HACETTEPE UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

FALL 2018

BBM 203 PROGRAMMING LAB. ASSIGNMENT 1 REPORT

Name and Surname:	Ege Berke Balseven
Number:	21590776
Subject:	Matrix Operations
Due Date:	04.11.2018

Contents

1 Problem definition

This assignment was to find the hidden treasure within a treasure map designed as a matrix.That's why we have key and map matrix. The map matrix and key matrix data should be read from the file. Then the key matrix has to be moved on the map depending on the rule. The result of the multiplication of the map matrix and the key matrix provides information about the location of the treasure.

Problems:

Size of the map matrix and key matrix are not fixed, so it should be determined by to the input that will be given at run time. The key matrix may be a square matrix but the map matrix may not, the input must be arranged accordingly. For both matrices dynamic allocation must be performed and segmentation fault should not be taken. Since the matrix is 2-dimensional, must be created accordingly. Parameters should be sent correctly when sending and receiving and must be correctly defined in the sent function. Since the search function is recursive, base case provided which is the search should end when the result of operations returns 0. Also if the result of the multiplication is negative, action must be taken. To decide the direction to go to find the treasure, if there is no place left, right, above or below, it has to go the opposite direction. Multiplication may give incorrect value in recursive function or may be mixed, so it should be done in another function. Where the sub matrix information is written to the file, recursive function can be inconvenience if you try it in, can be rewritten to the file, so that write function must be defined.

2 Methods and solutions

There is no problem because the key matrix is square, but i separated the map matrix row and column sizes with strtok. Because the matrices are two-dimensional, I used a double pointer to define. While dynamically allocating to matrices, i allocate for key matrix as key size but for map matrix firstly i allocate as map matrix row size, then i multiply rows with respect to column sizes. For example, even if the temp value i that counts row size, if i use i++ instead of ++i i got a segmentation fault on dev server. also in any case the size of the matrix is exceeded i got fault again. But i handled oversize checking in search function. After when we're done with the matrices,i have to deallocate with free(), memory needs to be emptied. Also I have to close the files after reading and writing that. I am done with the main function, after only parameter name is written when parameters are sent to search function but the search function has to write its type when receiving the parameters. It should specify if it is double pointer int or char* etc. When base case is provided in

search function that is, I get out of function when I find zero. I've defined different functions for multiplication operations and file write operations so that the Recursive function does not fail. The order in which these functions are called is important and they have to put in the right place. I'm sending that before entering the key matrix shifting. Because multiplication is done for the current sub matrix and if I would send the values to the output after sliding it, the values for the sub matrix were typed incorrectly. If the result of the multiplication is negative, the result of the mode operation may be negative and this is a problem. That is why i add 5 to mode operation result. Also since the row and column that i send to output should give the center of the sub matrix. So i add keysize/2 to current row, current column to print mid row, column of sub matrix. Before i move to find treasure, if the map matrix ends in the direction to go i'm going the other way. To determine this, for example rows are moved up to the key matrix size when going up and if there is no space above, then we try to go to the index under zero, so row - keysize gives us negative value. Then i have to add that instead of subtract with the key size to go to opposite direction. If row-key size was positive then no problem and i can go up with subtract that. This also applies to other situations. For example to go right or down if the map matrix size is exceeded, then the same procedures are done. Outgoing parameters for multiplication: current row and current column, key matrix, key matrix size, map matrix. Since the sub matrix we are going to process will be as much as key size and i will continue to proceed from where the sub matrix be in map matrix, i take keysize, current row and current column as parameters. With two simple for loops i multiply values for each row and column and i added the results to the count variable every time and return it. While the search function is recursive, i have written the file using append to avoid rewriting on top of each resulting file.

3 Functions implemented and not implemented

Main:
takes map matrix size
takes map matrix row size(str) with strtok
takes map matrix column size(str) with strtok
key matrix size(str)
map matrix row size
map matrix column size
key matrix size
curcolumn = initial current column for map matrix
currow = initial current row for map matrix

Dynamic memory allocation for key matrix
file read for key matrix
assigning values in the file
Dynamic memory allocation for map matrix
file read for map matrix
assigning values in the file
goes the search function with parameters
close files
deallocate matrices

searchtreasure:
keysize/2 for reaching sub map matrix(map*key)'s middle for output
multiresult = multiply result
resultvalue = mod 5 of multiply result for operations
if mod negative, turn positive
current row+a and current column+a shows that where is the sub mutrix
if (resultvalue==0) treasure found, exit
currow= current row of mapmatrix, curcolumn= current column of map matrix
for going up,if currow-keysize <=0 it in the map matrix so i can subtract keysize from currow
for going down,if it is not exceed the map matrix row, then i can add currow and keysize
for going right, if it is not exceed the map matrix column, then i can add curcolumn and keysize
for going left, if curcolumn-keysize =0 it in the map matrix so i can subtract keysize from curcolumn
for going up,if current row -keysize <0 it can not go up so it has to go down, i add currow and keysize
for going down, if it is not exceed the map matrix row, then i can subtract currow from keysize
for going right, if it is exceed the column, then it has go to left, so i subtract keysize from curcolumn
for going left, if curcolumn-keysize negative then it is not in the map matrix, then i add them for going right
multiply:
count = multiply result(multiresult)
for loop for multiply each key and map column
for loop for multiply each key and map row
count+=((mapmatrix[i+currow][j+curcolumn])*(keymatrix[i][j])): multiplies key matrix and sub matrix and adds count
output:

prints the results to the specified file name