User Manual

The program is easy to use, and the user can easily understand how the program works and how to use the program. Below are the user manual and rules that are as well printed in the program when starts.

```
---Instruction and Rules of the Game---
1. Your goal is to visit all the 13 cities, namely
   A, B, C, D, E, F, G, H, I, J, K, L, M, N
   with as less moves as possible.

2. Use key W, A, S, D to move around the board

3. Your starting point is randomised so make sure
   you know where your starting point is by looking
   for the symbol '@', which is your head, or cursor.

4. For every path you have visited you will leave a '='.

5. Use Q to quit the program.
```

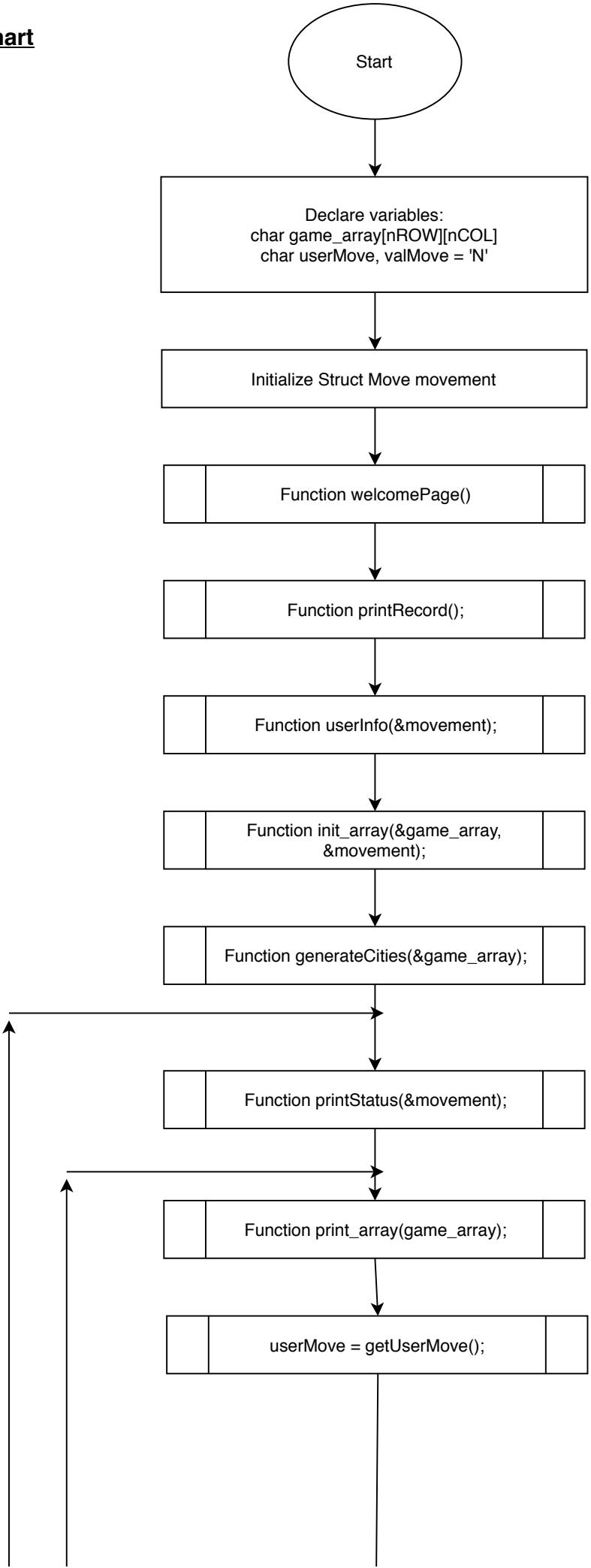
Description of Program Design

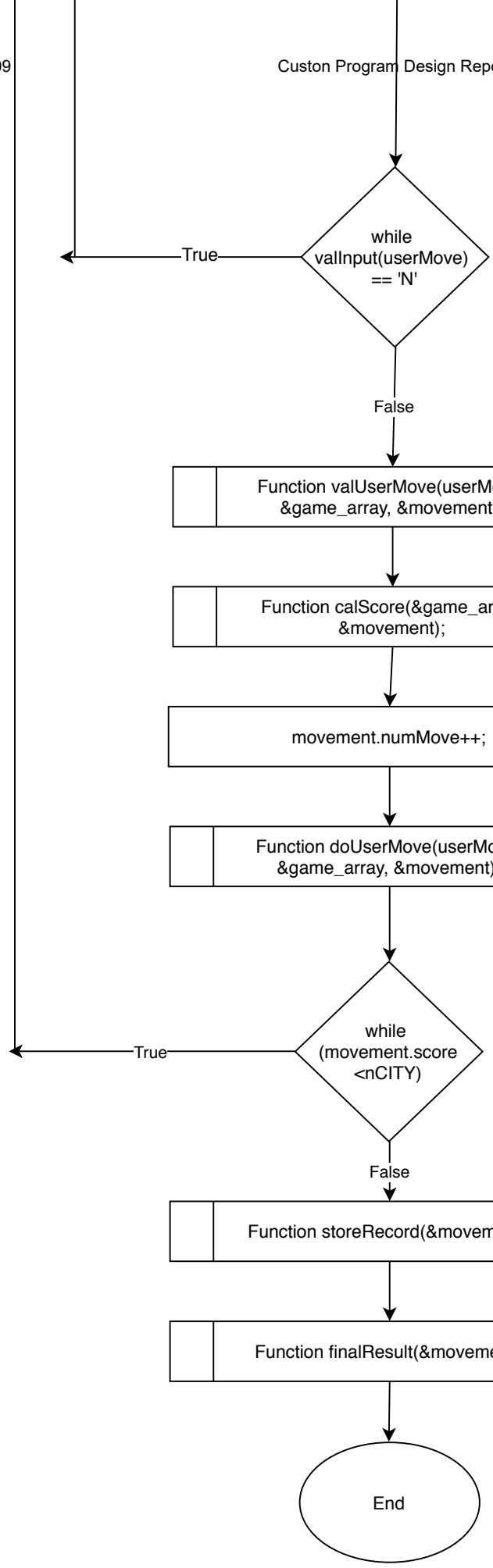
To describe the program design, I will start step by step of every code in the main() function.

1. The size of the board (number of row and number of the column) and the number of cities can be customised by going into the source code and change the value of the constant of nROW, nCOL and nCITY respectively. This provides great flexibility to the game.
2. Note that the head of the user is '@' so the user will know what position they are at real time. The tail '=' indicates where the user has been to on the array.
3. Function welcomePage. This function will print out the content of welcomeBanner.txt to the screen of the program. Inside the text file is a brief description of what does this program do and all the instruction and rules of the program for the user.
4. Function printRecord. This function will print out the content of result.txt to the screen of the program after the welcome banner. The data inside the text file are previous users' record of the program (name, number of moves and number of cities). The data is extracted and print it out after proper formatting, so it looks good.
5. Function userInfo. This function prompts the user to enter their name before they start the program. The name is used to record their playing history into result.txt after the game ends.
6. Function init_array. This function initializes the game array with "*" and randomly assign the starting point "@" of the game.
7. Function generateCities. This function will generate cities and assign them into the game array randomly. In this case, the function will assign 13 cities to the array.
8. Function printStatus. This function prints out the current status of the gameplay, namely the number of moves the user used and the number of cities the user had visited so far.
9. Function print_array. This function prints out the game_array every time after the user made a move so he or she can be updated of where they moved.
10. Function getUserMove(). This function get user move input. W to go up, S to go down, A to go left, D to go right and Q to quit the program. The move will get return to main to be validated.

11. Function valInput(userMove). This function validates whether the user entered the right input just now. If the input is wrong (not W, A, S, D), Function 9 and 10 will run again until the user entered the right input. This is done by the magic of do…while… loop.
12. Function valUserMove(). This function first replaces the current position of the user '@' to '=' so that next move can be done. Then the position of the new '@' will be calculated based on the user input move (W, A, S, D). This function also has a violation control to prevent the program from breaking when user tries to go over the border of the array. E.g. The program will break if the user tries to go upward when the index of row is already 0, that index will become -1 and there is no -1 index in every array and thus this will break the program, so the function will detect this violation and will have a safety measure to prevent the program from breaking by assigning the user to the same location, in this case, index of row is still 0. Then the number of move count will also be deducted by 1 as technically the user did not move at all. In another case where the user tries to go left (cCOL < 0), the user to try to go to the bottom (cROW > nROW) and the user try to go most right (cCOL > nCOL), this violation will happen, and this function prevents these from happening.
13. Function calScore(). This function calculates the core (number of cities visited) if the next move is not '=', '@' and '*', means they will be cities, then the score will add in by 1.
14. movement.numMove++. This line of codes run each time the loop of user input went through, means they successfully made a move thus the move count adds in by 1.
15. Function doUserMove(). This function replaces the coordinate of the user moved with '@' to indicates the user has made that move.
16. movement.score < nCITY. Function 8 to 15 will run again until the user has visited all the cities. This is done by the magic of do…while… loop.
17. Function storeRecord(). After all the cities have been visited, the game record or history will be automatically stored into the result.txt file.
18. Function finalResult(). This function prints out the final number of moves, number of cities visited and player's name as the end screen of the program.
19. Program ends.

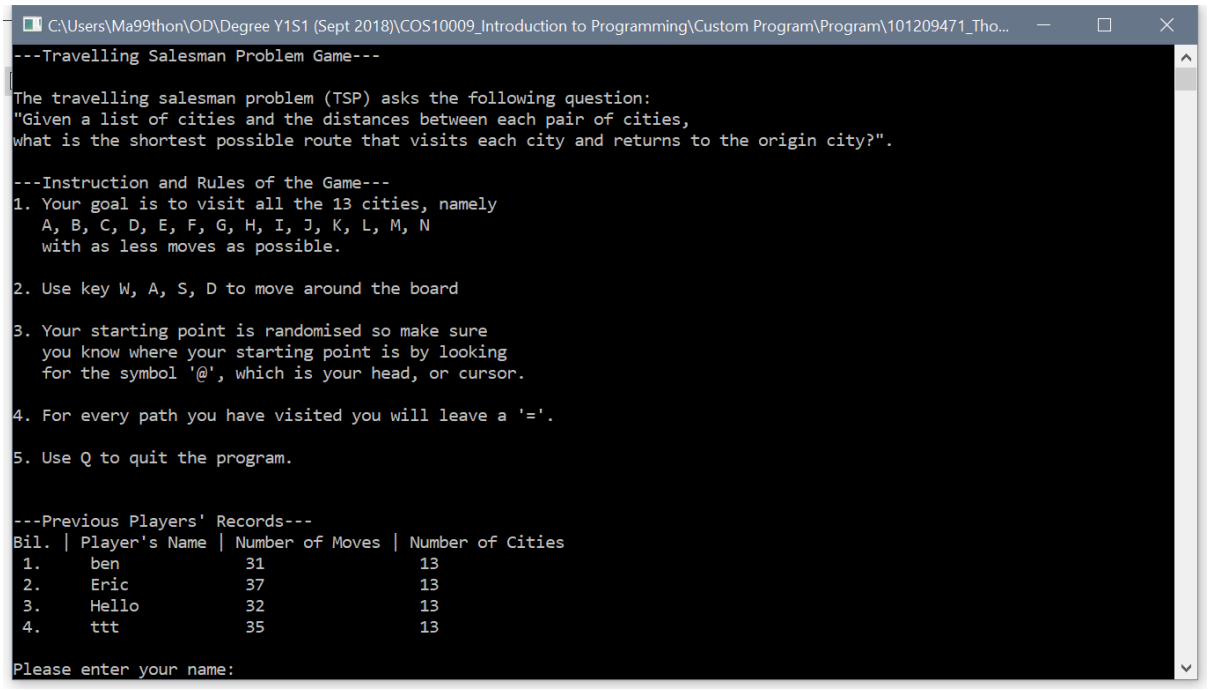
Flowchart


Start
Declare variables:
char game_array[nROW][nCOL]
char userMove, valMove = 'N'
Initialize Struct Move movement
Function welcomePage()
Function printRecord();
Function userInfo(&movement);
Function init_array(&game_array,
&movement);
Function generateCities(&game_array);
Function printStatus(&movement);
Function print_array(game_array);
userMove = getUserMove();

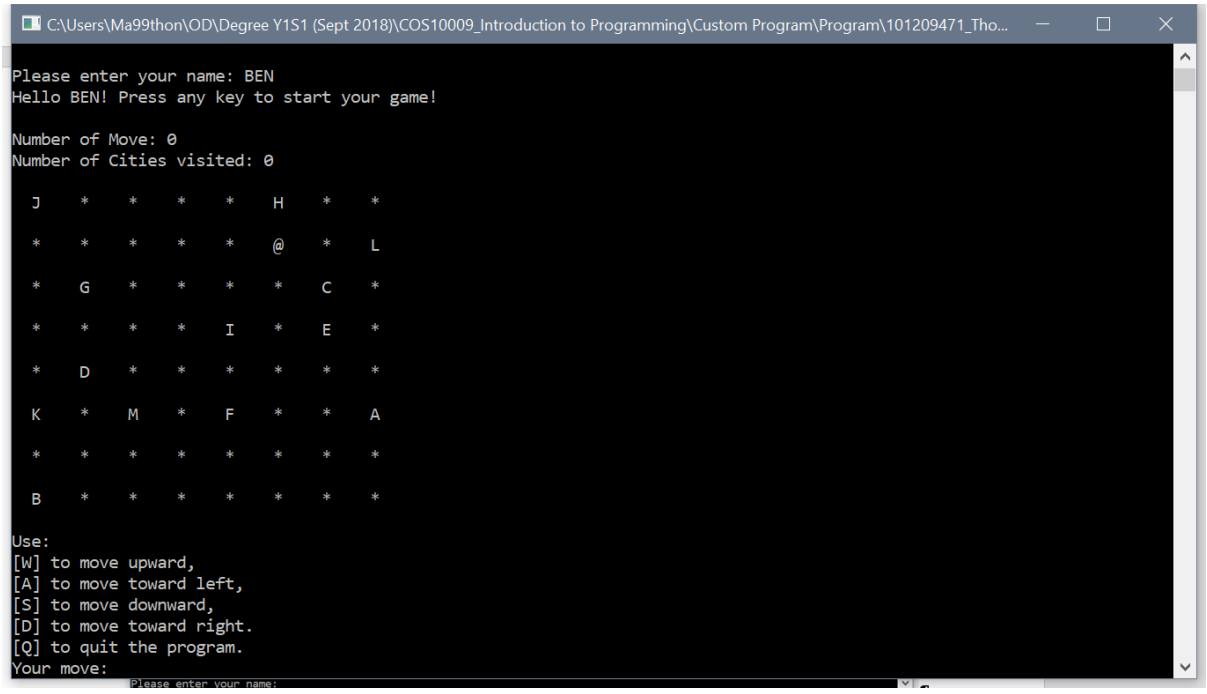

while valInput(userMove) == 'N'
True
False
Function valUserMove(userMove, &game_array, &movement);
Function calScore(&game_array, &movement);
movement.numMove++;
Function doUserMove(userMove, &game_array, &movement);
while (movement.score <nCITY)
True
False
Function storeRecord(&movement);
Function finalResult(&movement);
End

Screenshots

Screen when you first launch the program:

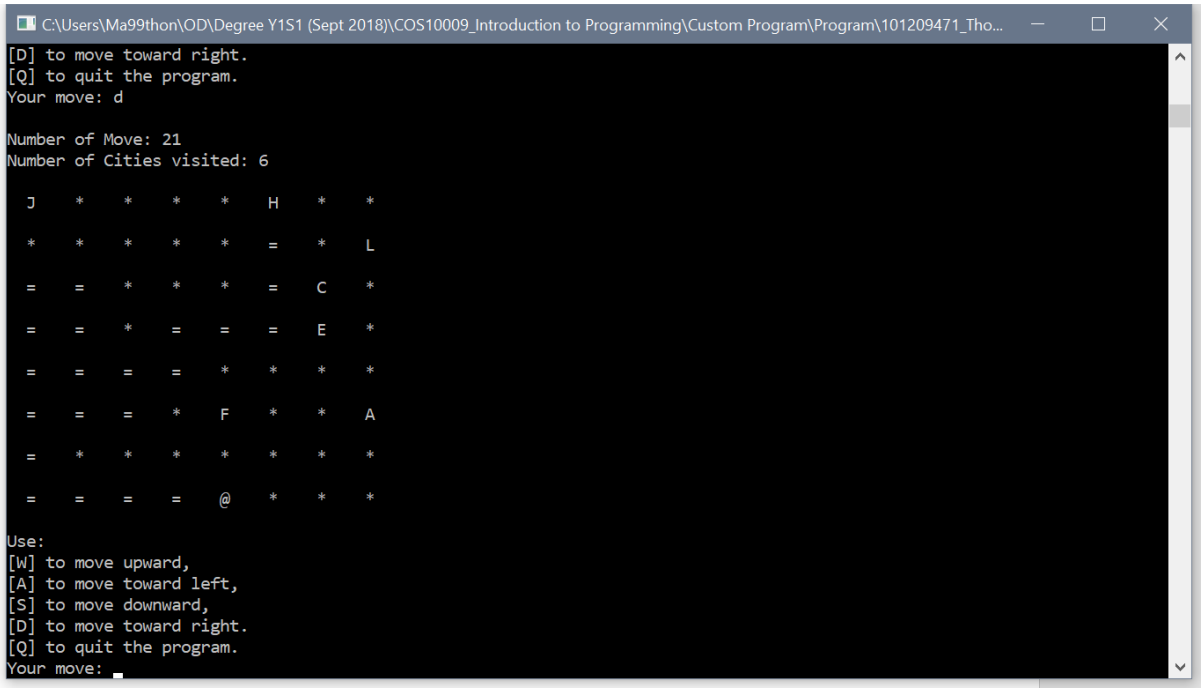


Screen when you first start your program after entering your name:

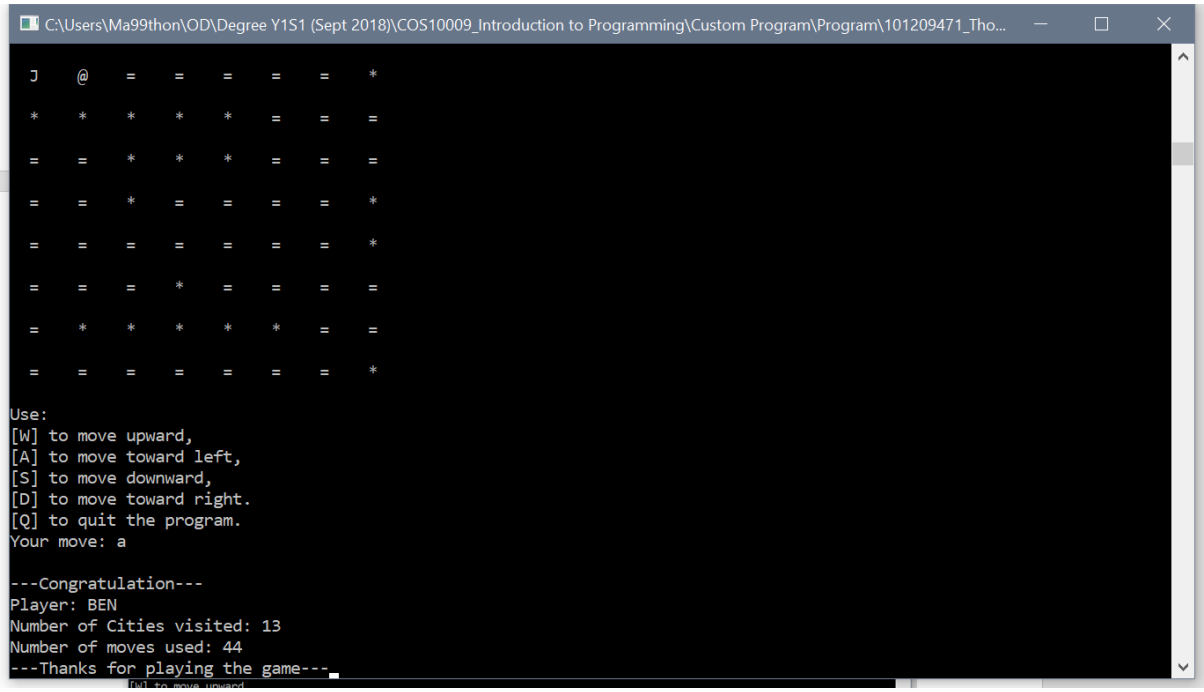


Screen of middle game play:

Note that '@' as head (where you currently is) and '=' as tail (where you have been)



Screen when completed the game:



Program Source Code

```
/*
      101209471_Thon.c
      Thon Pun Liang 101209471
      28/11/2018
*/

/* Include necessary libraries */
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

/*
   Define some constant
   - Constant Array Size: Row and Col
       - User can customize the size of the game board
   - Number of Cities
       - User can customize the number of cities to be visited
       - Recommended limit is 26 as only 26 letters
*/
#define nROW 8  // Number of Row of the array
#define nCOL 8  // Number of Col of the array
#define nCITY 13  // Number of cities of the game

typedef struct {
    int cROW;  // Current ROW coordinate
    int cCOL;  // Current COL coordinate
    int score;  // Cities visited
    int numMove;  // Moves used
    char name[30]; // User's name
} Move;

/* To print the array after evey move*/
void print_array(char array[nROW][nCOL]) {
    int x,y;
    for (x=0; x<nROW; x++) {
        for (y=0; y<nCOL; y++) {
            printf("  %c  ", array[x][y]);
        }
        printf("\n\n");
    }
}

/* To initialize the array */
void init_array(char (*array)[nROW][nCOL], Move *coor) {
    int x,y;
    for (x=0; x<nROW; x++) {
        for (y=0; y<nCOL; y++) {
             (*array)[x][y] = '*';  // Initialize with symbol '*'
        }
    }

    /* Codes below randomize starting point */
    srand((unsigned) time(NULL));
    (*coor).cROW = rand() % 8;
    (*coor).cCOL = rand() % 8;

    (*array)[(*coor).cROW][(*coor).cCOL] = '@';  // This is the head '@'
}
```

```
/* To generate cities and assign randomly into array */
void generateCities(char (*array)[nROW][nCOL]) {
    int x,row,col;

    srand((unsigned) time(NULL));

    /* Codes below assign cities randomly in the array */
    for (x=0; x<nCITY; x++) {
        row = rand() % nROW;
        col = rand() % nCOL;

        /* If the coordinates is not '*', go back again */
        /* Only replace '*' with cities (letter) */
        if ((*array)[row][col] != '*') {
            x -= 1;
        }
        else {
             (*array)[row][col] = 'A'+x;  // A,B,C,...
        }
    }
}

/* Get user movement from input */
char getUserMove() {
    char move;

    printf("Use:\n");
    printf("[W] to move upward,\n");
    printf("[A] to move toward left,\n");
    printf("[S] to move downward,\n");
    printf("[D] to move toward right.\n");
    printf("[Q] to quit the program.\n");
    printf("Your move: ");
    fflush(stdin);
    scanf("%c", &move);

    return move;
}

/* Validate user movement */
char valInput(char move) {

    if (move == 'w' || move =='W' ||
    move == 'a' || move == 'A' ||
    move == 's' || move == 'S' ||
    move == 'd' || move == 'D' ||
    move == 'q' || move == 'Q') {
        return 'Y';
    }
    else {
        return 'N';
    }
}
```

```
/* Validate Movement */
void valUserMove(char move, char (*array)[nROW][nCOL], Move *movement) {
    char exitx;

    /* After each move is done, replace the initial coor with tail '=' */
     (*array)[(*movement).cROW][(*movement).cCOL] = '=';

    /* Codes below determine the direction of the head '@' */
    if (move == 'w' || move == 'W') {
         (*movement).cROW = (*movement).cROW - 1;
    }
    else if (move == 'a' || move == 'A') {
        (*movement).cCOL = (*movement).cCOL - 1;
    }
    else if (move == 's' || move == 'S') {
         (*movement).cROW = (*movement).cROW + 1;
    }
    else if (move == 'd' || move == 'D') {
         (*movement).cCOL = (*movement).cCOL + 1;
    }

    /* If user decide to quit during game play */
    else if (move == 'q' || move == 'Q') {
        printf("Thanks for playing! See you again!");
        scanf("%c", &exitx);
        exit(0);  // Exit the program
    }

    /* Codes below prevent user to break the program
       by insisting to go over the barrier of the array
       ie if the max row is 8 the user try to continue move
       downward.
       The program will get malfunction if below's code is
       absent. */
    if ((*movement).cROW < 0) {
         (*movement).cROW = 0;
         (*movement).numMove--;   // To prevent the moves count even if go
over boarder
    }
    else if ((*movement).cROW > nROW-1) {
        (*movement).cROW = nROW-1;
        (*movement).numMove--;   // To prevent the moves count even if go
over boarder
    }

    if ((*movement).cCOL < 0) {
        (*movement).cCOL = 0;
        (*movement).numMove--;   // To prevent the moves count even if go
over boarder
    }
    else if ((*movement).cCOL > nCOL-1) {
         (*movement).cCOL = nCOL-1;
         (*movement).numMove--;  // To prevent the moves count even if go
over boarder
    }
}
```

```
/* Do movement by putting head '@' in the array*/
void doUserMove(char move, char (*array)[nROW][nCOL], Move *movement) {
     (*array)[(*movement).cROW][(*movement).cCOL] = '@';
}

/* Count score and movement */
// If not '*' '=' '@' score ++
void calScore(char (*array)[nROW][nCOL], Move *move) {
    if ((*array)[(*move).cROW][(*move).cCOL] != '*' &&
     (*array)[(*move).cROW][(*move).cCOL] != '=' &&
     (*array)[(*move).cROW][(*move).cCOL] != '@') {
         (*move).score += 1;
    }
}

/* Display final result */
void finalResult(Move *final) {
    char exit;
    printf("\n---Congratulation---\n");
    printf("Player: %s\n", (*final).name);
    printf("Number of Cities visited: %i\n", (*final).score);
    printf("Number of moves used: %i\n", (*final).numMove);
    printf("---Thanks for playing the game---");
    fflush(stdin);
    scanf("%c", &exit);
}

/* Display welcome page from textfile */
void welcomePage() {
    char c;
    FILE *welcome;

    welcome = fopen("welcomeBanner.txt", "r");

    if (welcome == NULL) {
        printf("Cannot open file \n");
        exit(0);  // Exit program if file is null
    }

    // Read contents from file line by line
    c = fgetc(welcome);
    while (c != EOF) {
        printf ("%c", c);
        c = fgetc(welcome);
    }

    fclose(welcome);
}

/* Get user's name and start the program */
void userInfo(Move *info) {
    char xxx;
    printf("\nPlease enter your name: ");
    scanf("%s", (*info).name);
    fflush(stdin);
    printf("Hello %s! Press any key to start your game!", (*info).name);
    scanf("%c", &xxx);
    fflush(stdin);
}
```

```
/* Store the game history to textfile */
void storeRecord(Move *rec) {
    FILE *result;

    result = fopen("result.txt", "a");

    if (result == NULL) {
        printf("Cannot open file \n");
        exit(0);    // Exit program if file is null
    }

    fprintf(result, "%s %i %i\n", (*rec).name, (*rec).numMove, (*rec).score);

    fclose(result);
}

/* Print previous users' histories */
void printRecord() {
    char namex[30];
    int x, y, bil=0, num=0;

    FILE *reco;
    reco = fopen("result.txt", "r");

    if (reco == NULL) {
        printf("Cannot open file \n");
        exit(0);    // Exit program if file is null
    }

    printf("\n\n---Previous Players' Records---\n");
    printf("Bil. | Player's Name | Number of Moves | Number of Cities\n");

    do {
        bil++;
        num = fscanf(reco, "%s %i %i", namex, &x, &y);
        if(num==3) {
            printf(" %i.\t%s\t\t%i\t\t  %i\n", bil, namex, x, y);
        }
    } while(num==3);

    fclose(reco);
}

/* Print real time game status */
void printStatus(Move *status) {
    printf("\nNumber of Move: %i\n", (*status).numMove);
    printf("Number of Cities visited: %i\n\n", (*status).score);
}

/* Main */
int main(void) {
    char game_array[nROW][nCOL];
    char userMove,valMove = 'N';

    /* Initialize value in struct Move */
    Move movement = {0, 0, 0, 0};

    welcomePage();  // Welcome page
    printRecord();  // Print previous records
    userInfo(&movement);  // Get user name and start game
    init_array(&game_array, &movement);  // Initialize array
```

```
    generateCities(&game_array);  // Generate cities into array

    /* Loop below codes until all cities has been visited */
    do {
        // Print status
        printStatus(&movement);

        /* Loop below codes if user input is wrong */
        do {
            print_array(game_array);  // Print gane array
            userMove = getUserMove();  // Get user's next move
        } while (valInput(userMove) == 'N');  // Validate user's move

        // Validate user move
        valUserMove(userMove, &game_array, &movement);
        // Cal Score if cities visited
        calScore(&game_array, &movement);
        // +1 to num of move
        movement.numMove++;
        // Actually move the head '@'
        doUserMove(userMove, &game_array, &movement);
    } while(movement.score<nCITY);

    storeRecord(&movement);  // Store record
    finalResult(&movement);  // Print final result

    return 0;
}
```