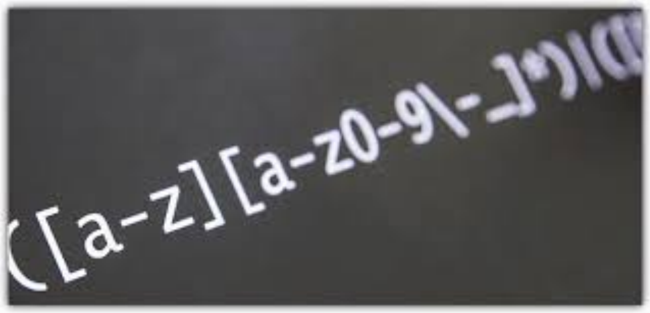
Project 8: `CodewordGenerator`



For this project you will implement a **codeword generator**.
A **codeword** in this context is any permutation of characters from an alphabet.
For this project, our **alphabet** is all uppercase letters **{ A, B, C, … ,Z}**
Our generator will create only codewords that match a given **pattern.**

Background - Regular Expressions:

A regular expression (or regex for short) is a special text string for describing a search pattern. You are probably familiar with the concept of wildcard and wildcard notations such as *.txt to find all text files in a file manager. This is a pattern: *"all strings that end in .txt".*
With regular expressions you can specify complex patterns. For example "[a-z0-9._% +-]+@[a-z0-9.-]+\.[a-z]{2,4}" would match any email address.
Lets break it down:

- The square brackets indicate a character class, and match (can be substituted by) any of the characters between the brackets
- [a-z0-9._%+-] matches any lowercase letter a-z or any digit 0-9 or period, underscore, percent, plus or minus symbol. So it would match any of the following: "a", "a9b", "-a9z1%azz54_". But it would not match "a&b9"
- + after a character class means "1 or more", so [a-z0-9._%+-]+ means *"any permutation of lowercase letters, digits and . _ % + _ of length 1 or more"*
- [a-z0-9._%+-]+@ matches the above followed by @

- … and so on, so [a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,4} means *"any permutation of lowercase letters, digits and . _ % + _ of length 1 or more followed by @ followed by one or more lowercase letters and/or digits and/or dot and/or dash followed by a dot followed by two to four lowercase letters"*
- A * instead of a + after a character class means "*0 or more of the characters in the character class"*

In C++ you can use the `<regex>` library to look for patterns in strings.

- To create a regular expression object given a string patters such as the one described above
  ```
  std::regex re("[a-z]+aba[a-z]*");
  ```

- To check if a string s matches the pattern
  ```
  std::regex_match(s, re)
  ```

  For example

  ```
  std::string s = "caba";
  std::regex_match(s, re) //true
  ```

will return true, while

```
std::string s = "aba";
std::regex_match(s, re) //false
```

will return false.

For this project this is enough knowledge of regular expressions, however I do encourage you to explore further, regex can be a very useful and powerful tool for future projects.

Implementation:

You will implement a class named `CodewordGenerator`.
You may design this class as you wish, but it MUST have the following two **public** methods:

- `std::string generateShortestWord(std::string pattern);`
  takes a string pattern and returns the shortest string that matches the pattern exploring the space of possible alphabet permutations in **Breadth First Search** order.
  The input string pattern is guaranteed to represent a valid regular expression pattern.

- `std::string generateLengthWord(std::string pattern, int length);`
  takes a string pattern and an integer n and returns a string of length n that matches the pattern exploring the space of possible alphabet permutations in **Depth First Search** order.
  Here too, the input string pattern is guaranteed to represent a valid regular expression pattern.

We discussed BFS and DFS algorithms using queue and stack in the Queue ADT Lecture19 slides (you may refer to the lecture slides for pseudocode)

Order of exploration of the permutation space is important for correctness!
If you return a string that matches the pattern but is not the one expected when exploring the space in BFS or DFS you will get partial credit.
The **alphabet** is all uppercase letters **{ A, B, C, ... ,Z}**

You may add as many private functions and data members as necessary.

Testing:

Beware of the complexity of these algorithms when testing. BFS will not handle very long strings. DFS will do better, but still may require excessive computation with certain patters. Test with simple patterns but make sure to cover different possibilities.

Submission:

There are no starter files for this project, but we will use GitHub classroom anyways. You will find the link on Blackboard under Course Materials/Project8. Please accept the assignment and work using the repo. I strongly encourage you to get into the habit of working with version control (which here translates to: it would be nice to see many more commits!!!).
Your project must be submitted to Gradescope through GitHub. The due date is Tuesday December 3 by 5pm. No late submissions will be accepted.

Have Fun!!!!!