SOFE 3950U
Operating Systems

Tutorial 3: Jeopardy

CRN: 74171

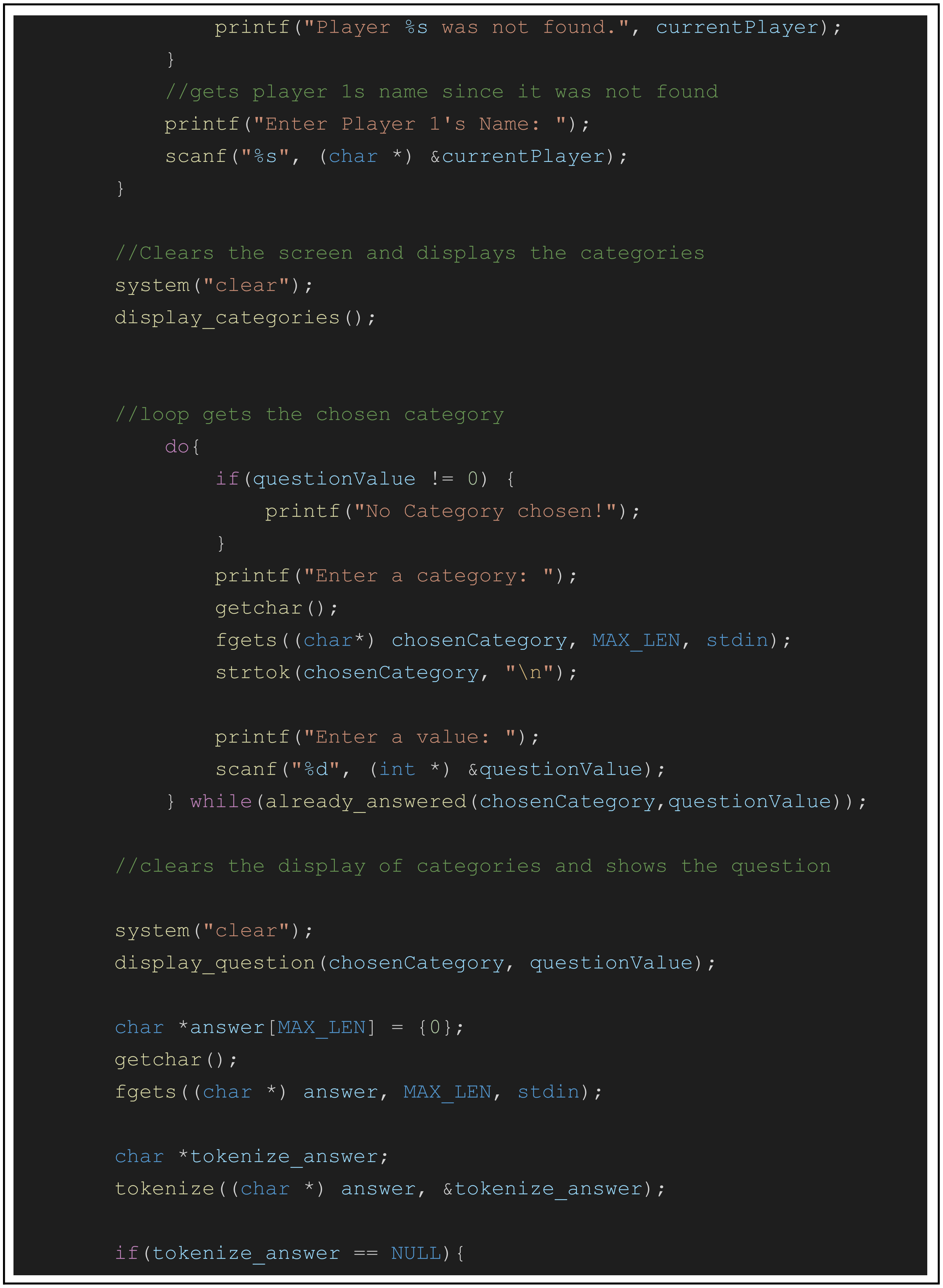
Group: A8

Date: Feb 20th, 2022

First Name	Last Name	Student Number
David	Fung	100767734
Anish	Patel	100751489
Raphaiel	Halim	100700318

jeopardy.c

```
/*
 * Tutorial 3 Jeopardy Project for SOFE 3950U / CSCI 3020U: Operating
Systems
 *
 * Copyright (C) 2015, <David Fung 100767734, Anish Patel 100751489,
Raphaiel Halim 100700318>
 * All rights reserved.
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "questions.h"
#include "players.h"
#include "jeopardy.h"

// Put macros or constants here using #define
#define BUFFER_LEN 256
#define NUM_PLAYERS 4

// Put global environment variables here

// Processes the answer from the user containing what is or who is and
tokenizes it to retrieve the answer.
//void tokenize(char *input, char **tokens);

// Displays the game results for each player, their name and final
score, ranked from first to last place
//void show_results(player *players, int num_players);


int main(int argc, char *argv[])
{
    // An array of 4 players, may need to be a pointer if you want it
set dynamically
    player players[NUM_PLAYERS];
    // Input buffer and and commands
    char buffer[BUFFER_LEN] = { 0 };

    // Display the game introduction and initialize the questions
    initialize_game();

    //Starting message
    printf("Welcome to Jeopardy!\n");

    // Prompt for players names and initialize each of the players in
the array
    for(int i = 0; i < NUM_PLAYERS; i++){
         printf("Enter your name Player %d: ", (i+1));
         scanf("%s", players[i].name);

    //Sets score to 0
        players[i].score = 0;

    }

    // Perform an infinite loop getting command input from users until
game ends
    while (fgets(buffer, BUFFER_LEN, stdin) != NULL)
    {

        //clears the command line
        system("clear");

        char chosenCategory[MAX_LEN] = "";
        char currentPlayer[MAX_LEN] = "";
        int questionValue;


        // Call functions from the questions and players source files

        //loop gets the current player
        while(!player_exists(players, 4, currentPlayer)){
            if(strcmp(currentPlayer, "") != 0) {
```



```c
            printf("Player %s was not found.", currentPlayer);
        }
        //gets player 1s name since it was not found
        printf("Enter Player 1's Name: ");
        scanf("%s", (char *) &currentPlayer);
    }

    //Clears the screen and displays the categories
    system("clear");
    display_categories();


    //loop gets the chosen category
        do{
            if(questionValue != 0) {
                printf("No Category chosen!");
            }
            printf("Enter a category: ");
            getchar();
            fgets((char*) chosenCategory, MAX_LEN, stdin);
            strtok(chosenCategory, "\n");

            printf("Enter a value: ");
            scanf("%d", (int *) &questionValue);
        } while(already_answered(chosenCategory,questionValue));

    //clears the display of categories and shows the question

    system("clear");
    display_question(chosenCategory, questionValue);

    char *answer[MAX_LEN] = {0};
    getchar();
    fgets((char *) answer, MAX_LEN, stdin);

    char *tokenize_answer;
    tokenize((char *) answer, &tokenize_answer);

    if(tokenize_answer == NULL){
            printf("Please try again and enter a valid answer.");
        } else if(valid_answer(chosenCategory, questionValue,
tokenize_answer)) {
            printf("You are CORRECT!!");
            printf("Player %s gets %d points \n", currentPlayer,
questionValue);
            update_score(players, 4, currentPlayer, questionValue);
        } else {
            printf("You are WRONG!!!");
            // print_answer(chosenCategory, questionValue);
        }

        show_results(players, 4);
        getchar();

    }
    return 0;
}



// tokenize function to get a valid answer
void tokenize(char *input, char **tokens) {
    const char delimiter = " ";

    char *stringTokens = strtok(input, delimiter);

    if (stringTokens != NULL){
        if (strcmp(stringTokens, "who") != 0 || strcmp(stringTokens,
"what") != 0) {
            return;
        }
        if (strcmp(stringTokens, "is") != 0) {
            return;
        }
    }

    *stringTokens = strtok(NULL, delimiter);
}
```

```c
//Gets the results and exits successfully
void show_results(player *players, int numPlayers) {
    int playerName = 0;
    int playerScore = 0;
    int winner = 0;

    for(int i = 0; i < numPlayers; i++) {
        if((int) strlen(players[i].name) > playerName)
            playerName = strlen(players[i].name);

        if(players[i].score > playerScore) {
            playerScore = players[i].score;
            winner = i;
        }
    }

    // prints the final scores of players
    printf("The final scores are: \n");
    for(int i = 0; i < numPlayers; i++)
        printf("%*s: %d\n", playerName + 1, players[i].name,
players[i].score);

    printf("The winner is: %s", players[winner].name);
    printf("Congrats!!!");

    return EXIT_SUCCESS;
}
```

players.c

```c
/*
 * Tutorial 3 Jeopardy Project for SOFE 3950U / CSCI 3020U: Operating
Systems
 *
 * Copyright (C) 2015, <David Fung 100767734, Anish Patel 100751489,
Raphaiel Halim 100700318>
 * All rights reserved.
 *
 */
#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) {
    //goes through each player
    for (int i = 0; i < num_players; i++){
        //comparing the names
        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){
    //goes through each player
    for (int i = 0; i < num_players; i++){
        //comparing the names
        if (strcmp(players[i].name, name) == 0){
            players[i].score += score; //update score of current player
        }
    }
}
```

questions.c

```c
/*
 * Tutorial 3 Jeopardy Project for SOFE 3950U / CSCI 3020U: Operating Systems
 *
 * Copyright (C) 2015, <David Fung 100767734, Anish Patel 100751489, Raphaiel Halim 100700318>
 * All rights reserved.
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "questions.h"

// Initializes the array of questions for the game
void initialize_game(void)
{
    // initialize each question struct and assign it to the questions array


    for (int i=0; i<12; i++){

        // set all questions to unanswered
        questions[i].answered = false;
    }

    // programming questions
    questions[0].value=200;
    strcpy(questions[0].category, "programming");
    strcpy(questions[0].question, "A data type of an ordered sequence of characters");
    strcpy(questions[0].answer, "string");

    questions[3].value=400;
    strcpy(questions[3].category, "programming");
    strcpy(questions[3].question, "A control flow statement that allows
code to be executed repeatedly based on a boolean condition");
    strcpy(questions[3].answer, "while loop");

    questions[6].value=600;
    strcpy(questions[6].category, "programming");
    strcpy(questions[6].question, "A special program that processes
statements in a programming language and converts it to machine
language");
    strcpy(questions[6].answer, "compiler");

    questions[9].value=800;
    strcpy(questions[9].category, "programming");
    strcpy(questions[9].question, "Linux was written in ___ Language");
    strcpy(questions[9].answer, "c");


    // algorithm questions
    questions[1].value=200;
    strcpy(questions[1].category, "algorithms");
    strcpy(questions[1].question, "The time complexity of hash maps");
    strcpy(questions[1].answer, "o(1)");

    questions[4].value=400;
    strcpy(questions[4].category, "algorithms");
    strcpy(questions[4].question, "A data structure where elements are
added or removed from the top in LIFO order");
    strcpy(questions[4].answer, "stack");

    questions[7].value=600;
    strcpy(questions[7].category, "algorithms");
    strcpy(questions[7].question, "A tree in which the value in each
internal node is greater or equal to the values in the children of that
node");
    strcpy(questions[7].answer, "max heap");

    questions[10].value=800;
    strcpy(questions[10].category, "algorithms");
    strcpy(questions[10].question, "The type of algorithm that follows
the problem-solving heuristic of making the locally optimal choice at
```

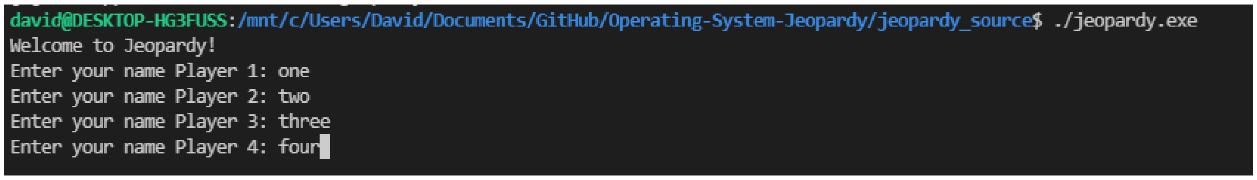
```
each stage");
    strcpy(questions[10].answer, "greedy algorithm");


    // database questions
    questions[2].value=200;
    strcpy(questions[2].category, "databases");
    strcpy(questions[2].question, "The standard language for storing,
manipulating, and retrieving data in databases");
    strcpy(questions[2].answer, "SQL");

    questions[5].value=400;
    strcpy(questions[5].category, "databases");
    strcpy(questions[5].question, "The SQL command to remove named
schema elements, such as tables, domains, or constraint");
    strcpy(questions[5].answer, "drop");

    questions[8].value=600;
    strcpy(questions[8].category, "databases");
    strcpy(questions[8].question, "Non-tabular databases that store data
differently than relational tables");
    strcpy(questions[8].answer, "nosql");

    questions[11].value=800;
    strcpy(questions[11].category, "databases");
    strcpy(questions[11].question, "The representational data model used
most frequently in traditional commercial DBMSs");
    strcpy(questions[11].answer, "relational data model");


}

// Displays each of the remaining categories and question dollar values
that have not been answered
void display_categories(void)
{
    // print categories and dollar values for each unanswered question
in questions array
```

```
    int column_size = 15;


    // print categories
    for(int i=0; i < 3; i++){
        printf("%-*s", column_size, categories[i]);
    }

    for(int i=0; i < 12; i++){

        if(i % 3 == 0){
            printf("\n");
        }

        //print questions if not answered yet
        if(!questions[i].answered){
            printf("%-*d", column_size, questions[i].value);
        }

        else{
            printf("%-*s", column_size, " --- ");
        }

    }

    printf("\n");

}

// Displays the question for the category and dollar value
void display_question(char *category, int value)
{

    for(int i=0; i < 12; i++){
        if(strcmp(questions[i].category, category) == 0 &&
questions[i].value == value){
            printf("%s\n", questions[i].question);
        }
    }
```

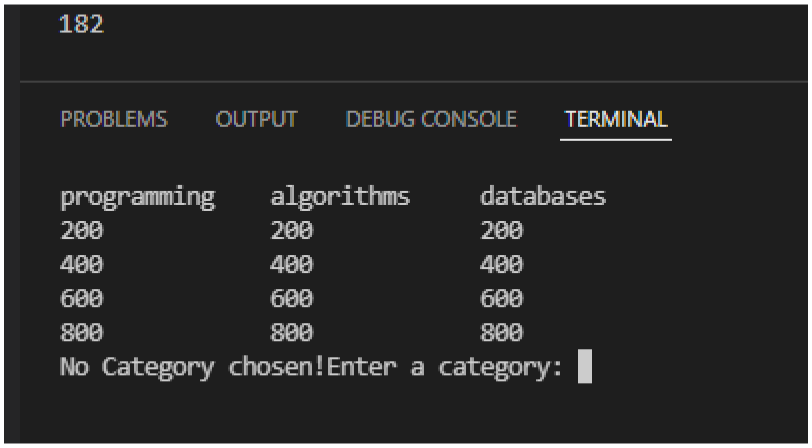
```
}

// Returns true if the answer is correct for the question for that
category and dollar value
bool valid_answer(char *category, int value, char *answer)
{
    // Look into string comparison functions

    // checks which question is equal to th
    for(int i = 0; i < 12; i++){
        if(strcmp(questions[i].category, category) == 0 &&
questions[i].value == value){
            if(strcasecmp(questions[i].answer, answer) == 0){
                questions[i].answered = true;

                return true;
            }

        }
    }

    return false;
}

// Returns true if the question has already been answered
bool already_answered(char *category, int value)
{
    // lookup the question and see if it's already been marked as
answered
    for (int i=0; i<12; i++){
        if(strcmp(questions[i].category, category == 0 ) &&
questions[i].value == value){
            if (questions[i].answered){
                printf("Question has already been answered");
                return true;
            }

            else {
```

```
                        return false;
                  }


            }



      }



      return false;
}
```

Screenshots:

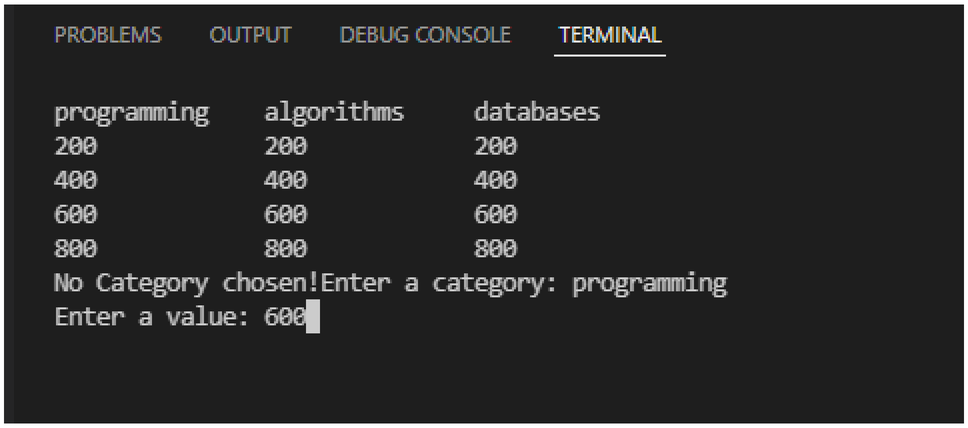


```
david@DESKTOP-HG3FUSS:/mnt/c/Users/David/Documents/GitHub/Operating-System-Jeopardy/jeopardy_source$ ./jeopardy.exe
Welcome to Jeopardy!
Enter your name Player 1: one
Enter your name Player 2: two
Enter your name Player 3: three
Enter your name Player 4: four
```

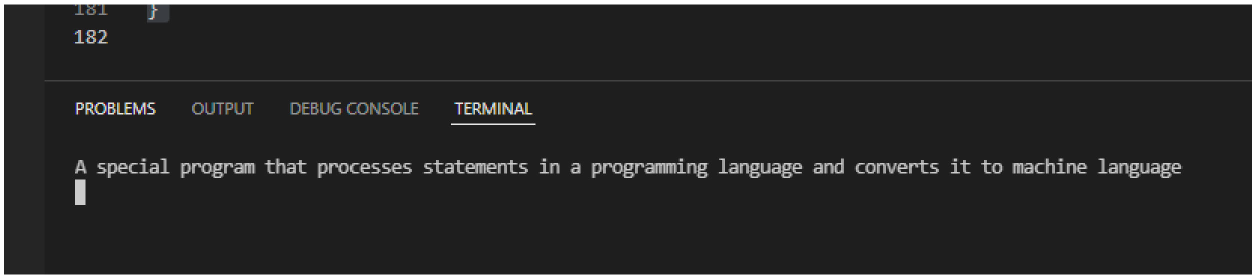


```
182
```

```
programming    algorithms    databases
200            200           200
400            400           400
600            600           600
800            800           800
No Category chosen!Enter a category:
```



```
programming    algorithms    databases
200            200           200
400            400           400
600            600           600
800            800           800
No Category chosen!Enter a category: programming
Enter a value: 600
```



```
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
A special program that processes statements in a programming language and converts it to machine language
```