CSE 241 Programming Assignment 1

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 *mastermind*. The idea of the game is, one side holds a secret number and the other side(user) proposes numbers and gets useful hints at each turn. The aim of the user is to guess the secret number.

The Rules of The *Mastermind*

- A digit is an integer in the range `[0,9]`
- A valid number is defined to be an `N` digit number ($N \leq 9$) where the high-most (leftmost) digit cannot be `0` and **any digit appears only once in the number**.
- Your program chooses a valid number (secret number).
  - First option is to generate a random number (it must be a valid number)
  - Second option is to use a given number (a number will be provided as a command-line argument)
- (At each turn) the code breaker (user) proposes a valid number. 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 digits of the proposed number that match in place of the secret number.
  - **Second count** ($C_{misplaced}$): The count of digits of the proposed number which do exist in the secret number 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

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

Output

- At each iteration, print hints.
- If the user input perfectly matches with the secret number, print the `FOUND` message and number of iterations and exit.

Example Run

Suppose that your source file is `student_name_PA1.cpp`.

First, it will be compiled:

```
g++ -std=c++11 student_name_PA1.cpp -o mastermind
```

Then your program can be called like the following:

`mastermind.exe -r 6`

or like the following if you are using a unix-like OS

`./mastermind -r 6`

With this call, user is expected to enter a `6` digit number. If user enters a number which has more or less than `6` digits, your program should print the following error message and exit.

`E1`

If the user enters something but not an integer, your program should print the following error:

`E2`

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

```
mastermind -r 6   ------> Assume that your program generates secret 130456
123456            ------> User enters 123456
4 1               ------> First count is 4, Second count is 1 (separated by a space)
132456            ------> User enters 132456
5 0               ------> First count is 5, Second count is 0 (separated by a space)
130456            ------> User enters 130456
FOUND 3           ------> User found the number in 3 iterations, program exits. (separated by a space)
```

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:

`mastermind -u 12345`

If this happens, your program will use 12345 (the given argument) as the secret number. The rest of your code will not change.

Error Checking

Check for any errors in program call. If there is any error, print the following and exit:

`E0`

Errors in program call include the following:

- Missing parameters.
- Wrong parameters.
- Undefined parameters.
- Negative value or `0` value following the `-r` option.
- Digits of the number followed by `-u` are not unique.

The other errors are specified in the `Example` section.

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 `constant` 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).