✓

C++ PROGRAMMING
(335)

REGIONAL – 2018

Production Portion:

Program 1: Pass Phrase Generator	_______________	(340 points)
TOTAL POINTS	_______________	***(340 points)***

Graders: Please double check and verify all scores and answer keys!

✓

Generating Pass Phrases

One of the main issues with standard password generation techniques is that they can prove too difficult to remember. This can force the user to store passwords using management tools that can easily be lost or forgotten, or found by others (i.e. an app or on paper).

If words are added to a string randomly, they can produce memorable yet complex passwords that are difficult to crack. For example, “InvisibleCrown” is easy to remember, but will still take some time for a brute-force cracking approach to find.

Requirements:

1. You must create a C++ console application named CPP_335_ContestantNumber, where ContestantNumber is your BPA assigned contestant number (including dashes). For example, CPP_335_01_2345_6789.
2. Your contestant number must appear as a comment at the top of the main source code file.
3. A function, named “getWordsFromLibrary” must be used to obtain the list of words from the library, which must also display an error and close the application if there is an error loading the library. The library is 20 words and is stored in “lib.txt”, but the function should be able to load a library of any reasonable number of words. Expect that the format will always be one word per line.
4. The getWordsFromLibrary function must populate a data structure (i.e. array or vector) of your choice. This data structure will be used throughout the rest of the program, but it cannot be set as a global variable.
5. The program must prompt the user to enter a positive integer that indicates the number of words to be generated to populate a pass phrase.
6. Each word in the generated pass phrase must be unique, i.e. “InvisibleInvisible” is unacceptable.
7. If the user inputs a number of words greater than the number of words, the unique word restriction is ignored, i.e. “InvisibleInvisible” is acceptable.

The output of the program must look similar to the following.

Sample Output:

```
Please enter the number of words in the phrase: 2
The result phrase: CrownBlunder
```

You will have ninety (90) minutes to complete your work.

Your name and/or school name should *not* appear on any work you submit for grading.

✓

Save a copy your entire solution/project to the flash drive provided. You must submit your entire solution/project so that the graders may open your project to review the source code and/or build and execute your solution/project. **Submissions that do *not* contain source code will *not* be graded**.

Development Standards

- Standard name prefixes must be utilized for variables.
- All subroutines, functions, and methods must be documented with comments explaining the purpose of the method, the input parameters (if any), and the output (if any).

✓

Your application will be graded on the following criteria:

Solution and Project

The project is present on the flash drive	____ 10 points
The project is named according to the naming conventions	____ 10 points

Program Execution

Code copied to flash drive and program runs from flash drive	____ 20 points

If the program does not execute, then the remaining items in this section receive a score of zero.

The program rejects negative numbers on word count input	____ 25 points
The program runs and produces random pass phrases	____ 20 points
The program runs and produces valid output for unique phrase	____ 25 points
The program runs and produces valid output for phrase with too many words	____ 25 points
The program runs and display error message library load fails	____ 25 points
The output matches the sample output in format and alignment	____ 50 points

Source Code Review

Code is commented at the top, for each function, and as needed	____ 15 points
Code uses reasonable and consistent variable naming conventions	____ 15 points
A data structure is used to store all the words in the library	____ 25 points
A function is used to populate the data structure of strings that is named `getWordsFromLibrary`	____ 50 points
The loading function can accept a library file of any number of words	____ 25 points

Total Points: 340 points

✓

```
#include <iostream>                          // used for console io
#include <fstream>                           // used for file io
#include <cstdlib>                           // allows use of srand()
#include <ctime>                             // used to initialize random number
generator
#include <string>                            // used for str.find() and string::npos

using namespace std;

const int wordsInLibrary = 20;          // Number of words stored in the library file
const string libPath = "lib.txt";  // Path to the library file

// Function prototype allowing easy retrieval of all words from a text file
// Results are stored in words[]
void getWordsFromLibrary(string words[]);

int main()
{
       int wordCount;                              // how many words to be added to
the phrase
       bool unique = true;                   // if each word in phrase is unique
       string result;                              // the result pass phrase
       string words[wordsInLibrary];      // stores all words from the library file

       getWordsFromLibrary(words);        // retrieves all words from library

       if (words[0] == "")                   // if first word is missing,
       {
              system("PAUSE");               // suspend application then close with
error code
              return 1;
       }

       srand((unsigned)time(0));          // initalizes random number generator

       cout << "Please enter the number of words in the phrase: ";
       cin >> wordCount;                     // retrieves # of words in phrase

       while (wordCount <= 0)
       {
              cout << "Please enter a positive number: ";
              cin >> wordCount;
       }

       if (wordCount > wordsInLibrary)
              unique = false;                      // if too many words requested,
cannot be unique

       while (wordCount-- > 0)
       {
              int random;                          // stores index of word to add to
phrase
              if (unique)
                     while (result.find(words[random = (rand() % wordsInLibrary)])
                            != string::npos) {} // until we find a word not in the phrase,
                                                              // keep randomizing the
index
              else
                     random = rand() % wordsInLibrary;
                                                              // get a value from 0 to
wordsInLibrary

              result += words[random];    // add word to phrase
       }

       cout << "The result phrase: " << result << endl << endl; // output final result

       system("PAUSE");                          // suspend application until any key
press
       return 0;
}

// Function prototype allowing easy retrieval of all words from a text file
// Results are stored in words[]
void getWordsFromLibrary(string words[])
{
       string line = "";
       ifstream lib(libPath);      // open file

       int index = 0;                     // index to store line at
       if (lib.is_open())          // if open, read each line and add to words
              while (lib >> line)
                     words[index++] = line;
       else
              cout << "Error reading library, closing application." << endl << endl;
}
```