College of Computer Studies
De La Salle University
CCPROG2 Machine Project
Word Trivia

Trivia games are very popular, be it in the television or in computer applications. For your project, you will create a simple word trivia game. So, there are 3 options in the Main Menu: Game, Admin, Exit.

For the Game Phase of your MP, the user is shown a board consisting of letters. To win, the player must correctly answer at least 1 trivia question per row. The letter on the board is the starting letter of the answer to the trivia question. If the player is not able to answer all questions in an entire row, the game is over.

| | | | | |
|---|---|---|---|---|
| C | O | M | P | R |
| O | D | E | A | U |
| X | G | A | M | I |
| N | G | O | F | W |
| O | R | D | S | Z |
| Y | X | H | F | W |
| V | U | T | S | R |

Assuming the player chose the letters in the shaded blocks and answered the trivia correctly, since there's a location answered correctly per row, the player wins.

| | | | |
|---|---|---|---|
| C | O | L | I |
| **S** | **M** | **O** | **R** |
| E | H | I | R |

The letters in boldface signify that the player answered incorrectly on those blocks. Since, all questions in a row were incorrectly answered, the game ends with the player losing.

Note that there could be different configurations of the board, depending on the player's preference. The smallest board would be 3x3. The largest is 10x10. However, the matrix need not have the same number of rows and columns.

| | | |
|---|---|---|
| E | A | S |
| Y | R | O |
| U | N | D |

The highlighting and boldfonts in the above visualization is not required. In your game, those answered correctly by the user are marked with * (to replace the letter). Those that are incorrectly answered should be marked with – (to replace the letter). The user has to choose a letter from the first row (cannot skip or jump rows), when answered correctly, then he chooses a letter from the next row, and so on. The letter chosen by the user may not be the letter in the first column, he could choose something in the middle or even the last. As stated in the visualization above, once the user gives wrong answers for all letters in a row, he cannot proceed to the next row and the game ends with the player losing. When all rows have 1 correct answer each, the player wins. The flow of the Game Phase is discussed further in page 3.

In the Admin Phase, this serves as the interface and features that will allow the Admin or game master manage the collection of words and the trivia/clues that will be used in the game. It is important to note that each word (the answer) can be at most 20 letters. And there can be at most 10 trivia per word. The trivia/clue is represented by a `relation` and the `relation value` pair. The `relation` and `relation value` can each be a phrase and each can have at most 30 characters. For example, for the word `Table`, the following can be some trivia information:

| Relation | Relation Value |
|---|---|
| Kind of | Furniture |
| Part | Top |
| Part | Leg |
| Height | Meter |
| Make | Wood, Plastic |
| Color | Brown, White, Black |

Note that there can be at most 150 entries or words(serving as the answers in the game), but a word can have at most 10 trivia/clues.

As mentioned, your program should allow the game master to manipulate these entries. Thus, your program should provide a user-friendly interface, as well as the functionality to do the following:

- ✯ ***Add Word***
  Your program should allow addition of a new word by asking the user to input the word (which could actually be a phrase) and at most 10 trivia (or the `relation` and `relation type` pairs).

Note that you are not allowed to ask the user how many trivia he wants to input. Make sure that in adding a word, it should have at least 1 character and it does not exist yet (meaning it has to be unique). Also, in asking input for the trivia, both **`relation`** and **`relation type`** should each have at least 1 character.

✯ ***Add Trivia***
Your program should ask under what word you want to add a clue to. A trivia can only be added if the word exists and if the list of trivia for that word has not yet reached 10. You are not allowed to ask the user the number of trivia he wants to encode for that particular word. However, the program may ask the user if he wants to encode another trivia. Both **`relation`** and **`relation value`** should each have at least 1 character.

✯ ***Modify Entry***
Your program should allow your user to modify an entry. This option will first display the listing of all words in alphabetical order before asking which he wants to modify. The input for this is the word. If this word is found, then the user is repeated asked what is to be modified (either the word or one of the clues) until the user chooses to end the modification of this entry. If the user chooses to modify the word, the new word should not exist in the list yet for it to be acceptable as a new word. If the choice was to modify the clue, first show all the clues for this word, Then, ask the user to input the number of which of the clues will be modified. The number should be a valid number (starting from 1 until n, where n is equivalent to the number of clues for this entry) before modification of either OR both the relation and relation value is modified. If the word is not found, a message is displayed and the program reverts back to asking for a new option under the maintenance phase. [Note that storing data in arrays should not have skipped entries, for example index 0 should not be null or contain garbage.]

✯ ***Delete Word***
Your program should allow your user to delete an entire word. Only an existing word may be deleted. Provide your user with a listing of all words in alphabetical order before asking which he wants to delete. Note that in deleting a word, all trivia under it will automatically be removed.

✯ ***Delete Clue***
Your program should allow your user to delete a clue. Show the list of existing words in alphabetical order to the user. From there, the user should input the word from which a clue would be deleted. Then, from the input word (if it is existing), provide your user with a listing the word's trivia list (which are the **`relation`** and **`relation value`**). Make sure that garbage values are not displayed (i.e., do not display the 7th, 8th, 9th, and 10th relation and relation values if there are only 6 entries). Ask the user to identify which clue he wants to be deleted by indicating a number. Make sure that the input number is valid (before you delete). A valid number is from 1 to 10 only and it should be an initialized entry (meaning, the user cannot choose 7 if there are only 6 trivia under the chosen word). Note that the first clue should be stored in index 0, but is referred as clue number 1 by the user.

✯ ***View Words***
Provide a listing of all words (in alphabetical order). The list of trivia of each word should also be shown. Display each word one at a time until all entries have been displayed. Provide a way for the user to view the next or previous word or exit the view (i.e., press 'N' for next, 'P' for previous, 'X' to end the display and go back to the menu).

✯ ***View Clues***
Show a listing of all words before asking your user to type the word whose clues he wants to view. The list of **`relation`**s and **`relation value`**s of the chosen word should be shown. Make sure that garbage values are not displayed. Using the example table above, this is view:

```
Object: Table
   Kind of: Furniture
   Part: Top
```

```
Part: Leg
Height: Meter
Make: Wood, Plastic
Color: Brown, White, Black
```

✯ ***Export***

Your program should allow all data to be saved into a text file. The data stored in the text file can be used later on. The system should allow the user to specify the filename. Filenames have at most 30 characters including the extension. If the file exists, the data will be overwritten.

This is a sample content of the text file. Make sure to follow the format.

```
Object: Table<next line>
Kind of: Furniture<next line>
Part: Top<next line>
Part: Leg<next line>
Height: Meter<next line>
Make: Wood, Plastic<next line>
Color: Brown, White, Black<next line>
<nextline>
Object: Picture Frame<nextline>
Kind of: Home Decor<nextline>
Make: Wood, Aluminum, Plastic, Glass<nextline>
Color: Assorted<nextline>
Height: Inch<nextline>
Width: Inch<nextline>
Contains: Picture<nextline>
<nextline>
<end of file>
```

✯ ***Import***

Your program should allow the data stored in the text file to be added to the list in the program. The user should be able to specify which file (input filename) to load. Note that if a word to be loaded already exists in the list, the system has to prompt the user if he wants the existing data (for that word) in your program to be overwritten. If the user said yes, your program should replace the existing word (including the trivia) with the one from the text file. On the other hand, if the user said no, your program retains the entries (for that word) in the program. If the word to be loaded is unique (does not exist yet in the list), it will be added to your array. The data in the text file is assumed to be in the format indicated in Export.

✯ ***Back to Main Menu***

The exit option will just allow the user to quit the Admin Phase. Only those exported to files are expected to be saved. The information in the lists should be cleared after this option.

## Game Phase

If the user chooses Game Phase from the Main menu, the program then asks the user what dimension the board (of letters) will be. Proceed by loading the text file of words and trivia (ask the user for the filename). From this, generate the letters to appear in the board. Make sure you have enough entries in the database for the grid. Meaning, if you only have 12 entries in the database, then you can have at most a 3x4, 4x3, 5x2, 2x5, 6x2, or 2x6 board, since the answers in the grid should be unique words. Note also that in a row, the letters should be unique. Once the board is initialized, the user can start to play.

The player starts the game by choosing which letter from the topmost row he wants. After which, the program randomly chooses and shows one trivia (relation and relation value) to the player. The player is then supposed to give the answer. If the answer is wrong, the player should choose another letter from the same row. If the answer is correct, the player now chooses another letter from the next row. The trivia that corresponds to the word answer is shown to the player and the program waits for the player to input his answer. This continues on until the game ends. The game ends when all letters in a row are answered incorrectly. Here the player loses. The player can also choose to exit the current game (anytime within the game). In which case, since the player has not reached the end, he loses as well. Once the player correctly answered the trivia in the last row, the player wins.

After the current game ends (whether the player won, lost, or chose to exit), the program should ask if the player wants to play another game. If the player chooses yes, then the player is again asked what dimension of board he wants and so on. If the player chooses no, the program reverts to the Main Menu (where the options are Admin Phase, Game Phase, and Exit). If in the Main Menu, the user chooses Admin Phase, the program reverts to show the features of Add Word, Add Trivia, etc. If the user chooses Game Phase, the program asks the user regarding the game board dimension, etc. If the user chooses to exit, the program ends.

## *Bonus*

A **maximum** of **10 points** may be given for features **over & above** the requirements (such as mouse control, 2-player game, timer, 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 1, 2024, 1200 noon via AnimoSpace**. No project will be accepted anymore after the submission link is locked and the grade is automatically 0.

  **Requirements:** Complete Program

  - Make sure that your implementation has considerable and proper use of arrays, 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** -std=C99 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 CCPROG2 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:

**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 address on or before the deadline

Legend:

***** Source code exhibit 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) [replace the pronouns as necessary if you are working with a partner]:

/*********************************************************************************************************
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>, DLSU 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 7, 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 AnimoSpace. You can submit multiple times prior to the deadline. However, only the last submission will be checked. Send also to your **DLSU GMail account** a **copy** of all deliverables.
7. 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 with 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 to 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 in the command prompt using gcc -Wall -std=C99. 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. For this project, you are allowed to do this individually or to be in a group of 2 members only. [Note that if you decide to work individually, you may still be sbujet to the Individual Demo Problem.] Should you decide to work in a group, the following mechanics apply:
**Individual Solution:** Even if it is a group project, each student is still required to create his/her INITIAL solution to the MP individually without discussing it with any other students. This will help ensure that each student went through the process of reading, understanding, solving the problem and testing the solution.
**Group Solution:** Once both students are done with their solution: they discuss and compare their respective solutions (ONLY within the group) -- note that learning with a peer is the objective here -- to see a possibly different or better way of solving a problem. They then come up with their group's final solution -- which may be the solution of one of the students, or an improvement over both solutions. Only the group's final solution, with internal documentation (part of comment) indicating whose code was used or was it an improved version of both solutions) will be submitted as part of the final deliverables. It is the group solution that will be checked/assessed/graded. Thus, only 1 final set of deliverables should be uploaded by one of the members in the AnimoSpace submission page. [Prior to submission, make sure to indicate the members in the group by JOINing the same group number.]
**Individual Demo Problem:** As each is expected to have solved the MP requirements individually prior to coming up with the final submission, both members should know all parts of the final code to allow each to INDIVIDUALLY complete the demo problem within a limited amount of time (to be announced nearer the demo schedule). This demo problem is given only on the day/time of the demo, and may be different per member of the group. Both students should be present during the demo, not just to present their individual demo problem solution, but also to answer questions pertaining to their group submission.
**Grading:** the MP grade will be the same for both students -- UNLESS there is a compelling and glaring reason as to why one student should get a different grade from the other -- for example, one student cannot answer questions properly

OR do not know where or how to modify the code to solve the demo problem (in which case deductions may be applied or a 0 grade may be given -- to be determined on a case-to-case basis).

11. Any form of **cheating** (**asking other people not in the same group for help**, **submitting as your [own group's] work part of other's work**, **sharing your [individual or group's] algorithm and/or code to other students not in the same group, etc.**) can be punishable by a grade of **0.0** for the **course** & a **discipline case**.

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

-----------There is only 1 deadline for this project: April 1, 12:00noon, 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 : Feb 16
    a. Determine if the player wins based on contents in the 2D array of characters.
    b. Determine if the game is over based on the contents in the 2D array of characters.
    c. Menu for the Admin Phase

2.) Milestone 2: March 1
    a. Initialize 2D array based on contents of array of words
    b. Display all words
    c. View Clues
    d. View Words
    e. Determine if word already exists in the array
    f. Add Word
    g. Add Trivia

3.) Milestone 3: March 15
    a. Modify Entry
    b. Delete Word
    c. Delete Clue
    d. Sort in alphabetical order

4.) Milestone 4: March 22
    a. Export
    b. Import

5.) Milestone 5: March 29
    a. Function Specs
    b. Integrated testing (as a whole, not per function/feature)
    c. Collect and verify versions of code and documents that will be uploaded
    d. Recheck MP specs for requirements and upload final MP
    e. [Optional] Implement bonus features