User Guide :: matrix.h

> **AUTHOR : MD. Shifat Hasan**
>
> **Email : shifathasangns@gmail.com**

Necessary `Header Files`

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
```

Important `Macros`

> **#define `OPERATION` VALUE**

```
#define ADDITION 0
#define SUBTRACTION 1
#define MULTIPLICATION 2
#define DETERMINANT 3
#define MINOR 4
#define COFACTOR 5
#define TRANSPOSE 6
#define ADJOINT 7
#define INVERSE 8
```

> **#define `TYPE_OF_MATRIX` VALUE**

```
#define ROW_MATRIX 9
#define COLUMN_MATRIX 10
#define SQUARE_MATRIX 11
#define DIAGONAL_MATRIX 12
#define SCALAR_MATRIX 13
#define IDENTITY_MATRIX 14
#define UNIT_MATRIX 14
#define NULL_MATRIX 15
#define ZERO_MATRIX 15
```

```
#define UPPER_TRIANGULAR_MATRIX 16
#define LOWER_TRIANGULAR_MATRIX 17
#define TRIANGULAR_MATRIX 18
#define INVOLUTORY_MATRIX 19
#define IDEMPOTENT_MATRIX 20
#define SYMMETRIC_MATRIX 21
#define SKEW_SYMMETRIC_MATRIX 22
#define NILPOTENT_MATRIX 23
```

> #define `TEXT_STYLE` VALUE

```
#define UPPER 24
#define LOWER 25
#define TITLE 26
```

Functions To `Parse Numbers From String`

> It checks whether a `string` contains a `number` or not...

```
bool is_number(char *string)
```

> It `splits` (by a character `c`) a `string` into `multiple strings`...

```
char **split(char *string, char c)
```

> It `parses` a `string` to retrieve `numbers`...

```
double *parse_number(char *string, int num)
```

`Matrix` Structure

```
struct matrix
{
    int rows;
    int cols;
    double **data;
};
```

```
typedef struct matrix* Matrix;
```

How To Make A `Matrix` Type Variable ?

```
int main()
{
    Matrix variable_name;

    return 0;
}
```

How To Work With `Functions` ?

```
int main()
{
    // If the 'function()' returns a Matrix
    Matrix matrix = function_name(...);

    // If the 'function()' returns a *double
    double *array = function_name(...);

    // If the 'function()' returns a double
    double number = function_name(...);

    // If the 'function()' returns a boolian value (true/false)
    bool something = function_name(...);

    return 0;
}
```

`Note` : Keep In Your Mind `That`

All the `index` starts counting from `0` , not from `1` ...

Important `Functions()`

> It checks whether a `Matrix` is `NULL` or not...

```
bool is_null(Matrix matrix)
```

> It checks whether a `Matrix` is perfect for the `OPERATION` or not...

```
bool is_perfect(Matrix matrix, int OPERATION)
```

> It checks whether the `Matrix(s)` are perfect for the `OPERATION` or not...

```
bool are_perfect(Matrix matrix_1, Matrix matrix_2, int OPERATION)
```

> It checks whether the `Matrix(s)` are `identical` or not...

```
bool are_identical(Matrix matrix_1, Matrix matrix_2)
```

`Create-Matrix` Functions()

> It creates a `<` rows `>` by `<` cols `>` `Matrix`

```
Matrix create_matrix(int rows, int cols)
```

> It creates a `Null/Empty Matrix`

```
Matrix null()
```

> It creates a `<` n `>` by `<` n `>` `Identity` or `Unit` `Matrix`

```
Matrix create_identity_matrix(int n)
```

`Input` Functions()

> Is takes `<rows>` by `<cols>` `Matrix` as a `input`

```
Matrix input_matrix(int rows, int cols)
```

> Is takes `<1>` by `<cols>` `Row-Matrix` as a `input`

```
Matrix input_row_matrix(int cols)
```

> Is takes `<rows>` by `<1>` `Column-Matrix` as a `input`

```
Matrix input_column_matrix(int rows)
```

> Is takes `<n>` by `<n>` `Square-Matrix` as a `input`

```
Matrix input_square_matrix(int n)
```

`Modify` Matrix

> It checks whether an Array `<*data>` can be converted into a `<row>` by `<cols>` `Matrix` or not

```
bool is_convertable(double *data, int rows, int cols)
```

> It `converts` an Array `<*data>` into a `<rows>` by `<cols>` `Matrix`

```
Matrix make_matrix_from_array(double *data, int rows, int cols)
```

> It `converts` the `<matrix>` into a `1D-Array`

```
double *make_array_from_matrix(Matrix matrix)
```

> It `Re-Forms` the `dimensions` and `index`(s) of the `<matrix>` if `possible`

```
Matrix reform(Matrix matrix, int rows, int cols)
```

> It `Re-Shapes` the dimensions of the `<matrix>`

```
Matrix reshape(Matrix matrix, int rows, int cols)
```

> It `Adds` a new `<row_matrix>` after the last row of the `<base_matrix>`

```
Matrix append_row_matrix(Matrix base_matrix, Matrix row_matrix)
```

> It `Adds` a new `<column_matrix>` after the last column of the `<base_matrix>`

```
Matrix append_column_matrix(Matrix base_matrix, Matrix column_matrix)
```

> It `Inserts` a new `<row_matrix>` at the `<index_of_row>`-th position of the `<base_matrix>`

```
Matrix insert_row_matrix(Matrix base_matrix, int index_of_row, Matrix row_matrix)
```

> It `Inserts` a new `<column_matrix>` at the `<index_of_col>`-th position of the `<base_matrix>`

```
Matrix insert_column_matrix(Matrix base_matrix, int index_of_col, Matrix column_matrix)
```

> It `Deletes` `<index_of_row>`-th `row` from the `<base_matrix>`

```
Matrix del_row_matrix(Matrix base_matrix, int index_of_row)
```

> It `Deletes` `<index_of_col>`-th `column` from the `<base_matrix>`

```
Matrix del_column_matrix(Matrix base_matrix, int index_of_col)
```

> It `Re-places` `<index_of_row>`-th `row` of the `<base_matrix>` with `<row_matrix>`

```
Matrix replace_row_matrix(Matrix base_matrix, int index_of_row, Matrix row_matrix)
```

> It `Re-places` `<index_of_col>`-th `column` of the `<base_matrix>` with `<column_matrix>`

```
Matrix replace_column_matrix(Matrix base_matrix, int index_of_col, Matrix column_matrix)
```

> It `Removes` the `last-row` from the `<base_matrix>`

```
Matrix pop_row_matrix(Matrix base_matrix)
```

> It `Removes` the `last-column` from the `<base_matrix>`

```
Matrix pop_column_matrix(Matrix base_matrix)
```

`Get` Matrix

> It `Returns` `<index_of_row>`-th `row` of the `<base_matrix>`

```
Matrix get_row_matrix(Matrix base_matrix, int index_of_row)
```

> It `Returns` `<index_of_col>`-th `column` of the `<base_matrix>`

```
Matrix get_column_matrix(Matrix base_matrix, int index_of_col)
```

> It `Prints` the `<matrix>`

```
void print_matrix(Matrix matrix)
```

> It `Prints` the list `<char** types>` which is made by the `Function()` named `types()`

```
void print_types(char **types)
```

`Mathematical` Functions()

> It `Returns` the `Principal-Diagonal` of `<matrix>` in a `Row-Matrix` form

```
Matrix principal_diagonal(Matrix matrix)
```

> It `Returns` the `Principal-Diagonal` of `<matrix>` in a `Diagonal-Matrix` form

```
Matrix principal_diagonal_matrix(Matrix matrix)
```

> The `Returns` the `Trace` of `<matrix>`

```
double trace(Matrix matrix)
```

> It `Returns` the `Secondary-Diagonal` of `<matrix>` in a `Row-Matrix` form

```
Matrix secondary_diagonal(Matrix matrix)
```

> It `Returns` the `Secondary-Diagonal` of `<matrix>` in a `Secondary-Diagonal-Matrix` form

```
Matrix secondary_diagonal_matrix(Matrix matrix)
```

> The `Returns` the `Secondary-Trace` of `<matrix>`

```
double secondary_trace(Matrix matrix)
```

> It `Adds` two `Matrix`(s) `<matrix_1>` and `<matrix_2>`

```
Matrix add(Matrix matrix_1, Matrix matrix_2)
```

> It `Adds` a `<row_matrix>` with all the `rows` of the `<base_matrix>`

```
Matrix add_row_matrix(Matrix base_matrix, Matrix row_matrix)
```

> It `Adds` a `<column_matrix>` with all the `columns` of the `<base_matrix>`

```
Matrix add_column_matrix(Matrix base_matrix, Matrix column_matrix)
```

> It `Subtracts` two `Matrix`(s) `<matrix_1>` and `<matrix_2>`

```
Matrix subtract(Matrix matrix_1, Matrix matrix_2)
```

> It `Subtracts` a `<row_matrix>` from all the `rows` of the `<base_matrix>`

```
Matrix subtract_row_matrix(Matrix base_matrix, Matrix row_matrix)
```

> It `Subtracts` a `<column_matrix>` from all the `columns` of the `<base_matrix>`

```
Matrix subtract_column_matrix(Matrix base_matrix, Matrix column_matrix)
```

> It `Multiplies` all of `index`(s) of the `<matrix>` by a `<scalar_number>`

```
Matrix multiply_by_scalar(Matrix matrix, double scalar_number)
```

> It `Multiplies` two `Matrix`(s) `<matrix_1>` by `<matrix_2>`

```
Matrix multiply(Matrix matrix_1, Matrix matrix_2)
```

> It `Calculates` the `<matrix>` to the `power` of `<n>`

```
Matrix power(Matrix matrix, int n)
```

> It `Returns` the `Minor-Matrix` of the index (`<index_row>`, `<index_col>`)

```
Matrix minor_matrix(Matrix matrix, int index_row, int index_col)
```

> It `Calculates` the `Determinant` of the `<matrix>`

```
double determinant(Matrix matrix)
```

> It `Calculates` the `Minor` of `<matrix>` for the index (`<index_row>`, `<index_col>`)

```
double minor(Matrix matrix, int index_row, int index_col)
```

> It `Calculates` the `Co-Factor` of `<matrix>` for the index (`<index_row>`, `<index_col>`)

```
double co_factor(Matrix matrix, int index_row, int index_col)
```

> It `Returns` the `Transpose-Matrix` of the `<matrix>`

```
Matrix transpose(Matrix matrix)
```

> It `Returns` the `Adjoint-Matrix` of the `<matrix>`

```
Matrix adjoint(Matrix matrix)
```

> It `Returns` the `Inverse-Matrix` of the `<matrix>`

```
Matrix inverse(Matrix matrix)
```

> It can `Solve` the `Simultaneous-Linear-Equations` by taking a `Square-Matrix` (It contains the `Coefficients`) and a `Column-Matrix` (It contains the `Constants` after the `'='` sign)

```
Matrix solve(Matrix coefficients_square_matrix, Matrix constants_column_matrix)
```

> It `Checks` the `<TYPE>` of a `<matrix>`

```
bool is_type_of(Matrix matrix, int TYPE)
```

> It `Lists` all possible `Types` of a `<matrix>` and `Returns` a `2D-Char-Array` where all the `items` will have the same `<TEXT_STYLE>`

```
char **types(Matrix matrix, int TEXT_STYLE)
```
