

**HACETTEPE UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING**

**BBM203 PROGRAMMING LAB.
ASSIGNMENT 1**

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

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

Dynamic

```
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
<code>malloc</code>	allocates the specified number of bytes
<code>realloc</code>	increases or decreases the size of the specified block of memory. Reallocates it if needed
<code>calloc</code>	allocates the specified number of bytes and initializes them to zero
<code>free</code>	releases the specified block of memory back to the system

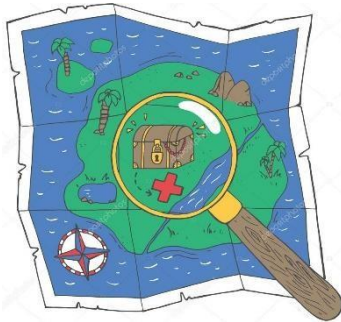
2.5. RECURSION

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. ONLINE RESOURCES

- Wikipedia C Entry: [https://en.wikipedia.org/wiki/C_\(programming_language\)](https://en.wikipedia.org/wiki/C_(programming_language))
- C Tutorial : <https://www.tutorialspoint.com/cprogramming>

3. EXPERIMENT



In this experiment, it is aimed to find the hidden treasure within a treasure map designed as a matrix.

For this purpose, the map and key data must be read from the file. Afterwards, the key must be moved on the map depending on the rules. The result of the multiplication of the map matrix and the key matrix will give an idea of the place of the treasure.

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

[illegible]

Table 1 An Example For Treasure Map Matrix

The treasure map is designed as a matrix as shown in Table 1. The size of the map matrix is not fixed but must be determined according to the input file that will be given at runtime. Each element of the map matrix is a number. The elements of the matrix placed in the edges are zeros. That show you the boundary of the map.

0	-1	0
-1	20	-1
0	-1	0

Table 2 An Example Of Key Matrix

As shown in Table 2, the key is designed as a square matrix with degree n that is an odd value (e.g. 3,5,7). n is a variable that will be determined at runtime. By using the key matrix, the hidden treasure can be found in the map.

In order to find treasure, we slide the key matrix on the map. The part of the map matrix that is covered by the key specify a sub-matrix on the map. By multiplying the key matrix by the obtained sub-matrix, we find the direction of the next sliding. The direction is computed by taking mod five of multiplication result (result % 5). The result of the mod operation is interpreted as follows:

- 0: Found treasure
- 1: Go up
- 2: Go down
- 3: Go right
- 4: Go left

The search operation must be start from the top left corner of the map matrix. The details of the first step are shown in Table 3. In order to get a full score from this assignment, you have to design search function recursively.



0	0	0	X	0	-1	0	=	58	$58 \% 5 = 3$			1,1:58
0	3	1		-1	20	-1						
0	1	5		0	-1	0						
A				B				C	D	E	F	G

Table 3 Start Search

- A: Sub-matrix
- B: Key Matrix
- C: Multiplication of matrices A and B
- D: Remaining part of the mod (5,C)
- E: Estimated direction
- F: Final direction
- G: Output: write center cell address of sub-matrix and C

In case of key matrix is placed on the boundary of map (e.g. most right or most left) where there is no way to slide through the determined direction, you must move in the opposite direction as shown in Table 4. Actually, the estimated direction may be changed by the mentioned exception.



0	0	0	X	0	-1	0	=	146	$146 \% 5 = 1$			4,1:146
5	8	6		-1	20	-1						
8	3	2		0	-1	0						
A'				B				C	D	E	F	G

Table 4 Handling the exception for the estimated direction.



2	9	3	X	0	-1	0	=	140	$140 \% 5 = 0$			4,4:140
5	8	1		-1	20	-1						
7	5	1		0	-1	0						
A''				B				C	D	E	F	G

Table 5 Treasure Found!

If the mod result is zero, the treasure is found as shown in Table 5. The place of the treasure is the midpoint of the sub-matrix. Your program must print out the midpoints of the sub-matrix in each step.

3.1. EXECUTION

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

findtreasure [size of map matrix] [size of key matrix] [input file name] [output file name]
e.g.: ./findtreasure 18x18 3 mapmatrix.txt keymatrix.txt output.txt

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 map matrix file, the numbers are separated by spaces and newlines. You do not need to check the input for errors. An example map matrix file can be given as follows:

```
00000000000000000000
031586947512928360
015832846937468930
016293589261897310
057581348515893520
094751294751292830
084693784693746890
058926158926189730
038134851538134850
075129475175129470
069378469369378460
092615892348515890
084693737294751290
058926161784693740
038134829158926180
075129481348846840
084693751294589580
000000000000000000
```

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

```
1,1:58
1,4:146
4,4:140
```

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.