Assigned: Saturday April 4, 2015, **Due: Wednesday April 22, 2015, No extensions past April 24th.**

Overview

For this project you will be given some template code for the game "Boggle." The user interface is complete and runs in the browser, however, the game is missing several pieces.

1) It lacks the code to solve the board
2) It lacks the code to check user answers
3) It lacks the code to score user answers

Your assignment is to write those three pieces.

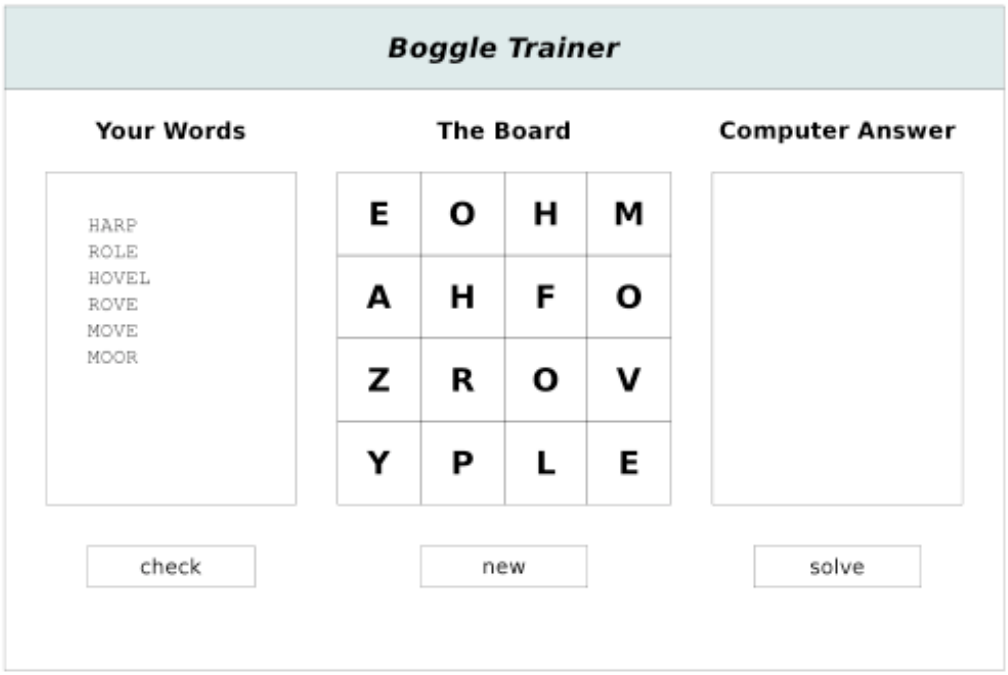
Implementation Specifics

Shown on the next page is a picture of a computer-based boggle game. In the middle of the screen is a boggle board. The challenge is to find sequences of adjacent letters that form words, and there is a time limit. Adjacent includes **vertical, horizontal, and diagonal**. Some actual words are shown in the panel on the left.

Usually people play boggle against other people. For this project, the user will play boggle against the computer.

The computer will do exactly four things:

a) generate the board
b) find all words in the board
c) check the user's words to make sure they are correct
d) score the user's words



We have already written a program to do step [a]. You have to write programs to do steps [b], [c], and [d]. That is the entire assignment -- write three programs.

Program 1: The Solver

The first program is the boggle solver. This is the most demanding of the three. Your program will read in two sets of data. First, it will read in a list of legal words (the dictionary). Then the program will read in a boggle board.

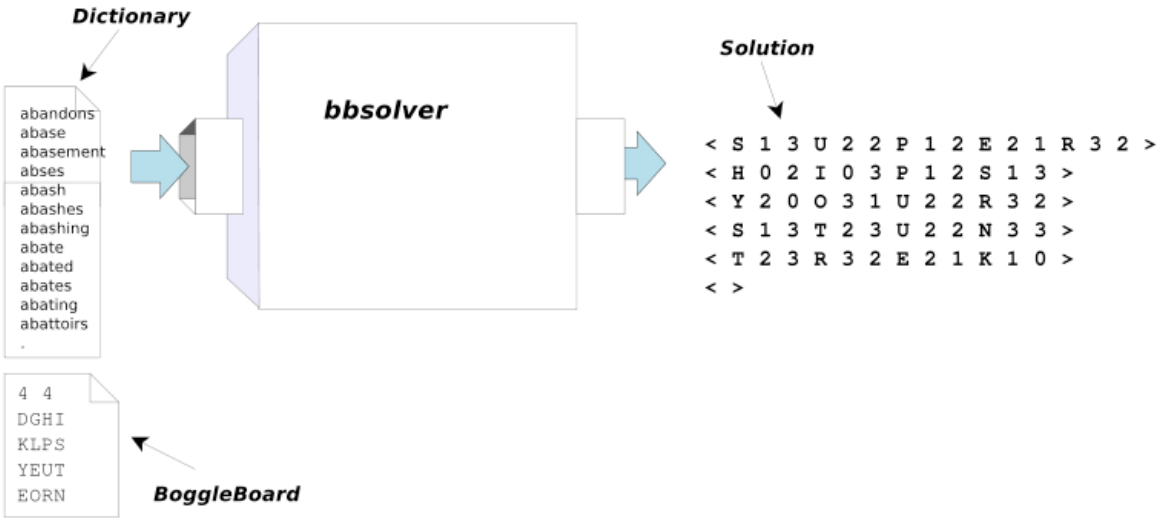
After reading in the dictionary and boggle board, the program will solve the board by searching for **all legal words** that can be formed by sequences of adjacent letters (including duplicate words). The same cell may not be used twice in the same word. Check online for details of the rules of boggle.

Your program will then print out its complete solution. The solution will be a list of the words it found. The output is more than just a list of strings, though. The output will also include the locations of the letters. And the output must use a very precise format. Here is an example. The words "ROLE" and "HAZY" appear on the board shown above. We assign coordinates (0,0) to the upper left cell. Giving row and column positions with each letter, we report these two words as follows:

```
< R 2 1 O 2 2 L 3 2 E 3 3 >
< H 1 1 A 1 0 Z 2 0 Y 3 0 >
```

Each word begins with "<" and ends with ">". Between those markers is a sequence of triples: letter, row number, column number. All items are separated by spaces. The end of the list is marked by the empty word: "< > “

Here is a picture showing the boggle solving program:



The program reads in the dictionary and the boggle board from cin. The program then prints to **`cout`** the list of words in this special format. This report format is called *Hescott Boggle Format*, or HBF for short (named in honor of Professor Ben Hescott).

Input Data Format

What does the input data looks like?
The dictionary is a list of words, one per line. They are mixed upper and lower case. You need to convert them to upper case. The end of the dictionary is marked with a sentinel of a single dot (.).

The boggle board looks like this:

```
4 4
EOHM
AHFO
ZROV
YPLE
```

The first line has two numbers: the number of rows and the number of columns of the board. Your program must work for boards of any size. After the first line are rows of letters. Each row is a single string with the correct width. These might be upper or lower case. You must convert all to upper case. There is no sentinel value for the board because you are told at the outset how many rows there will be.

Program 2: The Answer Checker

The second program will check the user's answers to see if they are valid answers. This program is a separate program from the solver. It will share some logic and data structures with the solver. This is an ideal use for classes. For example, both will use the Dictionary class. What other classes can you design that can be used in both programs?

Here is how the checker works. The checker first reads from **`cin`** the dictionary. The dictionary has the same format as before. The checker then reads from **`cin`** the boggle board the user worked on. This board has the same format described above. Then the program reads in words from **`cin`**.

The checker program then prints out a report on all the words the user gave it. The report has the following format:

```
OK ROLE
OK MOVE
NO LAZY
NO MOVE
...
```

Each user answer is printed preceded by OK or NO. OK means the word is a legal answer. NO means it is not a legal answer. There is more than one reason a word might not be a legal answer.

That is all there is to the checker. It reads in a dictionary and a board, then analyzes a list of words.

Program 3: The Answer Scorer

The third program is the simplest of the three. This program does not need to read in a dictionary or board. It just reads in a checker report as shown above. And it prints out each line with a score printed at the left. At the end of the output, the program will print the number of valid words and the total number of points. Check online for the scoring rules for boggle. The output for the sample data shown above is:

```
1 OK ROLE
1 OK MOVE
0 NO LAZY
0 NO MOVE
...
12 words 18 points
```

Putting It Together

The assignment is to write these three programs. Making these three programs into an enjoyable game requires a nice user interface. You do not have to do that. We shall provide a web-based system to show the user the board, run the timer, and allow the user to enter words. Then the web-based system will submit the words to your checker program. The output of the

checker program will be sent back to the webpage for the user to see. And the web based system will allow the user to click a button and ask your solver for its answers.

Your programs have to work with this webpage. Therefore, the input format and output format must match exactly what that page expects. If your output format does not match, the webpage will not work.

Program Requirements

Your program is a part of a larger system, so it has to meet certain specifications to work with the system. The solver has to be implemented as a class with specific names and functions. The checker has to be implemented as a class with specific names and functions. The details are described below.

The Solver

The solver must be implemented as a class with this definition:

```
//
// This must be in a class file called BogSolver.h
//
#ifndef BOGSOLVER_H
#define BOGSOLVER_H

#include "BogWordList.h"
#include "Dictionary.h"

class BogSolver
{
public:
      BogSolver();
      ~BogSolver();
      bool readDict();
      bool readBoard();
      bool solve(); // search board for words in dict
      int numWords(); // returns number of words found
      int numWords(int len); // number of words of length len
      BogWordList* getWords(); // returns all words found
      BogWordList* getWords(int len); // returns words of len len
      void printWords(); // print all words in HBF
      void printWords(int len); // print len-length words in HBF
      void listWords(); // print just the text, no coords
      void listWords(int len); // just the text, no coords
private:
      Dictionary dict; // must use a Dictionary
      // other private methods or data may appear here
};
#endif
```

You must implement all these functions. Two of the functions will return solutions as a list of words and the positions of the letters.

The data structure to store those lists is defined as a **`BogWordList`** defined as follows:

```
#ifndef BOGWORDLIST_H
#define BOGWORDLIST_H

#include<vector>

struct BogLett {
      char  c;
      int   row, col;
};

typedef std::vector<BogLett> BogWord;
typedef std::vector<BogWord> BogWordList;

#endif
```

IMPORTANT: You do not have to use this structure internally to solve or check the board (although you are welcome to do so). This data format is a compact way to export the results to an outside program and may not be the best way to manage data as you analyze the board.

Finally, the main function for the solver must be exactly this:

```
//
// this must be in a file called solverMain.cpp
//
int main()
{
      BogSolver solver;
      solver.readDict();
      solver.readBoard();
      solver.solve();
      solver.printWords();
      return 0;
}
```

Compile the solver with the Makefile:

`make bbsolver`

(you may have to adjust the Makefile depending on any other files you use).

The Checker

The checker program will be implemented as a class with this definition:

```
#ifndef BOGVALIDATOR_H
#define BOGVALIDATOR_H
//
// this must be in a header file called BogValidator.h
//
class BogValidator
```

```
{
public:
      BogValidator(); // constructor
      ~BogValidator(); // destructor
      bool readDict(); // read in a dictionary
      bool readBoard(); // read in a board
      bool isValid(string s); // validates one word
      void checkWords(); // validates cin. The end of input will
                         // simply return NULL to cin
private:
      Dictionary dict; // must use a Dictionary
      // other private methods or data may appear here

#endif
```

The main function for the checker program will be:

```
//
// this must be in a file called checkerMain.cpp
//
int main()
{
      BogValidator v;
      v.readDict();
      v.readBoard();
      v.checkWords();
      return 0;
}
```

You many not modify main. To compile your boggle answer checker, you type:

```
make bbchecker
```

The Scorer

The scorer may be written using any structure you like. You will call the main file **`scorerMain.cpp`**. The executable program must be called **`bbscorer`** .

Low Level Details

The Website

The boggle programs that solve a board, check some answers, and score answers are not very user friendly. Those programs are difficult enough to write without having to worry about putting a nice interface on them. Therefore, we have created a webpage that will use your programs to do the work.

This page will allow the user to see a board, enter answers, see the answers from your program, and have his/her answers checked and scored.

Setting Up the Site

We have a simple program that will set up a web page for this project. To run the program do this:

a) Make a folder for your program
Something like `Desktop/comp15/hw6 is good`
But any name is fine.
b) Change into that folder
c) Type: `/comp/15/public_html/assignments/hw6/files/bog/setup`
Answer the question or two, and wait a few seconds.

That is all there is to it. Your personal boggle game webpage is installed. Note: in the "data" folder, you will find example boards, and the word dictionary that we will use to test the code.

We have provide you with some template code (the .h files, the main files) and the Dictionary Class source code. The Dictionary is a modified version of the Trie we worked on in lab earlier in the semester. You are free to replace the Dictionary Class we provided with another Dictionary that you create (for "above and beyond" points), but it must follow the same interface.

Using Your Boggle Site

You can use any web browser to visit your page:
`http://www.cs.tufts.edu/~logname/bog15/`
where logname is your logname on the linux servers. There is a tilde just before your logname. The site will not do much except show you boards, allow you to enter answers, but no scoring or playing. To get it to do those things, you have to write the three programs then publish those programs to your site.

Above and Beyond

This assignment is challenging, but here are some options to make it more challenging. "(!)" denotes an exceptionally hard challenge (you will be rewarded greatly if you complete these!)

1. Create your own Dictionary class that meets the **`Dictionary.h`** interface.
2. Write your own 2D Dynamic Array functions to replace the **`vector<>`** definitions.
3. Provide a "hint" for users. E.g., provide partial matches that will produce words when other letters are used.
4. Provide a "suffix hint" that shows users just the suffixes of words in the game.
5. Limit the answers to palindromes only.
6. Create a "Scoreboard" class that will keep high scores in a file, with users' names.
7. Modify the game to use numbers (e.g., "Find primes" or "Find two numbers that will produce 193 when added" or "Find all numbers that alternate even/odd digits")
8. Modify the website code to do something unique. (!)
9. Allow two users to play against one another in real time through two different browsers with the same board. (!)
10. Have the computer write a poem/haiku with the words that are found on the board (!)
11. Do a search online for "Boggle Variations" and implement one of the interesting ones.
12. Be creative!

Filenames and Compiling

The web site that will use your tools to do the heavy lifting expects the three programs to be called:

```
bbsolver
bbchecker
bbscorer
```

The Makefile will tell us how to compile your programs (and again, you may need to adjust the Makefile for your own additional files).

Publishing Your Programs

When you write the programs **`bbsolve`**, **`bbcheck`**, and **`bbscore`**, you have to publish these to the site so the webpage can use them.

Publishing your programs is easy. Just type:

`./publish`

That will copy the current version of **bbsolver**, **bbchecker**, and **bbscorer** to the website for use by the webpage.

You can work on one file at a time, publish it, and see if that program works.

Sample Data

You may copy sample boggle boards and the word list from the comp15 directory with:

`cp /comp/15/public_html/assignments/hw6/files/bog/data* .`

You can play these sample boards with paper and pencil to get used to how to find the words.

Handling Qu

Boggle treats *Qu* as a single unit; the letter Q never appears on a tile by itself. How does that affect this program?

Here are the details:

a) In the BogLett struct, the tile is represented by a single 'Q'
b) In HBF, the Qu tile prints as the two-char string QU
c) The getWords(len) and listWords(len) methods consider Q to be two chars
therefore, QUICK is a 5-letter word, but is printed in HBF as
`< QU 1 1 I 1 2 C 1 3 K 1 4 >`