CUSTOM PROGRAM DESIGN REPORT
MUHAMMAD SHAEEL ABBAS
101213029

Background:

To navigate a path from point A to point B and maximise the number of attractions.
The map is a grid.
There are many attractions along every street in the city.
Purpose: The Tourist want to see as much attractions as possible
Limitation: They will only move South or East.

The city graph can be represented as a grid with numbers next to each line called **weights**.
The number will represent “Number of attractions” on every block
This entire this would be a **graph**
The intersections of streets are called **vertices**,
Street itself will be **edges** with **weight** associated to them.

User Manual:

When you run the program file, a console will appear. The console will display some instructions and give very brief instructions and it will firstly ask for your name, after which it will store that somewhere in the memory. After inputting your details, there will be a “grid” like structure that will be displayed in the console. It will consist of random numbers example maybe “0,4,7,1” and the symbols “-” “|”. The combination of these elements make up a grid. You cursor will start from the top left element(number) of the grid.This would be indicated by maybe “|4|”. The objective would be to reach the bottom right element. Which would also be a number that will be in pipes such as maybe “|2|”. As the cursor would go along the grid, it would leave it’s trace along the grid’s elements using “| |” besides the elements or “attractions”. Showing that these “attractions” have been “visited”. Furthermore, the console would also show the total number of attractions that have been successfully visited.

The ultimate task for the user is to visit the maximum number of “attractions” going only in the directions of either South (down) and East (right). A vital piece of information for us to note here is that once the user quits the game in between the session, the session will not write details(such as the name or the accumulated score that was stored in variables) in the text file. This is mentioned as a warning as a part of the introduction in the console.

Short Description:

The program uses a lot of elements that are unique to their own functionality. I'd like to briefly point out a few such as pointers, 2D Arrays, nested for loops, Functions,(Pass by value/reference) and the proper use of Struct (Structures). Things like if statements are used also. The program begins with an introduction of the scenario and instructions, followed by the console asking the name of the user.

Let me first lay out the fundamental elements of my entire program. We **essentially** have an 8 x 8 grid (64 elements) that are very much randomised. Of course, computers never truly "randomise" anything because they are not as yet capable of doing so, so hence I used the "<time.h>" library combined with srand to seed the values. The 2D array (member of a struct) is now filled with "random" integers. This is in one function called randomiseGrid.

Display Mechanism:
Using a 2D array, I then use a nested for loop and make it display as a grid using 8 rows and 8 columns with "weights" and "vertices". Weights and vertices are actually hardcoded and aligned manually using the printf function inside the nested for loop. This is actually inside one single function called "displayGrid". Eventually, the user is given the grid with the first element being the starting point. It is demonstrated by having 2 pipes between the number. The last element is also formatted like this. Once the user decides to make an action right using "r" or down by using "d" or use "q" to quit the program, selection statements are used.

Score and Movement mechanism:
Let me very briefly talk about the mechanism to "move" right and down. We must consciously keep in mind that our "grid" is simply a mix of rows and columns. To move "right" from a certain point, you are simply moving to the next column on the right.. Similarly, to move "down" from a certain point, you are simply down one row. This could be illustrated in variable forms as "column++" or "column=column+1" and same for row **(only for demonstration purpose, not my actual variables)** Hence, the score mechanism is just a variable that keeps adding elements with help of column++ and row++ and adding the index of specific indexes from the array values . When I say elements these are specific numbers in a 2D array that are specified by row number and column number. I have simplified the process in this text but the code is something like this.

```
int handleInput(gameGrid *grid, user *userData) /*pointer to grid type(struct) called grid and a pointer using user type(struct) called userData */
{
    /* declare variable to get user input and use it to perform appropriate action */
    char userInput;
    printf("\n \nPlease press r to go right, d to go down, q to quit, h to restart: ");
    scanf(" %c",&userInput);

    if((userInput=='r'|| userInput=='R') && grid->column < 7) /*Error handling, if lower case r or upper case R  */
    {   /* Only if the column value is less then 7 then our program will move towards the element on the right. Mainly because our index starts from 0 */
        (grid->column)++; /*add 1 to the value of pointer */
        (userData->score) += grid->indexes[grid->row][grid->column].value; /* score member of userData accumulates score corresponding the value at a specific row/gri
        grid->indexes[grid->row][grid->column].visited=1;   /*sets the specific element to visited, which means later on it can be printed in pipes*/
        return 1; /*gets stored in the main in continueGame*/
    }
    else if((userInput=='d' || userInput=='D') && grid->row < 7) /*similar pattern follows for row*/
    {
        (grid->row)++; /*add 1 to the value of pointer */
        (userData->score) += grid->indexes[grid->row][grid->column].value;  /* score member of userData accumulates score corresponding the value at a specific row/gr
        grid->indexes[grid->row][grid->column].visited=1; /*sets the specific element to visited, which means later on it can be printed in pipes*/
        return 1; /*gets stored in the main in continueGame*/
    }

    return handleInput2(grid, userData, userInput); /* Calls handleInput2, which will eventually return back 2.. */
}
```

Trace Mechanism:

Let me rewind a bit now. In my program, I thought of a struct called gameGrid with 3 members, int row; int column and gridIndex indexes[8][8]

gridIndex itself is a struct that has 2 members which are int value and int visited;
I then make a variable called grid with data type of gameGrid.

That means our variable grid is now inherited new properties. Value would be the random number/element and “visited” would simply be either 1 or 0. 1 being “visited”, 0 being not “visited”. Talking about the trace mechanism, our program is required to leave a simple “mark” of wherever it goes or whichever element is passes through. It uses “ | | “ to include the element/number inside the 2 pipes. If you relate these two ideas,you can set the first and last element’s “visited” property to 1. After setting this to 1, in the display function, I used a selection statement that if the grid element visited property is 1 then print out that element between two pipes. I would like to demonstrate this using a snapshot of code.

```
int handleInput(gameGrid *grid, user *userData) /*pointer to grid type(struct) called grid and a pointer using user type(struct) called userData */
{
    /* declare variable to get user input and use it to perform appropriate action */
    char userInput;
    printf("\n \nPlease press r to go right, d to go down, q to quit, h to restart: ");
    scanf(" %c",&userInput);

    if((userInput=='r'|| userInput=='R') && grid->column < 7) /*Error handling, if lower case r or upper case R  */
    {   /* Only if the column value is less then 7 then our program will move towards the element on the right. Mainly because our index starts from 0 */
        (grid->column)++; /*add 1 to the value of pointer */
        (userData->score) += grid->indexes[grid->row][grid->column].value; /* score member of userData accumulates score corresponding the value at a specific row/gri
        grid->indexes[grid->row][grid->column].visited=1;   /*sets the specific element to visited, which means later on it can be printed in pipes*/
        return 1; /*gets stored in the main in continueGame*/
    }
    else if((userInput=='d' || userInput=='D') && grid->row < 7) /*similar pattern follows for row*/
    {
        (grid->row)++; /*add 1 to the value of pointer */
        (userData->score) += grid->indexes[grid->row][grid->column].value;  /* score member of userData accumulates score corresponding the value at a specific row/gr
        grid->indexes[grid->row][grid->column].visited=1; /*sets the specific element to visited, which means later on it can be printed in pipes*/
        return 1; /*gets stored in the main in continueGame*/
    }

    return handleInput2(grid, userData, userInput); /* Calls handleInput2, which will eventually return back 2.. */
}
```

File Input/Output mechanism:

Using File pointer I found a way for my program to make a text file on a user’s PC called “custom.txt”. Using the append file property with mix with character array, I stored the user name in the text file. In the end, it also stores the score of the user in the same line as the name of the user. To read the file, I had to make another file pointer and using a while loop, I made sure it prints out the content, line by line.

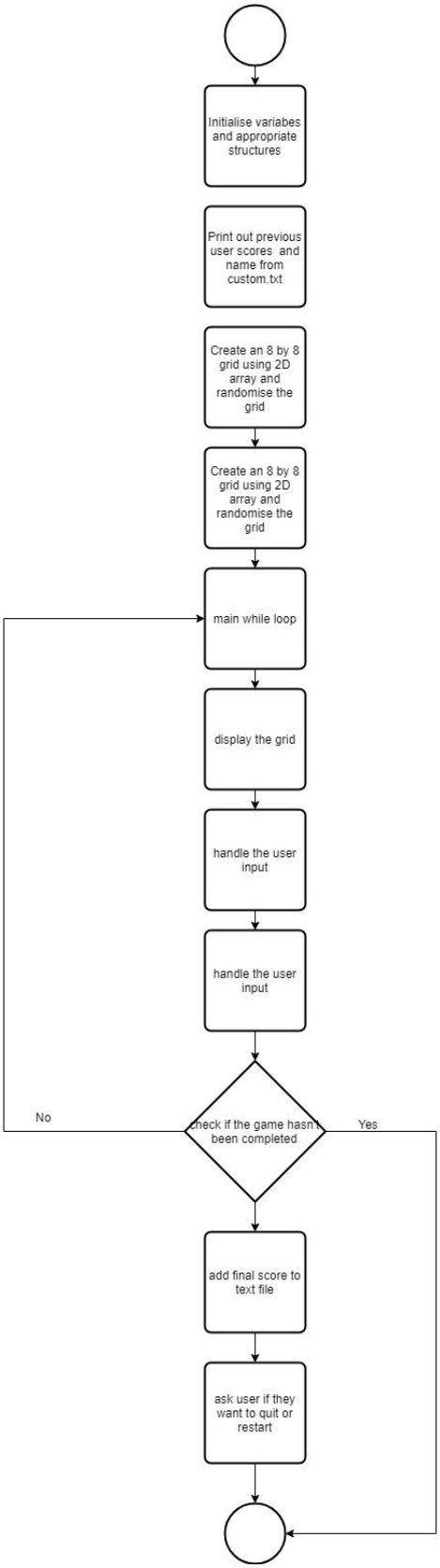
Additional Feature - Resetting the game at any given point of the program:

It's a really simple yet handy feature. I initialised/stored/set the value of the grid member row and column to 0. We then call our intoPara function and randomiseGrid function. The grid is randomised once again because we will be starting off fresh. This is simply done mostly by calling our old methods/functions that we had previously declared.

```
int handleInput2(gameGrid *grid, user *userData, char userInput)    /* Function for the user to restart the game */
{
    if(userInput=='q' || userInput=='Q')    /* if user presses q or Q to quit */
    {
            printf("You chose to quit the program, \nThank you %s, for playing, good bye.",userData->userName);
            return 0;
    }
    else if(userInput=='h' || userInput=='H') /*Once the user decides to replay the game*/
    {
        grid->row=0; /*initialise to 0*/
        grid->column=0;
        introPara(userData->userName); /*call methods*/
        randomiseGrid(grid, &userData->score);
        return 1;
    }

    return 2;   /* simply returns 2 to main function, if user does not press a valid choice */
}
```

Flow Chart(main method. XML version available on request, drawn from draw.io):


Initialise variabes and appropriate structures
Print out previous user scores and name from custom.txt
Create an 8 by 8 grid using 2D array and randomise the grid
Create an 8 by 8 grid using 2D array and randomise the grid
main while loop
display the grid
handle the user input
handle the user input
check if the game hasn't been completed
No
Yes
add final score to text file
ask user if they want to quit or restart