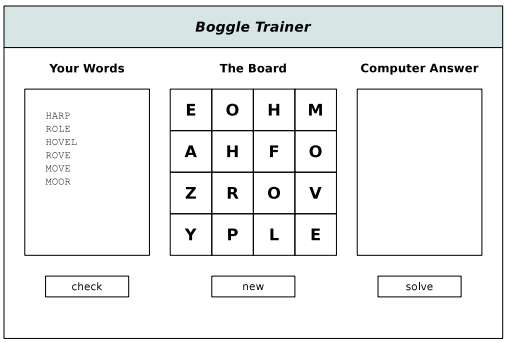
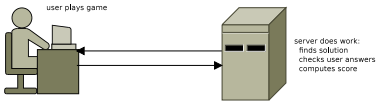




Playing Boggle on Line

Go to `http://www.cs.tufts.edu/~molay/bog/`

You will see the user interface for a boggle game. This game has a nice front-end, but lacks several important pieces. It lacks the code to solve the board, it lacks the code to check user answers, and it lacks code to score the user answers. Your assignment is to write those three pieces.

Building a Boggle Game

Shown above 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. 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. The same cell may not be used twice in the same word. Check online for details of the rules of boggle.

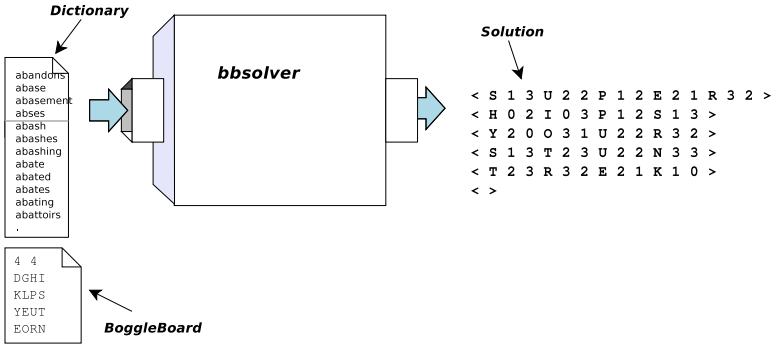
Your program will then print out its complete solution. The solution will be a list of the words it found. The output is more than just a list of strings, though. The output will also include the locations of the letters. And the output must use a very precise format. Here is an example. The words "ROLE" 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.

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 you wrote for part 1. 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 MOOR
NO LAZY
NO MOOR
  ...
```

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 MOOR
0 NO LAZY
0 NO MOOR
 ...
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

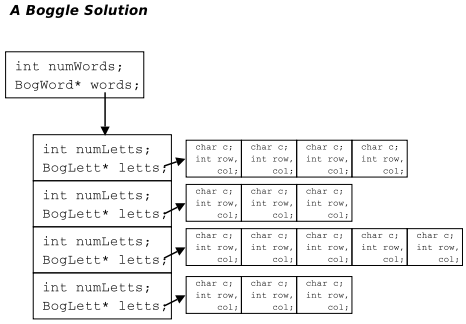
The solver must be implemented as a class with this definition:

```
//
// This must be in a class file called BogSolver.h
//
#include "BogWordList.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
};
```

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:

```
struct BogLett {
        char    c;
        int     row, col;
};
struct BogWord {
        int             numLetts;
        BogLett*        letts;
};

struct BogWordList {
        int             numWords;
        BogWord*        words;
};
```

Which is very similar to the transcript structure from earlier in the course.



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;
}
```

To compile your boggle board solver, you type:

```
g++ -Wall -Wextra -g solverMain.cpp BogSolver.cpp ... -o bbsolver
```

Where the "..." might be other class files you write. For example, it is very likely you will use the Dictionary class.

The Checker

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

```
//
// 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

   private:
        Dictionary dict;                // must use a Dictionary
        // other private methods or data may appear here
};
```

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:

```
g++ -Wall -Wextra -g checkerMain.cpp BogValidator.cpp ... -o bbchecker
```

Where the "..." might be other class files you write. For example, it is very likely you will use the Dictionary class.

The Scorer

The scorer may be written using any structure you like. You must call the main file solverMain.cpp . The executable program must be called bbchecker .

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
```

When you submit your work, you have to tell us how to compile your programs. Therefore, you must submit a file called README.txt that has one section that lists the three compile commands. There must be one line for each of the three programs. The lines have to look like:

```
bbsolver: g++ -Wall -Wextra ....
bbchecker: g++ -Wall -Wextra ....
bbscorer: g++ -Wall -Wextra ....
```

That is, the line begins with the name of the program, then a colon, then the compile command. These lines may appear anywhere in your README.txt file.

Sample Data

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

```
cp /comp/15/public_html/files/bog/*.* .
```

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