HACETTEPE UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

BBM203 PROGRAMMING LAB.
ASSIGNMENT 1

Subject	: Data Structures and Algorithms
Submission Date	: **24.10.2019**
Deadline	: **13.11.2019 23:59:59**
Programming Language	: C
Advisor	: R.A. Alaettin UÇAN

1. INTRODUCTION / AIM:

In this experiment, you are expected to gain knowledge on C language including read-write files, arrays, matrices, recursion and dynamic memory allocation. Prior knowledge on basic C syntax is assumed.

2. BACKGROUND INFORMATION

2.1. C LANGUAGE

C is an imperative procedural language. It was designed to be compiled using a relatively straightforward compiler, to provide low-level access to memory, to provide language constructs that map efficiently to machine instructions, and to require minimal run-time support.

Despite its low-level capabilities, the language was designed to encourage cross-platform programming. A standards-compliant and portably written C program can be compiled for a very wide variety of computer platforms and operating systems with few changes to its source code.

The language has become available on a very wide range of platforms, from embedded microcontrollers to supercomputers.

2.2. ARRAYS

Array types in C are traditionally of a fixed, static size specified at compile time. However, it is also possible to allocate a block of memory at run-time, using the standard library's malloc function. C's unification of arrays and pointers means that declared arrays and these dynamically allocated simulated arrays are virtually interchangeable.

Static

```c
int array[10];
```

Dynamic

```c
int * array = malloc(10 * sizeof(int));
```

Since arrays are always accessed via pointers, array accesses are typically not checked against the underlying array size. Array bounds violations are therefore possible and rather common in carelessly written code, and can lead to various repercussions, including illegal memory accesses, corruption of data, buffer overruns, and run-time exceptions. If bounds checking is desired, it must be done manually.

2.3. MULTI-DIMENSIONAL ARRAYS

C does not have a special provision for declaring multi-dimensional arrays, but rather relies on recursion within the type system to declare arrays of arrays, which effectively accomplishes the same thing. The index values of the resulting "multi-dimensional array" can be thought of as increasing in row-major order.

Multi-dimensional arrays are commonly used in numerical algorithms to store matrices. The structure of the C array is well suited to this particular task. However, since arrays are passed merely as pointers, the bounds of the array must be known fixed values or else explicitly passed to any subroutine that requires them, and dynamically sized arrays of arrays cannot be accessed using double indexing. (A workaround for this is to allocate the array with an additional "row vector" of pointers to the columns.).

2.4. DYNAMIC MEMORY ALLOCATION

The task of fulfilling an allocation request consists of locating a block of unused memory of sufficient size. Memory requests are satisfied by allocating portions from a large pool of memory called the heap or free store. At any given time, some parts of the heap are in use, while some are "free" (unused) and thus available for future allocations.

The C dynamic memory allocation functions are defined in stdlib.h header.

Function	**Description**
malloc	allocates the specified number of bytes
realloc	increases or decreases the size of the specified block of memory. Reallocates it if needed
calloc	allocates the specified number of bytes and initializes them to zero
free	releases the specified block of memory back to the system

2.5. RECURSION

Recursion is the process of repeating items in a self-similar way. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function. The C programming language supports recursion, i.e., a function to call itself. But while using recursion, programmers need to be careful to define an exit condition from the function, otherwise it will go into an infinite loop.

2.6. MEMORY LEAK

A memory leak is a type of resource leak that occurs when a computer program incorrectly manages memory allocations in such a way that memory which is no longer needed is not released. A memory leak may also happen when an object is stored in memory but cannot be accessed by the running code. You can use Valgrind to check for Memory Leak in your program.

2.7. ONLINE RESOURCES

• Wikipedia C Entry: https://en.wikipedia.org/wiki/C_(programming_language)
• C Tutorial : https://www.tutorialspoint.com/cprogramming
• Valgrind : http://valgrind.org/info/about.html

3. EXPERIMENT

In this experiment, it is aimed to write a program that can perform vector and matrix manipulation operations.

For this purpose, the vector, matrix data and the command list must be read from the file. Each line in the command list represents a manipulation process to be performed.

You should use your vector and matrix structure to implement all these operations in the C programming language. To store the vectors and matrices, you must use dynamic memory allocation.

Input files are consist of integer numbers separated by spaces and new lines. Vector files ".vec" and matrix files have ".mat" file extension. You can determine the type of an input file by looking at the file extensions.

Command types

- **Initialization**
    - **veczeros [name] [length]:** Creates a new vector of given length and name, filled with zeros.
      Parameters: “name”: name of created vector, “length”: length of vector
      Return: vector
      Output: created vector [name] [length]
      [elements of vector]

- **matzeros [name] [x] [y]**: Creates a new matrix of given x, y and name, filled with zeros.
  Parameters: “name”: name of created matrix, shape=“x”,“y”
  Return: matrix
  Output: created matrix [name] [x] [y]
  [elements of matrix]

- **vecread [filename]**: Reads the vector from the file with the given filename
  Parameters: “filename”: path of the vector file to read
  Return: vector
  Output: read vector [name] [length]
  [elements of vector]

- **matread [filename]**: Reads the matrix from the file with the given filename
  Parameters: “filename”: path of the sparse matrix file to read
  Return: matrix
  Output: read matrix [name] [x] [y]
  [elements of matrix]

- **Concatenation**
  - **vecstack [vector1] [vector2] [direction] [name]**: Combines “vector1” and “vector2” and creates a matrix.
    Parameters: Function takes the names of the vectors and the form of assembly as parameter. The "direction" parameter can be “row” or “column”. The new matrix name is given with the "name" parameter.
    Return: matrix
    Output: vectors concatenated [name] [x] [y] [newline]
    [elements of new matrix]

  - **matstack [matrix1] [matrix2] [where]**: Combines “matrix1” and “matrix2”.
    Parameters: Function takes the names of the matrices and the form of assembly as parameter. The "where" parameter can be “r” for right and “d” for down.
    Return: matrix
    Output: matrices concatenated [matrix1] [x] [y] [newline]
    [elements of matrix1]

  - **mvstack [matrix] [vector] [where]**: Combines “matrix” and “vector”.
    Parameters: Function takes name of the matrix, name of vector. The "where" parameter can be “r” for right and “d” for down.
    Return: matrix
    Output: matrix and vector concatenated [matrix] [x] [y] [newline]
    [elements of matrix]

- **Padding**
    - **pad [matrix] [x] [y] [mode]:** Enlarges the size of the matrix and sets the values of the new elements.

      Parameters: Function takes name of the matrix, x, y and the “mode” as parameter. The mode parameter can be “maximum” or “minimum”.
        - "**maximum**": Pads with the maximum value of all or part of the vector along each axis.
        - "**minimum**": Pads with the minimum value of all or part of the vector along each axis.

      Return: matrix

      Output: matrix paded [matrix] [x] [y] [newline]
      [elements of matrix]

    - **padval [matrix] [x] [y] [val]:** Enlarges the size of the matrix and sets the values of the new elements.
      Parameters: Function takes name of the matrix, x, y and the “val” as parameter.
      Return: matrix
      Output: matrix paded [matrix] [x] [y] [newline]
      [elements of matrix]

- **Slicing**
    - **vecslice [vector] [start] [stop] [name]:** Takes a part from the given vector.
      Parameters: Function takes "vector name", "start", "stop" and “name” as parameters. "start" is the start index and "stop" is the end index. The taken slice name is given with the "name" parameter.
      Return: vector
      Output: vector sliced [name] [elements]
      [elements of vector]

    - **matslicecol [matrix] [index] [start] [stop] [name]:** Takes a part of vector from the given matrix.
      Parameters: Function takes "matrix name", “index”, "start", "stop" and “name” as parameters. “index” is the column index, "start" is the start index and "stop" is the end index. The taken slice name is given with the "name" parameter.
      Return: vector
      Output: vector sliced [name] [elements]
      [elements of vector]

- **matslicerow [matrix] [index] [start] [stop] [name]:** Takes a part of vector from the given matrix.
  Parameters: Function takes "matrix name", “index”, "start", "stop" and “name” as parameters. “index” is the row index, "start" is the start index and "stop" is the end index. The taken slice name is given with the "name" parameter.
  Return: vector
  Output: vector sliced [name] [elements]
  [elements of vector]

- **matslice [matrix] [y1] [y2] [x1] [x2] [name]:** Takes a part of matrix from the given matrix.
  Parameters: Function takes "matrix name", "y1", “y2”, “x1”, "x2" and “name” as parameters. “y1” is the start column index, "y2" is the end column index, “x1” is the start row index, "x2" is the end row index. The taken matrix name is given with the "name" parameter.
  Return: matrix
  Output: matrix sliced [name] [x] [y] [newline]
  [elements of matrix]

- **Math**

  - **add [matrix1] [matrix2]:** calculates the element-wise sum of two matrices and overwrites matrix1
    Parameters: Function takes the names of the matrices.
    Return: matrix
    Output: add [matrix1] [matrix2] [newline]
    [elements of matrix1]

  - **multiply [matrix1] [matrix2]:** calculates the element-wise product of two matrices and overwrites matrix1
    Parameters: Function takes the names of the matrices.
    Return: matrix
    Output: multiply [matrix1] [matrix2] [newline]
    [elements of matrix1]

  - **subtract [matrix1] [matrix2]:** calculates the element-wise subtraction of two matrices and overwrites matrix1
    Parameters: Function takes the names of the matrices.
    Return: matrix
    Output: subtract [matrix1] [matrix2] [newline]
    [elements of matrix1]

3.1. EXECUTION

The name of the compiled executable program should be “matrixman”. Your program should read input/output file names from the command line, so it will be executed as follows:

```
matrixman [folder_of_input_files] [command_list_file_name] [output file name]
```

```
e.g.: ./matrixman inputs commands.cmd output.out
```

You can see sample input and output in piazza page. The program must run on DEV (dev.cs.hacettepe.edu.tr) UNIX machines. So make sure that it compiles and runs in one of the UNIX lab. If we are unable to compile or run, the project risks getting zero point. It is recommended that you test the program using the same mechanism on the sample files (provided) and your own inputs. You must compare your own output and sample output. If your output is different from the sample, the project risks getting zero point, too.

3.2. INPUT/OUTPUT FORMAT

In the commands file, the numbers are separated by spaces and newlines. You do need to check the input for errors. An example commands file can be given as follows:

```
matread m3.mat
matread m4.mat
subtract m3 m4
matslice m4 5 9 0 3 m8
matslicerow m4 1 2 6 v7
mvstack m8 v7 d
subtract m3 m8
```

Sample matrix and vector files:
v1.vec

```
7 1 9 4 6 8 1 9 1 9
```

m1.mat

```
1 1 1
1 2 1
1 1 1
```

m3.mat

```
1 2 3 4
4 5 6 7
7 8 9 9
1 1 1 1
```

Your program should write outputs to the output file, so that each line in the output file will contain the result of multiplication. If the above input file is given, the output file should be as below:

```
read matrix m3.mat 4 4
1 2 3 4
4 5 6 7
7 8 9 9
1 1 1 1
read matrix m4.mat 3 9
8 7 6 5 8 4 1 5 4
5 2 8 7 3 1 4 1 2
2 3 6 4 7 9 2 1 4
error
matrix sliced m8 3 4
4 1 5 4
1 4 1 2
9 2 1 4
vector sliced v7 4
8 7 3 1
matrix and vector concatenated m8 4 4
4 1 5 4
1 4 1 2
9 2 1 4
8 7 3 1
subtract m3 m8
-3  1 -2  0
 3  1  5  5
-2  6  8  5
-7 -6 -2  0
```

3.3. DESIGN EXPECTATIONS

You must use dynamic memory allocation for your solution. Writing spaghetti, or static arrays could render your entire assignment “unacceptable” and make it subject to huge (if not complete) grade loss.

3.4. VALID PLATFORMS

Your code will be compiled against gcc. You should not assume the availability of any other non-standard libraries/features. If you use a different compiler, it is your responsibility to ensure that your code has no compilation issues with the version of gcc in dev server.

4. EVALUATION

4.1. REQUIRED FILES

You should create and submit a ZIP archive in the following structure for evaluation. An invalid structured archive will cause you partial or full score loss.

You are required to submit a Makefile[1], which will be used to compile your program to generate the matrixman executable.

Directory	Files	Description
Source	*.c, *.h, Makefile	Program source/header files and Makefile
Report	*.pdf	Your report (Only pdf format is accepted)

4.2. REPORTS

You must write a report which is related to your program. The topics that should be included in the report are:

- Problem definition
- Methods and solution
- Functions implemented and <u>not</u> implemented

4.3. GRADING

Make sure that the source code you submitted is compiled with gcc compiler under Linux OS. Applications that produce compilation or runtime errors cannot be graded.

- The program you have submitted will be tested with five different inputs. You can achieve 13 points for each correct output file, total 65 points.
- Your source code must have "Dynamic memory allocation, recursion, multidimensional array, and comment lines". You can achieve 5 points for each item, total 20 points.
- You can achieve 5 points for each topic, total 15 points for report.
- Points you can get from report is directly correlated with the total points you get from execution and code review sections. For example if you get 57/85 from first two sections points you can from report section would be at most 10 points.
- In the event of a memory leak, you will be reduced by half the point you deserve.

[1] Make file example http://mrbook.org/tutorials/make/

LAST REMARKS:

- The output of your program will be graded automatically. Therefore, any difference of the output (even a smallest difference) from the sample output will cause an error and you will get 0 from execution. Keep in mind that a program that does not work 100% right is a program that works wrong.
- There will be incorrect lines in the input file. Your program should continue to run without giving a runtime error. For the wrong lines in the input file, you must type "error" in the corresponding line in the output file.
- Regardless of the length, use **UNDERSTANDABLE** names to your variables, classes and functions.
- Write **READABLE SOURCE CODE** block
- **You will use online submission system to submit your experiments. https://submit.cs.hacettepe.edu.tr/ Deadline is: 13.11.2019 23:59:59. No other submission method (email or etc.) will be accepted.**
- Do not submit any file via e-mail related with this assignment.
- **SAVE** all your work until the assignment is graded.
- The assignment must be original, **INDIVIDUAL** work. Duplicate or very similar assignments are both going to be punished. General discussion of the problem is allowed, but **DO NOT SHARE** answers, algorithms or source codes.
- You can ask your questions through course's piazza page and you are supposed to be aware of everything discussed in the page.

REFERENCES

1. The C Programming Language (Second Edition), B.W. Kernighan, D. M. Ritchie
2. Let Us C (Computer Science) 8th Edition, Y. P. Kanetkar