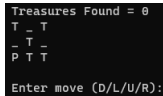


Question 01: Simple Game Development Using Arrays

Assignment Description:

Develop a simple console-based game in C, titled "Treasure Hunt," that uses arrays extensively. In this game, a 3x3 grid represents a map where treasures "T" are randomly placed. Each cell in the grid can either contain a treasure "T" or be empty "_". The player "P" starts at a random place of the grid and can move "U" up, "D" down, "L" left, or "R" right. The goal is to reach all treasures locations with a limited number of moves.



Requirements:

1. Use a two-dimensional array to represent the grid.
2. Implement functions to randomly place a fixed number of treasures on the grid.
3. Allow the player to input their move direction ('U' for up, 'D' for down, 'L' for left, 'R' for right). Keep capital letters.
4. Keep track of the player's position and the number of treasures found.
5. End the game when the player moves to all treasures locations or the player runs out of moves.

Instructions:

1. Your C code file should start with the following C code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define GRID_SIZE 3
#define TREASURE_COUNT 6
#define MIN_TREASURE_COUNT 5
#define MAX_MOVES 20

int treasuresFound = 0;
```

Use only the above H files

2. You must use the following command in the begging of your main() after the variable declaration:

```
srand(2024);
```

You should explain in the code what is this function do in the code and why do we use it.

3. Your main code must end with the following code lines:

```
if (treasuresFound >= MIN_TREASURE_COUNT)
    printf("\nCongratulations! You found all the treasures.\n");
else
    printf("\nSorry, you ran out of moves.\n");

return 0;
```

4. Write a function that define the grid and the initial setup of the game.
5. Implement functions for displaying the player movement and checking for treasures.

Question 02: Elevator Simulation Using Strings

Assignment Description:

Create a Linux terminal-based program that simulates an elevator moving between floors in a building. Use strings to represent the status of the elevator at each floor.

Instructions:

1. Your C code file should start with the following C code:

```
#include <stdio.h>
#include <string.h>
```

```
#define MAX_FLOORS 10;
```

Use only the above H files

2. Represent the building floors as a string array, with each element indicating a floor number.
3. Allow the user to input the number of floors the elevator should move up or down.
4. As a response to the user selection floor you should print "You requested to go to the XXXX" where XXXX the text name of the floor number. E.g. for 7 selection the XXXX will be replace with the string "Seventh floor" so the final printout message will be: "You requested to go to the Seventh floor"
5. As the elevator moves, print the current floor number (see the exampl02.out printouts).
6. Include error handling for invalid floor numbers or inputs.
7. Use a string array to store floor numbers.
8. Implement functions to move the elevator and display the current floor.

Question 03:: Smart Home System Simulation in C

Background:

As we delve into the world of Internet of Things (IoT) and smart home technology, understanding how to program these devices is crucial. In this assignment, you will be tasked with creating a simulation of a smart home system using C programming language. This simulation will run in a Linux terminal environment.

Objective:

Develop a C program that simulates the control and monitoring of a basic smart home system. Your program should represent a simple model of a smart home with basic functionalities like lighting control and temperature monitoring.

Given the following main() code:

```
#include <stdio.h>
#include <stdbool.h>
#include <unistd.h> // For sleep function

// Define a structure for the smart home state
typedef struct {
    bool lamp1;
    bool lamp2;
    bool lamp3;
    int dimmerALevel; // 0 to 4
    int dimmerBLevel; // 0 to 4
    int temperatureRoomLevel; // 0 to 9
} SmartHomeState;

// Function prototypes
void initializeSmartHome(SmartHomeState *state);
void updateLighting(SmartHomeState *state, bool lamp1, bool lamp2, bool lamp3, int dimmerA, int dimmerB);
void updateTemperature(SmartHomeState *state, int temperature);
void printSmartHomeState(const SmartHomeState *state);
void drawRoom(const SmartHomeState *state);
bool isValidInput(char d);
void getSimCode(int arr[]);

int main() {
    SmartHomeState state;
    int i, simcode[5];

    getSimCode(simcode); // It convert from char digits into integer digits

    initializeSmartHome(&state);

    // Simulate different situations
    for (i = 0; i < 5; i++) {
        // update new home state base on the given code (e.g 24553)
        updateLighting(&state, i % simcode[0] == 0, i % simcode[1] == 0, i % simcode[2] == 0, i % simcode[3],
        (4 - i) % simcode[4]);
        updateTemperature(&state, i);

        printSmartHomeState(&state);
        drawRoom(&state);

        sleep(1); // sleep for 1 second
    }

    return 0;
}
```

The mission

Implement the above prototype functions

Requirements:

1. Smart Home State:

- Design a structure (***SmartHomeState***) that encapsulates the state of the smart home. This should include:
 - Three boolean variables representing the state (ON/OFF) of three lamps.
 - Two integer variables representing the levels of two dimmers (The range for dimmer levels will have 5 levels 0-4).
 - An integer variable representing the room temperature (the range for temperature levels will have 10 levels 0-9).

2. Functionality:

- Implement functions to simulate the following:
 - Turning lamps ON and OFF.
 - Adjusting the dimmer levels.
 - Setting the room temperature.

3. Visualization:

```
Room State:
Lamps: 1[ON] 2[ON] 3[ON]
Dimmer Levels: A[0] B[4]
Temperature: 8°C
Lamp 1: *
Lamp 2: *
Lamp 3: *****
Temperature: --
```

- Implement a function to visually represent the state of the smart home in the terminal. This should include:
 - A visual representation of each lamp (using characters like '*' for ON based on dimmer level and '-' for OFF). Levels will be draw with "*" (e.g. level 3 will be draw as "*****")
 - A visual representation of the temperature as a horizontal bar. Levels will be draw with "--" (e.g. level 1 will be draw as "--")

4. Simulation:

- Use the loop in your main function to simulate those pre-defined scenarios of the smart home state. Use this loop to display how the state changes over time.

Code Structure:

Organize your code with appropriate functions for each task. Ensure good coding practices, such as clear variable names and comments explaining your logic.

General instructions

1. For each question, you receive a run code named **examplexx.out**, **inputxx.txt**, **outputxx.txt** where **xx** is the question number. Your code must behave like the given **examplexx.out** runtime code. In any case of a contradiction between the instructions in the question and the **examplexx.out** running code, the behavior of the **examplexx.out** prevails.
Your running code name should be **"t01qxx.out"** where **xx** is the question id (e.g. for question 2 the name will be **t01q02.out**).
2. For each question you have to write a new C source code file. The name of the C code should be **t01qxx.c** where **xx** is the question id (e.g. for question 2 the name will be **t01q02.c**)
3. You must use the following command in your Linux terminal to compile your C code:

gcc qxx.c -o t01qxx.out where **xx** is the question id

4. Run your running code:

./t01qxx.out where **xx** is the question id

Check your code using your keyboard and the printout on your terminal screen.

5. Before submitting to the model your files you should check your running code with the following command line:

./t01qxx/out< inputxx.txt> myOutputxx.txt where **xx** is the question id.

6. If your code is running OK then **myOutputxx.txt** should be the same as the given **outputxx.txt**
7. **A one new folder must be opened for all question.** The folder name is in the structure of **task01**. Inside the folder there will be at least 2 files:
 - a. Your C code file **qxx.c**
 - b. **gpt.txt** file, this file will include a copy of all your conversation, with your LLM (including the code that it produce for you). D) other files that might be requested in the question. The names of the files will be as indicated in each question, making sure to use upper and lower case letters.
 - c. Optional file name **t01.pdf**. Only one PDF file, containing special certificates. Approvals such as reserve approval. It is mandatory to **attach to the approval a letter explaining** the relief or request and the appropriate approval including specifying relevant dates.
8. Make sure that the program works properly both in the gcc compiler and that the program ran properly in the Linux environment. The dedicated input.txt file provided to you for each question must be used as a series of input commands and compared to the expected result shown in the output.txt file provided to you. Emphasized that the tester will test with other input files and output files and he will perform a much wider series of tests, including edge case and error tests.

9. You have to follow VPL instructions in order to submit the task.

Remarks:

1. 1 . It can be assumed that every input ends with pressing **Enter** key.
2. 2. **Numerical Input Validation:** Check inputs for validity (e.g., positive, negative, zero, whole, fractional). Invalid inputs include anything other than numbers, spaces, and tabs at the start or end.
3. 3. Normal numerical input is considered input with consecutive digits, spaces and tabs, when the spaces and tabs are either at the beginning or at the end of the input (or both) in sequence and the number should be in the requested range, any other input will be considered invalid.
4. 4. In any case of an error, or any unexpected response. The program will always print the same error message:
"MyERROR: An illegal operation was performed, so I have to stop the program."
There must be no situation where the program crashed or performed an illegal operation without this notification.
6. The exercise checking allows:
 - A. You can get compilation warnings on commands: scanf , strlen , strcpy , strcat , strcmp , gets .
 - B. You can get compilation warnings on while(1) and also on single line warnings.
5. No other compilation warnings or errors are allowed.
7. **Allowed Material:** Use only material studied until the publication of the work, including conditions, loops, arrays, strings (with studied functions), and two-dimensional arrays.
8. When entering the words and numbers, make sure that the user does not enter more than 200 characters in the input.
9. Pay attention to the notes on each and every section. In order to test your code use the given input file and check accordingly the output file. Your output file should be exactly the same as the given output file.
10. Wherever an anomaly is detected between the explanations and the given **examplexx.out** running code behavior, the given **examplexx.out** running code behavior must be followed.
11. **Submission in pairs**, ID numbers must be written in a comment inside the submission file. Everyone must submit separately in the model to receive a personal grade.