CSE 241 Programming Assignment 1

DUE

March 18, 2025, 23:55

Description

- This is an individual assignment. Please do not collaborate. You have to do a demo in order to get credit.
- If you think that this document does not clearly describe the assignment, ask questions before its too late.

This is the first assignment of the semester. This will get you ready for the upcoming assignments. Carefully execute every step.

You are going to implement an interactive program which enables user to play the game *SECRET*. The idea of the game is, one side holds a secret word and the other side(user) proposes words and gets useful hints at each turn. The aim of the user is to guess the secret word.

The Rules of The *SECRET*

- A character is a char element of an alphabet. Your program generates this list from a file called `alphabet.txt`.
- A valid word is defined to be an `N` character word (N $\leq$ length_of_the_alphabet) where **any character appears only once in the word**.
- Your program chooses a valid word (secret word).
  - First option is to generate a random word (it must be a valid word)
  - Second option is to use a given word (a word will be provided as a command-line argument)
- (At each turn) the code breaker (user) proposes a valid word. Turns are counted (Starting with 1).
- As response to the proposal, the code maker(your program) provides two counts:
  - **First count** (C_{exact}): The count of characters of the proposed word that match in place of the secret word.
  - **Second count** ($C_{misplaced}$): The count of characters of the proposed word which do exist in the secret word but are not in place.
- A C_{exact} value of `N` stops the game and the turn-count is recorded; otherwise the game continues with accepting user proposals.

Expectations

alphabet.txt

- the alphabet data is just a text file in the following format

```
***   ****
   **** ****
*** * *** ***
****  *** * * ** *
```

Here, `*` is a char from ASCII table. `,` (comma excluded), ` ` (space excluded)

- Example:

```
rrrrrv 45 6.x
  btcb gg:
```

- `alphabet = [r,v,4,5,6,.,x,b,t,c,g,:]`

Input

- Your program will take command-line arguments.
  - The first argument is either `-r` or `-u`.
  - If the first argument is `-r`, user has to provide another argument `N`: The number of characters. Your program will create a random word which has `N` characters.

  - If the first argument is `-u`, user has to provide another argument which will be used as a secret word.
- At each turn user enters a word, your program has to capture the word correctly, compare it with the secret and print hints.

Output

- **Initially, your program prints the alphabet**.
- At each iteration, print hints.
- If the user input perfectly matches with the secret word, print the `FOUND` message and number of iterations and exit.

Example Run

Suppose that your source file is `mysource.cpp`.

First, it will be compiled:

```
g++ -std=c++11 mysource.cpp -o SECRET
```

Then your program can be called like the following:

```
SECRET -r 6
```

or

```
./SECRET -r 6
```

With this call, user is expected to enter a word consisting of `6` characters.

If there is an unexpected form of input, your program should print the following and exit

```
INPUT ERROR
```

If the user enters a valid word your program should return the hints in the following format:

```
SECRET -r 4                 ------> Assume that your program generates secret helo
[x,y,a,t,b,c,h,f,e,k,o,l]   ------> Your program prints the alphabet read from alphabet.txt
hteo                        ------> User enters hteo
2 1                         ------> First count is 2, Second count is 1 (separated by a space)
heto                        ------> User enters heto
3 0                         ------> First count is 3, Second count is 0 (separated by a space)
helo                        ------> User enters helo
FOUND 3                     ------> User found the word in 3 iterations, program exits.
```

If the user cannot find in `100` iterations, print the following message and exit the program:

```
FAILED
```

Your program can also be called like the following:

```
SECRET -u abc
```

If this happens, your program will use abc (the given argument) as the secret word. The rest of your code will not change.

Error Checking

Errors in program call include the following:

- Missing parameters.
- Wrong parameters.
- Undefined parameters.
- Entering characters which don't belong to the alphabet
- Words of with non-unique characters

Remarks

- Error checking is important.
- Your program should be immune to the whitespace before any user input.
- Do not submit your code without testing it with several different scenarios.
- Write comments in your code. Extensive commenting is required. Comment on every variable, constant, function and loop. (`10pts`)
- Do not use `#define` and define macros. Instead use `const` keyword and define constant variables. If you use macros, you will loose `5` points for each of them.
- Do not create an output file. Everything happens through `stdin` and `stdout`.
- Be very careful about the input and output format. Don't print anything extra(including spaces).

Turn in:

- Source code of a complete C++ program. Name of the file should be in this format: `<full_name>_PA1.cpp`. If you do not follow this naming convention you will loose `-10` points.
- Example: `deckard_cain_PA1.cpp`. Please do not use any Turkish special characters.
- You don't need to use an IDE for this assignment. Your code will be compiled and run in a command window.
- Your code will be compiled and tested on a Linux machine(Ubuntu). GCC will be used.
- Make sure you don't get compile errors when you issue this command : `g++ -std=c++11 <full_name>_PA1.cpp`.
- A script will be used in order to check the correctness of your results. So, be careful not to violate the expected output format.
- Provide comments unless you are not interested in partial credit. (If I cannot easily understand your design, you may loose points.)
- You may not get full credit if your implementation contradicts with the statements in this document.

Late Submission

- Not accepted.

Grading (Tentative)

- `Max Grade` : `100`.
- Multiple tests will be performed.

All of the followings are possible deductions from `Max Grade`.

- Do **NOT** use hard-coded values. If you use you will loose `10pts`.
- No submission: `-100`.
- No show/ failed demo / failed to explain ANY part of the code submitted = No submission.
- Compile errors: `-100`.
- Irrelevant code: `-100`.
- Major parts are missing: `-100`.
- Unnecessarily long code: `-30`.
- Inefficient implementation: `-20`.
- Using language elements and libraries which are not allowed: `-100`.
- Not caring about the structure and efficiency: `-30`. (avoid using hard-coded values, avoid hard-to-follow expressions, avoid code repetition, avoid unnecessary loops).
- Significant number of compiler warnings: `-10`.
- Not commented enough: `-10`. (Comments are in English. Turkish comments are not accepted).
- Source code encoding is not `UTF-8` and characters are not properly displayed: `-5`. (You can use 'Visual Studio Code', 'Sublime Text', 'Atom' etc... Check the character encoding of your text editor and set it to `UTF-8`).
- Missing or wrong output values: `Fails the test`.
- Cannot produce numbers with unique characters: `Fails the test`.
- Output format is wrong: `-30`.
- Infinite loop: `Fails the test`.
- Segmentation fault: `Fails the test`.
- Fails 5 or more random tests: `-100`.
- Fails the test: `deduction up to 20`.
- Prints anything extra: `-30`.
- Unwanted chars and spaces in output: `-30`.
- Submission includes files other than the expected: `-10`.
- Submission does not follow the file naming convention: `-10`.
- Sharing or inheriting code: `-200`.