CHANAKYA
UNIVERSITY

SUBMITTED BY:

SHREYAS G

24U400459

PROBLEM STATEMENT:

You are required to model a 2D grid of characters as a graph, where each cell acts as a node, and edges exist between horizontally and vertically adjacent cells (no diagonal connections). Given a target word, your program should search for occurrences of the word in the grid horizontally (left to right) or vertically (top to bottom).

For each occurrence of the word found, print the start node and end node coordinates. If there are no occurrences, print Word not found.

Input Format

An integer m number of rows in the grid.

An integer n number of columns in the grid.

A grid of m rows and n columns containing uppercase letters (A-Z).

A target word (string) to be searched in the grid.

Output Format

Start: (row_start, col_start) End: (row_end, col_end)

If no occurrence is found, print "Word not found

Additional Requirement

Design your own grid in such a way that it contains your name (in uppercase) as the target word, arranged horizontally and vertically.

Use that grid and your name as input to test your program.

Make sure at least 2 valid occurrences exist. If the length of your name string is too long, you can take first 5 letters of your name and demonstrate the output.

Instructions for submission:

1. Push the code file to your GitHub repo which you have shared for the evaluation.

2. Create a pdf file that contains your name, USN and screenshots of all the outputs.

CODE :

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAX_ROWS 50
#define MAX_COLS 50
#define MAX_WORD_LEN 50
#define MAX_NAME_LEN 6
typedef struct {
    int start_row;
    int start_col;
    int end_row;
    int end_col;
} Occurrence;
int dr[] = {-1, 1, 0, 0, -1, -1, 1, 1};
int dc[] = {0, 0, -1, 1, -1, 1, -1, 1};
int check_direction(char grid[MAX_ROWS][MAX_COLS], int M, int N, int row, int col,
const char* word,
                    int dr_dir, int dc_dir, Occurrence* occurrence) {
    int len = strlen(word);
    int r = row;
    int c = col;
 for (int k = 0; k < len; k++) {
        if (r < 0 || r >= M || c < 0 || c >= N) {
            return 0;
        }
        if (grid[r][c] != word[k]) {
            return 0;
        }
```

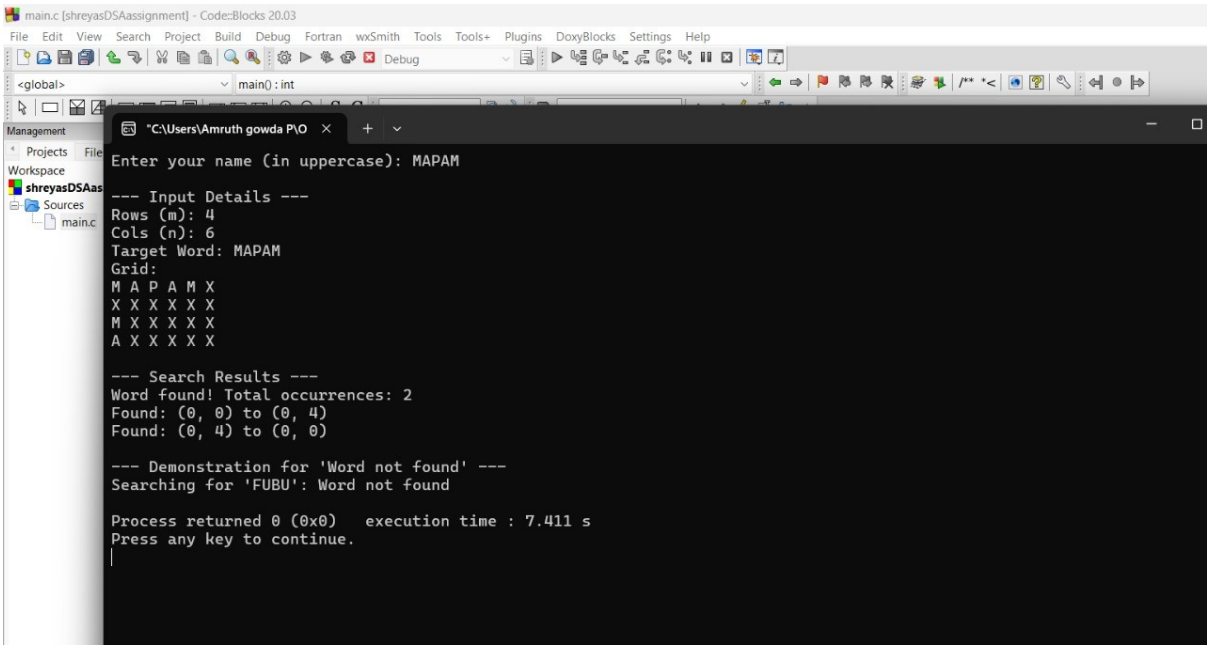
```
r += dr_dir;
        c += dc_dir;
    }
  occurrence->start_row = row;
    occurrence->start_col = col;
    occurrence->end_row = r - dr_dir;
    occurrence->end_col = c - dc_dir;
  return 1;
}
int search_word(char grid[MAX_ROWS][MAX_COLS], int M, int N, const char* word,
                Occurrence results[MAX_ROWS * MAX_COLS * 8]) {
    int count = 0;
    int word_len = strlen(word);
 if (word_len == 0) return 0;
 for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            if (grid[i][j] == word[0]) {
                for (int k = 0; k < 8; k++) {
                    Occurrence current_occurrence;
                    if (check_direction(grid, M, N, i, j, word, dr[k], dc[k], &current_occurrence)) {
                        results[count++] = current_occurrence;
                    }
                }
            }
        }
    }
    return count;
}
void print_occurrence(const Occurrence* occ) {
    printf("Found: (%d, %d) to (%d, %d)\n",
           occ->start_row, occ->start_col, occ->end_row, occ->end_col);
```

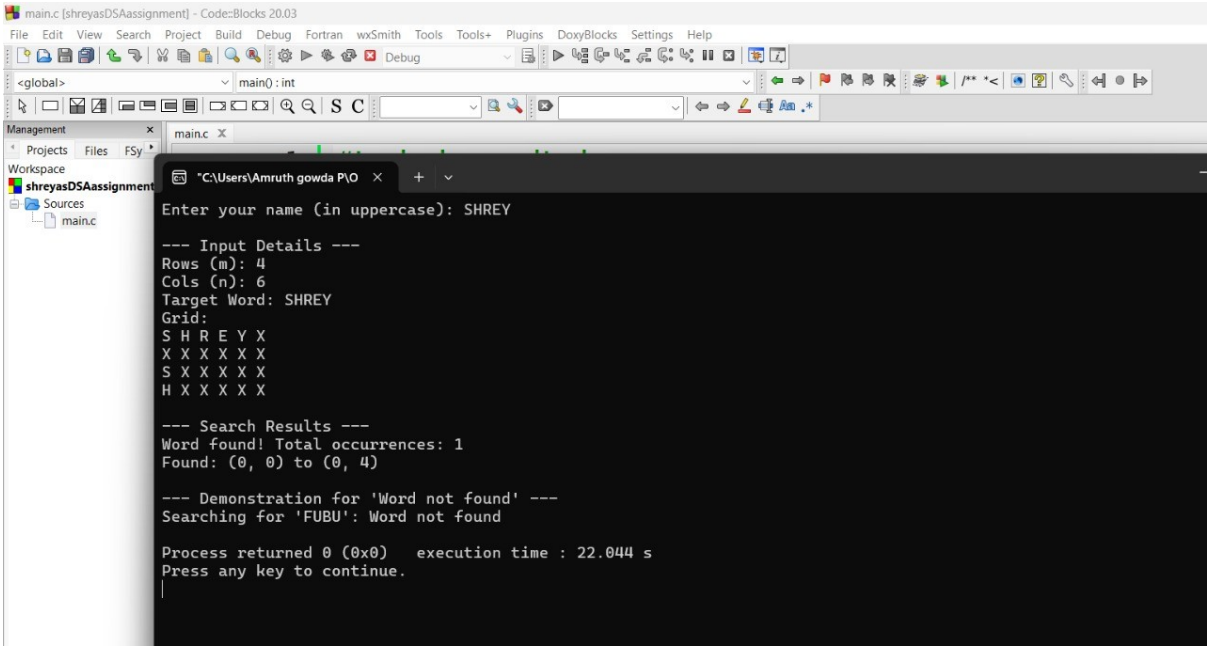
```
}
void design_grid(int M, int N, char grid[MAX_ROWS][MAX_COLS], const char*
name_part, char *target_word) {
    int i, j;
    int len = strlen(name_part);
  strcpy(target_word, name_part);
 for (i = 0; i < M; i++) {
        for (j = 0; j < N; j++) {
            grid[i][j] = 'X';
        }
    }
 for (j = 0; j < len; j++) {
        if (j < N) grid[0][j] = target_word[j];
    }
 if (1 + len - 1 < M && 1 + len - 1 < N) {
         for (i = 0; i < len; i++) {
            grid[1 + i][1 + i] = target_word[i];
        }
    } else {
        for (i = 0; i < len; i++) {
            if (2 + i < M) grid[2 + i][0] = target_word[i];
        }
    }
}

int main() {
    int M, N;
    char target_name[MAX_WORD_LEN];
    char target_word[MAX_WORD_LEN];
 printf("Enter your name (in uppercase): ");
    scanf("%s", target_name);
```

```
 int name_len = strlen(target_name);
   if (name_len > 5) {
      target_name[5] = '\0';
      name_len = 5;
   }
  M = 4;
   N = 6;
    char grid[MAX_ROWS][MAX_COLS];
  design_grid(M, N, grid, target_name, target_word);
  printf("\n--- Input Details ---\n");
   printf("Rows (m): %d\n", M);
   printf("Cols (n): %d\n", N);
   printf("Target Word: %s\n", target_word);
   printf("Grid:\n");
   for (int i = 0; i < M; i++) {
      for (int j = 0; j < N; j++) {
         printf("%c ", grid[i][j]);
      }
      printf("\n");
   }
Occurrence results[MAX_ROWS * MAX_COLS * 8];
   int occurrence_count = search_word(grid, M, N, target_word, results);
  printf("\n--- Search Results ---\n");
 if (occurrence_count > 0) {
      printf("Word found! Total occurrences: %d\n", occurrence_count);
      for (int i = 0; i < occurrence_count; i++) {
         print_occurrence(&results[i]);
      }
   } else {
      printf("Word not found\n");
   }
```

```
  printf("\n--- Demonstration for 'Word not found' ---\n");
   char not_found_word[] = "FUBU";
   int not_found_count = search_word(grid, M, N, not_found_word, results);
   if (not_found_count > 0) {
      printf("ERROR: Word '%s' was unexpectedly found.\n", not_found_word);
   } else {
      printf("Searching for '%s': Word not found\n", not_found_word);
   }
 return 0;
}
```

SREEN SHOT OF OUTPUT :




main.c [shreyasDSAassignment] - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Debug
<global>
main() : int
Management
Projects Files FSy
Workspace
shreyasDSAassignment
Sources
main.c
main.c
"C:\Users\Amruth gowda P\O
Enter your name (in uppercase): SHREY
--- Input Details ---
Rows (m): 4
Cols (n): 6
Target Word: SHREY
Grid:
S H R E Y X
X X X X X X
S X X X X X
H X X X X X
--- Search Results ---
Word found! Total occurrences: 1
Found: (0, 0) to (0, 4)
--- Demonstration for 'Word not found' ---
Searching for 'FUBU': Word not found
Process returned 0 (0x0) execution time : 22.044 s
Press any key to continue.

****