SOFE 3950U / CSCI 3020U: Operating Systems

TUTORIAL #4: Jeopardy

Objectives

- Learn the fundamentals of C
- Gain experience writing C programs with multiple source files
- Create a command line version of the game Jeopardy

Important Notes

- Work in groups of **3-4** students
    - All reports must be submitted as a PDF on Canvas, if source code is included submit everything as an archive (e.g. zip, tar.gz)
- Save the file as <tutorial_number>_<first student's id>.pdf (e.g. tutorial4_100123456.pdf)

Notice

It is recommended for this tutorial and others that you save/bookmark the following resources as they are very useful for C programming.

- http://en.cppreference.com/w/c
- http://www.cplusplus.com/reference/clibrary/
- http://users.ece.utexas.edu/~adnan/c-refcard.pdf
- http://gribblelab.org/CBootcamp

The following resources are helpful as you will need to perform string tokenization and use POSIX functions.

- http://www.tutorialspoint.com/c_standard_library/c_function_strtok.htm

Tutorial Setup

1. For the purpose of this tutorial and future tutorial activities create a new repository on GitHub for the course (if you haven't already!). You will need to go into the settings of your repository and add all of your tutorial group members as collaborators.

2. Add the provided source code template for this tutorial to your github repository, if you forget how to use git please refer to the following resources. The main commands that you will need to use are: **git add, git commit, git push**
    - https://help.github.com/articles/set-up-git
    - https://try.github.io
    - http://git-scm.com/docs/gittutorial

3. Ensure that you are able to use the Makefile and C source code examples to build the code using **make**. If you are having issues you may need to modify the makefile and set the **CC** variable to **gcc** instead of **clang**.

Tutorial Activity

For the purpose of the tutorial activity you will be working as a group on a single project to create a command line version of Jeopardy with support for up to four players, use the source files provided and the following project requirements and guidelines to complete the project.

In addition, it is also advised that when working on the code you frequently push your code to GitHub so that other group members can stay in sync. Also, it is recommended that each group member work on **separate source files**. This avoids **merge conflicts**, which can happen when two or more people are working on the same section of code.

Project Requirements

These are the main guidelines, you are free to add more than is required here (coloured output, terminal animations, etc.) to personalize your project and make it your own.

1. Your program must have a command line prompt that is used for all interactions with the Jeopardy system.

2. The program must ask for the names of the four players who will be playing Jeopardy, each player must enter their name and have it recorded in the system and their initial earnings set as 0.

3. The program then starts the game of Jeopardy and prints out each of the question categories and displays the dollar values for each question.

4. The program must accept the name of the person selected to pick the category and question.
    - You can select the person any way you choose (first person to raise their hand, random selection, etc.).
    - The program must validate that the name of the person entered is correct and matches to one of the players.

5. Once the player's name has been entered the program must prompt the player for the category and dollar amount question.
    - The program must verify that the category and dollar amount question has not already been answered.

6. The program must then display the question for the player to then answer.
    - The questions and answers can be defined for each category and for each dollar amount for each category.
    - However, if you want to make the game more interesting for multiple playthroughs you can have a bank of question and randomly select each question for each category from this question bank.

7. After the question is displayed, the program must prompt the player for the answer, the player **must** enter the answer starting with one of the following: **what is** or **who is**.
    - You will need to use string tokenization to parse the answer, use the resources above to help.
    - To make the game easier to program you can have every question only required a one word answer.

8. After the player answers the question the program must display whether they got it **correct** or **incorrect.**
    - If the player answered the question correctly the program must update their score.
    - If they answered incorrectly the program will display the answer and no user will receive any points.
    - The question must then be marked as answered so it cannot be used again.

9. After the question has been completed, the program must print the remaining categories and question dollar amounts, then repeat requirements 4 to 9, until all of the questions have been completed.

10. Once the players have completed all of the questions the game must then print each of the players from the first place to last place and print their total earnings.
    - The player with the highest earnings is the winner.

11. **Bonus marks** will be awarded to the group with the best project, feel free to add any enhancements to your project to improve it (e.g. randomly chosen categories and questions, double jeopardy, coloured terminal output, etc.) The class will use the group's project to play Jeopardy at the start of the next tutorial.

Project Guidelines

The following are project guidelines intended to help you in completing the project, you are free to deviate from them in order to complete the project.

Questions source files

1. The questions source files should contain all of your categories, questions, and answers as well as any functions to help in getting questions or validating

answers.

2. The list of categories can be stored in an array of strings **categories[ ][ ]**.

3. Each of the questions can be stored in a struct called **question** containing the following members:
    - **category**, a string containing the category
    - **question**, a string containing the question
    - **answer**, a string containing the answer
    - **value**, the dollar value of the question
    - **answered**, a true/false value if the question has been answered

4. The following functions would be beneficial to have for interacting with questions.
    - **void initialize_game(void)** -- initializes the array of questions for the game
    - **void display_categories(void)** -- displays each of the remaining categories and question dollar values that have not been answered
    - **void display_question(char *category, int value)** -- displays the question for the category and dollar value
    - **bool valid_answer(char *category, int value, char *answer)** -- returns true if the answer is correct for the question for that category and dollar value
    - **bool already_answered(char *category, int value)** -- returns true if the question has already been answered

Players source files

1. The players source files should contain all of the data structures and functions related to players.

2. Each player could be stored in a struct called **player** containing the following members:
    - **name**, a string containing the player's name
    - **score**, the player's score/earnings

3. The following functions would be beneficial to have for interacting with players.
    - **bool player_exists(player *players, char *name)** -- returns true if the player name matches one of the existing players

- **void update_score(player *players, char *name, int score)** -- updates the score for that player given their name

Jeopardy source files

1. The jeopardy source files contain all the data structures and functions related to the game itself.

2. The source file should contain an array for storing the player structs and buffers for processing the command line input from the users.

3. The following functions would be beneficial to have for the game:
    - **void tokenize(char *input, char **tokens)** -- processes the answer from the user containing **what is** or **who is** and tokenizes it to retrieve the answer.
    - **void show_results(player *players)** -- displays the game results for each player, their name and final score, ranked from first to last place