Programming Assignment #3

Due: Midnight Monday, April 18th, 2016

Points: 100 + 15 bonus

Worth: 30% of your programming assignments grade

Objectives

1. Implementing a provided interface.
2. Recursive programming techniques.
3. Data structures: linked lists.
4. Searching techniques: binary search.
5. Reading and writing files.

Problem Statement

WordGrid is a word-finding game similar to the Parker Brothers game Boggle, but for a single player.

Rules

The game board is a four-by-four grid of letters. Each position in the grid gets a randomly-chosen uppercase letter each game. Letters are chosen according to their frequency in English, so that 'E' is much more common than 'X'.

Letters in the grid may be used to spell words. A word is formed by choosing a sequence of letters from the grid, each adjacent to the previous letter horizontally, vertically, or diagonally. Squares can be re-used, but not twice in a row (because a square is not adjacent to itself). For example, the board in ***Figure 1*** contains the words "SIZE", "EFFECT", and "FIFES", as indicated by the letters and lines in red in ***Figure 2***. Note that "FIFES" re-uses the same "F" square for its first and third letters.

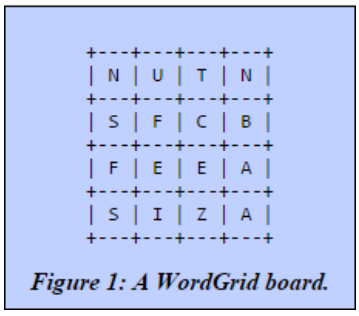

Figure 1: A WordGrid board.

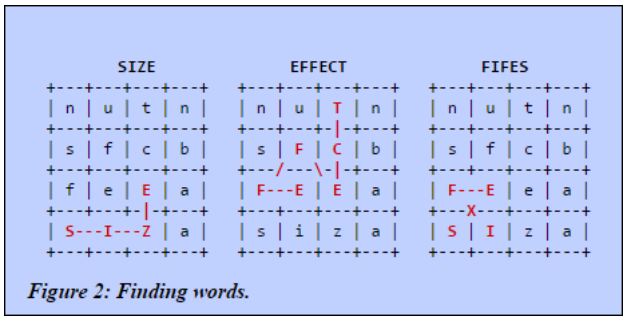

Figure 2: Finding words.

Every turn, the user enters a word, or 'q' by itself to quit the game. They may enter words with any capitalization: words are all converted to uppercase before searching. If the user's word is a real word, is found on the board, and hasn't already been entered by the user, they get n^2 (n squared) points, where n is the length of the word. Otherwise (if the word was not a real word, if it was not on the board, or if the

user already entered the word), the user receives a "miss". The game ends when the user has five misses or when he/she quits.

Before every turn, display to the user the board, their total score, their number of misses, and the list of words they have found so far. After every miss, tell the user why they missed (the word isn't a real word, the word wasn't found, or the word was already entered). At the end of the game, print the list of found words and the score.

Here is a sample session.

Letter Frequencies

When your program randomizes the board, it should select letters based on their frequencies in English. You can use the provided `random_letter` function to do so, or you can implement your own function.

You need to seed the random number generator at the beginning of `main`, which is given in the `main.cpp`.

Word List

Your program must be able to check whether a word that the user entered is a "real" word. The project includes a text file, `wordlist.txt`, with a fairly extensive list of words, one per line. This file is already sorted, so once you load it into an array or vector, you can use binary search to check whether a word is there. For the purposes of this program, the "real" words are precisely those listed in the file.

Implementation specifications

1. The board should be printed as in ***Figure 1***, with borders draw with '|', '-', and '+' characters.
2. Use binary search to check whether a word appears in the word list.
3. Use a recursive function to search for words in the grid. See the Hints section below.
4. Your program does not need to handle boards larger or smaller than 4 x 4 letters (but see the Bonus section).

Design specifications

Your implementation should include at least the following classes. You are free to add as many other classes or methods as you would like.

- `WordList`: a sorted list of words from a file. Contains a vector of strings. Methods:
  - The constructor or another method should read lines from the input file into the vector.
  - Check whether a word is in the list (using binary search).
  - (Private) Recursive helper function for binary search.
- `Board`: a 4 x 4 game board. Should contain a 2D array of characters. Methods:
  - The constructor or another method should randomize the board.
  - Print the board to the screen.

  - Check whether a word occurs on the board.
  - (Private) Recursive helper function for word search.
- `Game`: the current state of the game as a whole. Should contain a WordList, a Board, the score, the number of misses, and the list of already-used words. Methods:
  - Constructor.
  - Check whether the game is over.
  - Check whether a word is legal (in the word list, on the board, and not already used).
  - Take a turn: print the board/score/words, prompt for and read a word, and assign pointers or misses.
    You may split this into multiple methods if you prefer.
  - Print a game over message.

Documentation

Your source file should begin with a "header comment" with title (class the assignment number), your name, the date you created the file and made the last change, and a brief description of what the program does. For example,

```
/*************************************************************
 * Title:        CS 237 Programming Assignment #3
 * Author:       Your Name Here
 *               your.n.here@uwrf.edu
 * Created:      April 1st 2016
 * Modified:     April 15th 2016
 * Description:  Implementation of the WordList class
 * Team:         Team Mate
 *************************************************************/
```

Each function in your program, including main, should be preceded by a comment with the following components. See pause_237 for a detailed example.

- The function's name and a short (a few words) synopsis of what the function is used for. For example, "Compute Fibonacci numbers".
- Description: A longer (a few sentences) narrative description of what the function does.
- Inputs: The names, types, and meaning of each parameter. Also list any stream input (standard input or from a file) the function performs. For each input (whether from a parameter or stream), list any restrictions or assumptions you make about its value.
- Outputs: The type and meaning of your function's return value. Also list any stream output (standard output or file) the function performs, and any changes it makes to its call-by-reference parameters.
- Notes (optional): Any additional information you feel is relevant: Why the function exists, what role it plays in the entire program, hints for using the function, and so on.

In general, your internal documentation should be targeted towards someone who knows C++ but doesn't know your code. The reader should be able to figure out from your documentation exactly how to use your function **without having to read your code.**

What to submit

A skeleton project will be provided through D2L. Unzip the folder `pa3`.You can double-click the solution (`.sln`) file in this folder to load the code and project into Visual Studio. If you do not use Visual Studio, all the source code stubs are packaged in `pa3_CodeStubs.zip`. The main function is provided in `main.cpp`. You could do any change as you need. The class interfaces are given in the form of header files. You will need to create additional source files to implement every class. The `wordlist.txt` file is put in the Resource Files folder.

After you finish the assignment, zip the whole project and submit to the dropbox on D2L. When you zip your submission folder, you do not need to include your executable, debugging, or intermediate files (the `ipch` and `Debug` directions, and the `pa3.sdf` file). You can submit as many times as you want. I will grade the latest version before the due date.

Make sure your project is complete. Make sure that you can view, compile, and run your program with your submission version. If this doesn't work, I will not be able to grade your assignment.

Grading

A total of 100 points, plus 15 bonus points, are possible.

Points	Task
70	**Correctness**
5	Displays the contents of the board before each turn
5	Board is correctly formed
5	Displays the score, number of misses, and word list before each turn
5	Prompts for a word, converts to uppercase, quits on 'Q'
5	Randomizes the board.
5	Reads the word list from a file.
10	Check for the world in the word list using binary search.
15	Recursive function checks for the word on the board.
5	Checks for repeated words from the user.
5	Score and misses calculation is correct
5	Detects end of the game, prints the score and word list.
30	**Style and design**
5	WordList implemented as a class, appropriate members.
5	Board implemented as a class, appropriate members.
5	Game implemented as a class, appropriate members.
5	Each class has its own source and header file.
10	Code has proper documentation.
15	**Bonus**
5	No Repeats.
5	High Scores.
5	Big Boards.

Bonus

Note: you will not receive bonus points for any late, incomplete, or failing submission. Only attempt bonus points after you have finished the rest of the assignment. Please note at the top of your source file whether you are attempting bonus points.

No Repeats (5 points)

One way that WordList differs from Boggle is that it allows re-using the same square multiple times in one word. For five bonus points, implement both a normal search and "Boggle rules" search, which only allows using each square once per word. If the user's word was found using normal search but not using Boggle rules search, print a message warning the user, but do not assign any points or misses for the word.

In order to do a Boggle rules search, your grid search helper function (see Hints below) will need an additional parameter to hold the set of grid positions that have already been used in this word.

High Scores (5 bonus points)

For five bonus points, keep a high score list of the top 10 winners of the game. Keep the list sorted in order of score, with the highest-scoring game listed first. After a game, ask the user for their name, add the new entry to the appropriate place in the list, and save and print the updated top 10 scores.

You should save the high score list in a text file, so it persists from one run to another. Your program should work correctly if the score file does not exist. You can use a hardcoded name for the file; you don't need to ask the user for the filename.

Your program should have a class representing a score entry (name and points), and a class representing the score list. The score list class should have methods to create the high score file if it doesn't exist, read scores from the file, add a score to the list, write the updated score list back out to the file, and display all the scores in the list. You can use either a linked list or vector to implement the scores list.

Big Boards (5 bonus points)

The standard WordGrid board is four by four squares. For five bonus points, allow the user to play on a board of any board size 2 x 2 or larger. At the beginning of the program, ask the user for the width and height of the board as integers. You should handle input errors and re-prompt until the user has specified two numbers each greater than 1.

You will need to use a dynamically allocated 2D array in your Board class, and will need to implement the appropriate destructors and copy constructors.

Hints

1. The provided `letters.cpp` contains a function `random_letter` to choose a random letter based on English letter frequencies. It also contains a function `uppercase` to convert a string to all uppercase; you will need this function in order to convert the user's input, since the grid and the word list are both all-uppercase.
2. It can take a very long time to read the word list. You might try using a smaller word list for testing: just make sure it is sorted and all-uppercase.
3. You can search for a word in the grid recursively. You will need a recursive helper function with extra parameters indicating the row and column of the grid to begin the search. To search for a word at a given position:
    - The empty string can always be found.
    - If the letter at the starting position of the grid is not the first letter of the word, the search fails.
    - If it is the first letter, search for the rest of the word at each of the adjacent positions.
        - If one of those searches returns success, the search succeeded. Otherwise, it failed.
4. You can use a linked list (`StringList` in the lab 7) /or use a standard C++ list class object to store the found words.

A sample session

Text in bold was entered by the user.

```
WordGrid by J. Random Student - CS 237 PA3
Bonus features implemented: none
Loading wordlist.txt . . . Found 62887 words.

+---+---+---+---+
| D | A | N | C |
+---+---+---+---+
| K | Y | E | S |
+---+---+---+---+
| E | N | S | A |
+---+---+---+---+
| E | D | S | C |
+---+---+---+---+
Score: 0   Misses: 0/5
Your words:

Enter a word: day

+---+---+---+---+
| D | A | N | C |
+---+---+---+---+
| K | Y | E | S |
+---+---+---+---+
| E | N | S | A |
+---+---+---+---+
| E | D | S | C |
```

```
+---+---+---+---+
Score: 9   Misses: 0/5
Your words: DAY

Enter a word: end

+---+---+---+---+
| D | A | N | C |
+---+---+---+---+
| K | Y | E | S |
+---+---+---+---+
| E | N | S | A |
+---+---+---+---+
| E | D | S | C |
+---+---+---+---+
Score: 18   Misses: 0/5
Your words: DAY, END

Enter a word: deny

+---+---+---+---+
| D | A | N | C |
+---+---+---+---+
| K | Y | E | S |
+---+---+---+---+
| E | N | S | A |
+---+---+---+---+
| E | D | S | C |
+---+---+---+---+
Score: 34   Misses: 0/5
Your words: DAY, DENY, END

Enter a word: sand
I could not find that word on the board.

+---+---+---+---+
| D | A | N | C |
+---+---+---+---+
| K | Y | E | S |
+---+---+---+---+
| E | N | S | A |
+---+---+---+---+
| E | D | S | C |
+---+---+---+---+
Score: 34   Misses: 1/5
Your words: DAY, DENY, END

Enter a word: seyed
That isn't a real word.

+---+---+---+---+
| D | A | N | C |
+---+---+---+---+
| K | Y | E | S |
+---+---+---+---+
| E | N | S | A |
+---+---+---+---+
```

```
| E | D | S | C |
+---+---+---+---+
Score: 34   Misses: 2/5
Your words: DAY, DENY, END

Enter a word: casen
That isn't a real word.

+---+---+---+---+
| D | A | N | C |
+---+---+---+---+
| K | Y | E | S |
+---+---+---+---+
| E | N | S | A |
+---+---+---+---+
| E | D | S | C |
+---+---+---+---+
Score: 34   Misses: 3/5
Your words: DAY, DENY, END

Enter a word: deny
You have already used that word.

+---+---+---+---+
| D | A | N | C |
+---+---+---+---+
| K | Y | E | S |
+---+---+---+---+
| E | N | S | A |
+---+---+---+---+
| E | D | S | C |
+---+---+---+---+
Score: 34   Misses: 4/5
Your words: DAY, DENY, END

Enter a word: key

+---+---+---+---+
| D | A | N | C |
+---+---+---+---+
| K | Y | E | S |
+---+---+---+---+
| E | N | S | A |
+---+---+---+---+
| E | D | S | C |
+---+---+---+---+
Score: 43   Misses: 4/5
Your words: DAY, DENY, END, KEY

Enter a word: and
I could not find that word on the board.


Game over!
Your score was 43 points.
Your words: DAY, DENY, END, KEY
```

```
Press ENTER to continue.
```

Pause_237

```
/* pause_237() - Wait for the user to press ENTER.
*
* Description:
*   Optionally read and discard the remainder of the user's last
*   line of input.  Then, prompt the user to press ENTER, then
*   read and discard another line of input.
*
* Inputs:
*   bool have_newline:
*     True if the user has already entered a newline that the
*     program has not yet read.  If true, this function first
*     discards remaining input up to and including that newline.
*
*   Reads two lines from standard input if have_newline is true,
*   one line if it is false.  Lines are assumed to be less than
*   two hundred characters long.
*
* Outputs:
*   No return value.
*
*   Prints a prompt to standard output.
*
* Notes:
*   This function is intended to be used at the end of a program,
*   just before returning from main().  Reading another line of
*   input prevents the console window from closing immediately.
*
*   In general, have_newline should be true if the last user input
*   from cin used the extraction operator (>>), and false if there
*   has been no user input or if the last input used getline().
*/
void pause_237(bool have_newline)
{
      if (have_newline) {
            // Ignore the newline after the user's previous input.
            cin.ignore(200, '\n');
      }

      // Prompt for the user to press ENTER, then wait for a newline.
      cout << endl << "Press ENTER to continue." << endl;
      cin.ignore(200, '\n');
}
```