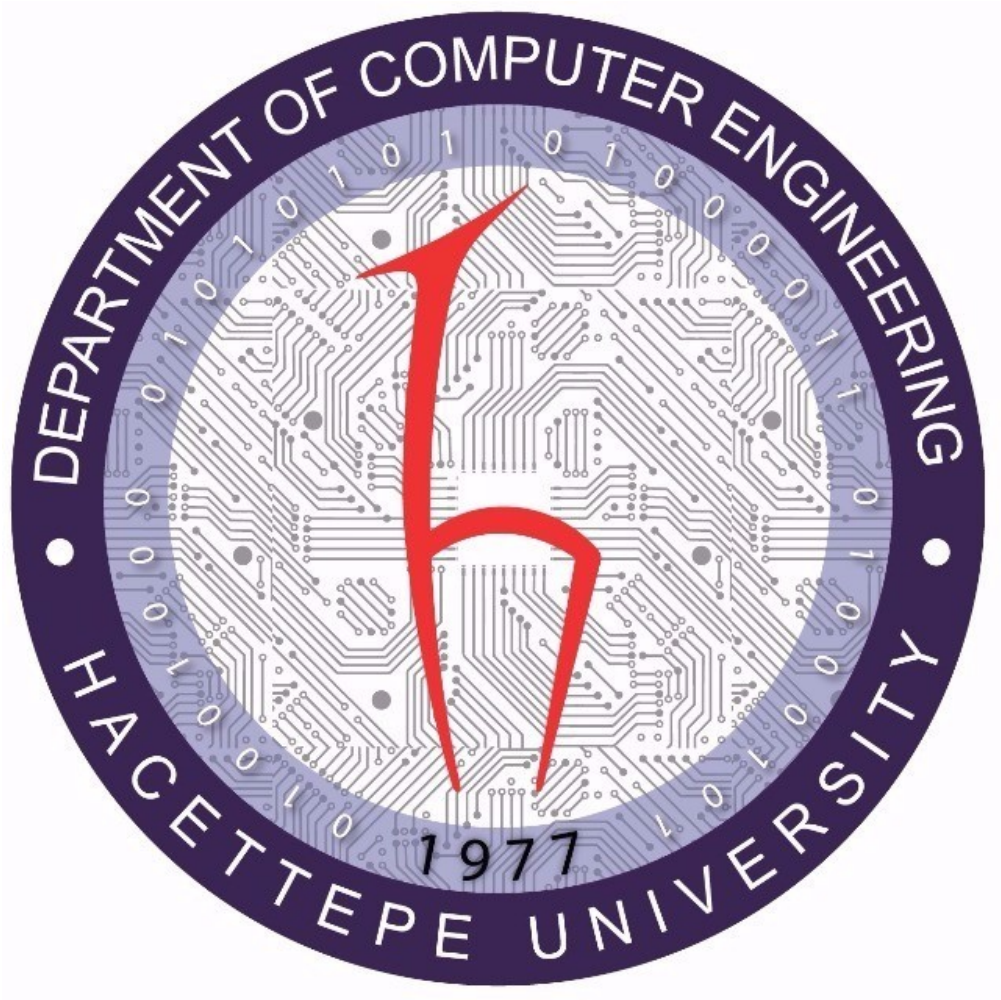

HACETTTEPE UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

GÖKHAN ÖZELOĞLU

b21627557@cs.hacettepe.edu.tr

21627557

BBM 203 - ASSIGNMENT – 1

FIND TREASURE

1. Problem Definition

The problem is finding hidden treasure with giving key matrix and map matrix. Also, this searching operation must be implemented recursively. The other aim is that using dynamic memory allocation while storing the maps. There are some rules for searching operation. The direction rules are like this:

0 – Found Treasure

1 – Go Up

2 – Go Down

3 - Go Right

4 – Go Left

If searching direction goes to out side of the map, it changes the direction into opposite side. While finding above numbers, we did matrix multiplication and took modulus by 5. After that, map matrix is divided into sub-matrix which its size is the same with key matrix size. Also, the searching operation starts in leftmost in the map matrix.

2. Methods and Solution

I implemented my program modular, readable and maintable as much as possibe. I divided my program into little pieces . For instance, reading input files operation is done in separate functions outside of the main function. Moreover, I used helper functions to do small operations like multiplication or tokenizing string. I will explain these funtions below. Dividing into small pieces like functions gave me power to maintain my proogram or debugging. Also, I was careful that not using unnecessary variables, pointers and functions.

3. Functions

```
int main(int argc, char const *argv[]) {
```

The program starts with **main** function. All the other functions are called from here. Generally, I created pointers, opened the files, handled the error situations, calls the recursive and helper functions, closed the files end of the program and freed the all pointers.

```
void read_map_matrix(FILE *file, int **map_matrix, struct Inputs input)
```

This function reads the map matrix and returns nothing because of its type, **void**. It takes a **FILE** pointer and **struct** object. Firstly, the function allocates the each columns by using **malloc** dynamically. Finally, storing the values into **map_matrix** is being done with nested for loop. Outer loop is responsible for rows and inner loop is responsible for columns. I used

fscanf. function to read. There is no need to return **map_matrix** because I just send the address of the map array.

```
void read_key_matrix(FILE *file, int **key_matrix, struct Inputs input)
```

This function reads the key matrix. This is very similiar with above -**read_map_matrix**- function. Firstly, the function allocates the memory dynamically by using **malloc** for each row. Then, assign operation is being done by nested for loop. **fscanf** assigns the each element in matrix into **key_matrix**.

```
int* parse_row_column(char cartesian_notation[])
```

This is a helper function for parse operation. Command line argument **{rowSize}x{columnSize}** is parsed here. Firstly, a pointer array which length is 2. Then, **token** variable defined by using **strtok**. Due to being not sure, I defined **strtok** for parsing upper case and lower case **X**.

```
int multiple_matrix(int **map_matrix, int **key_matrix, int mid_point_row, int mid_point_column, int size_key_matrix)
```

This function calls while searching treasure. One of the most important part is matrix multiplication in this assignment. Function multiplies the key and map matrix with nested for loop. Boundries of the map matrix is set with parameters and for loop. Finally, the function returns the multiplication result.

```
int control_direction(int direction, int mid_row, int mid_column, struct Inputs input, int bound_num)
```

Another helper function is **control_direction**. This controls the direction of searching operation. While searching treasure, after matrix multiplication, takes modulus to find direction. According to modulus result, program must control the boundaries. If it goes to out of the boundaries, this function just changes the direction. If there is not any problem for boundaries, it returns the same number.

```
void search_treasure(FILE *file_output, int **map_matrix, int **key_matrix, int mid_row, int mid_column, struct Inputs input, int i)
```

The most important part is this recursive function in this assignment. Firstly, multiplication, taking modulus, controlling the direction and writing file operations are executed. Then, I looked at the cases by using **if-else** statements. My base case that found treasure is controlled in the first **if** statement. Other **else-if** statements just updates the middlee point of sub-matrix. After updating the middle points, the function calls itself recursively.

```
struct Inputs {
    int row_size;
    int column_size;
    int key_matrix_size;
};
```

This struct just strores the command line arguments to send less parameter into functions.

NOT IMPLEMENTED FUNCTIONS

I do not have any **not implemented** functions. I am thinking that I used all necessary functions that I need to use.