comp1511 week 05

Catherine Cheng

admin

- assignment 1 has been released!
  - due 8th July (week 7 Monday)
- no tutorials or labs next week (flex week)
- there will be help sessions and revision sessions!

agenda for today

- 2d arrays
- arrays and functions
- strings
- assignment 1 style

2d arrays practice

In this section we will write a short program that deals with 2D arrays.

The objective of this program is to simulate the exploration of a galaxy by placing celestial bodies such as stars, planets, and nebulae within a grid.

Functionality

1. Firstly, the program should prompt for celestial bodies with `Enter planets and nebulae:` and take input for planets and nebula until `q` is pressed.
    - Planets are added with `p [row] [col] [points]`
    - Nebulae are added with `n [row] [col]`

You may assume you will always be given valid input.

2. After populating the map with various celestial bodies your program will need to scan in the player's starting position. This will be given as a pair of integers which denotes the row and column (in that order). If the starting position is already occupied by a celestial body, the program should print out `Invalid starting position!` . Then the program should prompt with `Re-enter starting position:` and re-scan the position of the player, repeating until a valid position is scanned in.
3. After spawning the player, the program should prompt for stars with `Enter the position and points of the star(s):` and take input in the form `[row] [col] [points]` until `ctrl + D` is pressed. Again, you may assume that the provided input is always valid.
4. Finally, after spawning all celestial bodies, the program should print the galaxy with the `print_galaxy` function provided.

arrays and functions

- can we split what we've written into helper functions?

```c
int main(void) {
    struct celestial_body galaxy[SIZE][SIZE];
    int row, col;

    // Initialise the galaxy
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            galaxy[i][j].entity = EMPTY;
            galaxy[i][j].points = 0;
        }
    }

    // Place the planets and nebulae in the galaxy
    printf("Enter planets and nebulae: ");
    int points;
    char type;
    scanf(" %c", &type);
    while (type != 'q') {
        scanf(" %d %d", &row, &col);
        if (type == 'p') {
            scanf("%d", &points);
            galaxy[row][col].entity = PLANET;
            galaxy[row][col].points = points;
        } else if (type == 'n') {
            galaxy[row][col].entity = NEBULA;
            galaxy[row][col].points = NEBULA_POINTS;
        }
        scanf(" %c", &type);
    }

    // Place the player in the galaxy
    printf("Enter the starting position of the player: ");
    scanf("%d %d", &row, &col);
    while (row < 0 || row >= SIZE || col < 0 || col >= SIZE ||
           galaxy[row][col].entity != EMPTY) {
        printf("Invalid player position!\n");
        printf("Please re-enter the starting position of the player: ");
        scanf("%d %d", &row, &col);
    }
    galaxy[row][col].entity = SPACESHIP;
    galaxy[row][col].points = 0;

    // Place the stars in the galaxy
    printf("Enter the position and points of the star(s): \n");
    while(scanf("%d %d %d", &row, &col, &points) == 3) {
        galaxy[row][col].entity = STAR;
        galaxy[row][col].points = points;
    }

    print_map(galaxy);
}
```

```c
int main(void) {
    struct celestial_body galaxy[SIZE][SIZE];
    int row, col;

    // Initialise the galaxy
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            galaxy[i][j].entity = EMPTY;
            galaxy[i][j].points = 0;
        }
    }

    // Place the planets and nebulae in the galaxy
    printf("Enter planets and nebulae: ");
    int points;
    char type;
    scanf(" %c", &type);
    while (type != 'q') {
        scanf(" %d %d", &row, &col);
        if (type == 'p') {
            scanf("%d", &points);
            galaxy[row][col].entity = PLANET;
            galaxy[row][col].points = points;
        } else if (type == 'n') {
            galaxy[row][col].entity = NEBULA;
            galaxy[row][col].points = NEBULA_POINTS;
        }
        scanf(" %c", &type);
    }

    // Place the player in the galaxy
    printf("Enter the starting position of the player: ");
    scanf("%d %d", &row, &col);
    while (row < 0 || row >= SIZE || col < 0 || col >= SIZE ||
           galaxy[row][col].entity != EMPTY) {
        printf("Invalid player position!\n");
        printf("Please re-enter the starting position of the player: ");
        scanf("%d %d", &row, &col);
    }
    galaxy[row][col].entity = SPACESHIP;
    galaxy[row][col].points = 0;

    // Place the stars in the galaxy
    printf("Enter the position and points of the star(s): \n");
    while(scanf("%d %d %d", &row, &col, &points) == 3) {
        galaxy[row][col].entity = STAR;
        galaxy[row][col].points = points;
    }

    print_map(galaxy);
}
```

```c
int main(void) {
    struct celestial_body galaxy[SIZE][SIZE];

    // Initialize the galaxy
    initialise_galaxy(galaxy);

    // Place the planets and nebulae in the galaxy
    place_planets_and_nebula(galaxy);

    // Place the player in the galaxy
    place_player(galaxy);

    // Place the stars in the galaxy
    place_stars(galaxy);

    // Print the galaxy
    print_map(galaxy);

    // Print the sum of the points in the galaxy
    print_galaxy_sum(galaxy);
}
```

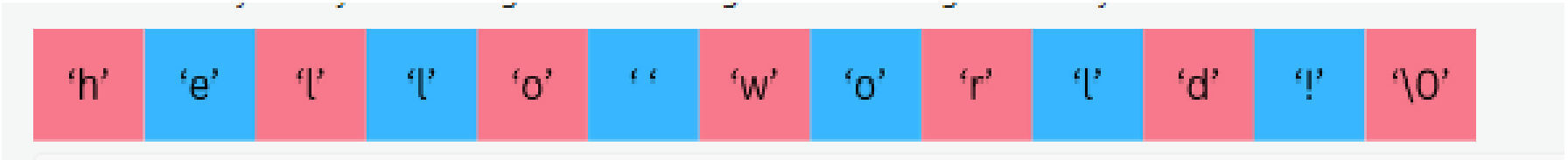
```c
// Function to initialize the galaxy
//
// Parameters:
// - galaxy: the 2D array representing the galaxy
//
// returns: nothing
void initialise_galaxy(struct celestial_body galaxy[SIZE][SIZE]) {
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            galaxy[i][j].entity = EMPTY;
            galaxy[i][j].points = 0;
        }
    }
    return;
}

// Function to place the player in the galaxy
//
// Parameters:
// - galaxy: the 2D array representing the galaxy
//
// returns: nothing
void place_player(struct celestial_body galaxy[SIZE][SIZE]) {
    int row, col;
    printf("Enter the starting position of the player: ");
    scanf("%d %d", &row, &col);
    while (row < 0 || row >= SIZE || col < 0 || col >= SIZE ||
           galaxy[row][col].entity != EMPTY) {
        printf("Invalid player position!\n");
        printf("Please re-enter the starting position of the player: ");
        scanf("%d %d", &row, &col);
    }
    galaxy[row][col].entity = SPACESHIP;
    galaxy[row][col].points = 0;
    return;
}
```

```c
// Function to place the planets and nebulae in the galaxy
//
// Parameters:
// - galaxy: the 2D array representing the galaxy
//
// returns: nothing
void place_planets_and_nebula(struct celestial_body galaxy[SIZE][SIZE]) {
    int row;
    int col;
    printf("Enter planets and nebulae: ");
    int points;
    char type;
    scanf(" %c", &type);
    while (type != 'q') {
        scanf(" %d %d", &row, &col);
        if (type == 'p') {
            scanf("%d", &points);
            galaxy[row][col].entity = PLANET;
            galaxy[row][col].points = points;
        } else if (type == 'n') {
            galaxy[row][col].entity = NEBULA;
            galaxy[row][col].points = NEBULA_POINTS;
        }
        scanf(" %c", &type);
    }
    return;
}

// Function to print the sum of the points in the galaxy
//
// Parameters:
// - galaxy: the 2D array representing the galaxy
//
// returns: nothing
void print_galaxy_sum(struct celestial_body galaxy[SIZE][SIZE]) {
    int sum = 0;
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            sum += galaxy[i][j].points;
        }
    }
    printf("The total points in the galaxy is: %d\n", sum);
}

// Function to place the stars in the galaxy
//
// Parameters:
// - galaxy: the 2D array representing the galaxy
//
// returns: nothing
void place_stars(struct celestial_body galaxy[SIZE][SIZE]) {
    int row;
    int col;
    int points;
    printf("Enter the position and points of the star(s): \n");
    while(scanf("%d %d %d", &row, &col, &points) == 3) {
        galaxy[row][col].entity = STAR;
        galaxy[row][col].points = points;
    }
}
```



strings

how do we declare and initialise a string?

how do we loop through a string?

how do we print a string?

string practice

```c
// Functions to implement:

// 1.
// returns the number of lowercase letters in `char *string`
int count_lowercase(char *string);

// 2.
// modifies `char *string` by converting all its vowels to uppercase
void make_vowels_uppercase(char *string);

// 3..
// shortens a string so that it ends after the first word
// e.g. "This is a sentence" should turn into:
//      "This"
//
// (hint. what defines when a string ends?)
void delete_following_words(char *string);
```

helper functions provided in the tute questions

string practice (helper functions)

```c
// Provided char functions

// Returns : 1 if `c` is a lowercase letter
//         : 0 otherwise.
int is_lowercase(char c) {

    return 'a' <= c && c <= 'z';
}

// Returns : 1 if `c` is an uppercase letter
//         : 0 otherwise.
int is_uppercase(char c) {

    return 'A' <= c && c <= 'Z';
}

// Returns : 1 if `c` is a letter
//         : 0 otherwise.
int is_letter(char c) {

    return is_lowercase(c) || is_uppercase(c);
}

// Returns : `c` converted to lowercase, if it was an uppercase letter
//         : `c` unmodified, otherwise
char to_lowercase(char c) {
    if (is_uppercase(c)) {
        return c - 'A' + 'a';
    }

    return c;
}
```

```c
// Returns : `c` converted to uppercase, if it was a lowercase letter
//         : `c` unmodified, otherwise
char to_uppercase(char c) {
    if (is_lowercase(c)) {
        return c - 'a' + 'A';
    }

    return c;
}

// Returns : 1 if `c` is an uppercase or lowercase vowel
//         : 0 otherwise.
int is_vowel(char c) {
    char lower_c = to_lowercase(c);

    return lower_c == 'a'
        || lower_c == 'e'
        || lower_c == 'i'
        || lower_c == 'o'
        || lower_c == 'u';
}
```

string practice (solutions)

```c
// 1.
// returns the number of lowercase letters in `char *string`
int count_lowercase(char string[MAX_CHARS]) {
    int count = 0;
    for (int i = 0; string[i] != '\0'; i++) {
        if (is_lowercase(string[i])) {
            count++;
        }
    }
    return count;
}

// 2.
// modifies `char *string` by converting all its vowels to uppercase
void make_vowels_uppercase(char string[MAX_CHARS]) {
    for (int i = 0; string[i] != '\0'; i++) {
        if (is_vowel(string[i])) {
            string[i] = to_uppercase(string[i]);
        }
    }
    return;
}

// 3..
// shortens a string so that it ends after the first word
// e.g. "This is a sentence" should turn into:
//      "This"
//
// (hint. what defines when a string ends?)
void delete_following_words(char string[MAX_CHARS]) {
    for (int i = 0; string[i] != '\0'; i++) {
        if (string[i] == ' ') {
            string[i] = '\0';
        }
    }
    return;
}
```

assignment 1 style!

- **include a header comment!**
- run 1511 style cs_sokoban.c
- #define all constants AND all char commands
  - e.g. #define ADD_WALL_COMMAND ‘W’ and #define NUM_ROWS 10
- make sure output matches examples in spec exactly
- use helper functions!
- function comments

any questions?