Project One: Hangman

Out: May 25, 2018; Due: June 3, 2018

Motivation

This project will give you experience in using basic C++ constructs including I/O, arithmetic operators, branch, and loop.

Introduction

Hangman is a guessing game. The player needs to guess a word by suggesting letters. At the beginning of the game, a random word is selected from a dictionary and displayed as a sequence of dashes, one for each letter of the word, separated by a space for legibility. For instance, for the word "`hangman`":

```
_ _ _ _ _ _ _
```

During one turn, the player chooses one letter. If it appears in the word, the display is updated and shows where it appears in the word. For instance, if 'a' is selected:

```
_ a _ _ _ a _
```

If the letter does not appear in the word, the score, which represents the number of erroneous trials, is increased by one. The initial score is zero.

When the score reaches 10, the game is over and the player has lost. If after the letter selection all the letters of the word have been revealed, the player has won. Otherwise, the player chooses another letter.

Programming Assignment

You will implement a program for this game.

Input/Output

Your program should first display a randomly chosen word (called W) with each of its letters replaced by dashes. Then it will repeatedly prompt for a character until it reads a letter from 'a' to 'z' (it only accepts lowercase letters) that have not been proposed before:

```
"Please enter a letter (a-z): "
```

The score is incremented by one if the read letter does not appear in word W. If the score reaches 10, the word is revealed and the game ends with the following message `"You lost!"`. If all the letters of word W have been guessed, the word is revealed and the game ends with the following message `"You won!"`. Otherwise, your program outputs the state of the game: the score is printed, then the word is displayed again such that each letter that has been guessed is printed normally and the other letters are replaced by '_', and finally your program also displays all the characters that have been proposed and that do not belong to the word (in alphabetic order, separated by one space). Then the program loops and asks for another letter.

Note: Our online judge ignores the trailing spaces in a line. Thus, it does not matter whether in your output you put a space or not at the end of any line. If the user inputs an illegal character, it is ignored and the program outputs the current state of the game (see examples below).

Here is one example of a possible play:

```
Score: 0

_ _ _
Already proposed characters:


Please enter a letter (a-z): a
Score: 1

_ _ _
Already proposed characters:
a
Please enter a letter (a-z): e
Score: 1

_ e _
Already proposed characters:
a
Please enter a letter (a-z): {
Score: 1

_ e _
```

```
Already proposed characters:
a
Please enter a letter (a-z): E
Score: 1
_ e _
Already proposed characters:
a
Please enter a letter (a-z): i
Score: 2
_ e _
Already proposed characters:
a i
Please enter a letter (a-z): e
Score: 2
_ e _
Already proposed characters:
a i
Please enter a letter (a-z): s
Score: 2
_ e s
Already proposed characters:
a i
Please enter a letter (a-z): y
The word was: yes
You won!
```

Here is another example of a possible play:

```
Score: 0
```

```
_ _ _
Already proposed characters:


Please enter a letter (a-z): a
Score: 1
_ _ _
Already proposed characters:
a
Please enter a letter (a-z): b
Score: 2
_ _ _
Already proposed characters:
a b
Please enter a letter (a-z): c
...
Score: 9
_ e _
Already proposed characters:
a b c d f g h i j
Please enter a letter (a-z): h
The word was: yes
You lost!
```

Implementation Requirements

You should put **all** the functions you write in a single file, called `p1.cpp`. You may only include `<iostream>` (e.g., `cin`, `cout`). No other system header files may be included, and you may not make any call to any function in any other library.

The list of words used by your program is provided in a header file `p1.h` that we provide. It defines two variables `words` and `nWords`. The former corresponds to an array of C strings that contains all the words your program can use and the latter corresponds to the number of words. Therefore, you should have the following line `include "p1.h"` on the top of your file `p1.cpp`.

The header file `p1.h` also includes functions (declared and defined in `rand.h` and `rand.cpp`, which are also provided) to generate (pseudo)random numbers. To select a random integer, you can use the function `p1_rand()` that returns a (pseudo)random `int`. For example, to obtain a number between 1 and 100, you can use `p1_rand()%100+1`. Before using the function `p1_rand()`, the function `p1_srand(seed)` is generally called with an integer `seed` to initialize the pseudorandom sequence. Parameter `seed` is generally taken as equal to `time(NULL)` from the `ctime` library. However, to simplify testing, we require you to ask the user to input the seed at the start of the program (after printing this message first: "`Please input seed: `") and then make the following call just after getting the input `seed` from the user: `p1_srand(seed)`. This way, Online Judge can control the selection of the word.

Recall that a character is encoded in ASCII (e.g., `'a' == 97` or `'<' == 60`) and can also be seen as a number between 0 and 255. Therefore `'b' - 'a' == 1`.

Compiling and Testing

To compile, type the following Linux command:

```
g++ -Wall -o p1 p1.cpp rand.cpp
```

You should test your program extensively.

Submitting and Due Date

You need to submit `p1.cpp`, `rand.cpp`, `p1.h` and `rand.h`. Please name `p1.cpp` exactly like this, and **do not** modify the codes we provide. They should be submitted via Online Judge system. Please see the announcement from the TAs for details about submission.

The due date is 11:59 pm on June 3rd, 2018.

Grading

Your program will be graded along three criteria:

1. Functional Correctness

2. Implementation Constraints

3. General Style

An example of Functional Correctness is whether or not you produce the correct output. Implementation Constraints checks whether you stick to the implementation requirements. General Style speaks to the cleanliness and readability of your code. We don't need you to follow any particular style, as long as your style is consistent and clear. Some typical style requirements include: 1) appropriate use of indenting and white space, 2) program appropriately split into subroutines, 3) variable and function names that reflect their uses, and 4) informative comments at the head of each function.