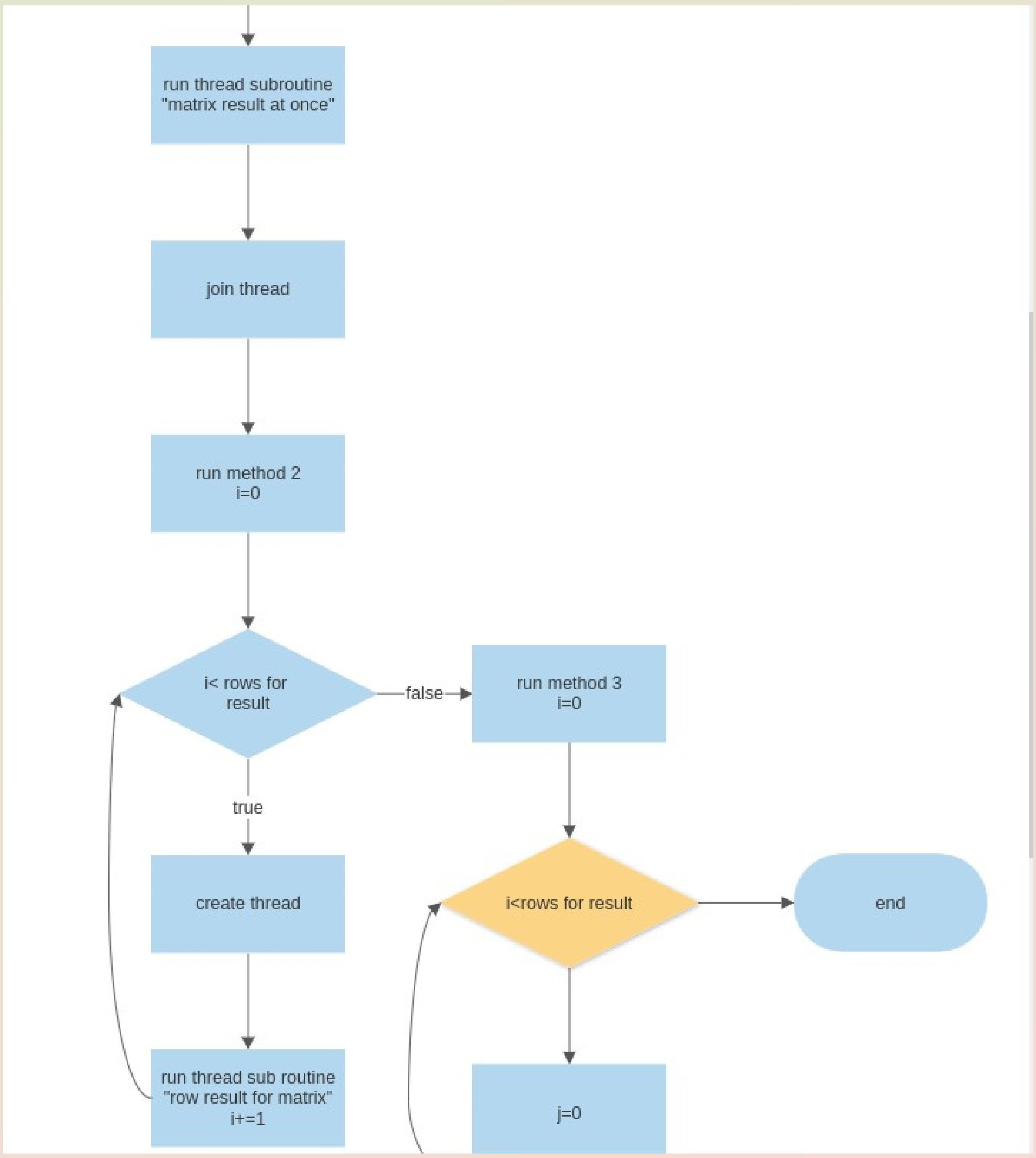
كريم فتحي عبد العزيز (20011116)

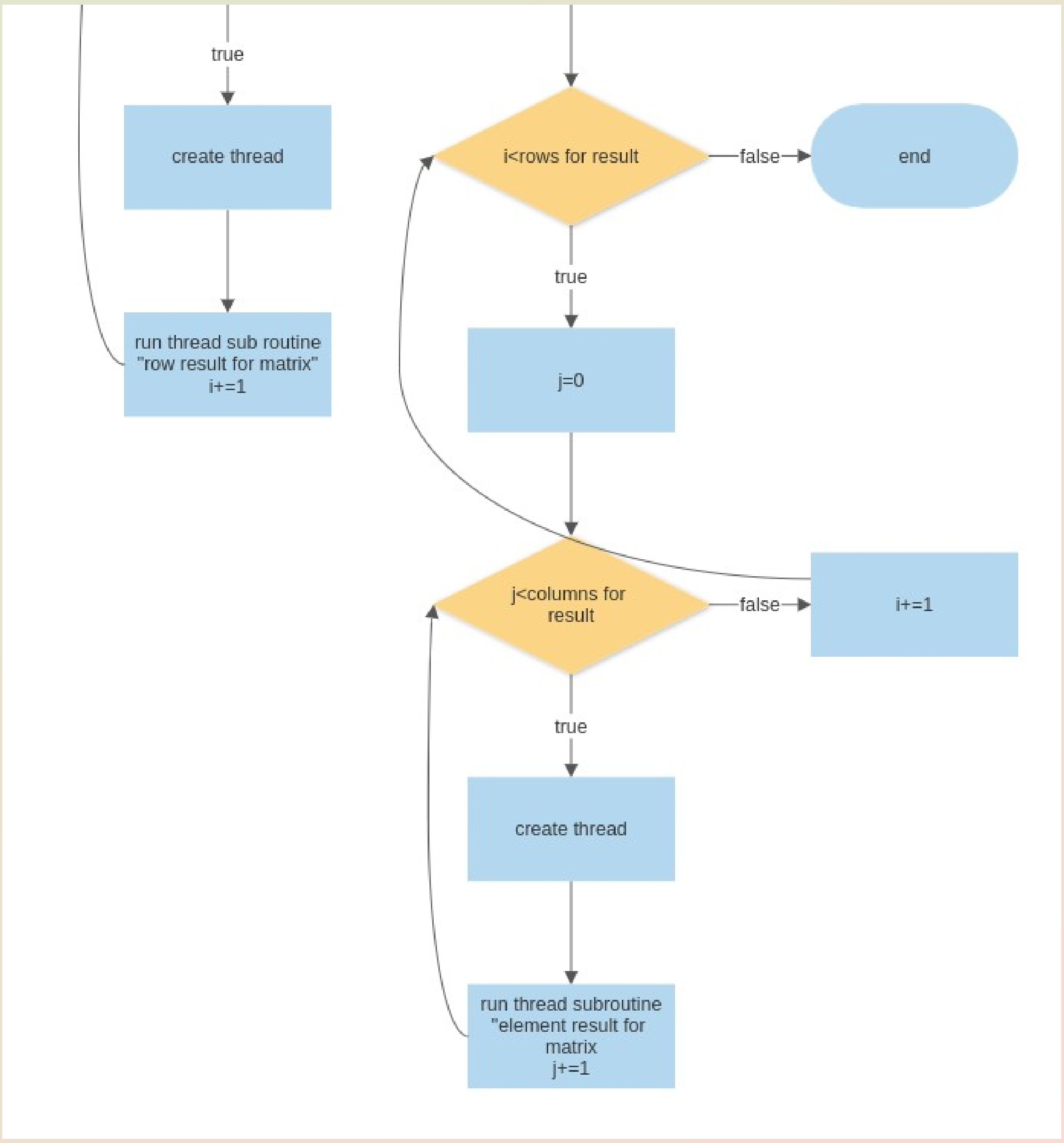
test cases video

1) Code organization:

a) Flow chart:


Start
check arguments
if not sufficient
set file pointers with default file names
if sufficient
set file pointers with file names provided in arguments
read matrices in files and put their values in global 2d arrays and get dimensions for the resulting matrix
run method 1
create thread
run thread subroutine "matrix result at once"

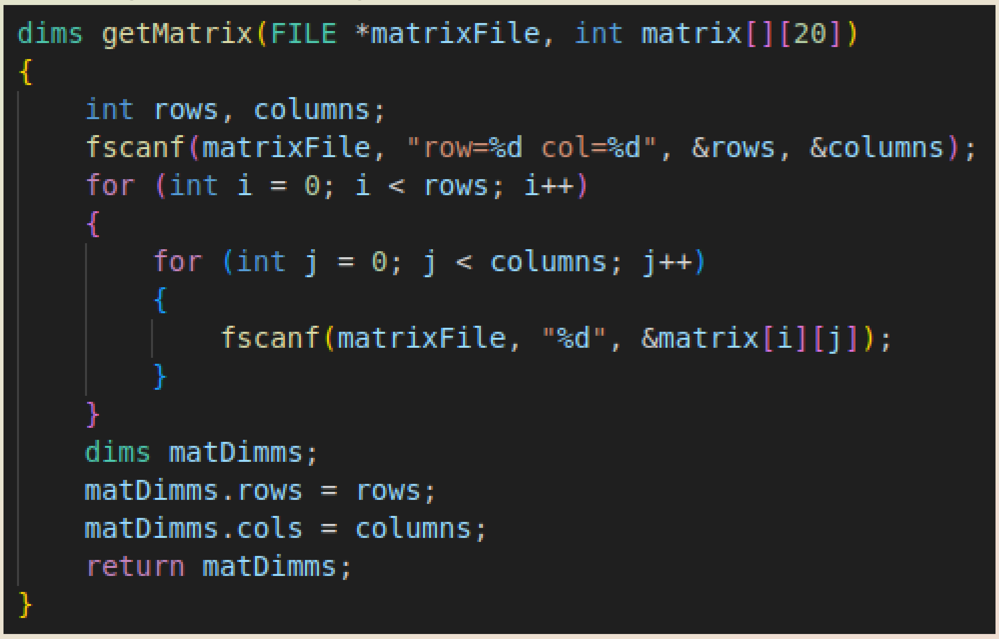

run thread subroutine "matrix result at once"
join thread
run method 2 i=0
i< rows for result
false
true
run method 3 i=0
create thread
i<rows for result
end
run thread sub routine "row result for matrix" i+=1
j=0


true
create thread
run thread sub routine "row result for matrix" i+=1
i<rows for result
false
end
true
j=0
j<columns for result
false
i+=1
true
create thread
run thread subroutine "element result for matrix j+=1

b) Project files:
   - all functions,structs and pointers are in one file called "matMultp.c", needed functions are relatively small no need for "*.h" files.Therefore ,no dependency management is needed

2) Main functions:

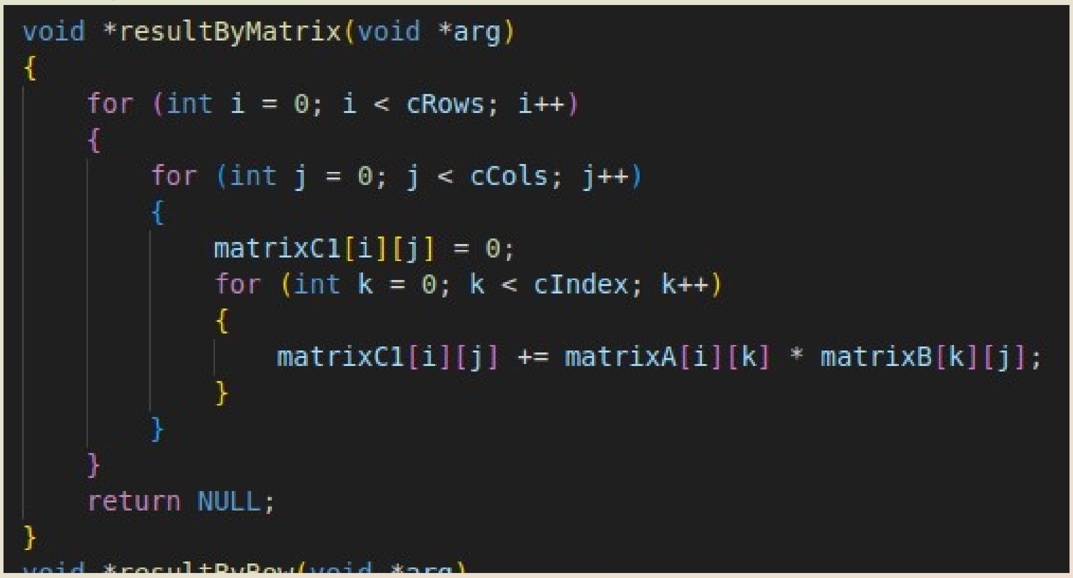
a) Reading matrix from given file:



```c
dims getMatrix(FILE *matrixFile, int matrix[][20])
{
    int rows, columns;
    fscanf(matrixFile, "row=%d col=%d", &rows, &columns);
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < columns; j++)
        {
            fscanf(matrixFile, "%d", &matrix[i][j]);
        }
    }
    dims matDimms;
    matDimms.rows = rows;
    matDimms.cols = columns;
    return matDimms;
}
```

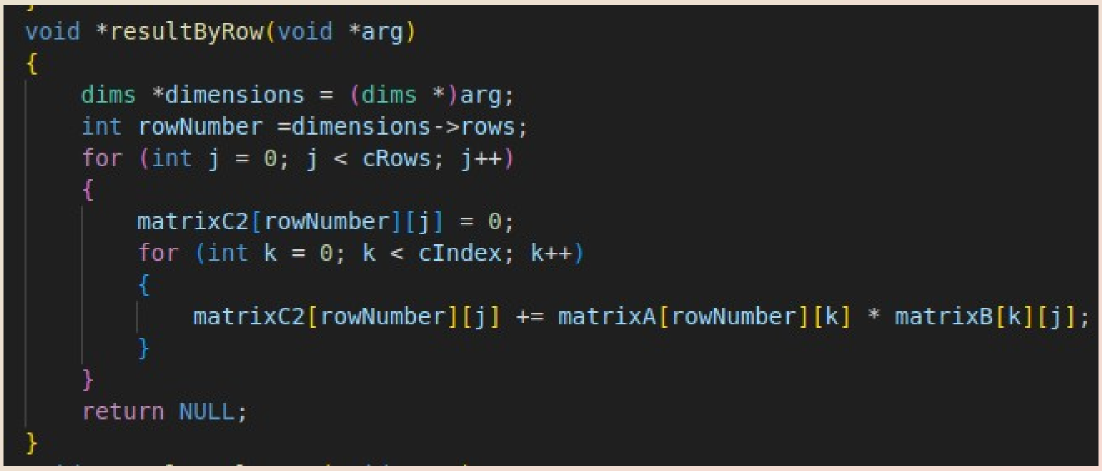
b) Threads subroutines:

i) Getting the result of the whole matrix at once:



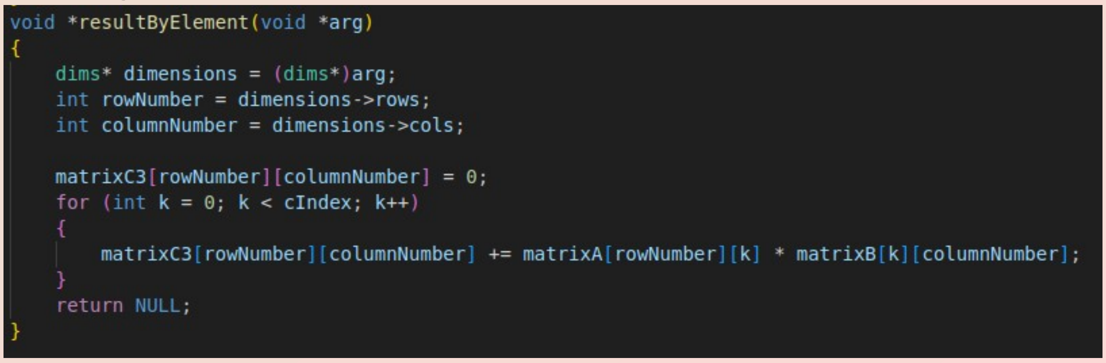
```c
void *resultByMatrix(void *arg)
{
    for (int i = 0; i < cRows; i++)
    {
        for (int j = 0; j < cCols; j++)
        {
            matrixC1[i][j] = 0;
            for (int k = 0; k < cIndex; k++)
            {
                matrixC1[i][j] += matrixA[i][k] * matrixB[k][j];
            }
        }
    }
    return NULL;
}
```

ii) Getting the result of a row in matrix at once:



```c
void *resultByRow(void *arg)
{
    dims *dimensions = (dims *)arg;
    int rowNumber =dimensions->rows;
    for (int j = 0; j < cRows; j++)
    {
        matrixC2[rowNumber][j] = 0;
        for (int k = 0; k < cIndex; k++)
        {
            matrixC2[rowNumber][j] += matrixA[rowNumber][k] * matrixB[k][j];
        }
    }
    return NULL;
}
```

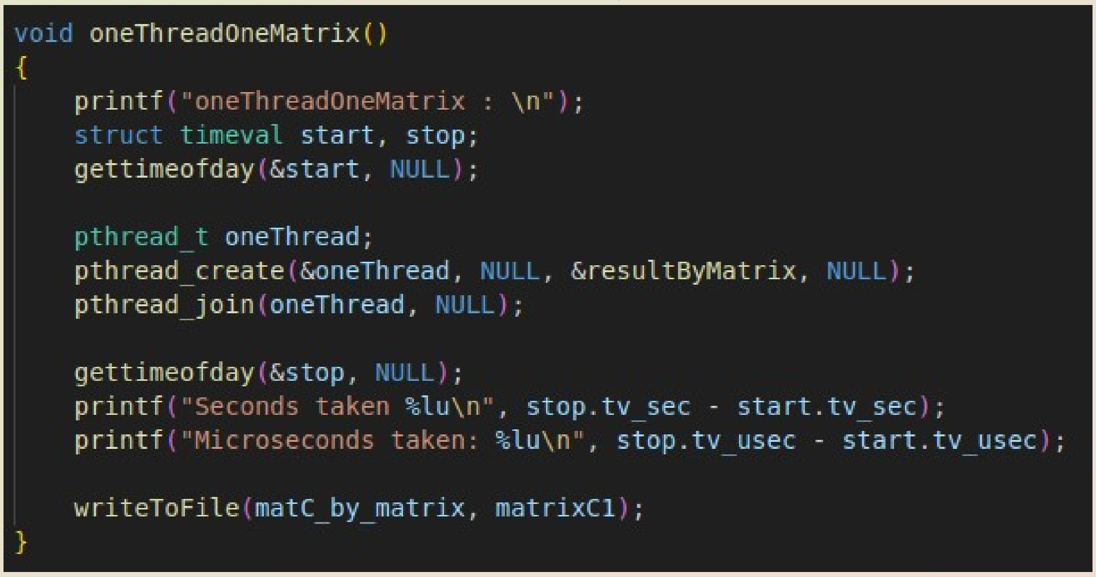
iii) Getting the result of an element in matrix at once:



```c
void *resultByElement(void *arg)
{
    dims* dimensions = (dims*)arg;
    int rowNumber = dimensions->rows;
    int columnNumber = dimensions->cols;

    matrixC3[rowNumber][columnNumber] = 0;
    for (int k = 0; k < cIndex; k++)
    {
        matrixC3[rowNumber][columnNumber] += matrixA[rowNumber][k] * matrixB[k][columnNumber];
    }
    return NULL;
}
```

c) Getting the final result by three methods:

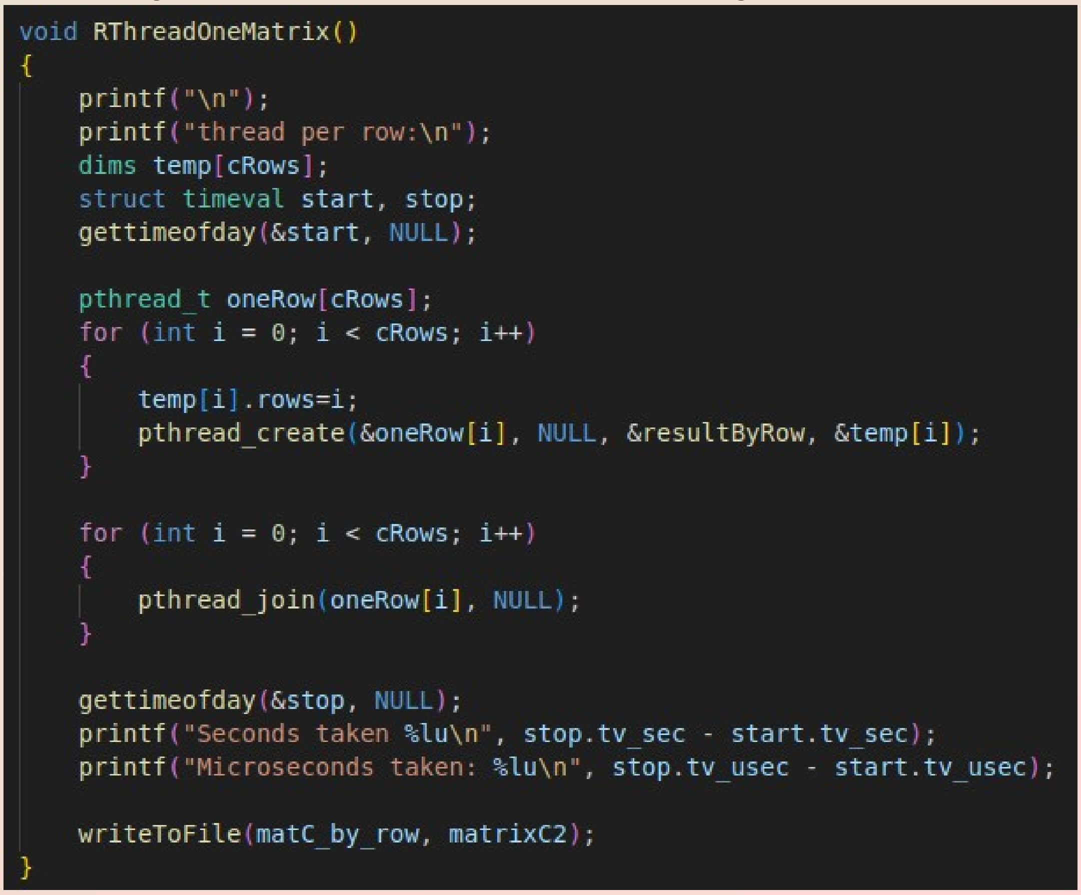
i) One thread for the whole matrix multiplication:



```c
void oneThreadOneMatrix()
{
    printf("oneThreadOneMatrix : \n");
    struct timeval start, stop;
    gettimeofday(&start, NULL);

    pthread_t oneThread;
    pthread_create(&oneThread, NULL, &resultByMatrix, NULL);
    pthread_join(oneThread, NULL);

    gettimeofday(&stop, NULL);
    printf("Seconds taken %lu\n", stop.tv_sec - start.tv_sec);
    printf("Microseconds taken: %lu\n", stop.tv_usec - start.tv_usec);

    writeToFile(matC_by_matrix, matrixC1);
}
```

It calculates time taken during that operation,and writes the result in the corresponding file.

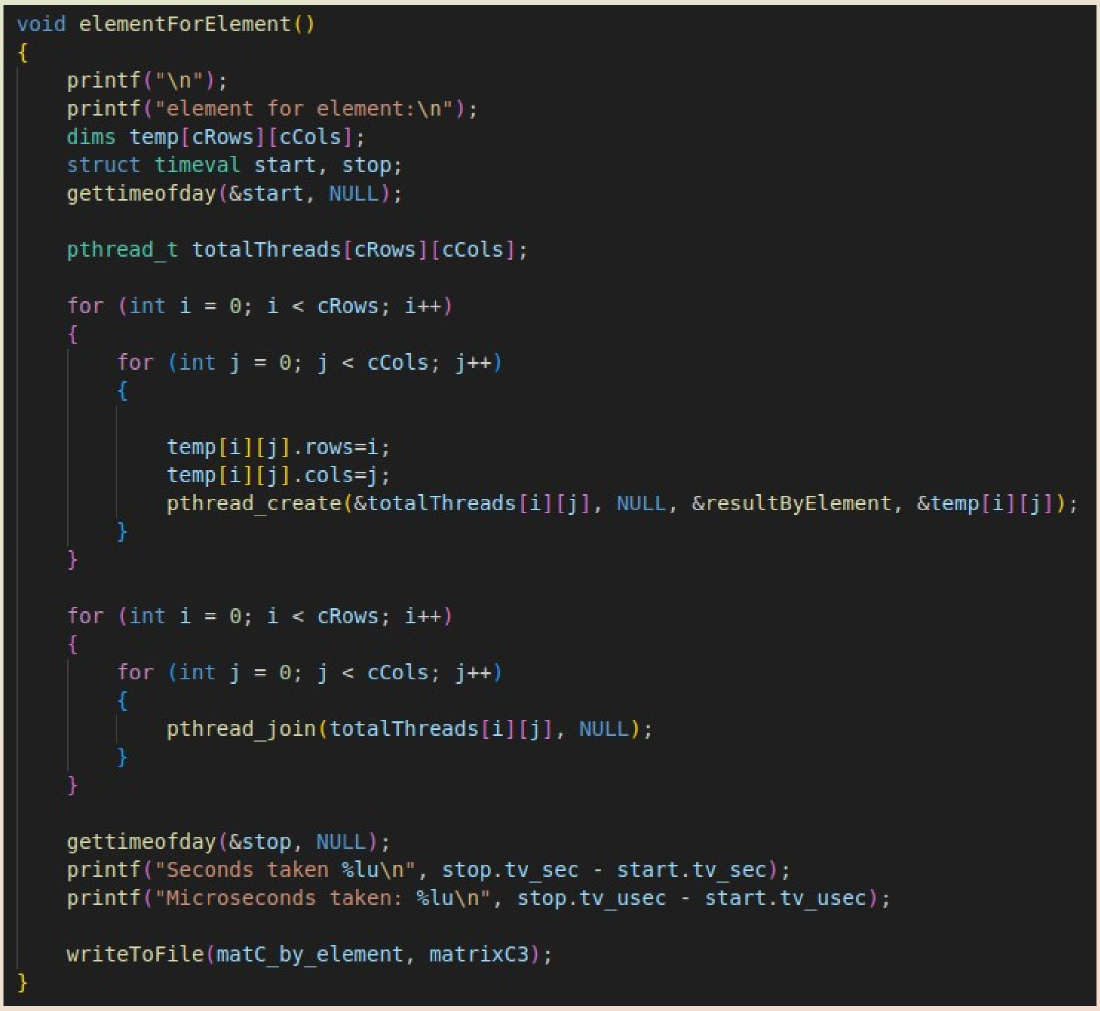
ii) A working thread for each row in the resulting matrix:



```c
void RThreadOneMatrix()
{
    printf("\n");
    printf("thread per row:\n");
    dims temp[cRows];
    struct timeval start, stop;
    gettimeofday(&start, NULL);

    pthread_t oneRow[cRows];
    for (int i = 0; i < cRows; i++)
    {
        temp[i].rows=i;
        pthread_create(&oneRow[i], NULL, &resultByRow, &temp[i]);
    }

    for (int i = 0; i < cRows; i++)
    {
        pthread_join(oneRow[i], NULL);
    }

    gettimeofday(&stop, NULL);
    printf("Seconds taken %lu\n", stop.tv_sec - start.tv_sec);
    printf("Microseconds taken: %lu\n", stop.tv_usec - start.tv_usec);

    writeToFile(matC_by_row, matrixC2);
}
```

It calculates time taken during that operation,and writes the result in the corresponding file.

iii) A working thread for each element in the resulting matrix:



```c
void elementForElement()
{
    printf("\n");
    printf("element for element:\n");
    dims temp[cRows][cCols];
    struct timeval start, stop;
    gettimeofday(&start, NULL);

    pthread_t totalThreads[cRows][cCols];

    for (int i = 0; i < cRows; i++)
    {
        for (int j = 0; j < cCols; j++)
        {

            temp[i][j].rows=i;
            temp[i][j].cols=j;
            pthread_create(&totalThreads[i][j], NULL, &resultByElement, &temp[i][j]);
        }
    }

    for (int i = 0; i < cRows; i++)
    {
        for (int j = 0; j < cCols; j++)
        {
            pthread_join(totalThreads[i][j], NULL);
        }
    }

    gettimeofday(&stop, NULL);
    printf("Seconds taken %lu\n", stop.tv_sec - start.tv_sec);
    printf("Microseconds taken: %lu\n", stop.tv_usec - start.tv_usec);

    writeToFile(matC_by_element, matrixC3);
}
```

It calculates time taken during that operation, and writes the result in the corresponding file.

3) The way to compile code:

a) Open the terminal then run “cd” command to change current directory to the directory of the project

b) Create two files for the 2 matrices you wish to multiply and put them in the directory of the project folder

c) Run the command:

```
make build
```

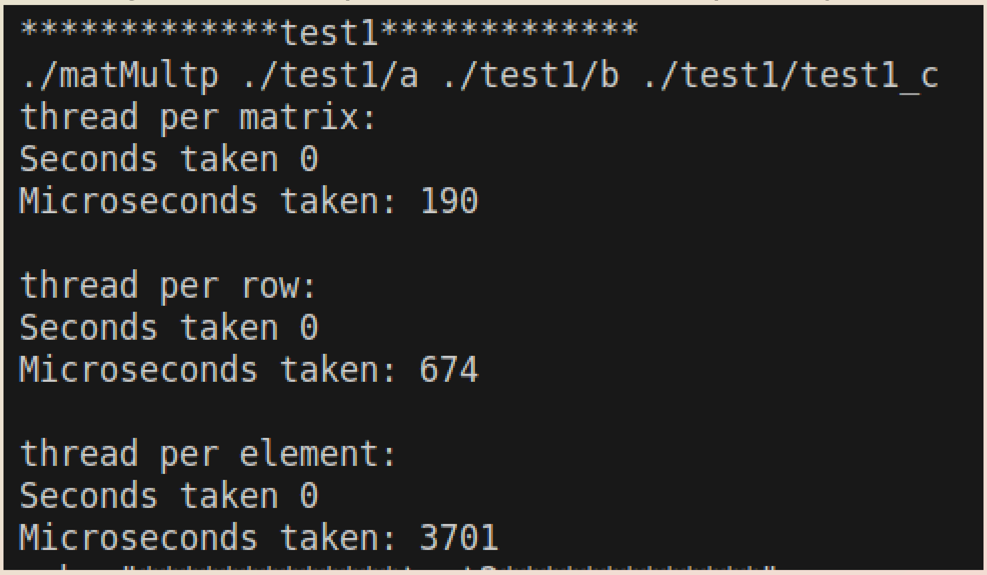
d) Run command:

```
./matMultp <first_matrix_file_name>
<second_matrix_file_name>
```

```
<output_matrix_file_name_prefix>
```
e.g:
```
./matMultp matrixA matrixB matrixC
```
p.s:
If no arguments are given the code will run on default names(a b c). Therefore, files “a.txt” and “b.txt” should be in the directory of the project.

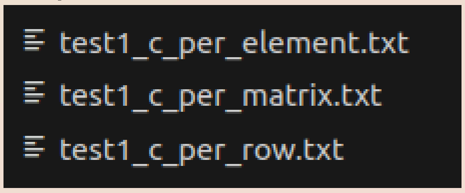
4) Sample runs:

a) Running test case 1 provided in the lab2 repository:



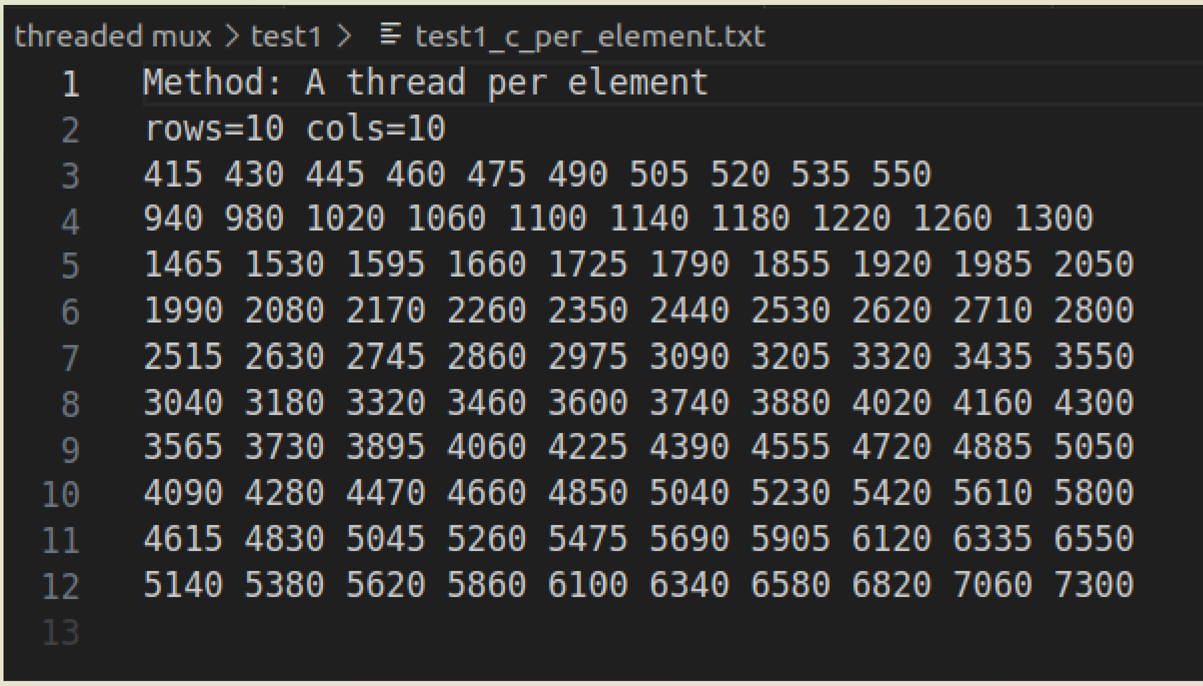
```
*************test1*************
./matMultp ./test1/a ./test1/b ./test1/test1_c
thread per matrix:
Seconds taken 0
Microseconds taken: 190

thread per row:
Seconds taken 0
Microseconds taken: 674

thread per element:
Seconds taken 0
Microseconds taken: 3701
```

output files for test1:



test1_c_per_element.txt
test1_c_per_matrix.txt
test1_c_per_row.txt

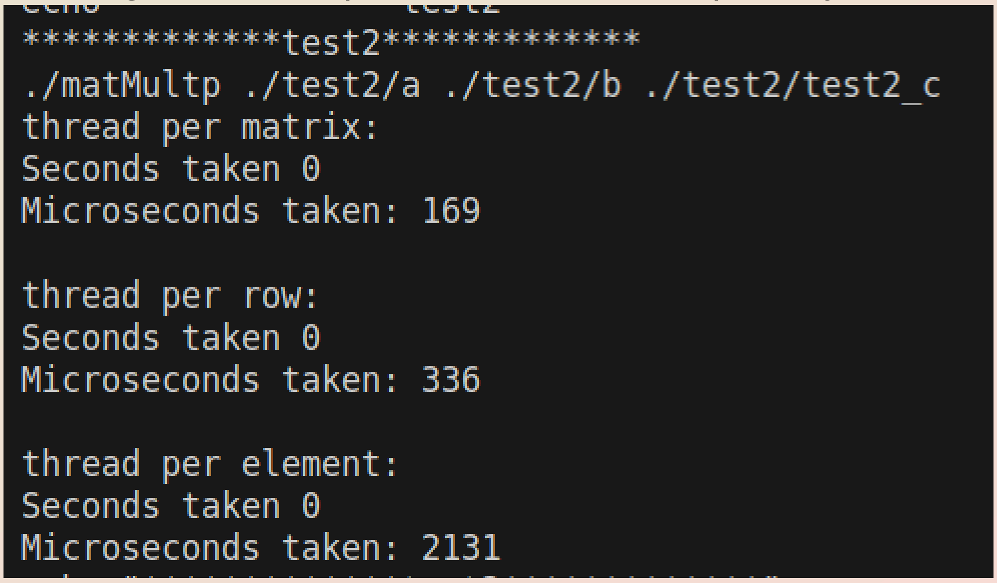
The results in each one of them:



threaded mux > test1 > test1_c_per_element.txt

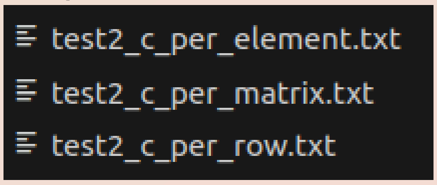
```
Method: A thread per element
rows=10 cols=10
415 430 445 460 475 490 505 520 535 550
940 980 1020 1060 1100 1140 1180 1220 1260 1300
1465 1530 1595 1660 1725 1790 1855 1920 1985 2050
1990 2080 2170 2260 2350 2440 2530 2620 2710 2800
2515 2630 2745 2860 2975 3090 3205 3320 3435 3550
3040 3180 3320 3460 3600 3740 3880 4020 4160 4300
3565 3730 3895 4060 4225 4390 4555 4720 4885 5050
4090 4280 4470 4660 4850 5040 5230 5420 5610 5800
4615 4830 5045 5260 5475 5690 5905 6120 6335 6550
5140 5380 5620 5860 6100 6340 6580 6820 7060 7300
```

b) Running test case 2 provided in the lab2 repository:



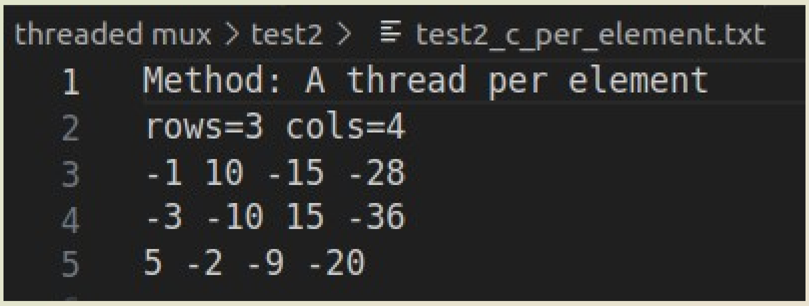
```
*************test2*************
./matMultp ./test2/a ./test2/b ./test2/test2_c
thread per matrix:
Seconds taken 0
Microseconds taken: 169

thread per row:
Seconds taken 0
Microseconds taken: 336

thread per element:
Seconds taken 0
Microseconds taken: 2131
```

Output files for test2:



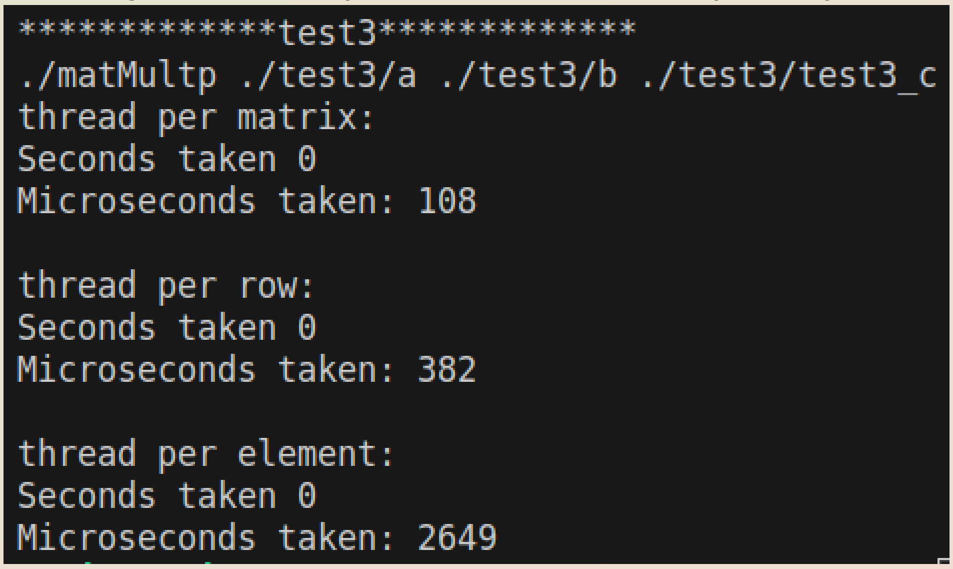
```
test2_c_per_element.txt
test2_c_per_matrix.txt
test2_c_per_row.txt
```

The results in each one of them:



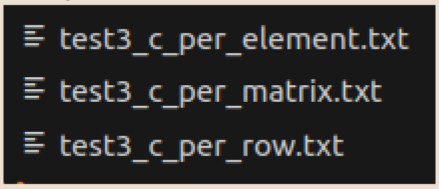
```
threaded mux > test2 > test2_c_per_element.txt
Method: A thread per element
rows=3 cols=4
-1 10 -15 -28
-3 -10 15 -36
5 -2 -9 -20
```

c) Running test case 3 provided in the lab2 repository:



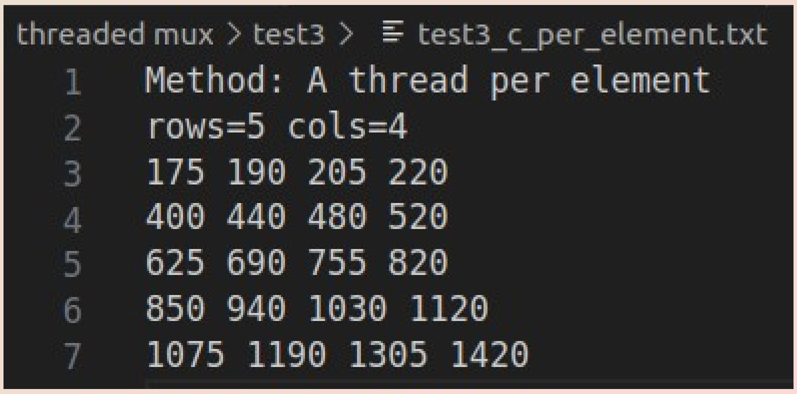
```
*************test3*************
./matMultp ./test3/a ./test3/b ./test3/test3_c
thread per matrix:
Seconds taken 0
Microseconds taken: 108

thread per row:
Seconds taken 0
Microseconds taken: 382

thread per element:
Seconds taken 0
Microseconds taken: 2649
```

Output files for test2:



```
test3_c_per_element.txt
test3_c_per_matrix.txt
test3_c_per_row.txt
```

The results in each one of them:



```
threaded mux > test3 > test3_c_per_element.txt
Method: A thread per element
rows=5 cols=4
175 190 205 220
400 440 480 520
625 690 755 820
850 940 1030 1120
1075 1190 1305 1420
```

5) **We can compare between the three methods by using average time for execution:**

method	Attempt 1 “test1”(µs)	Attempt 2 “test2”(µs)	Attempt 3 “test 3”(µs)	Avg. time(µs)
Method 1 “Thread per matrix”	190	169	108	155.66667
Method 2 “Thread per row”	674	336	382	464
Method 3 “Thread per element”	3701	2131	2649	2827

We notice that method 3 is the slowest by far, due to the time taken to create a thread for each element .