ARRAYS
Lecture 6-7 Assignments

1. a) Revise line 16 such that you use a designated initializer to set pathways 0 and 2 to true, and the rest will be false. Make the initializer as short as possible.

```
bool pathway[8] = {[0]true, [2]true};
```

b) Revise line 16 such that the initializer will be short as possible (without using a designated initializer)

```
bool pathway[8] = {true, false, true};
```

2. **As a programming assignment (see last page for full code):**
   - **Declare and initialize a `road_networks` multidimensional array that represents the adjacency matrix**

```
/* MAIN FUNCTION */
int main(void){

    /* INITIALIZATION */
    int location;
    int road_networks[NUM][NUM] = {     // used jagged arrays since there are trailing zeroes
        {1, 1, 0, 0, 0, 1},             // A
        {1, 1, 1},                      // B
        {0, 1, 1, 0, 1, 1},             // [C] charging station
        {0, 0, 0, 1, 1},                // [D] charging station
        {0, 0, 0, 1, 1},                // E
        {1, 0, 1, 0, 0, 1},             // F
        {1, 0, 0, 1, 0, 0, 1},          // G
        {0, 0, 0, 0, 0, 1, 0, 1},       // H
    };

    char points[NUM] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'};     // array for points/destinations
```

For initializing a multidimensional array, I decided to use jagged arrays since the rest of the elements for each row are just zeroes. I also set the size of its rows and columns to the macro NUM which has a value of 8. So, we have an 8 x 8 adjacency matrix that represents road_networks. We also have another array for our points or labels of the rows and columns of the said matrix.

   - **Display the adjacency matrix. Put a bracket to the points/destinations that are considered as charging stations, e.g. [c], [d]**

```
void print_matrix(char arr1[NUM], int arr2[NUM][NUM]) {       // FUNCTION for printing the adjacency matrix

    printf("    ");
    for (int a = 0; a < NUM; a++){                            // loop for printing the column labels A-H
        switch (a) {
            case 2: case 3:                                   // special cases for formatting [C] and [D]
                printf("      [%c]", arr1[a]); break;
            default:                                          // rest of the labels
                printf("%8c ", arr1[a]); break;
        }
    }
    printf("\n");

    for (int i = 0; i < NUM; i ++) {                          // nested for loop for printing the matrix
        if (i == 2 || i == 3){                                // if-else statement for printing row labels A-H
            printf("[%c]", arr1[i]);
        } else {
            printf("%-3c", arr1[i]);
        }
        for (int j = 0; j < NUM; j++) {                       // iterates through each element of the array and print it
            printf("%9d", arr2[i][j]);
        }
        printf("\n");
    }
}
```

```
/* MATRIX OUTPUT */
print_matrix(points, road_networks);
```

→ function call

I defined a function for displaying the matrix. So, we have 2 arrays as parameters, which are the road networks and the points (A-H). For printing the labels/points, I used switch case to format it properly using

format specifiers and for loop. In the case of the `road_networks` or the binary values, I used nested for loop to print the actual matrix, while inserting the row labels A-H. The inner loop is responsible for iterating through each element of the array and printing the values.

- **Given a point / destination, determine the nearest charging station. For example, if you are in point a, the nearest charging station is point c. If you are in point e, the nearest charging station is point d.**

```c
int inp_validation() {           // FUNCTION for input validation
    int location;

    while(1) {                   // while true loop that will break once input is valid --- continues to run if input is false/invalid
        printf("\nWhich point are you located?  0 - A, 1 - B, 2 - C, 3 - D, 4 - E, 5 - F, 6 - G, 7 - H");
        printf("\nENTER integer (0-7) based on the corresponding points (A-H): ");
        scanf("%d", &location);

        if (location < 0 || location > 7) {              // checks if input is within the range 0-7
            printf("!!! ERROR !!! Invalid input! Must be an integer between 0-7. Please try again.\n");

        } else {                 // input is valid
            return location;     // ends the function and returns the value of input to the calling function
        }
    }
}
```

```c
/* USER INPUT */
location = inp_validation();
printf("At Point: %c", points[location]);
```

→ function call

First, I made a function for input validation since we need to have a user input to determine the current location or point the user is at. For this function, if the user's input is not within 0-7 then it is invalid, and we will continue to ask for input until the user enters the correct value.

```c
void navigate(int location, int arr1[NUM][NUM], char arr2[NUM]) {     // FUNCTION for navigating the pathway towards charging station

    if (location == 2 || location == 3) {                             // if input is 2 or 3, then we are already at a charging station
        printf("\n%c is a charging station", arr2[location]);
    } else if (arr1[location][2]) {
        printf("\nNow at point C");
        printf("\nArrived at charging station C!");
    } else if (arr1[location][3]) {
        printf("\nNow at point D");
        printf("\nArrived at charging station D!");
    } else {                                                          // for loop will run if we need to go at another point before reaching a charging station
        for (int new_location = 0; new_location < NUM; new_location++) {
            if ((location != new_location) && (arr1[location][new_location])) {
                printf("\nNow at point %c", arr2[new_location]);
                navigate(new_location, arr1, arr2);                   // recursive call to navigate with the new point and check if it is now near the charging station
                break;
            }
        }
    }
}
```

```c
/* NEAREST CHARGING STATION */
navigate(location, road_networks, points);
```

→ function call

Then I also made a function for navigating the pathway towards the nearest charging station, so if the input was 2 or 3 then we display a message that the user is already at a charging station. The other else ifs will be useful when we are at a different point rather than C or D as an origin. So, if the user inputs A, and it cannot travel directly towards the C charging station, then it has to go first to a closest point with a pathway; hence, the point B in our matrix. So, this is actually the purpose of our else statement with a for loop because it will allow us to navigate first with a different close point before reaching the charging station.

I also have a recursive call within the else statement because we need to navigate the charging station again using our new location and check if we can already reach it. If not, then we will loop again.

- **Bonus: Use a macro to define the size of the 2d array**

```c
#define NUM 8                    // macro for the size of the array
```

This is the macro for the size of the array, and it was also used in other parts of the code which requires the value 8.

GitHub Link: https://github.com/tinapayy/CMSC21.git

FULL CODE FOR NUMBER 2

```c
/***  CELIS, KRISTINA | ASSIGNMENT 2 | as2  ***/

#include<stdio.h>
#define NUM 8                       // macro for the size of the array

/* FUNCTIONS */
int inp_validation() {              // FUNCTION for input validation
    int location;

    while(1) {                     // while true loop that will break once input is valid --- continues to run if input is false/invalid
        printf("\nWhich point are you located?  0 - A, 1 - B, 2 - C, 3 - D, 4 - E, 5 - F, 6 - G, 7 - H");
        printf("\nENTER integer (0-7) based on the corresponding points (A-H): ");
        scanf("%d", &location);

        if (location < 0 || location > 7) {              // checks if input is within the range 0-7
            printf("!!! ERROR !!! Invalid input! Must be an integer between 0-7. Please try again.\n");

        } else {                    // input is valid
            return location;        // ends the function and returns the value of input to the calling function
        }
    }
}

void print_matrix(char arr1[NUM], int arr2[NUM][NUM]) {       // FUNCTION for printing the adjacency matrix

    printf("    ");
    for (int a = 0; a < NUM; a++){                            // loop for printing the column labels A-H
        switch (a) {
            case 2: case 3:                                   // special cases for formatting [C] and [D]
                printf("      [%c]", arr1[a]); break;
            default:                                          // rest of the labels
                printf("%8c ", arr1[a]); break;
        }
    }
    printf("\n");

    for (int i = 0; i < NUM; i ++) {                          // nested for loop for printing the matrix
        if (i == 2 || i == 3){                                // if-else statement for printing row labels A-H
            printf("[%c]", arr1[i]);
        } else {
            printf("%-3c", arr1[i]);
        }
        for (int j = 0; j < NUM; j++) {                   // iterates through each element of the array and print it
            printf("%9d", arr2[i][j]);
        }
        printf("\n");
    }
}

void navigate(int location, int arr1[NUM][NUM], char arr2[NUM]) {     // FUNCTION for navigating the pathway towards charging station

    if (location == 2 || location == 3) {                         // if input is 2 or 3, then we are already at a charging station
        printf("\n%c is a charging station", arr2[location]);
    } else if (arr1[location][2]) {
        printf("\nNow at point C");
        printf("\nArrived at charging station C!");
    } else if (arr1[location][3]) {
        printf("\nNow at point D");
        printf("\nArrived at charging station D!");
    } else {                                                      // for loop will run if we need to go at another point before reaching a charging station
        for (int new_location = 0; new_location < NUM; new_location++) {
            if ((location != new_location) && (arr1[location][new_location])) {
                printf("\nNow at point %c", arr2[new_location]);
                navigate(new_location, arr1, arr2);               // recursive call to navigate with the new point and check if it is now near the charging station
                break;
            }
        }
    }
}
```

GitHub Link: https://github.com/tinapayy/CMSC21.git

```c

/* MAIN FUNCTION */
int main(void){

    /* INITIALIZATION */
    int location;
    int road_networks[NUM][NUM] = {        // used jagged arrays since there are trailing zeroes
        {1, 1, 0, 0, 0, 1},                // A
        {1, 1, 1},                         // B
        {0, 1, 1, 0, 1, 1},                // [C] charging station
        {0, 0, 0, 1, 1},                   // [D] charging station
        {0, 0, 0, 1, 1},                   // E
        {1, 0, 1, 0, 0, 1},                // F
        {1, 0, 0, 1, 0, 0, 1},             // G
        {0, 0, 0, 0, 0, 1, 0, 1},          // H
    };

    char points[NUM] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'};     // array for points/destinations

    /* MATRIX OUTPUT */
    print_matrix(points, road_networks);

    /* USER INPUT */
    location = inp_validation();
    printf("At Point: %c", points[location]);

    /* NEAREST CHARGING STATION */
    navigate(location, road_networks, points);

    return 0;
}
```

GitHub Link: https://github.com/tinapayy/CMSC21.git