Hacettepe University

Computer Science and Engineering

BBM203 Programming Lab.

Assignment 1

Ahmet Faruk BAYRAK

21426716

22.10.2018 - 04.11.2018

R.A. Alaettin UÇAN

1. Software Using Documentation

1.1. Software Usage

The programs runs on command prompt. Takes inputs as a command line argument. To run the program: firstly you go to the file path which contains main.c file. And write the “make” command. This command is provided by Makefile and compiles main.c file. After that program is ready to start. There are 5 arguments: first one is size of map matrix, second one is size of key matrix, third one is input file name of map matrix, fourth one is input file name of key matrix and last argument is output file name. Example of command line argument is: ./findtreasure 18x18 3 mapmatrix1.txt keymatrix1.txt output1.txt.

1.2. Error Messages

Program does not have any error messages but there are some error handling.

2. Software Design

2.1. Description of the Program

2.1.1. Problem

In this experiment, the problem that has been expected us to solve is find the hidden treasure within a treasure map designed as a matrix. Program reads the map and key data from the file. After that we would move on the map according to the rules. For solving this problem, we were expected to use dynamic memory allocation, recursion and multidimensional array.

2.1.2. Solution

For the solution, first of all I took the arguments and assign them. One problem while assigning the argument is, size of map was given like “18x18”. So I had to parse it according to “x” and assign these numbers as a row and a column. I used these number of row and column in malloc function for creating multidimensional array dynamically. After that I read the map and key data from file. I used fscanf() function for reading. I took the data and created the multidimensional array. When reading was done, I initialized the starting variables such as starting point and called my recursive function, findTreasure().

In my findTreasure() function, firstly program calls matrixMult() function for calculating result of multiplication of sub-matrix of map matrix and key matrix. When calculating is done, program prints the current center of sub-matrix and result to the output file. After that program takes the mod5 of result and switch-case situations welcomes to mod of result. According to the mod of result there are 5 cases:

- First one is mod = 0, it means treasure is founded. In this case program frees multidimensional arrays and exits.

- Second one is mod = 1, it means go up. In this situation program decreases the x variable of first element of sub-matrix by size of key matrix so sub-matrix goes up. Program also checks that whether sub-matrix could go to up. To do this, program checks if x variable of first element of sub matrix is less than zero. If this situation is true, it means there is no up then program increases the x variable of first element of sub-matrix by 2*(key matrix of sub matrix) so sub-matrix goes down.

- Third one is mod = 2, it means go down. In this situation program increases the x variable of first element of sub-matrix by size of key matrix so sub-matrix goes down. Program also check that whether sub-matrix could go to the down. To do this, program checks if x variable of first element of sub-matrix is greater than (number of row in map matrix - size of key matrix). If it is true, it means there is no down then I decreases the x variable of first element of sub-matrix by 2*(key matrix of sub matrix) so sub-matrix goes up.

- Fourth one is mod = 3, it means go right. In this situation program increases the y variable of first element of sub-matrix by size of key matrix so sub-matrix goes right. Program also check that whether sub-matrix could go to the right. To do this, program checks if y variable of first element of sub-matrix is greater than (number of column in map matrix - size of key matrix). If it is true, it means there is no right then I decreases the y variable of first element of sub-matrix by 2*(key matrix of sub matrix) so sub-matrix goes left.

- Last one is mod = 4, it means go left. In this situation program increases the y variable of first element of sub-matrix by size of key matrix so sub-matrix goes left. Program also check that whether sub-matrix could go to the right. To do this, program checks if x variable of first element of sub matrix is less than zero. If it is true, it means there is no left then I increases the y variable of first element of sub-matrix by 2*(key matrix of sub matrix) so sub-matrix goes right.

In each case, after determining of the next movement of sub-matrix, program calls recursively itself while sending new points of sub-matrix as a parameter until mod5 is zero which is first case.

There were some restrictions which we had to handle. For example the result of mod says sub-matrix should go up but if this movement exceed the boundary of map matrix then sub-matrix should go down. This rule is same to horizontal (right and left). I mentioned how program handles these situations above.

Here is the functions that I used in the program:

- int** readFile(FILE *file, int iRow, int iCol)

This function takes data input as a FILE*, int and int. First parameter is the file which we desired to read. Second and third parameter are number of row and column of multidimensional array which is size of matrix in the read file.

First of all function creates empty multidimensional array dynamically according to second and third argument. When creating is done program reads file data and fill the multidimensional array. After that program closes the file and return the multidimensional array.

- int matrixMult(int keyLength, int** mapMaze, int centerX, int centerY, int** keyMaze)

This function takes data input as an int, int**, int, int and int**. First parameter is the size of key matrix. Second parameter is multidimensional array of map matrix. Third and fourth arguments are x and y variable of first element of sub-matrix. And the last argument is multidimensional array of key matrix.

In this function, program multiply each element of sub-matrix and key matrix and add them. After that program returns final result.

- void calculation(int **mapMaze, int **keyMaze, int centerX, int centerY, int keyLength, int mapRow, int mapCol, FILE *outFile)

This function is my recursive function. First parameter is multidimensional array of map matrix. Second parameter is multidimensional array of key matrix. Third and fourth parameters are x and y variable of first element of sub-matrix. Fifth parameter is size of key matrix. Sixth and seventh parameters are number of row and column of map matrix. And the last parameter is output file.

First of all program calls matrixMult() function and calculate result of multiplication. Then program write the center of current sub-matrix and result to file using fprintf() function. After that program takes mod5 of result and switch-case structure welcomes to mod. If mod5 = 0, program closes the output file, frees multidimensional arrays and exit program. If mod5 = 1 sub-matrix moves up as long as there is up. If moving up means exceeding of map matrix then sub-matrix moves down. If mod5 = 2 sub-matrix moves down as long as there is down. If moving down means exceeding of map matrix then sub-matrix moves up. If mod5 = 3 sub-matrix moves right as long as there is right. If moving right means exceeding of map matrix then sub-matrix moves left. If mod5 = 4 sub-matrix moves left as long as there is left. If moving left means exceeding of map matrix then sub-matrix moves right. Each case call the findTreasure() function recursively. I mentioned the details of how this switch-case structure works above.

There are no functions which I did not implement.

2.1.3. Algorithm

1) Program takes arguments
2) Program takes first argument and split it. After splitting assign them number of row and column of map matrix
3) Program takes second argument and assign it to size of key matrix
4) Program takes third, fourth and fifth argument which are file name of map matrix, key matrix and output. And open the files.
5) Program sends map matrix file and key matrix file to readFile()
6) readFile() function creates and fills the multidimensional arrays.
7) After reading program calls recursive function, findTreasure() function.
8) findTreasure() function firstly calls matrixMult() function for matrix multiplication.
9) matrixMult() function calculates result of multiplication and return total.
10) According to mod5 of this result, findTreasure() function starts switch-case structure.
11) Each case calls findTreasure() function recursively until mod5 = 0
12) If mod5 = 0 program exits.