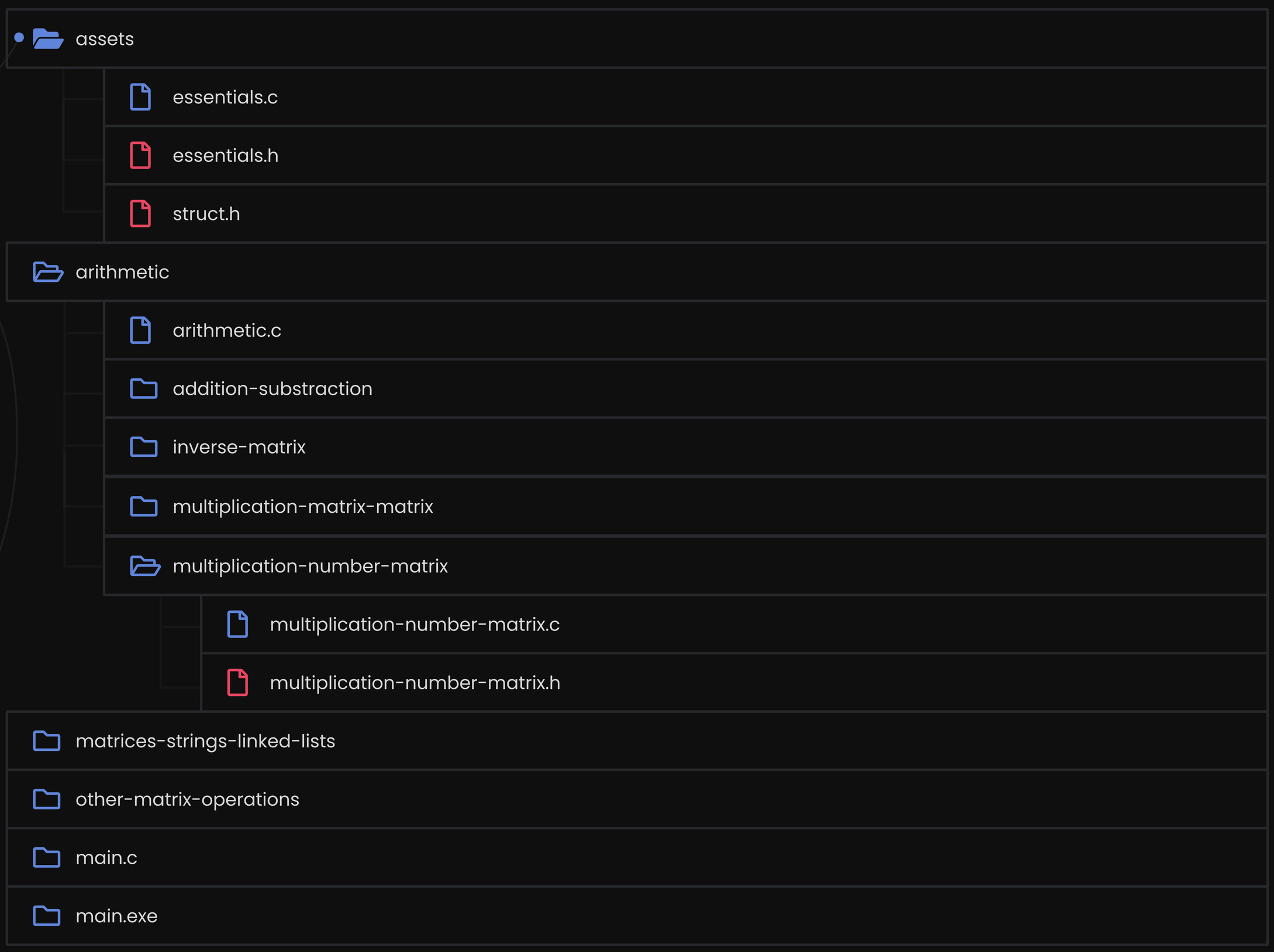
CHAPTER

ZERO

BEFORE WE START

1. Directory Hierarchy

In This Chapter I Will Talk A Little Bit About How Do I Structured The Project Folder



1.1 Assets

The Assets Folder Is The Core Of The Project, I Included In There The Most Essential And Reused Functions (Allocation, Creation .. Etc) By All Parts, I Also Included The Structs And Libraries That I Have Used Later To Avoid Repeating My Self.

1.1.1 struct.h

I Defined All The Structs That I Used In The Project And Stored Them In One Single File To Keep The Folder Clean And Easy Accessible

```c
typedef struct word_element {

    char * word;
    struct word_element* next;

} word_element ;

typedef struct {

    char letter;
    struct word_element* next;

} words_vector_element;
```

screenshot from the struct.h file

1.1.2 essential.h & essential.c

The Essential.H File Included All The Standard Libraries And The Ones That I Used During The Development Of The Mini Project And It Also Included The Structs (Struct.H) File The Prototypes Of All The Essential Functions Stored In Essential.C File.

```c
1
2   // required libraries
3   #include <stdio.h>
4   #include <stdlib.h>
5   #include <string.h>
6   #include <ctype.h>
7   #include <math.h>
8   // structs
9   #include "./struct.h"
```

all of the labraries included in the essential.h file

```c
// for allocations

int* allocate_vector( int length );
int** allocate_matrix( int rows );


// for creation
int* create_vector( int length );
int** create_matrix( int rows, int columns );


// for scaning

int** scan_matrix( int* rows, int* columns );
char* scan_text( int text_length );

// text utilites

int text_words_count( char *text );
char* text_to_lower_case( char *text );

// for clearing memory

void free_matrix( int** matrix, int rows );
void free_float_matrix( float** matrix, int rows );

 // for printing

 char* counter( int a );
 void print_matrix( int** matrix, int rows, int columns );
 void print_float_matrix( float** matrix, int rows, int columns
 void print_vector( int* matrix, int length );
```

the prototypes of the essential functions

allocate_vector allocates he memory space for an array of length integers.

allocate_matrix allocates the memory spacefor an array of rows pointer to integers.

create_vector allocates the memory spaceusing allocate_vector and scan the values to fill the array.

create_matrix allocates the memory space using allocate_matrix and fills every singe row by using the function create_vector.

scan_matrix prints a guide massages to the user and call the function create_matrix .

scan_text allocate the space for a text with a maximaum length of text_length and scan it

text_words_count retruns an integer of words count of a text

text_to_lower_case allocates the space for new text and transform UPPERCASE letters of the given text to lowercase letter and puts them into the new lowercase_text.

free_matrix takes a matrix of integers and clear it from the memory row by row than the matrix itself.

free_float_matrix the same as free_matrix but for a matrix of floats

print_matrix prints a matrix of an intigers

print_float_matrix prints a matrix of an intigers

print_vector prints a vector of an intigers

1.2 Project Parts

As We've Seen In The First Illustration Every Part Of The Project Has A Sub Folder That Includes Sub Folders Of That Part Questions And A C File That Contains The Part Menu Function And Includes The Sub Questions Files.

CHAPTER
ONE
ARITHMETIC

1. addition / subtraction two matrices

To Keep The Code Reusable, I Was Dividing A Function Into Small Simple Functions, To Reuse Them Later (Or Even Out Of This Project), And The Addition/Subtraction Function Is Not An Exception, So I Will Explain The Small Fonctionalites To Understand The Bigger Picture.

1.1 matrices_sum

This Function Accepts As An Arguments Two Matrices With The Same Dimensions, And Another 3 Integers Represents The Matrices Rows Count, Matrices Columns Count And The Operation Type, Basically What This Function Do Is Allocating Space For A New Matrix And Depending On The Operation Type (0 Or -1) It Fills The New Matrix With The Sum Or The Difference Of The Two Matrices, And Returns The New Matrix.

PROTOTYPE

```
fonction matrices_sum( E/ ^^matrix_a, ^^matrix_b : entier; rows, columns, operation : entier ) : ^^entier;
```

```
int** matrices_sum( int **matrix_a, int **matrix_b, int rows, int columns, int operation );
```

1.2 add_matrices & substract_matrices

Basically, Add_Matrices And Substract_Matrices Do The Same Job They Both Accepts Two Matrices And Their Dimensions And Pass It To Matrices_Sum But The Difference Here's That Add_Matrices Passes As The Fourth Argument 0 In The Other Side Substract_Matrices Passes - 1, And Both Of These Function Return Substract_Matrices;

PROTOTYPE

```
fonction add_matrices( E/ ^^matrix_a, ^^matrix_b : entier; rows, columns : entier ) : ^^entier;
```

```
int** add_matrices( int **matrix_a, int **matrix_b, int rows, int columns );
```

PROTOTYPE

```
fonction substract_matrices( E/ ^^matrix_a, ^^matrix_b : entier;  rows, columns  : entier ) : ^^entier;
```

```
int** add_matrices( int **matrix_a, int **matrix_b, int rows, int columns );
```

1.3 do_sum_matrices

It Plays The Role Of A User Interface, It's Tells The User To Choose Between Addition And Subtraction, Depending On His Response The Function Calls Add_Matrices Or Substract_Matrices After Allocating And Filling Matrices Using The Essential Function Scan_Matrix, After That It Print The Results And Clean The Memory.

PROTOTYPE

```
procedure do_sum_matrices();
```

```
void do_matrices_sum();
```

1.4 Example

```
addition or substraction [a/s]
a
[ 1nd martix ]
please enter your matrix rows count
2
please enter your matrix columns count
2

[ 1st row ]
  enter the 1st value: 1
  enter the 2nd value: 2

[ 2nd row ]
  enter the 1st value: 4
  enter the 2nd value: 5

[ 2nd martix ]

[ 1st row ]
  enter the 1st value: 6
  enter the 2nd value: 7

[ 2nd row ]
  enter the 1st value: 8
  enter the 2nd value: 9

[ result ]
7       9
12      14
```

2. multiplication of a matrix by a number

2.1 multiply_matrix_number

It Allocates The Space For A New Matrix With The Same Dimensions Of The Passed Matrix, Then It Fills It With The Passed Matrix Values Multiplied By Multiplier, Then It Returns The New Matrix.

PROTOTYPE

```
fonction multiply_matrix_number(E/S ^^matrix: entier; rows, columns,multiplier:entier):^^entier
```

```
int** multiply_matrix_number( int** matrix, int rows, int columns, int multiplier );
```

2.2 do_multiply_matrix_number

Almost All Functions That's Has To Prefix Of "Do_" Does The Same Thing, In This Case, It Scans The Matrix Then The Multiplier, Then Passes Those Variables To Multiply_Matrix_Number, Prints The Results, And Cleans The Memory.

PROTOTYPE

```
procedure do_multiply_matrix_number();
```

```
void do_multiply_matrix_number();
```

2.3 Example

```
please enter your matrix rows count
2
please enter your matrix columns count
2

[ 1st row ]
  enter the 1st value: 1
  enter the 2nd value: 7

[ 2nd row ]
  enter the 1st value: 5
  enter the 2nd value: 6
please enter the multiplier
3

[ result ]
3       21
15      18
```

3. multiplication of a matrix by a matrix

3.1 multiply_matrix_matrix

For Us That's Was A Little Bit Challenging, So We Did A Little Bit Of Research About The Matrices Multiplication Formula Then We Translated The Formula Into A Function(The Resources That We Used Will Be Included In The Last Page Of This Document).

PROTOTYPE

```
fonction mutliply_matrix_matrix( E/ ^^matrix_a : entier; rows_a, columns_a :entier; ^^matrix_b : entier; rows_b, columns_b :entier ): ^^entier
```

```
int** mutliply_matrix_matrix( int** matrix_a, int rows_a, int columns_a ,int** matrix_b, int rows_b, int columns_b );
```

3.2 do_multiply_matrix_matrix

As Usually The "Do_" Function Talks To The User, Scans The Matrices, Passes Them To Multiply_Matrix_Matrix Then Prints The Results, Cleans The Memory

PROTOTYPE

```
procedure do_multiply_matrix_matrix();
```

```
void do_mutliply_matrix_matrix();
```

2.3 Example

```
enter the first matrix info
please enter your matrix rows count
2
please enter your matrix columns count
2

[ 1st row ]
  enter the 1st value: 5
  enter the 2nd value: 7

[ 2nd row ]
  enter the 1st value: 4
  enter the 2nd value: 11
enter the second matrix info
please enter your matrix rows count
2
please enter your matrix columns count
2

[ 1st row ]
  enter the 1st value: 3
  enter the 2nd value: 6

[ 2nd row ]
  enter the 1st value: 4
  enter the 2nd value: 5

[ Result ]
43      65
56      79
```

4. inversion of a matrix

4. 1 How to inverse a matrix

That's The Hardest Part Of The Project, Because When We Want To Invert A Matrix The Common Method Is By Keep Subtracting Adding And Switching Rows, And Apply The Same Operations To The Matrix Of Identity, Until We Get A Matrix Of Identity From The Original Matrix, We Take The Modified Matrix Of Identity As The Inverse... But This Method Relays On Intelligence So We Can't Translate It Into A Function... But After Some Research We Found That The Inverse Of The Matrix Is The Matrix Of Cofactors Transposed Divided By The Determinant Of The Original Matrix.

4. 2 ignore_matrix_ligne_column

Ignore_Matrix_Ligne_Column Is A Simple Function That Creates A New Matrix Excluding The Specified Row And Column, This Function Is Necessary For This Part

PROTOTYPE

```
fonction ignore_matrix_ligne_column ( E/ ^^matrix :entier; rows, columns, ignored_rows, ignored_columns :entier)^^entier
```

```
int** ignore_matrix_ligne_column( int **matrix, int rows, int columns, int ignored_row, int ignored_column );
```

4. 3 matrix_determinant

It's A Simple Recursive Function That's Keep Calling Itslef With Smaller Matrix Until The Matrix Width <= 2 According To This Formula

$$
\begin{aligned}
|A| &= \begin{vmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{vmatrix} \\
&= A_{11} \begin{vmatrix} A_{22} & A_{23} & A_{24} \\ A_{32} & A_{33} & A_{34} \\ A_{42} & A_{43} & A_{44} \end{vmatrix} - A_{21} \begin{vmatrix} A_{12} & A_{13} & A_{14} \\ A_{32} & A_{33} & A_{34} \\ A_{42} & A_{43} & A_{44} \end{vmatrix} \\
&\quad + A_{31} \begin{vmatrix} A_{12} & A_{13} & A_{14} \\ A_{22} & A_{23} & A_{24} \\ A_{42} & A_{43} & A_{44} \end{vmatrix} - A_{41} \begin{vmatrix} A_{12} & A_{13} & A_{14} \\ A_{22} & A_{23} & A_{24} \\ A_{32} & A_{33} & A_{34} \end{vmatrix}.
\end{aligned}
$$

PROTOTYPE

```
fonction matrix_determinant (E/ ^^matrix :entier; width :entier ) :entier
```

```
int matrix_determinant( int **matrix, int width );
```

4. 4 matrix_cofactors

Basically, It Loops Through The Matrix And Excludes The Row And The Column Of Each Element Of The Matrix Calculate The Determinant For The Rest Of The Matrix

PROTOTYPE

```
fonction matrix_cofacteurs ( e/ ^^matrix:entier; width:entier ): ^^entier
```

```
int** matrix_cofactors( int **matrix, int width );
```

4. 5 matrix_switch_row_column

That Simple Function Transpose A Square Matrix

PROTOTYPE

```
fonction matrix_switch_row_column( E/ ^^matrix :entier; width :entier ) :^^entier
```

```
int** matrix_switch_row_column( int **matrix, int width );
```

4. 6 invert_matrix

Itcombines All Of The Previous Functions To Create The Inverse Matrix

PROTOTYPE

```
fonction invert_matrix( E/ ^^matrix :entier; width :entier ) : ^^reel
```

```
float** invert_matrix( int **matrix, int width );
```

4. 7 do_invert_matrix

That's The Function Thats Scans The Matrix Pass It To Invert_Matrix And Prints The Inverted Matrix, Clears The Memory

4. 8 Example

```
[ 1st row ]
  enter the 1st value: 3
  enter the 2nd value: 7
  enter the 3rd value: 88

[ 2nd row ]
  enter the 1st value: 4
  enter the 2nd value: 5
  enter the 3rd value: 66

[ 3rd row ]
  enter the 1st value: 199
  enter the 2nd value: 5
  enter the 3rd value: 33

[ RESULT ]
-0.03   0.04    0.00
2.76    -3.69   0.03
-0.21   0.29    -0.00
```

CHAPTER
TWO

OTHER MATRIX OPERATIONS

1. Transpose a matrix

1.1 transpose_matrix

The Transpose Of A Matrix Is Simply A Flipped Version Of The Original Matrix. We Can Transpose A Matrix By Switching Its Rows With Its Columns,.

PROTOTYPE

```
fonction transpose_matrix( E/ matrix : ^^entier; rows, columns : entier ) : ^^entier;
```

```
int** transpose_matrix( int **matrix, int rows, int columns );
```

1.2 do_transpose_matrix

The Usual, Scans The Matrix, Pass It To Transpose_Matrix, Prints The Results, Free The Memeroy.

1.3 Example

```
=> 5. exctract sub matrices
1
please enter your matrix rows count
2
please enter your matrix columns count
3

[ 1st row ]
  enter the 1st value: 1
  enter the 2nd value: 8
  enter the 3rd value: 7

[ 2nd row ]
  enter the 1st value: 9
  enter the 2nd value: 6
  enter the 3rd value: 4
1       9
8       6
7       4
```

2. sort a matrix

2.1 sort_matrix

Sorting A Matrix Is Not A Complicated Operation, We Started By Extractingxthe Matrix Into A Flat Array, Then Applied The Swipes Sorting Algorithm To It Then We Put It Back Into A New Matrix, Of Course The Sort Type Depend On The Last Argument Of The Function.

PROTOTYPE

```
fonction transpose_matrix( E/ matrix : ^^entier; rows, columns, sort_type : entier ) : ^^entier;
```

```
int** sort_matrix( int **matrix, int rows, int columns, int sort_type );
```

2.2 sort_matrix_ascending & sort_matrix_descending

Both Of These Function Passes A Matrix To The Sort_Matrix Function With A Teawk In The Sort Type

PROTOTYPE

```
fonction sort_matrix_ascending ( E/ matrix : ^^entier; rows, columns : entier ) : ^^entier;
```

```
int** sort_matrix_descending( int **matrix, int rows, int columns );
```

2.3 Example

```
ascending or descending [a/d]
a
please enter your matrix rows count
3
please enter your matrix columns count
3

[ 1st row ]
  enter the 1st value: 1
  enter the 2nd value: 2
  enter the 3rd value: 4

[ 2nd row ]
  enter the 1st value: 7
  enter the 2nd value: 8
  enter the 3rd value: 9

[ 3rd row ]
  enter the 1st value: 66
  enter the 2nd value: 5
  enter the 3rd value: 4
1       2       4
4       5       7
8       9       66
```

3. max row & max column

3.1 exctract_max_row_vector & exctract_max_row_vector

The Functions Is Similar With Few Changes The Principe Is The Same, So We Star By Initialize Our Max Integer With The First Element Then We Compare The Rest Elements Of The Row (Or The Column), When We Arrive To The End Of The Row (Or The Column) We Store Our Max In The Vector

PROTOTYPE

```
fonction extract_max_row_vector ( E/ matrix : ^^entier; rows,columns :entier ) :^entier
```

```
int* exctract_max_row_vector( int **matrix, int rows, int columns );
```

3.2 Example

```
please enter your matrix rows count
3
please enter your matrix columns count
3

[ 1st row ]
  enter the 1st value: 4
  enter the 2nd value: 7
  enter the 3rd value: 8

[ 2nd row ]
  enter the 1st value: 9
  enter the 2nd value: 4
  enter the 3rd value: 7

[ 3rd row ]
  enter the 1st value: 9
  enter the 2nd value: 7
  enter the 3rd value: 44
8       9       44
```

4. Sub Matrices Exctraction

4.1 exctract_sub_matrix

What This Function Simply Do, Is Extract A Small Matrix Form A Bigger Matrix From Certain Position

PROTOTYPE

```
fonction extract_sub_matrix ( E/ matrix: ^^entier; offset_row,offset_columns, rows_length, columns_length :entier ) :^^entier
```

```
int** extract_sub_matrix( int** matrix, int offset_row, int offset_column, int rows_length, int column_length );
```

4.1 exctract_sub_matrices

This Function Uses A Basic Mathematical Calculation To Use The Previous Function To Exctract All Possible Sub Matrices

PROTOTYPE

```
fonction extract_sub_matrices ( E/ matrix : ^^entier; rows, columns, sub_rows, sub_columns :entier):^^entier
```

```
int*** extract_sub_matrices( int** matrix, int rows, int columns, int sub_rows, int sub_columns );
```

3.2 Example

```
please enter your matrix rows count
3
please enter your matrix columns count
3

[ 1st row ]
  enter the 1st value: 1
  enter the 2nd value: 5
  enter the 3rd value: 7

[ 2nd row ]
  enter the 1st value: 5
  enter the 2nd value: 4
  enter the 3rd value: 6

[ 3rd row ]
  enter the 1st value: 4
  enter the 2nd value: 2
  enter the 3rd value: 6
enter the rows of the sub matrix that you would extract
2
enter the columns of the sub matrix that you would extract
2
1       5
5       4
============
5       7
4       6
============
5       4
4       2
============
4       6
2       6
============
```

CHAPTER

THREE

MATRICES STRINGS LINKED LISTS

1. Words To Matrix

1.1 words_matrix

I Will Explain This Function Briefly Because There Is A Lot Of Cases Like What If There's Two Spaces, Or What If The Text Started With A Lot Of Spaces, What If The Whole Text Is A Single Word, How Much Memory Should I Allocate For Each Word... Etc, Without Diving Into Details Words_Matrix Split The Text By The Spaces And Returns A Array Of Words

PROTOTYPE

```
fonction words_matrix ( E/ ^text :chaine ) :^^chaine
```

```
char** words_matrix( char* text );
```

1.2 Example

```
1
Enter Your Text :
The quick brown fox jumps over the lazy dog
[ your text words ]
The
quick
brown
fox
jumps
over
the
lazy
dog
```

2. Words To Vector

2.1 words_matrix_vector

This Function Relay On The Previous One We Will Just Extract The Words Of The Matrix And Put Linked List Element

PROTOTYPE

```
fonction words_matrix_vector ( E/ ^text :chaine ) :^^word_element
```

```
word_element** words_matrix_vector( char* text );
```

1.2 Example

```
1
Enter Your Text :
The quick brown fox jumps over the lazy dog
[ your text words ]
The
quick
brown
fox
jumps
over
the
lazy
dog
```

3. Orginize Words Into Structure

3.1 organize_words

We Will Use The Words_Matrix_Vector As A Resource For Words So We Don't Have To Allocate Anything. What I Did Here Is I Intialiated The Array V With All Letters In Alphabet, And I Started Filling The Words So I Don't Worry About Allocation And Stuff And I Think Its The Easiest, Another Thing That I Was Aware Of Is Save The Same Word Twice.

PROTOTYPE

```
fonction organize_words ( E/ ^text :chaine ) :^^words_vector_element
```

```
words_vector_element* orginize_words( char* text );
```

3.2 Example

```
3
Enter Your Text :
Lorem Ipsum is simply dummy text of the printing and typesetting industry.
A
        and
B
        been
D
        dummy
E
        ever
H
        has
I
        industry's
        industry.
        is
        ipsum
L
        lorem
O
        of
P
        printing
S
        since
        standard
        simply
T
        typesetting
        the
        text
```

4. Remove/Add Word Structure

4.1 organize_words

There's Nothing Special About It, It's Like Adding And Remove Elements From Any Regular Linked List

PROTOTYPE

```
fonction organize_words ( E/ ^text :chaine ) :^^words_vector_element
```

```
words_vector_element* orginize_words( char* text );
```

3.2 Example

```
Enter Your Text :
The quick brown fox jumps over the lazy dog
add or remove [a/r]
a
enter the word that you want to add: cat
B
        brown
D
        dog
F
        fox
J
        jumps
L
        lazy
O
        over
Q
        quick
T
        the
```

```
============================
B
        brown
C
        cat
D
        dog
F
        fox
J
        jumps
L
        lazy
O
        over
Q
        quick
T
        the
```

CONCLUSION

Team Work Is Important.

Compilers Doesn't Act The Same Way.

How To Find A Solution

Modular Programming Is A Great Tool

https://www.youtube.com/watch?v=uUqXLixmM9E&t=12s

https://www.youtube.com/watch?v=S4n-tQZnU6o

https://www.baeldung.com/java-matrix-multiplication