CMSC 202 Spring 2024

Project 1 – UMBC Fortune Wheel

Assignment: Project 1 – UMBC Fortune Wheel

Due Date: Tuesday, February 27th at 8:59pm on GL

Value: 80 points

1. Overview

In this project, you will:

- Write a program that calls multiple functions,
- Use simple file input/output, and
- Pass arrays to functions.

2. Background

Wheel of Fortune is an American television game show created by Merv Griffin. The show has aired continuously since January 1975. It features a competition in which contestants solve word puzzles, similar to those in Hangman, to win cash and prizes determined by spinning a giant carnival wheel. The current version of the series, which airs in nightly syndication, premiered on September 19, 1983. It stars Pat Sajak and Vanna White as hosts, who have hosted the nighttime version since its inception.

For this project, we are going to be creating a version of the Wheel of Fortune game where the computer will load a file that has both categories and puzzles. The computer will randomly choose one category/puzzle for the single player to solve. The players will then guess a letter. The program will check three things: 1. If the guess is a letter from the puzzle, 2. If the guess completes the puzzle, and 3. If the guess has not already been guessed. The game ends when all of the letters in the puzzle have been guessed. In this version, the player can never lose. Image 1 shows an example of the category (food & drink) and the puzzle (Fresh Tropical Fruit).

Image 1. Sample Wheel of Fortune Puzzle

3. Assignment Description

Your assignment is to develop the code to read a file with puzzle categories and puzzles. The program will randomly choose one of the puzzles for the player to complete. The player will guess letters which will either be in the puzzle or not. Letters can only be guessed once otherwise the player will be notified that the letter has already been guessed. The game ends when all of the letters in the puzzle have been guessed.

4. Starting Files

For project 1, there is one starting file, which is the list of categories and their corresponding puzzle named **`proj1_data.txt`**. It is your job to create the **`proj1.cpp`** file and all the functions required to make this project work. We understand that this is not a complete list of all words!

There is one sample run file named **`proj1_sample.txt`**. These files can be copied from my directory to yours by following these instructions.

First, navigate to your project 1 directory (Recommend that you use **`202/projects/proj1`** or something similar). Then run this command:

```
cp /afs/umbc.edu/users/j/d/jdixon/pub/cs202/proj1/* .
```

Once you have run this command, you should be able to type:

```
ls
```

and see the following:

```
proj1_data.txt and proj1_sample.txt .
```

5. Requirements:

This is a list of the requirements of this application. For this project, it is up to you exactly how you want to implement it. For you to earn all the points, however, you will need to meet all the defined requirements.

- You must follow the coding standard as defined in the CMSC 202 coding standards (found on Blackboard under course materials). This includes comments as required.
- The project must be turned in on time by the deadline listed above on GL. The project must run on GL.
- This assignment is completely individual. You may not work with anyone else on this project. Do not share your code with anyone who doesn't work for CMSC 202 at UMBC.
- The project must be completed in C++. You may not use any libraries or data structures that we have not learned in class. These are the only libraries that you are allowed to use in this project **`<iostream>`, `<ctime>`, `<string>`, `<fstream>`, `<iomanip>`, `<cmath>`, and `<cstdlib>`.** You should only use **`namespace std`**.
- You must use a variety of functions (main plus at least 5 additional functions) including passing parameters to those functions and returning information from those functions. At least one time, an array (of any size) must be passed to a function (although you may do this more than once). At least one-time, meaningful data must be returned from a function.
- All user input must be validated. However, the user is expected to always enter the correct data type. i.e., if the user is asked to enter an integer, they will. If they are asked to enter a character, they will. You do not need to worry about checking for correct data types.
- All words from the starting file must be loaded into an array.
- The starting category and puzzle are randomly chosen from the provided file.

- A letter can only be guessed once, otherwise the program indicates that it has already been guessed.
- The program should print the puzzle only when a guessed letter matches a letter in the answer. In other words, when the guessed letter is found in the answer, the updated puzzle should be displayed. However, if the guessed letter does not match any letter in the answer, the puzzle remains unchanged and is not printed.
- The game ends when all letters in the puzzle have been guessed.
- Specific coding requirements include:
    - The project must run as closely to the sample output here and the provided sample output file (**`proj1_sample.txt`**) as possible. Significant deviation from the sample output may result in a reduction in points.
    - Must use at least 5 different functions. (**`main`** + 5 additional functions)
    - Must pass at least one array to a function.
    - Must have at least one nested loop.
    - Must not use any global variables.
    - You must use global constants for the number of lines in the file (48), and the name of the input file (**`proj1_data.txt`**).
    - Must use input validation (assume the data is the correct type).
    - Must use at least one do..while loop.
    - No explicit pass-by-reference. No pointers. No vectors. No structs, classes, or objects. No recursion. No EOF usage.
    - No ChatGPT or other AI based code generator is allowed.
    - No warnings or errors when you turn it in.

6. Recommendations

You are free to implement this with your own functions. These are just some ideas for how to break up the project into smaller pieces. You will probably want to use at least three arrays: one to store the list of categories loaded in from the input file, one to store the list of puzzles, and one to store all characters guessed. You may want to use a multidimensional array to store the categories and the puzzles (that is up to you). While not required, these are some functions that you may want to include:

- Load File – This function opens the input file of categories and puzzles and loads them into an array. Start by loading the file of categories and puzzles into an array. You can use two arrays or a multidimensional array. Do not use EOF for this function.
- getAnswer – Randomly chooses a puzzle (and its corresponding category).
- inPuzzle – Checks to see if a guess is in the puzzle. There are many ways of doing this – including just keeping an array of letters guessed. See the hint below for another way.

Hint: For inPuzzle, one easy solution would be to create an array of integers the complete size of the ASCII table (256). Then, for each letter in the starting puzzle, you could increment the count for the letter in the corresponding location of the array. For example, if the letter was an 'A', then you would increment the array in location 65. Characters can convert to their decimal location very efficiently! You can use this method both for the puzzle and to track the guesses.

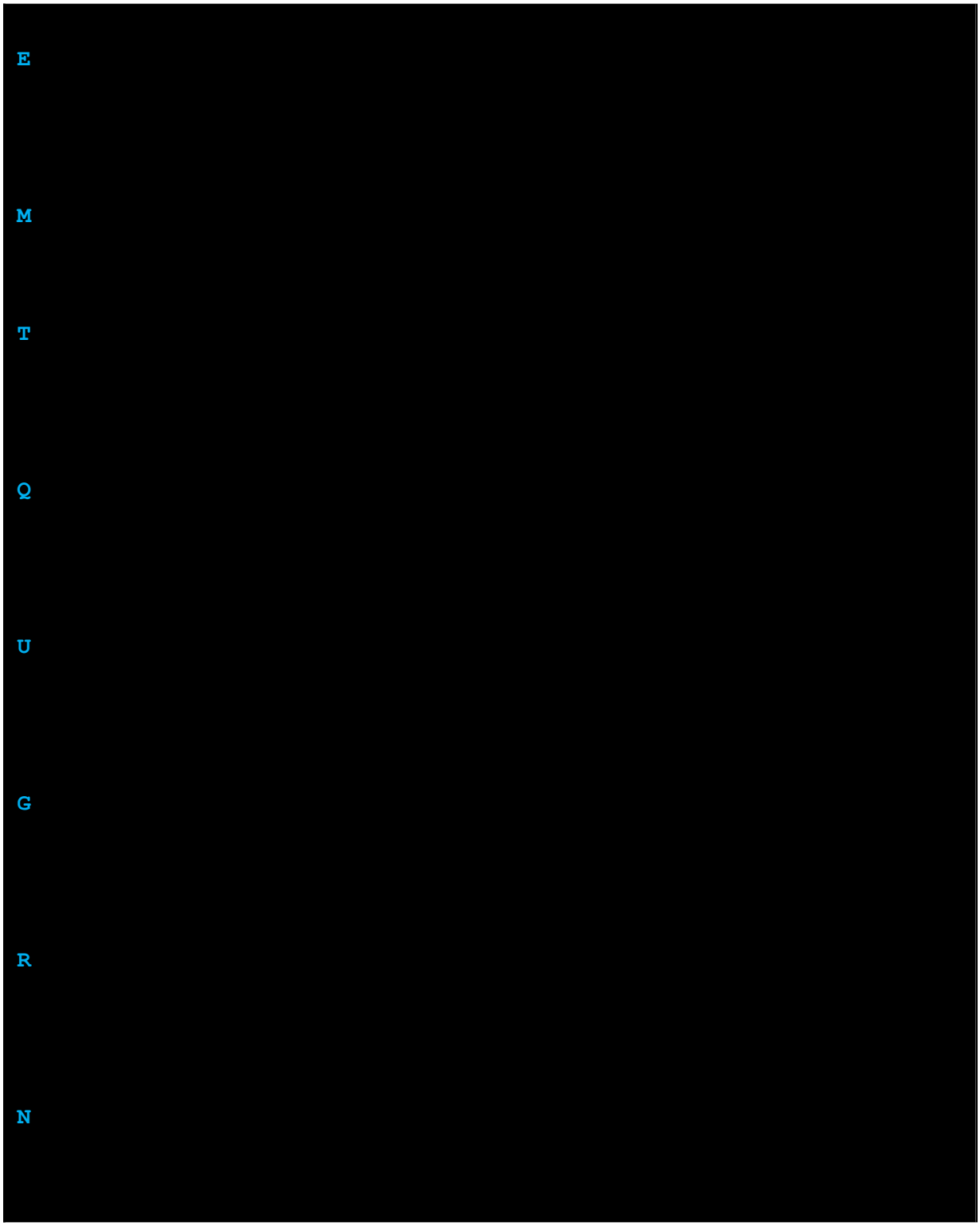
If you are using this method, don't forget to assign 256 as a global constant rather than using the literal.

- Get Input – Allows the player to guess a letter. All guesses should be in upper (or lower) case. Just be consistent.
- IsCompleted – Checks to see if all characters in the puzzle have been guessed. If they have, the program ends and the player has won.

- printPuzzle – Prints the puzzle. It uses _ if that letter hasn't been guessed yet. A puzzle with the solution of "CANDY BAR" would appear _____ ___

7. Sample Input and Output

In the sample output below, user input is colored blue for clarity. After compiling and running proj1, here is a sample run:


E
M
T
Q
U
G
R
N

```
S
```

Here is another run showing validation of entries:

```
Welcome to UMBC Fortune Wheel!
24 puzzles loaded.
Category is: Thing

_ ________ ________
What letter would you like to try?
A
1 A found in the puzzle.
A ________ ________
What letter would you like to try?
C
No C found in the puzzle.
What letter would you like to try?
D
2 D found in the puzzle.
A _____D__ _____D__
What letter would you like to try?
S
No S found in the puzzle.
What letter would you like to try?
T
No T found in the puzzle.
What letter would you like to try?
R
3 R found in the puzzle.
```

```
A _R___D__ R____D_R
What letter would you like to try?
E
3 E found in the puzzle.
A _R_E_D__ RE___DER
What letter would you like to try?
e
Enter an upper case letter only
What letter would you like to try?
P
No P found in the puzzle.
What letter would you like to try?
N
2 N found in the puzzle.
A _R_END__ RE__NDER
What letter would you like to try?
I
2 I found in the puzzle.
A _RIEND__ RE_INDER
What letter would you like to try?
M
1 M found in the puzzle.
A _RIEND__ REMINDER
What letter would you like to try?
F
1 F found in the puzzle.
A FRIEND__ REMINDER
What letter would you like to try?
L
1 L found in the puzzle.
A FRIENDL_ REMINDER
```

```
What letter would you like to try?
Y
1 Y found in the puzzle.
A FRIENDLY REMINDER
You won!
```

See the provided **`proj1_sample.txt`** file for additional examples.

8. Compiling and Running

To compile your program, enter the following command at the Linux prompt:

`g++ -Wall proj1.cpp -o proj1` (use this command to show warnings – which should be eliminated before turning your code in)

This command runs the GNU C++ compiler (**`g++`**). The option **`-Wall`** instructs the compiler to be verbose in its production of warning messages; the option **`-o proj1`** (hyphen followed by the letter "o", not the digit zero), instructs the compiler to give the executable program the name **`proj1`**. If the program compiles correctly, the executable file **`proj1`** will be created in the current directory. *Your project files should have no warnings or errors when turned in.*

At the Linux prompt, enter the command **`./proj1`** to run your program. It should look like the sample output provided above.

9. Completing your Project

When you have completed your project, you can copy it into the submission folder. You can copy your files into the submission folder as many times as you like (before the due date). We will only grade what is in your submission folder.

Test your code!

For this project, you should submit these files to the **`proj1`** subdirectory:

`proj1.cpp` — should include your implementations of the required functions.

After you have set up your symbolic link (as from lab 1), copy the project files into your proj1 folder. Execute these commands:

a. cd to your project 1 folder. An example might be:

```
cd ~/202/projects/proj1
```

b. `cp proj1.cpp ~/cs202proj/proj1`

You can check to make sure that your files were successfully copied over to the submission directory by entering the command:

```
ls ~/cs202proj/proj1
```

You can check that your program compiles and runs in the `proj1` directory, but please clean up any `.o` and executable files. Again, do not develop your code in this directory and you should not have the only copy of your program here.

IMPORTANT: If you want to submit the project late (after the due date), you will need to copy your files to the appropriate late folder. If you can no longer copy the files into the proj1 folder, it is because the due date has passed. You should be able to see your proj1 files, but you can no longer edit or copy the files in to your proj1 folder. (They will be read only)

- If it is 0-24 hours late, copy your files to `~/cs202proj/proj1-late1`
- If it is 24-48 hours late, copy your files to `~/cs202proj/proj1-late2`
- If it is after 48 hours late, it is too late to be submitted.