Programming Assignment 5
Mystery Word

Time due: 9:00 PM Tuesday, November 15

You have been contracted by the producers of the highly-rated Mystery Word TV game show to write a program that lets fans play a home version of the game. Here's how one round of the game works: The computer picks a mystery word of four to six letters and tells the player how many letters are in the word. The player tries to determine the mystery word by presenting the computer with a series of trial words. Each trial word is a four to six letter word. If the trial word is the mystery word, the player wins. Otherwise, the computer responds to the trial word with an integer from 0 to the length of the mystery word, indicating the number of letters in the trial word that are in the mystery word. For example, if the mystery word is EAGER, the response to the trial word GOOSE would be 2 (because of the G and one E), the response to EERIE would be 3 (the R and two Es), the response to GAME would be 3 (the G, the A, and the E), the response to BONUS would be 0 (no letters in common), and the response to AGREE or to GREASE would be 5 (one A, one G, one R, and two Es). The player's score for each round is the number of trial words needed to get the correct word (counting the trial word that matched the mystery word).

Your program must start by asking the player how many rounds to play, and then play that many rounds of the game. After each round, the program must display some statistics about how well the player has played the rounds so far: the average score, the minimum score, and the maximum score.

Here is an example of how the program must interact with the player (player input is in **boldface**):

```
How many rounds do you want to play? 3

Round 1
The mystery word is 5 letters long
Trial word: assert
3
Trial word: xyzzy
I don't know that word
Trial word: bred
2
Trial word: mucus
0
Trial word: never
4
```

```
Trial word: enter
3
Trial word: raven
You got it in 7 tries
Average: 7.00, minimum: 7, maximum: 7

Round 2
The mystery word is 5 letters long
Trial word: eerie
3
Trial word: game
3
Trial word: agree
5
Trial word: eager
You got it in 4 tries
Average: 5.50, minimum: 4, maximum: 7

Round 3
The mystery word is 4 letters long
Trial word: monkey
0
Trial word: Hello
Your trial word must be a word of 4 to 6 lower case letters
Trial word: what?
Your trial word must be a word of 4 to 6 lower case letters
Trial word: wrap-up
Your trial word must be a word of 4 to 6 lower case letters
Trial word: stop it
Your trial word must be a word of 4 to 6 lower case letters
Trial word: sigh
You got it in 6 tries
Average: 5.67, minimum: 4, maximum: 7
```

You can assume the user will always enter an integer for the number of rounds (since you haven't learned a clean way to check that yet). If the number of rounds entered is not positive, write the message `The number of rounds must be positive` and end the program.

In order for us to thoroughly test your program, it must have at least the following components:

- A main routine that declares an array of C strings. This array exists to hold the list of words from which the mystery word will be selected. The response to a trial word will be an integer only if the trial word is in this array. (From the example transcript above, we deduce that "xyzzy" is not in the array.) The declared number of C strings in the array must be at least 9000. (You can declare it to be larger if you like, and you don't have to use every element.)

  Each element of the array must be capable of holding a C string of length up to 6 letters (thus 7 characters counting the zero byte). So a declaration such

as `char wordList[9000][7];` is fine, although something like `char wordList[MAXWORDS][MAXWORDLENGTH+1];`, with the constants suitably defined, would be stylistically better.

Along with the array, your main routine must declare an int that will contain the actual number of words in the array (i.e., elements 0 through one less than that number are the ones that contain the C strings of interest. The number may well be smaller than the declared size of the array, because for test purposes you may not want to fill the entire array.

Your main routine must call `loadWords` (see below) to fill the array.

- A function named `loadWords` with the following prototype:
- `int loadWords(char words[][7], int maxWords);`

(Instead of `7`, you can use something like `MAXWORDLENGTH+1`, where `MAXWORDLENGTH` is declared to be the constant 6.) This function puts words into the `words` array and returns the number of words put into the array. The array must be able to hold at least `maxWords` words. You *must* call this function exactly once, before you start playing any of the rounds of the game. If your main routine declares `wordList` to be an array of 10000 C strings and `nWords` to be an int, you'll probably invoke this function like this:

```
int nWords = loadWords(wordList, 10000);
```

If you wish, you may use this implementation of `loadWords`. (Don't worry if you don't understand every part of the implementation.) It fills the array with the four-to-six-letter words found in a file named `Z:\words.txt` that you would put in the top level folder on your network drive (the Samba Server) on a SEASnet machine. You may use this 7265-word file if you want a challenging game. If you want to use this implementation of `loadWords`, but call the file something else or put it somewhere else, change the string literal `"Z:/words.txt"` in the function appropriately. (Note the use of the forward slash in the string literal.)

Another way to implement `loadWords` for doing simple testing is something like this:

```
int loadWords(char words[][7], int maxWords)
{
    strcpy(words[0], "eager");
    strcpy(words[1], "goose");
    return 2;
}
```

Whatever implementation of `loadWords` you use, it must return an int no greater than `maxWords`. If `loadWords` returns a value less than 1, your main routine can do whatever reasonable thing you want, provided its behavior is not undefined. One reasonable thing to do would be to write a message like `No words were loaded, so I can't play the game` and terminate the program. When we test your program, we will replace your implementation of `loadWords` with our own special testing implementation.

If `loadWords` returns a value in the range from 1 to `maxWords` inclusive, your program must write no output to `cout` other than what is required by this spec. If you want to print things out for debugging purposes, write to `cerr` instead of `cout`. When we test your program, we will cause everything written to `cerr` to be discarded instead — we will never see that output, so you may leave those debugging output statements in your program if you wish.

- A function named `manageOneRound` with the following prototype:
- `int manageOneRound(char words[][7], int num, int wordnum);`

(Again, instead of `7`, you can use something like `MAXWORDLENGTH+1`.)
Using `words[wordnum]` as the mystery word, this function plays one round of the game. It returns the score for that round. In the transcript above, this function is responsible for this much of the round 1 output, no more, no less:

```
Trial word: assert
3
Trial word: xyzzy
I don't know that word
Trial word: bred
2
Trial word: mucus
0
Trial word: never
4
Trial word: enter
3
Trial word: raven
```

Your program must call this function to play each round of the game. Notice that this function does *not* select a random number; the *caller* of this function does, and passes it as the third argument. Notice also that this function does *not* write the message about the player successfully determining the mystery word. **If you do not observe these requirements, your program will fail most test cases.**

> The parameter `num` represents the number of words in the array; if it is not positive, or if `wordnum` is less than 0 or greater than or equal to `num`, then `manageOneRound` must return −1 without writing anything to `cout`.

To make the program interesting, it must pick mystery words at random. This brief tutorial describes the tools you need to do this.

Your program must **not** use any std::string objects; you must use C strings. You may assume (i.e., we promise when testing your program) that any line entered in response to the trial word prompt will contain fewer than 100 characters (not counting the newline at the end).

Your program must **not** use any global variables whose values may change. Global *constants* are all right; it's perfectly fine to declare `const int MAXWORDLENGTH = 6;` globally, for example. The reason for this restriction is that part of our testing will involve replacing your `manageOneRound` function with ours to test some aspects of your `main` function, or replacing your `main` with ours to test aspects of your `manageOneRound`. For this reason, you must not use any non-const global variables to communicate between these functions, because our versions won't know about them; all communication between these functions must be through the parameters (for `main` to tell `manageOneRound` the words, number of words, and mystery word number for a round), and the return value (for `manageOneRound` to tell `main` the score for that round). Global constants are OK because no function can change their value in order to use them to pass information to another function.

Microsoft made a controversial decision to issue by default a warning when using certain functions from the standard C and C++ libraries (e.g., `strcpy`). These warnings are not relevant in this class; to eliminate them, put the following line in your program *before* any of your `#include`s:

```
#define _CRT_SECURE_NO_DEPRECATE
```

It is OK and harmless to leave that line in when you build your program using g++.

What you will turn in for this assignment is a zip file containing these two files and nothing more:

1. A text file named **game.cpp** that contains the source code for your C++ program. Your source code should have helpful comments that tell the purpose of the major program segments and explain any tricky code.
2. A file named **report.doc** or **report.docx** (in Microsoft Word format), or **report.txt** (an ordinary text file) that contains:

a. A brief description of notable obstacles you overcame.
b. A description of the design of your program. You should use pseudocode in this description where it clarifies the presentation.

Your report does not need to describe the data you might use to test this program.

By November 14, there will be links on the class webpage that will enable you to turn in your zip file electronically. Turn in the file by the due time above.