Assignments

Assignment 5 - Returned

Honor Pledge Accepted
Draft - In progress
Submitted
Returned

Assignment Details

Title	Assignment 5
Student	Ramtin Sirjani
Submitted Date	Nov 29, 2022 11:23 PM
Grade	**100.00 (max 100.00)**
History	Nov 29, 2022 11:23 PM EST Ramtin Sirjani (rsirjani) saved draft Nov 29, 2022 11:23 PM EST Ramtin Sirjani (rsirjani) submitted

Instructions

Assignment overview

The 2211 World Cup is fast approaching. Your work on building worldCupDB.c (Assignment 4) was a great start but it's time to build version 2.0. The old version was missing a few important features:

- You need to keep track of an arbitrary number of teams, not just the 32 who qualified for the tournament
- There is no way to delete teams
- You need to keep a database of players as well
- The program is a little large. It's time to break it up into individual files

There is very little code from Assignment 4 which will be applicable for Assignment 5. You should consider this to be a fresh assignment. Don't try to back stitch Assignment 5 into your existing Assignment 4.

Purpose

This assignment will ask you to apply the following concepts from class:

- Basic programming concepts
- Intermediate concepts added in Assignment 4
- Linked Lists and memory management
- Makefiles
- Program organization

Design

As before, each election team will be stored as a structure with the following attributes:

- Team code (eg. 0, 1, 2, 3, 4, etc.)
  - Each team code must be unique. There is no limit to the number of teams but each code will be greater than or equal to 0
- Team name (eg. "Australia", "Cameroon", "Canada", "Argentina", "Belgium", etc.)
  - Only team names up to 25 characters in length are acceptable (including the null character - So think of this as 24+1). Each team name must be unique
- Group seeding (eg. "A1", "B3", "F2", etc.)
  - Only groups A-H and seeds 1-4 are acceptable. (So only A1, A2, A3, A4, B1, B2, ..., H3, H4). Each group seeding must be unique
- Primary kit (uniform) colours (eg. "Red", "Orange", "Yellow", "Green", "Blue", "Indigo", and "Violet")

- Only the values "R", "O", "Y", "G", "B", "I", "V" are acceptable.

Now, you will *also* need to store each team's player as a structure with the following attributes:

- Player code (eg. 0, 1, 2, 3, 4, etc.)
  - Each player code must be unique. There is no limit to the number of players but each code will be greater than or equal to 0
- Player name (eg. "John Smith", "Jane Doe", etc.)
  - Only names up to 50 characters in length are acceptable (including the null character - So think of this as 49+1)
- Player age (eg. 17, 18, 19, 20, etc.)
  - Only integers 17-99 are acceptable
- Professional club affiliation (eg. "Manchester City", "Los Angeles FC", "Toronto FC", "AFC Ajax", etc.)
  - Only names up to 50 characters in length are acceptable (including the null character - So think of this as 49+1)

Your program will then use a linked list of structures to represent all teams and a linked list of structures to represent all players.

Implementation

Your program should continuously prompt the user for one of four possible commands:

1. Print help (using command `h`)
   - Print a simple message or messages describing how to use the program.
2. Quit (using command `q`)
   - Quit the program. Yes, all data is lost when quitting your program. You do not need to maintain the data across multiple runs.
3. Control teams (using command `t`)
   - See below
4. Control players (using command `p`)
   - See below

Teams

As in worldCupDB.c (Assignment 4) all the same commands exist as before:

1. Insert a new team (using command `i`)
   - Prompt the user for the team code
     - Assume the user will enter one integer
     - This must be unique in your database and cannot conflict with an existing team code. If there is a conflict with an existing code, or if the database is full (you cannot allocate any more memory), tell the user the error. The user can try again or you can return the user to the main prompt
   - Prompt the user for the name of the team
     - Assume the user will enter a string of characters of any length
     - If the team name is longer than the acceptable length, you should accept as many characters as you can and ignore any additional characters. If there is any other issue, tell the user the error. The user can try again or you can return the user to the main prompt
   - Prompt the user for the group seeding of the team
     - Assume the user will enter two characters: a letter representing the group, and a number representing the seeding
     - If the letter is not A-H or if the number is not 1-4, tell the user the error. The user can try again or you can return the user to the main prompt
   - Prompt the user for the area of the primary kit (uniform) colour
     - Assume the user will enter one character value
     - If the character is not in the list "R", "O", "Y", "G", "B", "I", or "V", tell the user the error. The user can try again or you can return the user to the main prompt
2. Search for an team in the database and print it out (using command `s`)
   - Prompt the user for the team code
     - If the team code is found, print out all the values for this team only (see the print command below for more details)

    - If the team code is not found, tell the user the error. The user can try again or you can return the user to the main prompt

3. Update a team in the database (using command `u`)
    - Prompt the user for the team code
        - If the team code is found, prompt the user to update all the values for the team (see the insert command above for details)
        - If the team code is not found, tell the user the error. The user can try again or you can return the user to the main prompt
4. Print the entire list of teams (using command `p`)
    - Print out a table listing all the teams in your database with all the attributes:
        - Team Code
        - Team Name
        - Group Seeding
        - Primary Kit Colour

You should implement one additional command:

1. Delete team (using command `d`)
    - Prompt the user for the team code
        - If the team code is found, delete the team safely by removing the element from the linked list and `free`ing the memory
        - If the team code is not found, tell the user the error. The user can try again or you can return the user to the main prompt

Players

Implement the following commands:

1. Insert a new player (using command `i`)
    - Prompt the user for the player code
        - Assume the user will enter one integer
        - This must be unique in your database and cannot conflict with an existing player code. If there is a conflict with an existing code, or if the database is full (you cannot allocate any more memory), tell the user the error. The user can try again or you can return the user to the main prompt
    - Prompt the user for the name of the player
        - Assume the user will enter a string of characters of any length
        - If the player name is longer than the acceptable length, you should accept as many characters as you can and ignore any additional characters. If there is any other issue, tell the user the error. The user can try again or you can return the user to the main prompt
    - Prompt the user for the age of the player
        - Assume the user will enter one integer
        - If the number is less than 17 or greater than 99, tell the user the error. The user can try again or you can return the user to the main prompt
    - Prompt the user for the professional club affiliation of the player
        - Assume the user will enter a string of characters of any length
        - If the professional club affiliation name is longer than the acceptable length, you should accept as many characters as you can and ignore any additional characters. If there is any other issue, tell the user the error. The user can try again or you can return the user to the main prompt
2. Search for a player in the database and print it out (using command `s`)
    - Prompt the user for the player code
        - If the player code is found, print out all the values for this player only (see the print command below for more details)
        - If the player code is not found, tell the user the error. The user can try again or you can return the user to the main prompt
3. Update a player in the database (using command `u`)
    - Prompt the user for the player code

        - If the player code is found, prompt the user to update all the values for the player (see the insert command above for details)
        - If the player code is not found, tell the user the error. The user can try again or you can return the user to the main prompt
4. Print the entire list of players (using command `p`)
    - Print out a table listing all the players in your database with all the attributes:
        - Player Code
        - Player Name
        - Age
        - Club
5. Delete player (using command `d`)
    - Prompt the user for the player code
        - If the player code is found, erase the player safely by removing the element from the linked list and `free`ing the memory
        - If the player code is not found, tell the user the error. The user can try again or you can return the user to the main prompt

Other implementation notes

- You are welcome to create any number of helper functions you wish
- You are welcome to use any C libraries you wish but you should be able to get by with `stdio.h`, `stdlib.h`, and `string.h`
- You are welcome to use whatever wording you would like for your prompts. It does not have to precisely match the example below.

Sample output

Refer to Assignment 4 for a sample output. Apply the same principles for the player portion of your database.

Program structure

Your program should consist of the following structure:

```
251xxxxxx-Assignment5
   |-------- worldCupDB.h          <--- Header containing macros and definitions for worldCupDB.c
   |-------- worldCupDB.c          <--- The main program
   |-------- worldcup_team.h       <--- Header containing macros and definitions for teams
   |-------- worldcup_team.c       <--- The program containing all the functions used to manage teams
   |-------- worldcup_player.h     <--- Header containing macros and definitions for players
   |-------- worldcup_player.c     <--- The program containing all the functions used to manage players
   |-------- Makefile              <--- A Makefile allowing the user to compile all or part of your program
```

See also the attached diagram.

Testing and submitting

You should test your program by compiling and running it on Gaul before submitting (This is how the TA will be testing your program). Capture the screen of your testing by using screenshots. Demonstrate your program works with at least two runs using different input.

Place your program into a `251xxxxxx-Assignment5` folder.

Create a single tarball called `251xxxxxx-Assignment5.tar` containing a directory structure that looks like this:

```
251xxxxxx-Assignment5
   |-------- worldCupDB.h
   |-------- worldCupDB.c
   |-------- worldcup_team.h
   |-------- worldcup_team.c
```

```
    |-------- worldcup_player.h
    |-------- worldcup_player.c
    |-------- Makefile
    |-------- worldCupDB.png
```

Download your tarball and upload it to Assignment 5 in OWL.

This is easy!

For a 10% bonus, include a fifth member of the player structure called `team_code` and use the insert function to prompt the user for this data. Include a sixth command to the team menu called `r` to print all the players registered on the team the user chooses. For example, if player 23 is registered in team code 12, then I know that player 23 is a player for team 12. Therefore, if I print the registrants of team 12, player 23 should show in the list. Note the following rules:

- The team code entered by the user must already exist as a valid team in your database. In other words, a player cannot be a member of a team which doesn't exist yet.
- It is not possible to remove a team that still has players registered for it. One must either re-assign or remove all players first, then the team could be removed.

I need help

Check to see if your question has already been posted in the forums. If not, post your question.

https://owl.uwo.ca/x/yvBGmc

Additional resources for assignment

- Assignment5-worldcup-diagram.png (12 KB; Aug 15, 2022 9:56 pm)

Submitted Attachments

- 250680632-Assignment5.tar (187 KB; Nov 29, 2022 11:23 pm)

Assignment 5 - C Programming (Structures, Memory allocation) Unix (Makefiles and file organization)

Program 1 - Organization and Style

Instructions followed. Properly documented and commented. Useful naming conventions. Appropriate spacing.

Inadequate or Incomplete

Did not follow instructions at all. Commenting is inadequate or missing. Naming is consistently inadequate. Spacing and program layout is difficult to follow.

0 Points

Poor

Did not follow many key instructions. Commenting is generally inadequate. Naming is generally inadequate. Spacing and program layout is difficult to follow.

3.5 Points

Fair

Followed most instructions. Commenting is generally okay. Naming is generally okay. Spacing and program layout is generally okay.

5 Points

Good

Followed most instructions. Commenting is generally good. Naming is generally good. Spacing and program layout is generally good.

7.5 Points

Exceptional

Followed instructions. Commenting is exceptional. Naming is exceptional. Spacing and program layout is exceptional.

10 Points

Comment for Program 1 - Organization and Style

Done

0 0

Program 1 - Functionality

Program compiles. Program does what is asked. Program is not buggy.

Inadequate or Incomplete

Program does not compile or compiles with many adjustments. Program is very buggy and/or rarely produces correct results. Or, no submission received.

0 Points

Poor

Program does compile or compiles with minimal adjustments. Program is very buggy and/or rarely produces correct results.

15 Points

Fair

Program does compile. Program has only few bugs and usually produces correct results.

30 Points

Good

Program does compile. Program has no obvious bugs and usually produces correct results.

45 Points

Exceptional

Program does compile. Program has no bugs and consistently produces correct results.

60 Points

Comment for Program 1 - Functionality

Done

0 0

Makefile - Organization and Style

Instructions followed. Properly documented and commented. Useful naming conventions. Appropriate spacing.

Inadequate or Incomplete

Did not follow instructions at all. Commenting is inadequate or missing. Naming is consistently inadequate. Spacing and program layout is difficult to follow.

0 Points

Poor

Did not follow many key instructions. Commenting is generally inadequate. Naming is generally inadequate. Spacing and program layout is difficult to follow.

1.25 Points

Fair

Followed most instructions. Commenting is generally okay. Naming is generally okay. Spacing and program layout is generally okay.

2.5 Points

Good

Followed most instructions. Commenting is generally good. Naming is generally good. Spacing and program layout is generally good.

3.75 Points

Exceptional

Followed instructions. Commenting is exceptional. Naming is exceptional. Spacing and program layout is exceptional.

5 Points

Comment for Makefile - Organization and Style

Done

0 0

Makefile - Functionality

Program does what is asked. Program is not buggy.

Inadequate or Incomplete

Program does not run or runs with many adjustments. Program is very buggy and/or rarely produces correct results. Or, no submission received.

0 Points

Poor

Program does run or runs with minimal adjustments. Program is very buggy and/or rarely produces correct results.

6.25 Points

Fair

Program runs. Program has only few bugs and usually produces correct results.

12.5 Points

Good

Program runs. Program has no obvious bugs and usually produces correct results.

18.75 Points

Exceptional

Program runs. Program has no bugs and consistently produces correct results.

25 Points

Comment for Makefile - Functionality

Done

0 0

Total: **0**

Back to list