SOFE 3950U Operating Systems

TUTORIAL REPORT

Tutorial Report TITLE #: **Tutorial 4 (Jeopardy)**

Date: **Feb 10, 2025**

LAB GROUP NUMBER #: **8**

CRN: **74027**

LAB GROUP MEMBERS				
#	Surname	Name	ID	Signature
1	Binukumar	Abinav	100851953	A.B
2	Yang	Gitae	100794460	G.Y
3	Hanoosh	Abdullah	100749026	A.H.
4	LIN	SI THU	100992271	S.T.L

Deliverables

In this tutorial session, we were given a task to provide C codes to make a simple jeopardy game of our own. The template was provided by the TA and we added our codes into it to generate an actual working file.

jeopardy.c

```c
// Tokenizes the user's input answer
void tokenize(char *input, char **tokens)
{
    input[strcspn(input, "\n")] = 0; // Remove newline

    // Skip "what is" or "who is" and extract the actual answer
    if (strncmp(input, "what is ", 8) == 0) {
        tokens[2] = input + 8;
    } else if (strncmp(input, "who is ", 7) == 0) {
        tokens[2] = input + 7;
    } else {
        tokens[2] = input; // Default to entire input if format is wrong
    }
}

// Displays the game results for each player, ranked by score
void show_results(player *players, int num_players)
{
    // Simple sorting to rank players by score
    for (int i = 0; i < num_players - 1; i++) {
        for (int j = 0; j < num_players - i - 1; j++) {
            if (players[j].score < players[j + 1].score) {
                player temp = players[j];
                players[j] = players[j + 1];
                players[j + 1] = temp;
            }
        }
    }
```

This is the very first function we have at the top which we tokenize all the answers that user inputs. This way it will differentiate the user's answer with the jeopardy question answering form of 'what is' and 'who is'. The second function you see below is the show_result which outputs the score of the players in descending order once the jeopardy game has been finished.

```
initialize_game(); // Initialize questions

// Prompt for player names
for (int i = 0; i < NUM_PLAYERS; i++) {
    printf("Enter player %d name: ", i + 1);
    fgets(players[i].name, MAX_LEN, stdin);
    players[i].name[strcspn(players[i].name, "\n")] = 0; // Remove newline
    players[i].score = 0;
}

// Start game loop
while (true) {
    display_categories();

    // Ask for player's name until valid
    while (true) {
        printf("\nEnter the player's name who is selecting: ");
        fgets(buffer, BUFFER_LEN, stdin);
        buffer[strcspn(buffer, "\n")] = 0; // Remove newline

        if (player_exists(players, NUM_PLAYERS, buffer)) {
            break; // Valid player name
        } else {
            printf("Player not found. Try again.\n");
        }
    }
```

This is the screenshot of the code where it asks to enter the name of the player. During the game, the prompt will be asking to enter the name of the player that will be playing the current round. If the user types any player names that does not exist/did not type in the beginning of the game, it will ask to retype the name until it is valid.
It is not shown in the screenshot above, but it will do the same thing for the category and the amount of money they are willing to pay for.

```
// Ask for answer
printf("Enter answer (format: 'what is X' or 'who is X'): ");
fgets(answer, MAX_LEN, stdin);
answer[strcspn(answer, "\n")] = 0; // Remove newline

char *tokens[3] = { NULL };
tokenize(answer, tokens);

// Validate the answer
if (valid_answer(category, value, tokens[2])) {
    update_score(players, NUM_PLAYERS, buffer, value);
    printf("Correct!\n");
} else {
    for (int i = 0; i < NUM_QUESTIONS; i++) {
        if (strcasecmp(questions[i].category, category) == 0 && questions[i].value == value) {
            printf("Incorrect. The correct answer was: %s\n", questions[i].answer);
            break;
        }
    }
}

// Check if all questions are answered
bool all_answered = true;
for (int i = 0; i < NUM_QUESTIONS; i++) {
    if (!questions[i].answered) {
        all_answered = false;
        break;
    }
}
if (all_answered) break;
```

This is checking if the answers from the users align with what they have and prompt incorrect or correct. If it is incorrect, it will tell you the answer at the end. Once all the questions have been answered, it will either ask a new player to begin the next round or sum it up using the show_result to notify that the game is over.

players.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "players.h"

// Returns true if the player name matches one of the existing players
bool player_exists(player *players, int num_players, char *name)
{
    for (int i = 0; i < num_players; i++) {
        if (strcmp(players[i].name, name) == 0) {
            return true;
        }
    }
    return false;
}

// Go through the list of players and update the score for the player given their name
void update_score(player *players, int num_players, char *name, int score)
{
    for (int i = 0; i < num_players; i++) {
        if (strcmp(players[i].name, name) == 0) {
            players[i].score += score;
            return;
        }
    }
}

```

This is where it checks the player name matches with the existing players once the jeopardy game has started. Also, it will update the scores accordingly once the player has finished their round.

questions.c

```c
// Converts a string to lowercase and removes trailing newline
void clean_input(char *str) {
    int len = strlen(str);
    if (len > 0 && (str[len - 1] == '\n' || str[len - 1] == '\r')) {
        str[len - 1] = '\0'; // Remove newline
    }
    for (int i = 0; str[i]; i++) {
        str[i] = tolower(str[i]); // Convert to lowercase
    }
}

// Initializes the array of questions for the game
void initialize_game(void)
{
    // History Questions
    strcpy(questions[0].category, "history");
    strcpy(questions[0].question, "What year did WW2 End?");
    strcpy(questions[0].answer, "1945");
    questions[0].value = 100;
    questions[0].answered = false;

    strcpy(questions[1].category, "history");
    strcpy(questions[1].question, "Who killed JFK?");
    strcpy(questions[1].answer, "Lee Harvey Oswald");
    questions[1].value = 200;
```

This is the first part of the questions.c file we have which contains all the information about the questions that are used in this game. First, to avoid any interactions with the prompt, we have it made case sensitive and line sensitive when the user is inputting the answers. Next is the game question part which is under initialize_game function. We have three categories of history, geography, and pop culture. The screenshot shows only the history questions.

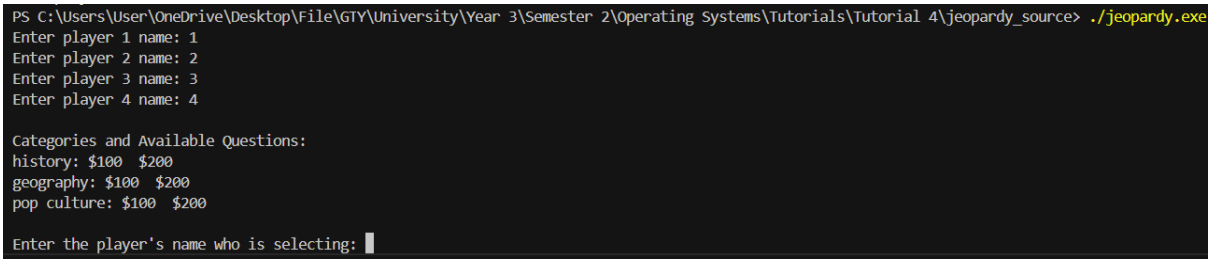
```c
// Displays all categories with their available $100 and $200 questions
void display_categories(void)
{
    printf("\nCategories and Available Questions:\n");

    // Define possible question values
    int values[] = {100, 200};
    int num_values = sizeof(values) / sizeof(values[0]);

    // Iterate over all categories
    for (int c = 0; c < NUM_CATEGORIES; c++) {
        printf("%s: ", categories[c]);  // Print category name

        // Ensure each category is printed with $100 and $200 values
        for (int v = 0; v < num_values; v++) {
            bool available = false;

            // Iterate over all questions to check if this value exists for the category
            for (int i = 0; i < NUM_QUESTIONS; i++) {
                if (strcasecmp(questions[i].category, categories[c]) == 0 &&
                    questions[i].value == values[v] &&
                    !questions[i].answered) {
                    available = true;
                    break;
                }
```

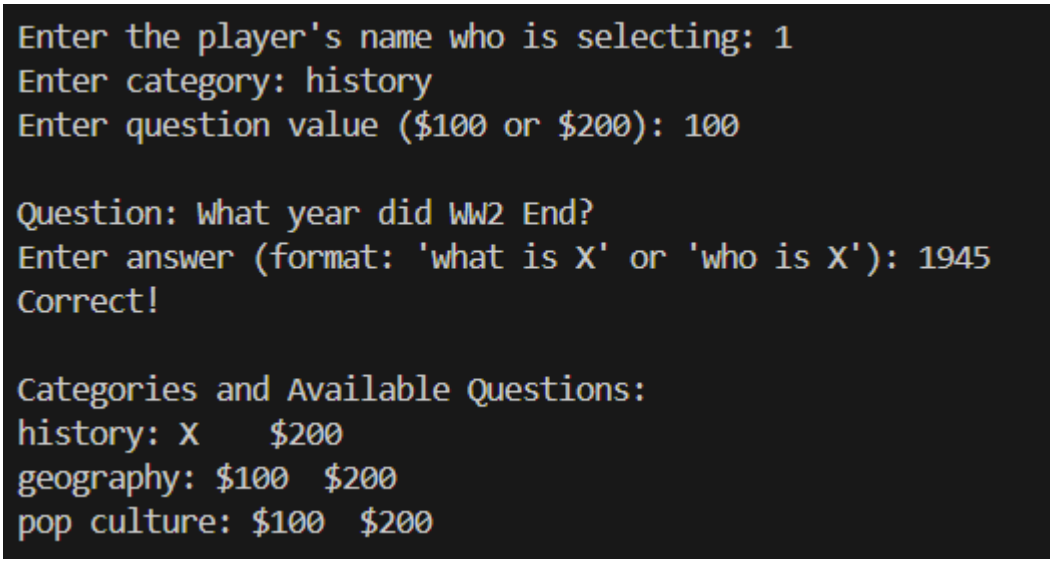
This is showing how the categories are displayed when the user reaches this point. It will display the 3 categories we have in this code and the amount they are playing for. If a game has already been played for a specific amount for a specific category, it will be displayed with ‘x’ to indicate it cannot be played.

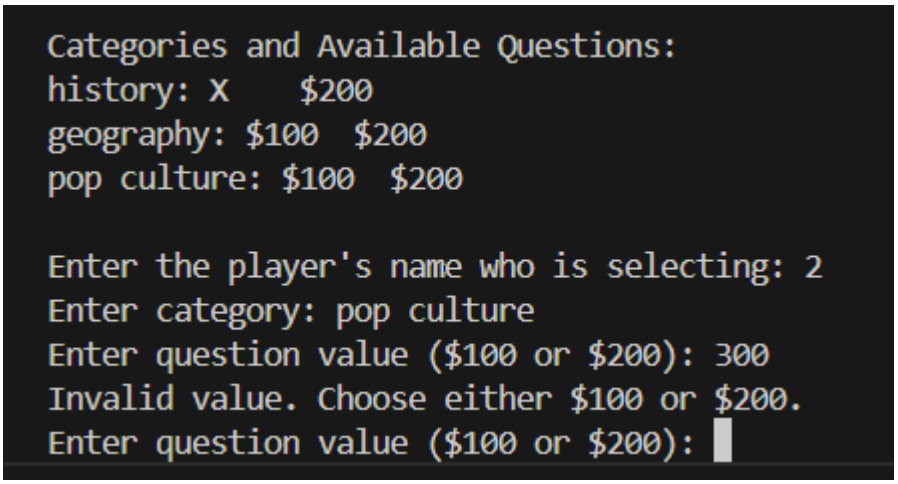
```c
// Checks if the answer is correct
bool valid_answer(char *category, int value, char *answer)
{
    clean_input(answer);  // Standardize user input

    for (int i = 0; i < NUM_QUESTIONS; i++) {
        if (strcasecmp(questions[i].category, category) == 0 &&
            questions[i].value == value &&
            !questions[i].answered) {

            char correct_answer[MAX_LEN];
            strcpy(correct_answer, questions[i].answer);
            clean_input(correct_answer); // Clean correct answer

            if (strcmp(correct_answer, answer) == 0) {
                questions[i].answered = true;
                return true;
            }
        }
    }
    return false;
}

// Checks if the question has already been answered
bool already_answered(char *category, int value)
{
    for (int i = 0; i < NUM_QUESTIONS; i++) {
        if (strcasecmp(questions[i].category, category) == 0 && questions[i].value == value) {
```

This checks if the answer is correct from the user and it also checks if the question has already been answered by other players indicating that it cannot be played further on.

Output

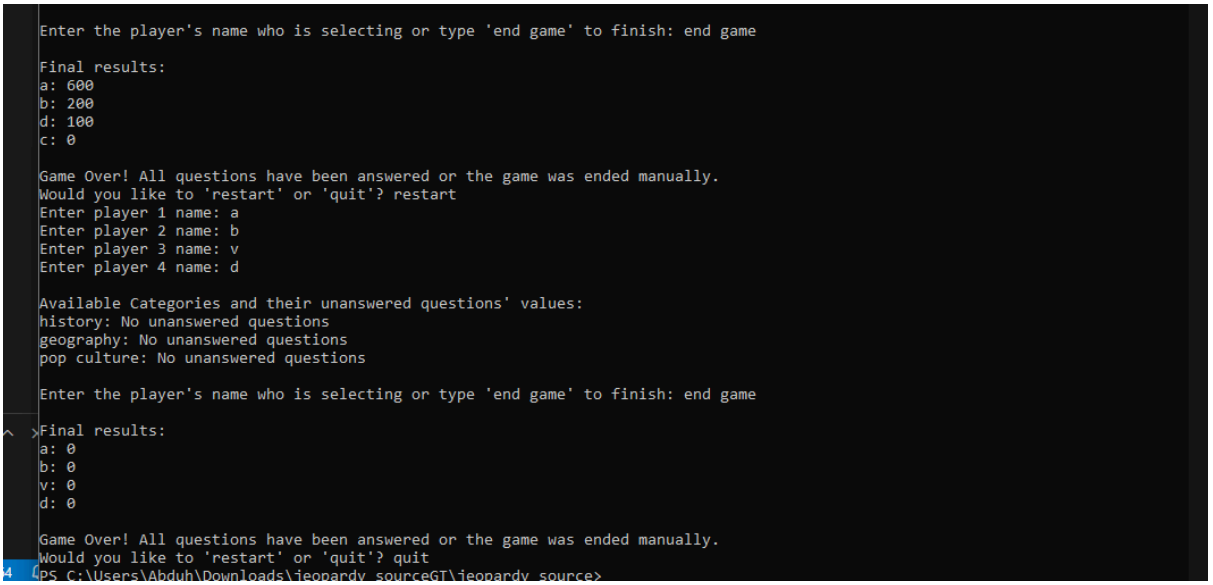






These screenshots show how the game performs in the terminal. It will ask to type the player first and by turns they will answer specific questions within the category chosen. Once done, it will be passed on to the next player and it will be repeated.

the last screenshot is a test we did if the user inputs something other than the existing values or categories, the prompt will continue to ask until the user finally answers correct one.



This screenshot shows that the users can end the game if they choose by inputting ‘end game’ , this will then show the results of the game in order with each player name and their points, as well as, ask if the users would like to restart, at which point the application will restart from the beginning, or end the game, which terminates the program.