Trivia Quiz

(Deadline of submission on/before April 11 8AM)

The Latin aphorism *"Scientia potentia est"* means "Knowledge is power" in English. This phrase is often credited to Sir Francis Bacon in his work *Meditationes Sacrae* written in 1597 and is now a renowned phrase to appreciate one's knowledge. Contrary to the in-depth knowledge of a single subject, general knowledge is the understanding of a variety of topics.

A person's knowledge base can be widened by having general knowledge, which can benefit people in both their academic and personal lives. In addition to enhancing their overall development, it will make them smarter in every aspect of life. Knowing the answers to trivia and general knowledge tests is only one aspect of how important general knowledge is. A person's personality can be assessed based on his current general knowledge of a variety of topics and current events in numerous fields, which makes him an intelligent person with whom everyone wants to communicate and who they would also like to seek advice from. All these apply to almost everyone from beginners to professionals to business owners to homemakers to retirees.

To offer a venue for people to test their general knowledge, you are to create a Quiz game as your Machine Project. This project will require you to design a system that will make use of the new programming concepts discussed in class while also reviewing and/or widening your own general knowledge.

MAIN MENU

Your Quiz Game should display a **MAIN MENU** with a user-friendly (text-based) interface where the user chooses whether to ***"Manage Data"*** as an *admin*, ***"Play"*** the game as a *player*, or ***"Exit"*** the program.

MANAGE DATA

In order to have enough collection of question items for the game, you are to manage ***records*** that store seven (7) information needed which consists of a *topic, question number, question, 3 multiple choices,* and *the correct answer.* Below is an example record:

Topic	Geography	one word (20)
Question Number	1	
Question	Which country features a shipwreck on its national flag?	multiple words (150)
Choice 1	Vanuatu	one word (30)
Choice 2	Bermuda	one word (30)
Choice 3	Croatia	one word (30)
Answer	Bermuda	one word (30)

Your designed program should allow the *admin* to **MANAGE DATA** by providing the functionality to do the following operations:

- **Ask for a "password"**

  After choosing the option MANAGE DATA, the program should first ask the admin password in order to limit access to the collection of records. You have to set ONE admin password. Just like a usual password input, you should display only asterisks (*) in place of each character in the password. In case of an incorrect password input, your program should be able to ask for password again and an option to go back to **MAIN MENU**. Once logged in, a user-friendly interface should be available for other operations *(listed below)*.

- **Add a record**

  In adding a record, your program should ask first for a *question & answer* and check if the pair is already existing in the records. If yes, then display the record and a message saying it is already listed. If not, then your program should display the given *question & answer* and ask for the remaining input - *topic and the 3 choices*. ***Take note*** that the *question number* should automatically be assigned after getting the last number of questions from the given topic. After successful record addition, go back to the **MANAGE DATA** menu.

- **Edit a record**

  To edit a record, your program should display all the [UNIQUE] available *topics* first then the admin can choose any from there. After choosing a topic, all the *questions* in that *topic* should be displayed plus the *question number* that the admin can choose to edit that specific record. Note that after choosing which record to edit, your program should also ask which specific field is to be modified - *topic, question, 1-2-3 choices,* or *answer*. It will also help if you can display the current record as a guide. Once editing is done, go back to displaying all the topics which should have an option to go back to the **MANAGE DATA** menu.

- **Delete a record**

  In the Delete option, same as in Edit, your program should display available [UNIQUE] *topics* first, then list all the *question numbers* and *questions* on that specific topic. Your program should now ask the admin which question number to delete, but before actually deleting that record, a confirmation message should be displayed first. After successful deletion, the admin may opt to delete another record or go back to the **MANAGE DATA** menu.

- **Import data**

  In the import option, your program should be able to read a list of entries in a given text file. The admin should first specify the filename of the text file to load. If not found, display a message then ask again for input and provide an option to go back to the MANAGE DATA menu.

  The contents of the text file should look like the given one below. Make sure to follow this specific format. Those that are in <> are supposed to be replaced with the character '\n'. No <> will be saved in the text file. No additional content should be stored in the text file.

  Geography**<next line>**

  1**<next line>**

  Which country features a shipwreck on its national Rag?**<next line>**

  Vanuatu**<next line>**

  Bermuda**<next line>**

  Croatia**<next line>**

  Bermuda**<next line>**

<next line>
Technology**<next line>**
1**<next line>**
What is the part of a database that holds only one type of information?**<next line>**
Report**<next line>**
File**<next line>**
Field**<next line>**
Field**<next line>**
<next line>
<end of file>

- **Export data**
  Aside from reading data from a text file, your program should also allow writing into one. In the export option, admin should be able to write the filename of the exported text file. Filenames can have up to 30 characters - already including the file extension. If the given filename already exists in your computer, then it will be overwritten. Format of the data in the text file should be the same as given above in the Import option. The data in this exported file can be used in the import option.

- **Go back to MAIN MENU**
  Lastly, this go back option will lead the user back to the **MAIN MENU**. Only those exported to files are expected to be saved. The information in the lists should be cleared after this option. If the user wants to access the **MANAGE DATA** functionalities again, password should be asked again.

QUIZ GAME

After choosing the option **PLAY**, the player will be given three options - Play, View Scores, and Exit. A ***score.txt*** file should automatically load after entering this option so View Scores should already be able to display scores. File format should look like below:

Ben**<next line>**
7**<next line>**
<next line>
Jun**<next line>**
5**<next line>**
<next line>
<end of file>

- **Play**
  For the play option, your program should ask for the player's name first. During every turn, the player can choose from the available topics and your program should display any random question from that topic. User will then answer the question. If correct, add a score. If not, display a sorry message and ask for another topic to generate a question again. Scores should be visible all throughout the game. Players should also have an *'end game'* option every turn. If the end game option is chosen, display a message together with the final accumulated score then go back to the menu.

- **View Scores**
  In view ranking option, the available scores (both from file and currently accumulated score) should be displayed with a row number, player name, and the score.

- **Exit**
  This Exit will lead the player back to the **MAIN MENU**. Make sure to save the updated scores to the ***score.txt*** file before exiting. Only those data exported to files are expected to be saved. The information in the lists should be cleared after this option.

The program only terminates when the user chooses to **Exit** from the **MAIN MENU**.

YOUR TASK

Implement all the requirements as described above. You may change or design a different approach in some parts you deem useful or appropriate as long as you can fully support and explain your decisions. For your code versioning, you are expected to use git which is widely used to track changes in the source code and the service of GitHub. Note that only pulling copy and pushing changes using git will be explained, other functionalities such as branching will not be covered.

Bonus

A **maximum** of **10 points** may be given for features **over & above** the requirements, like (1) sorting the scores from high to low both on display and on file; (2) limit the question generated from the records to those not yet answered in the game; or other features not conflicting with the given requirements or changing the requirements) subject to **evaluation** of the teacher. **Required features** must be **completed first** before bonus features are credited. Note that use of conio.h, or other advanced C commands/statements may **not** necessarily merit bonuses.

Submission & Demo

Final MP Deadline: April 11, 2023 (M), 0800 via Canvas. After the indicated time, no more submissions will be accepted and grade will automatically be 0.

Requirements: Complete Program

- Make sure that your implementation has considerable and proper use of arrays, strings, structures, files, and user-defined functions, as appropriate, even if it is not strictly indicated.
- It is expected that each feature is supported by at least one function that you implemented. Some of the functions may be reused (meaning you can just call functions you already implemented [in support] for other features. There can be more than one function to perform tasks in a required feature.
- Debugging and testing was performed exhaustively. The program submitted has
  a. NO syntax errors
  b. NO warnings - make sure to activate -Wall (show all warnings compiler option) and that C99 standard is used in the codes
  c. NO logical errors -- based on the test cases that the program was subjected to

Important Notes:

1. Use **gcc -Wall** to compile your C program. Make sure you **test** your program completely (compiling & running).
2. Do not use brute force. Use **appropriate conditional** statements **properly**. Use, **wherever appropriate**, **appropriate loops** & **functions properly**.
3. You **may** use topics outside the scope of the subject but this will be **self-study**. Goto *label*, exit(), break (except in switch), continue, global variables, calling main() are **not allowed**.
4. Include **internal documentation** (comments) in your program.
5. The following is a checklist of the deliverables:

Legend:

Checklist:

Upload via AnimoSpace submission:

- source code*
- test script**
- sample text file exported from your program

email the softcopies of all requirements as attachments to **YOUR** own email addres

* Source code exhibits readability with supporting inline documentation (not just comments before the start of every function) and follows a coding style that is similar to those seen in the course notes and in the class discussions. The first page of the source code should have the following declaration (in comment):

/**
This is to certify that this project is my own work, based on my personal efforts in studying and applying the concepts learned. I have constructed the functions and their respective algorithms and corresponding code by myself. The program was run, tested, and debugged by my own efforts. I further certify that I have not copied in part or whole or otherwise plagiarized the work of other students and/or persons.

<your full name>, Your ID# <number>
**/

Example coding convention and comments before the function would look like this:

```
/* funcA returns the number of capital letters that are changed to small letters
   @param strWord - string containing only 1 word
   @param pCount - the address where the number of modifications from capital to small are
                   placed
   @return 1 if there is at least 1 modification and returns 0 if no modifications
   Pre-condition: strWord only contains letters in the alphabet
*/
int     //function return type is in a separate line from the
funcA(char strWord[20] ,    //preferred to have 1 param per line
      int * pCount)            //use of prefix for variable identifiers
{ //open brace is at the beginning of the new line, aligned with the matching close brace
   int    ctr;        /* declaration of all variables before the start of any statements
   –
                        not inserted in the middle or in loop- to promote readability */

   *pCount = 0;
   for (ctr = 0; ctr < strlen(strWord); ctr++) /*use of post increment, instead of pre-
                                                 increment */
   { //open brace is at the new line, not at the end
      if (strWord[ctr] >= 'A' && strWord[ctr] <= 'Z')
      {    strWord[ctr] = strWord[ctr] + 32;
           (*pCount)++;
      }
      printf("%c", strWord[ctr]);
   }

   if (*pCount > 0)
      return 1;
   return 0;
}
```

Test Script should be in a table format. There should be at least 3 categories (as indicated in the description) of test cases **per function. There is no need to test functions which are only for screen design (i.e., no computations/processing; just printf).

Sample is shown below.

Function	#	Description	Sample Input Data	Expected Output	Actual Output	P/F
sortIncreasing	1	Integers in array are in increasing order already	aData contains: 1 3 7 8 10 15 32 33 37 53	aData contains: 1 3 7 8 10 15 32 33 37 53	aData contains: 1 3 7 8 10 15 32 33 37 53	P
	2	Integers in array are in decreasing order	aData contains: 53 37 33 32 15 10 8 7 3 1	aData contains: 1 3 7 8 10 15 32 33 37 53	aData contains: 1 3 7 8 10 15 32 33 37 53	P
	3	Integers in array are combination of positive and negative numbers and in no particular sequence	aData contains: 57 30 -4 6 -5 -33 -96 0 82 -1	aData contains: -96 -33 -5 -4 -1 0 6 30 57 82	aData contains: -96 -33 -5 -4 -1 0 6 30 57 82	P

Test descriptions are supposed to be unique and should indicate classes/groups of test cases on what is being tested. Given the sample code in page 6, the following are four distinct classes of tests:

i.) testing with strWord containing all capital letters (or rephrased as "testing for at least 1 modification")
ii.) testing with strWord containing all small letters (or rephrased as "testing for no modification")
iii.) testing with strWord containing a mix of capital and small letters
iv.) testing when strWord is empty (contains empty string only)

The following test descriptions are **incorrectly formed**:

- Too specific: testing with strWord containing "HeLlo"
- Too general: testing if function can generate correct count OR testing if function correctly updates the strWord
- Not necessary -- since already defined in pre-condition: testing with strWord containing special symbols and numeric characters

6. Upload the softcopies via Submit Assignment in Canvas. You can submit multiple times prior to the deadline. However, only the last submission will be checked. Send also to your **mylasalle account** a **copy** of all deliverables.

7. Use **<LastnameFirstInitial>.c** & **<LastnameFirstInitial>.pdf** as your filenames for the source code and test script, respectively. Please make sure that your test script (pdf file) is readable and does not need zooming in to read. You are allowed to create your own modules (.h) if you wish, in which case just make sure that filenames are descriptive.

8. During the MP **demo**, the student is expected to appear on time, to answer questions, in relation to the output and to the implementation (source code), and/or to revise the program based on a given demo problem. **Failure to meet these requirements could result in a grade of 0 for the project.**

9. It should be noted that during the MP demo, it is expected that the program can be compiled successfully and will run. If the program does not run, the grade for the project is automatically 0. However, a running program with complete features may not necessarily get full credit, as implementation (i.e., code) and other submissions (e.g., non-violation of restrictions evident in code, test script, and internal documentation) will still be checked.

10. The MP should be an HONEST intellectual product of the student/s and should be worked on individually..

Any requirement not fully implemented or instruction not followed will merit deductions.

----------There is only 1 deadline for this project: April 11 0800 AM, but the following are **suggested** targets.----------
*Note that each milestone assumes fully debugged and tested code/function, code written following coding convention and included internal documentation, documented tests in test script.

1. Milestone 1: February 17
    a. Setup git and GitHub account
    b. Initial git push
    c. Menu options and transitions
    d. Preliminary outline of functions to be created
    e. Password entry
2. Milestone 2: March 3
    a. Add a Record
    b. Edit Record
    c. Delete Record

3. Milestone 3: March 17
    a. Accept player name
    b. Randomize question from certain topic
    c. Accumulate and display score after a game
4. Milestone 4: March 31
    a. Import
    b. Export
        ** both for question records and scores*
5. Milestone 5: April 7
    a. Integrated testing (as a whole, not per function/feature)
    b. Collect and verify versions of code and documents that will be uploaded
    c. Recheck MP specs for requirements and upload final MP
    d. [Optional] Implement bonus features