```
// Project 5

#define _CRT_SECURE_NO_DEPRECATE
#ifdef _MSC_VER
#pragma warning(disable:6262)
#endif

#include "utilities.h"
#include <iostream>
#include <cstring>
#include <cctype>
using namespace std;

   //**********************
   //***  Replace the "Z:/words.txt" in the initialization of WORDNAMEFILE below
   //***  with a path to the file you want to use as your word file.
   //***
   //***  On a Windows system, if you provide a path for the wordfilename,
   //***  use / in the string instead of the \ that Windows usually uses
   //***  (e.g., "Z:CS31/P5/mywordfile.txt").
   //
   //***  On a Mac, it's probably easiest to use the complete pathname to
   //***  the words file, e.g. "/Users/yourUsername/words.txt" or
   //***  "/Users/yourUsername/CS31/P5/words.txt".
   //
   //***  On a SEASnet Linux server, if you put the words.txt file in the
   //***  same directory as your .cpp file, you can use "words.txt" as the
   //***  file name string.

const char WORDFILENAME[] = "Z:/words.txt";

const int MAXTRIALLEN = 100;
const int MAXWORDS = 9000;

int playOneRound(const char words[][MAXWORDLEN+1], int num, int wordnum);
bool contains(const char words[][MAXWORDLEN+1], int num, const char str[]);
void countMatches(const char s1[], const char s2[], int& nFlowers, int& nBees);

int main()
{
      // Get word list

    char wordList[MAXWORDS][MAXWORDLEN+1];

    int nWords = getWords(wordList, MAXWORDS, WORDFILENAME);
    if (nWords < 1)
    {
        cout << "No words were loaded, so I can't play the game." << endl;
        return 1;
    }

    cout.setf(ios::fixed);
    cout.precision(2);

    cout << "How many rounds do you want to play? ";
    int nRounds;
    cin >> nRounds;
    cin.ignore(10000, '\n');
    if (nRounds <= 0)
    {
        cout << "The number of rounds must be positive." << endl;
        return 1;
    }
```

```
    int totalTrials = 0;
    int minTrials;
    int maxTrials;

      // Play rounds

    for (int round = 1; round <= nRounds; round++)
    {
        cout << endl << "Round " << round << endl;

          // Select random word as mystery word

        int wordnum = randInt(0, nWords-1);
        cout << "The mystery word is " << strlen(wordList[wordnum])
             << " letters long." << endl;

          // Play a round with that word

        int nTrials = playOneRound(wordList, nWords, wordnum);
        if (nTrials == -1)
        {
            cout << "**** Internal program error:  playOneRound returned -1!"
                 << endl;
            return 1;
        }
        cout << "You got it in " << nTrials;
        if (nTrials == 1)
            cout << " try";
        else
            cout << " tries";
        cout << "." << endl;

          // Update and print statistics

        totalTrials += nTrials;
        if (round == 1)
        {
            minTrials = nTrials;
            maxTrials = nTrials;
        }
        else if (nTrials < minTrials)
            minTrials = nTrials;
        else if (nTrials > maxTrials)
            maxTrials = nTrials;
        cout << "Average: " << static_cast<double>(totalTrials)/round
             << ", minimum: " << minTrials
             << ", maximum: " << maxTrials << endl;
    }
}

int playOneRound(const char words[][MAXWORDLEN+1], int num, int wordnum)
{
      // If impossible to play a round, return failure

    if (wordnum < 0  ||  wordnum >= num)
        return -1;

      // Repeatedly get trial words until word is guessed

    for (int trialNum = 1; ;)
    {
          // Get trial word

        cout << "Trial word: ";
        char trial[MAXTRIALLEN+1];
```

```
        cin.getline(trial, MAXTRIALLEN+1);

          // If it's the mystery word, return

        if (strcmp(trial, words[wordnum]) == 0)
            return trialNum;

          // See if trial word has a valid number of lower case letters...

        int k;
        for (k = 0; islower(trial[k]); k++)
            ;
        if (trial[k] != '\0'  ||  k < MINWORDLEN  ||  k > MAXWORDLEN)
            cout << "Your trial word must be a word of " << MINWORDLEN
                 << " to " << MAXWORDLEN << " lower case letters." << endl;

          // ... and is in the word list ...

        else if (! contains(words, num, trial) )
            cout << "I don't know that word." << endl;

          // ... and if so, report number of flowers and bees
        else
        {
            int nFlowers;
            int nBees;
            countMatches(trial, words[wordnum], nFlowers, nBees);
            cout << "Flowers: " << nFlowers << ", Bees: " << nBees << endl;
            trialNum++;
        }
    }
}

  // Return true if str is in list
bool contains(const char words[][MAXWORDLEN+1], int num, const char str[])
{
    for (int k = 0; k < num; k++)
        if (strcmp(words[k], str) == 0)
            return true;
    return false;
}

  // Determine numbers of flowers and bees between s1 and s2
void countMatches(const char s1[], const char s2[], int& nFlowers, int& nBees)
{
    const char FLOWER_BLOT = '#';  // any nonword character will do
    const char BEE_BLOT    = '@';  // any nonword character will do

    char s1copy[MAXWORDLEN+1];
    strcpy(s1copy, s1);

      // First, count and blot out the flowers so they won't be matched later.
      // Check every position for a flower, stopping at the end of the
      // shorter string

    nFlowers = 0;
    int k;
    for (k = 0; s1copy[k] != '\0'  &&  s2[k] != '\0'; k++)
    {
        if (s1copy[k] == s2[k])
        {
            nFlowers++;
            s1copy[k] = FLOWER_BLOT;
        }
    }
```

```
    int shorterLen = k;

      // Now count the bees.  For every character in s2

    nBees = 0;
    for (int k2 = 0; s2[k2] != '\0'; k2++)
    {
          // Don't count flowers

        if (k2 < shorterLen  &&  s1copy[k2] == FLOWER_BLOT)
            continue;

          // For every character in the copy of s1

        for (int k1 = 0; s1copy[k1] != '\0'; k1++)
        {
              // If they match, blot it out of the copy of s1
              // so it won't be matched again

            if (s1copy[k1] == s2[k2])
            {
                nBees++;
                s1copy[k1] = BEE_BLOT;
                break;
            }
        }
    }
}
```