C++ 712 A: Word Game assignment

NOTE: Please pay attention to the **Assignment Submission** section below.

For this assignment, you need to write a word game. You will have a M X N board, and each cell in the board will have one letter from the alphabet. Your program will need to find all valid words that can be made from the letters on the board.

Words can be made by:

1. Traversing adjacent cells (up, down, left, right, and diagonally, which means a cell can have up to 8 immediate neighbors).
2. Each cell can only be **visited once** when looking for a word.

For example, from the board below, some of the words that can be made are:

ape apes apt pot pots pottery try trying rye and so on.

ape is made by starting at top left corner ('a'), going right to 'p', and again right to 'e'

pot is made by starting at 'p', going down to 'o', and right to 't'. If you went further right one cell, you come to 's', and that would make the word *pots*. Note that from 'o', if you went down to 't', that would also make *pot*, so your program would typically cover both paths, but eventually we **do not want duplicates to be printed**.

gin is made by starting at 'g'(bottom right corner), going up diagonally left to 'i', and then right to 'n'. If you then go up to 's', you get another word, *gins*.

pest is made by starting at 'p', and then 'e', 's' and 't'. Note that you cannot make *peste* because that would mean you have to visit 'e' again, but 'e' has already been visited for this word. (See rule #2 above)

a	**p**	**e**	**x**
n	**o**	**t**	**s**
e	**t**	**i**	**n**
r	**y**	**e**	**g**

Board Diagram

Your program should take the following as parameters

1. Number of rows in the board
2. Number of cols in the board
3. Maximum length of a word to look for
    a. For example, for the board above, if we say max length of the words we want to find is 3, then once your program finds the word "try", it would not look for anything beyond that for that particular string, and as a result it would not find the word "trying".
    b. Note if you do not put the maximum length restriction, your program will try to find words that are up to total number of cells on board (16 for a 4X4 board), and that could

take a while for the program to run. So its best that you limit it to smaller word lengths, especially when developing and testing it.
Something like 5 or 6 could be a good maximum length to start with when you have a, say, 3X3 or 4X4 board.
NOTE: For the board in this assignment (the board specified above), you should set **max word length to 9**, that way I can compare your program output that you submit.

Important:
Another thing you may want to do in the beginning is to have a small board, say, 2X2, because there you can manually determine what word possibilities are there, and then see if your program is computing those words.
For generating the program output, you will then use a 4X4 board as shown in diagram above.

If you parametrize the above, then you can run, say, a 2X2 or a 3X3 or a 4X4 or 4X5 word game board just by changing the parameter values, and not having to change anything else in your code.

The classes and methods mentioned below **SHOULD** be used.

You can have more classes and methods if you need, but the ones mentioned below **should be used** in your submission.

class WordGame

```
void   Run();
```

class Board

```
int NumRows() const;
int NumCols() const;
```

This class will have storage for the board. It could be like a vector< string >

class BoardInitializer

```
void InitializeBoardDebug(std::vector<std::string> & board, int size);
```

Note that I am passing a vector of string for the board, but if you want to use some other way to store the board, you can. BoardInitializer methods will initialize the board, i.e., assign characters to the board cells. The `InitializeBoardDebug` method should initialize your board to the **letters shown in the board diagram above**. You can add other initialize methods for your testing, but for generating the output of your program (mentioned below), you **should use the** letters show in the board diagram above. That way I can compare the output to what I have.

class WordFinder

```
TWordsList    FindWords();
```

`TWordsList` is a collection of the found words. You can use whatever container you think is appropriate here.

class Dictionary

```
bool Contains(const std::string & word) const;
```

if `word` is a valid word, i.e., is present in the Dictionary, `Contains` returns true, else false.

Program Output: You should print

1. 1st line of output:
    a. **Max word length: <number>**
    b. This prints the maximum length of words searched for. (For the given board above, you should set this to the number provided in description above)
2. 2nd line of output:
    a. **Number of words found: <number>**
    b. Count of the unique words found
3. 3rd line of output:
    a. All the unique words found, separated by a space.

Assignment Submission

Your assignment submission should contain the following:

1. Source code files (.cpp and .h)
2. A file name **WordsOutput.txt** that contains the program output as specified in the **Program Output** section above.

Your main program should look something like the following:

Note: The code below may not compile, I added it below to show what it could look like:

```
int main(int argc, char **argv)
{
        // parametrize num rows and cols, so that we can change these only here,
        // and have a bigger board.

        int numRows = 2;
        int numCols = 2;

        WordGame   wg( numRows, numCols, wordsFile);
        // wordFile (provided in the assignment) contains a bunch of words,
        // that should be read in to the Dictionary object.

        wg.Run();

        return 0;
}
```