Programming Assignment 5
Rock On

Time due: 9:00 PM Tuesday, November 17

You have been contracted by the producers of the highly-rated Rock On 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 secret word of four to six letters and tells the player how many letters are in the word. The player tries to determine the secret word by presenting the computer with a series of probe words. Each probe word is a four to six letter word. If the probe word is the secret word, the player wins. Otherwise, the computer responds to the probe word with two integers: the number of rocks and the number of pebbles. Rocks and pebbles are pairings between a letter in the probe word and the same letter in the secret word. A *rock* is a pairing of a letter in the probe word and the same letter in the secret word in the same position. A *pebble* is a pairing between a letter in the probe word and the same letter in the secret word, but *not* in the same position as in the probe word, provided the two letters are not involved in a rock or another pebble. The player's score for each round is the number of probe words needed to get the correct word (counting the probe word that matched the secret word).

As an example, suppose the secret word is EAGER. Here are some examples of the rock and pebble counts for various probe words:

```
               EAGER        EAGER        EAGER        EAGER  GOOSE        RAVE
EERIE        EERIE    p   p        pr r         rpp     or    r p p
0 r / 2 p    2 r / 1 p    1 r / 2 p    1 r / 2 p     EAGER        EAGER
EAGER        EAGER    AGREE        BONUS        GREASE       CONDOR        ppprp
```

Your program must ask 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
secret word is 5 letters long.       Probe word: assert    Rocks: 1,
Pebbles: 2     Probe word: xyzzy      I don't know that word.
        Probe word: bred       Rocks: 0, Pebbles: 2   Probe word: mucus      Rocks: 0,
        You got it in 7 tries.       Average: 7.00, minimum: 7,
maximum: 7     Round 2        The secret word is 5 letters long.
        Probe word: eerie      Rocks: 1, Pebbles: 2   Probe word: rave       Rocks: 2,
is 4 letters long.     Probe word: monkey     Rocks: 0, Pebbles: 0
        Probe word: puma       Rocks: 0, Pebbles: 0   Probe word: Hello      Your prob
word must be a word of 4 to 6 lower case letters.    Probe word: stop
it      Your probe word must be a word of 4 to 6 lower case letters.
```

```
        Probe word: **sigh**       You got it in 3 tries.        Average:
4.67, minimum: 3, maximum: 7
```

Notice that unknown words and probe strings that don't consist of exactly 4 to 6 lower case letters don't count toward the number of tries for a round.

You can assume the player 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.
```

(not preceded by an empty line) and terminate the program immediately.

The program will be divided across three files: `rocks.cpp`, which you will write; [`utilities.h`], which we have written and which you must not change; and [`utilities.cpp`], which we have written but you may modify in a limited way. You will turn in only `rocks.cpp`; when we test your program, our test framework will supply `utility.h` and our own special testing version of `utilities.cpp`.

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

- In `rocks.cpp`, a main routine that declares an array of C strings. This array exists to hold the list of words from which the secret word will be selected. The response to a probe word will be the number of rocks and pebbles only if the probe 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][MAXWORDLEN+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 elements 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.

  Before prompting the player for the number of rounds to play, your main routine must call `loadWords` (see below) to fill the array. The only valid words in the game will be those that `loadWords` puts into this array.

If the player's score for a round is not 1, the message reporting the score must be

```
You got it in n tries.
```

where *n* is the score. If the score is 1, the message must be

```
You got it in 1 try.
```

- In `utilities.cpp`, a function named `loadWords` with the following prototype:

  ```
  int loadWords(char words[][7], int maxWords);
  ```

  (Instead of `7`, you can use something like `MAXWORDLEN+1`, where `MAXWORDLEN` is declared to be the constant 6, as in `utilities.h`.) 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);
  ```

  We have given you an 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 Windows 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.) The comment in the `loadWords`implementation file tells you how to change the file name string if you're running on a different Windows system, a Mac, or a SEASnet Linux server.

  To do simple testing, you can change the implementation of `loadWords` to something like this:

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

Whatever implementation of `loadWords` you use, each C string that it puts into the array must consist of four to six lower case letters; the C strings must not contain any characters that aren't lower case letters.
The `loadWords` function must return an int no greater than `maxWords`. If it returns a value less than 1, your main routine must write

```
  No words were loaded, so I can't play the game.
```

to `cout` and terminate the program immediately, without asking the player for the number of rounds to play, etc.

When we test your program, we will replace `utilities.cpp` (and thus any changed implementation of `loadWords` you might have made) 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.

- In `rocks.cpp, a function named manageOneRound with the following prototype:`

```
  int manageOneRound(const char words[][7], int nWords, int
wordnum);
```

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

```
      Probe word: assert
      Rocks: 1, Pebbles: 2
       Probe word: xyzzy
       I don't know that word.
      Probe word: bred
       Rocks: 0, Pebbles: 2
       Probe word: mucus
      Rocks: 0, Pebbles: 0
      Probe word: never
      Rocks: 2, Pebbles: 2
      Probe word: enter
      Rocks: 1, Pebbles: 2
      Probe word: river
      Rocks: 3, Pebbles: 0
      Probe word: raven
You got it in 7 tries.
```

```
Average: 7.00, minimum: 7, maximum: 7
Round 2
The secret word is 5 letters long.
Probe word: eerie
Rocks: 1, Pebbles: 2
Probe word: rave
Rocks: 2, Pebbles: 1
Probe word: agree
Rocks: 1, Pebbles: 4
Probe word: eager
You got it in 4 tries.
Average: 5.50, minimum: 4, maximum: 7
```

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

The parameter nWords 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 nWords, then manageOneRound must return −1 without writing anything to cout.

If for a probe word the player enters a string that does not contain four to six lower case letters or contains any character that is not a lower case letter, the response must be

```
Your probe word must be a word of 4 to 6 lower case letters.
```

If the player enters a string consisting of exactly four to six letters, but that string is not one of the words in the array of valid words, then the response must be

```
I don't know that word.
```

To make things interesting, your program must pick secret words at random using the function randInt, contained in utilities.cpp:

```
int randInt(int min, int max);
```

Every call to randInt returns a random integer between min and max, inclusive. If you use it to generate a random position in an array of ninteresting items, you should invoke it as randInt(0, n-1), not randInt(0, n), since the latter might return n, which is not a valid position in an n-element array.

Your program must not use any std::string objects (C++ strings); you must use C strings.

You may assume (i.e., we promise when testing your program) that any line entered in response to the probe 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 during execution. Global *constants* are all right; it's perfectly fine to declare const int MINWORDLEN = 4; globally, for example. The reason for this restriction is that part of our testing will involve replacing yourmanageOneRound function with ours to test some aspects of your main function, or replacing your main with ours to test aspects of yourmanageOneRound. 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 secret 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 an error or warning when using certain functions from the standard C and C++ libraries (e.g., strcpy). These warnings call that function unsafe and recommend using a different function in its place; that function, though, is not a Standard C++ function, so will cause a compilation failure when you try to build your program under clang++ or g++. Therefore, for this class, we do not want get that error or warning from Visual C++; to eliminate them, put the following line in your program *before* any of your #includes:

```
#define _CRT_SECURE_NO_DEPRECATE
```

It is OK and harmless to leave that line in when you build your program using clang++ or 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 **rocks.cpp** that contains main, manageOneRound, and other functions you choose to write that they might call. (You must *not*put implementations of loadWords or randInt in rocks.cpp; they belong in utilities.cpp.) 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. Document the design of your main routine and any functions you write. Do not document the loadWords or randInt functions.

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

By November 16, 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.