National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute”

SOFTWARE SYSTEM DESIGN PROJECT

prepared by :-Solomon Abrha

sub.To Ruslan

Table of Contents

list of Drawings

Chapter one

Introduction

The programming language used for the course is java Script. So along with is let fs=require(`fs`) module was used. The project uses command line argument to execute a tasks. The tasks are to be executed are as listed. Matrix operation; multiplication , transpose ,subtract and insertion sorting

The whole project is have four different files. The main file app,js which main program is taking place, the function file fun.js which handles all the function running on the app.js. The fun.js and the app.js are linked with export and import methods.

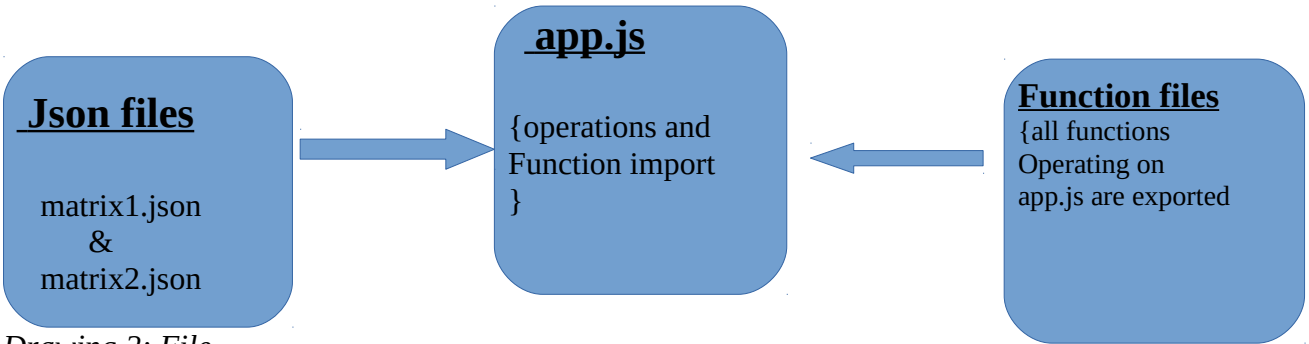
Command line argument; process.argv is an array of two elements initially. With the file directory to node environment and and the file directory.

```
solomon@solomon-Satellite-C850-C019:~/Documents/java and js/javaScript/CORSE PROJECT/pr
oject one$ node app.js
[ '/usr/bin/node',
  '/home/solomon/Documents/java and js/javaScript/CORSE PROJECT/project one/app.js' ]
```

Drawing 1: process.argv

Requirements and Analysis

For the project there are four different files



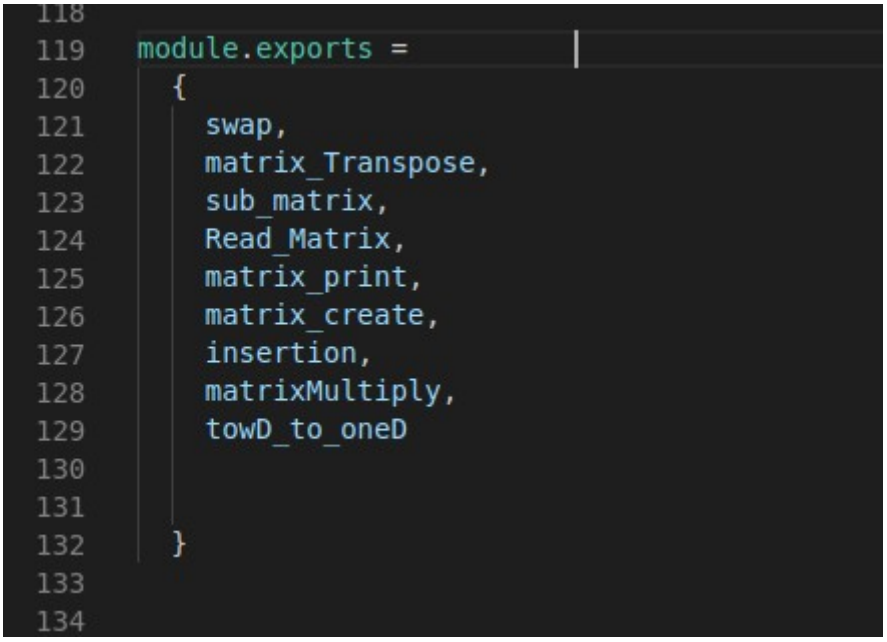
Drawing 2: File structure

And the other library imported is file system which is written with the following syntax

```
let fs = require(`fs`)
```

Import export

The program run on two main parts which are linked with import and export method. All the functions in the fun.js file are exported to app.js as shown in the Drawing 3 below.



Drawing 3: Export File

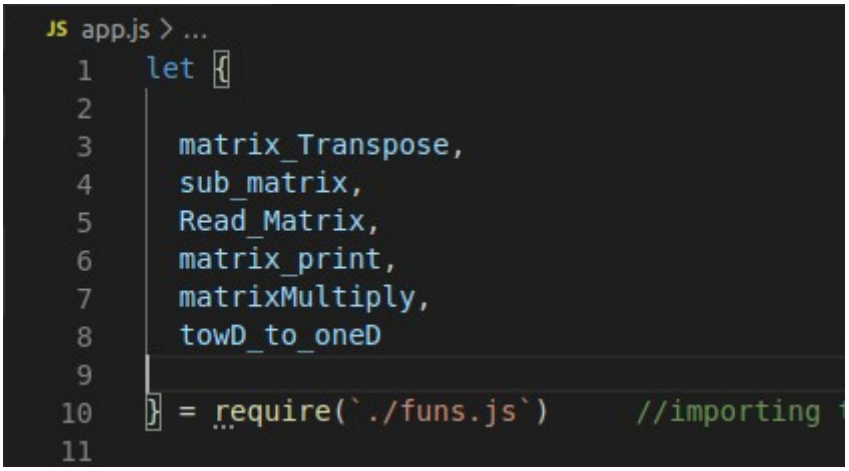
module.exports=

{

//all methods and functions to be exported

}

the exported files are again imported to where needed in this case to app.js. And how to import is shown on Drawing 4 below



Drawing 4: Importing functions

Operations

All operations on the program are performed on matrix. The command to are as follows:

- ➔ node app.js matrix `operation type` files(./matrix1.json ./matrix2.json)

operation types

Matrix subtraction

- ➔ **The command (node app.js matrix sub ./matrix1.json ./matrix2.json)**

Matrix subtraction uses two matrix files to output the difference. The function for the operation is as follows

```
49
50  function sub_matrix(a, b) {
51
52   let k = []
53    k = new Array(a.length);
54    for (let i = 0; i < k.length; i++) {
55      if( a[i].length!= b[i].length ) throw Error (" Error: each matr
56      if(a.length!=b[i].length) throw Error (" Error: each matrix mus
57      if(b.length!=b[i].length) throw Error (" Error: each matrix mus
58
59      k[i] = new Array(a[0].length)
60      for (let j = 0; j < a[i].length; j++) {
61
62        k[i][j] = a[i][j] - b[i][j]
63      }
64
65    }
66    return k
67  }
68
```

Drawing 5: subtraction function

The function operates first by creating an empty matrix “k” by equating the width and length to one of the matrix

let k= [] // this declaration will create new array

k=new Array(a.length) // this will equate number of rows of new ‘K’ array to a’s

Again equate the number of columns of new array ‘K’ to a’s by k[i]=a[0] in side the first loop.

Therefore at this stage there will be new empty matrix with number of rows and columns equal to `matrix a`.

Again create a nested loop for the to execute the operation `k[i][j] = a[i][j] - b[i][j]`

and populate the new matrix ‘K[i][j]’ with the result of the operation

Error conditions

For subtraction operation to be execrated the two matrices must be equal to each other and as for the our case needs to be square.

```
if( a[i].length!= b[i].length ) throw Error (" Error: each matrix must be square with equal size in between => n X n and m X m n=m ")
if(a.length!=b[i].length) throw Error (" Error: each matrix must be square with equal size in between => n X n and m X m n=m ")
if(b.length!=b[i].length) throw Error (" Error: each matrix must be square with equal size in between => n X n and m X m n=m ")
```

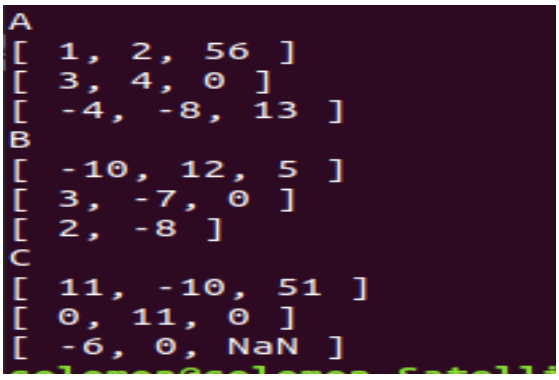
The above there conditions must be satisfied for the function to execute. Not only the size of the matrix but quantitative number of elements must be at the same amount.

To illustrate the above statement the following drawing added

<u>case one:</u>- missing one element

Drawing 6: matrix with one element missing

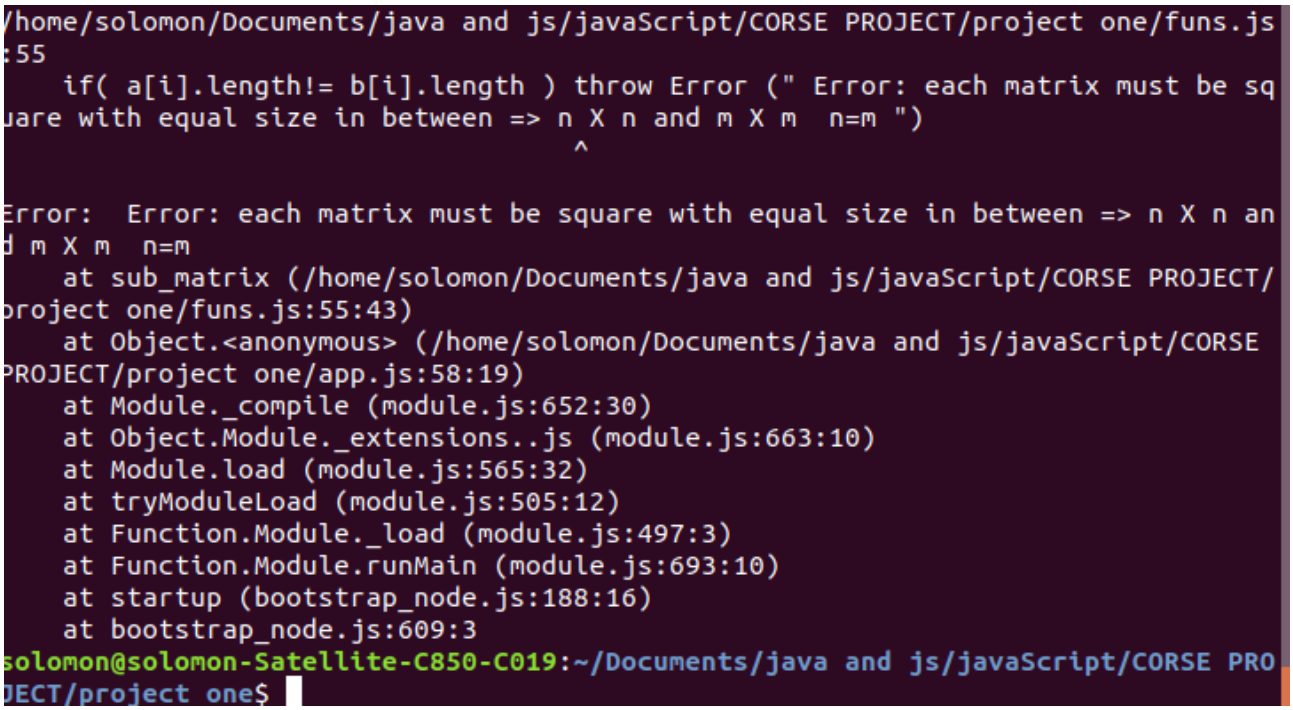
on the above matrix element [3][3] is missing and the result for the subtraction will be as follows. The logical scenes here is to expect error. Because not only number of columns and row must be equal to each other but also quantitative amount of each elements as well



Drawing 7: result of unequal matrix

On the place of the missing element the program console out NaN(not a number). So there should be an error condition for it as well.

The result after the stating the Error condition is:-



Drawing 8: Error return for unequal matrices

The error states that `each matrix must be square with equal size in between =>nXn and mXm where n=m

case two:- missing one row

As stated on the beginning the two matrices must have equal number of rows and columns with both being square.

Matrix multiplication

➔ **The command (node app.js matrix multiply ./matrix1.json ./matrix2.json)**

Multiplication of a matrix is operated on two different matrices having the number of column of the first matrix equal to the number of row of the second matrix.

Matrix multiply is also executed with the same method as subtraction with creating new empty matrix

```
Js funs.js > sub_matrix
  }
  function matrixMultiply(a, b) {

    let widthA = a[0].length
    let heightB = b.length

    if (heightB !== widthA) throw new Error(`can not multiply this matrices`)

    let newArr = [];
    for (let i = 0; i < a.length; i++) {

      newArr[i] = [];
      for (let j = 0; j < b[0].length; j++) {
        if( a[i].length!= b[i].length ) throw Error (" Error: Invalid matrix ")

        let sum = 0;
        for (let k = 0; k < a[0].length; k++) {
          sum += a[i][k] * b[k][j];
        }
        newArr[i][j] = sum;
      }
    }
    return newArr;
  }


```

Drawing 9: Matrix multiply function

for multiplication function three nested loops are needed

Error conditions

Matrix multiply conundrum have the same error conditions with subtraction

Matrix Transpose

➔ **The command (node app.js matrix transpose ./matrix1.json or ./matrix2.json)**

Transpose of a matrix is a new matrix whose rows are the columns of the original. (This makes the columns of the new matrix the rows of the original).

$$\begin{pmatrix} 5 & 4 & 3 \\ 4 & 0 & 4 \\ 7 & 10 & 3 \end{pmatrix}^{T} = \begin{pmatrix} 5 & 4 & 7 \\ 4 & 0 & 10 \\ 3 & 4 & 3 \end{pmatrix}$$

The function to Transpose matrix operates by creating a new empty matrix.

After creating the matrix set the dimension to it to the original matrix.

`matrix_create` function is called here and height and width of the original matrix passed. This will set the dimension of the new matrix with the original matrix ‘m’.

```
function matrix_Transpose(m) {  // Transpose matrix
  let height = m.length
  let width = m[0].length
  //console.log(height, width, m)
  if (height != width) throw Error("invalid matrix: it should be square matrix")
  let temp = matrix_create(height, width)
  for (let i = 0; i < height; i++) {
    for (let j = 0; j < width; j++) {
      temp[i][j] = m[j][i]
    }
  }

  return temp
}
```

Drawing 10: matrix transpose function

The Error condition here is to check if the original matrix is square or not.

Matrix Insertion sort

- ➔ **The command (node app.js matrix insertion ./matrix1.json ./matrix2.json)**
- ➔ **The command (node app.js matrix insertion ./matrix1.json or ./matrix2.json)**

The insertion algorithm sort the matrix value to increasing order. This operation undergoes three major steps.

Step 1:- changing the two dimensional array to one dimension.

```
function towD_to_oneD(arr) {
  let temp = [];
  for (let i = 0; i < arr.length; i++) { //the .concat(a[i]) function will convert
    temp = temp.concat(arr[i]);
  }       // the two dimensional arry to one dimensional
  for (let i = 0; i < temp.length; i++) {     //find the minimum index from the 1d array
    insertion(temp)
  }
  return temp
}
```

Drawing 11: function to convert two dimensional array to one

The very firs step was to crate one dimensional empty array and populate the elements of the array in one line

let tmp= [] this declaration will create a new empty array

and inside the for loop setting the “temp =temp.concat(arr[i]) will covert the two dimensional array to one dimension and set the elements to the new array.

After converting the array the next step will be to sort the elements by insertion sorting method.

```
68
69   function insertion(a) {
70
71     for (let i = 1; i < a.length; i++) {
72       let key = a[i];
73       let j = i - 1;
74       while (j >= 0 && a[j] > key) {
75         a[j + 1] = a[j];
76         j = j - 1;
77       }
78       a[j + 1] = key;
79     }
80   }
81
```

Drawing 12: insertion sorting

The when passing the array thorough this function the array will be sorted to increasing order.

After sorting the array will again be converted to two dimensional array again by the following method

```
}else if (operationType ==="insertion")
{
  let matrixApath = process.argv[4]
  let matrixs = Read_Matrix(matrixApath)
  let unsorted_matrixs = Read_Matrix(matrixApath)

let fin=towD_to_oneD(matrixs)


  let converted_twoD = [];      // take the converted One dimensionalarr and pass the elements here
  while (fin.length) converted_twoD.push(fin.splice(0, unsorted_matrixs[0].length));
  console.log("Frst matrix :")
 matrix_print(converted_twoD)
 console.log("  ")
 let secondmatrix=process.argv[5]
 if(secondmatrix==="./matrix2.json"){
 let matrixBpath = process.argv[5]
  matrixs = Read_Matrix(matrixBpath)
  unsorted_matrixs = Read_Matrix(matrixBpath)

let fin2=towD_to_oneD(matrixs)
```

Drawing 13: function to convert one dimensional array to two

the process.agrv position for the first matrix is on 5thplace and is the user wants to sort both matrix at once there is additional else if condition set

```
if(secondmatrix==="./matrix2.json"){   //'if' function inside 'else if ' to add anothe
let matrixBpath = process.argv[5]
 matrixs = Read_Matrix(matrixBpath)
 unsorted_matrixs = Read_Matrix(matrixBpath)

let fin2=towD_to_oneD(matrixs)


 converted_twoD = [];      // take the converted One dimensionalarr and pass the eleme
while (fin2.length) converted_twoD.push(fin2.splice(0, unsorted_matrixs[0].length));
console.log("Second matrix :")

matrix_print(converted_twoD)
 }
```

Drawing 14: if function for second position in insertion sort

Array Insertion sort

➔ **The command (node app.js sort ./array.json)**

the basic deference between this operation and the insertion sort for matrix is the diension of the matrix. meaning here there is no need to convert the array to one dimension and reconvert it.

```
99
100  }else if(type==='sort'){
101    let arrayPath = process.argv[3];
102      let array = Read_Matrix(arrayPath)
103      for (let i = 0; i < array.length; i++) {     //find the minimum index from the 1d array
104        insertion(array)
105      }
106
107      console.log(array)
108
109
110  }
111  else {
112    console.log("Error : unkown command!")
113  }
114
115
```

Drawing 15: insertion sorting of array

The operation type is ‘sort’. The insertion function is with in a for loop set to a limit of length of the array passing.

Since the position of the “arrayPath” is right after the ‘type’ it’s placed on process.argv[3]

Results

Matrix subtraction

```
solomon@solomon-Satellite-C850-C019:~/Documents/java and js/javaScript/C
JECT/project one$ node app.js matrix sub ./matrix1.json ./matrix2.json
A
[ 1, 2, 56 ]
[ 3, 4, 0 ]
[ -4, -8, 13 ]
B
[ -10, 12, 5 ]
[ 3, -7, 0 ]
[ 2, -8, 13 ]
C
[ 11, -10, 51 ]
[ 0, 11, 0 ]
[ -6, 0, 0 ]
solomon@solomon-Satellite-C850-C019:~/Documents/java and js/javaScript/C
JECT/project one$
```

Drawing 16: subtraction result

Matrix multiply

```
solomon@solomon-Satellite-C850-C019:~/Documents/java and js/javaScript/CORSE PR
JECT/project one$ node app.js matrix multiply ./matrix1.json ./matrix2.json
A
[ 1, 2, 56 ]
[ 3, 4, 0 ]
[ -4, -8, 13 ]
B
[ -10, 12, 5 ]
[ 3, -7, 0 ]
[ 2, -8, 13 ]
the product of the two matrices is:
C
[ 108, -450, 733 ]
[ -18, 8, 15 ]
[ 42, -96, 149 ]
solomon@solomon-Satellite-C850-C019:~/Documents/java and js/javaScript/CORSE PR
JECT/project one$
```

Drawing 17: Matrix multiply result

Matrix Insertion

```
JECT/project one$ node app.js matrix insertion ./matrix1.json ./matrix2.json
Frst matrix :
[ -8, -4, 0 ]
[ 1, 2, 3 ]
[ 4, 13, 56 ]

Second matrix :
[ -10, -8, -7 ]
[ 0, 2, 3 ]
[ 5, 12, 13 ]
solomon@solomon-Satellite-C850-C019:~/Documents/java and js/javaScript/CORSE P
JECT/project one$
```

Drawing 18: Two- Matrix insertion result

```
solomon@solomon-Satellite-C850-C019:~/Documents/java and js/ja
JECT/project one$ node app.js matrix insertion ./matrix1.json
Frst matrix :
[ -8, -4, 0 ]
[ 1, 2, 3 ]
[ 4, 13, 56 ]

solomon@solomon-Satellite-C850-C019:~/Documents/java and js/ja
JECT/project one$
```

Drawing 19: One-Matrix insertion

Array Insertion

```
solomon@solomon-Satellite-C850-C019:~/Documents/
JECT/project one$ node app.js sort ./array.json
[ -77, -23, 0, 1, 8, 12, 28, 63, 98 ]
solomon@solomon-Satellite-C850-C019:~/Documents/
JECT/project one$
```

Drawing 20: Array-Insertion