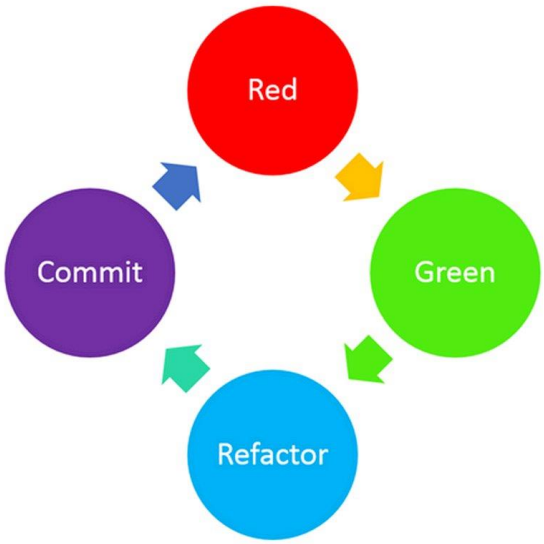
# Design Spec

## Introduction

As the behavior of this requirement (input and output) is very clear, it is fit for Test Driven Design - TDD. So test cases will be designed and created first, then implement these methods, followed by refactor the code. Additionally, there is no complex class or other component that need to be instantiated, mock is not necessary here. If there are further requirements coming, we should follow an iterative workflow as:  Create Test Cases(Red) -> Implement features and make all test cases pass(Green) -> Refactor the code(Blue) -> Commit to git(Purple) -> Create Test Cases with new features.




## Limitation

1. It is very useful to implement the feature that generates the string list based on an original string. However, it will out of scope for this release
2. As verifying the result using super long string and tens of or hundreds of strings in string list will be very hard to do manually, only short string and a couple of strings in the list will be used in the test cases to verify the correctness of the app.
3. If this is a normal project, I will do one commit per test case for code quality purpose, but as this is a test, to save time, I did batch commit.

## Assumption

1. Assume file extension is .txt, each line only has one string
2. If two strings have no overlap, it will be concatenated as the order of their positions of input parameter. For instance, ConcatTwoString(s1,s2, overlap) will get s1 + s2 instead of s2 + s1.
3. If two strings have more than one overlaps, only the first overlap will be used. For example, the overlap used of "abcde" and "abdef" will be "ab", not "de".

4. When concatenating two strings, they are aligned using the overlap substring, if both strings have prefix and suffix, the shorter one will be used. For instance, the concatenated string of "abcdefg" and "xcdy" will be "xcdy".
5. When one string has 2 overlaps, use the first one to align with other string and concatenate.

## Structure

This feature can break down into 4 steps underneath, and step 2 is the most crucial part, the performance of using different algorithms may vary dramatically. There are typically 2 ways of doing it:

1. One is the brute force by recursively comparing all substring of 2 strings, the complexity is $O(2^{m+n})$, refers to here.
2. A better way is dynamic programming that checks the common suffix, the complexity is O(m*n), refers to here.

## Steps

1. Read file to string list.
2. Loop in the string list, compare every 2 strings, find the maximum overlapped substring and corresponding 2 strings.
3. Concatenate these 2 strings to a new string using max overlapped string, insert it into the string list and remove the original 2 strings.
4. If the Count of string list is 1, output the last string, otherwise, go back to Step 2.

## Pseudo Code

```
string FindCommonString(List<string> stringList){

        while (stringList.Count>1){
                For loop in stringList {
                    FindLongestSubstring(s1,s2)
                }
                concatString = ConcatTwoString(s1,s2,overlap)
                stringLIst.Add(concatString)
                stringList.Remove(s1)
                stringList.Remove(s2)
        }

        return stringList[0];
}
```

## Test Cases

```
/// Method that finds the maximum overlapped string
///String FindLongestSubstring(String s1, String s2)
//1 overlap String, return the overlapped string
s1 = "abcd"
s2 = "cdef"
result = "cd"
```

// No overlap, return null
s1 = "abcd"
s2 = "efgh"
result = null

//2 overlap strings, return 1 of 2 Strings
s1 = "abcd"
s2 = "cdab"
result = "ab" or "cd"

//s1 contain s2, return s2, or s2 contains s1, return s1
s1 = "abcd"
s2 = "bc"
result = "bc"

s1 = "bc"
s2 = "abcd"
result = "bc"

//s1 contains 2 overlap substring, return the first substring
S1="abcdabef"
S2="abh"
Result = "ab"

//s1 is null or s2 is null, or both null
s1 = null
s2 = null
result = ArgumentNullException

/// Method that concatenates 2 strings by providing overlapped string between them
///String ConcatTwoString(String s1, String s2, String overlap)
//has overlap
s1 = "abcde"
s2 = "bcfgh"
overlap = "bc"
result = "abcfgh"

// No overlap
s1 = "abcde"
s2="fghij"
overlap=null
result = "abcdefghij" or "fghijabcde"

// s1 is null or s2 is null, or both null
s1 = null
s2 = null
overlap=null
result = ArgumentNullException

// s1 contains s2, return s1; s2 contains s1, return s2;

S1 = "abcd"
S2 = "bc"
overlap = "bc" = s2
result = s1

s1 = "bc"
s2 = "abcd"
overlap = "bc" = s1
result = s2

//s1 contains 2 overlap substring, use the first overlap for concatenation
S1="abcdabef"
S2="abh"
result = "ab"

//when align s1 and s2 by the overlap, both the prefix and suffix of s1 are longer than s2, e.g. "abcdefg","xcdy", should return shorter one "xcdy"

//when s1 has 2 overlaps, use the 1st one to align with s2
s1 = "abcdgcd";
s2 = "cdef";
overlap = "cd";
result = "abcdef";

///The wrapper method, accept a string list and return the common string
///String FindCommonString(List<string> stringList)
//stringList is null stringList.Count is 0
stringList = null
stringList.Count = 0
result = ArgumentNullException

//provide 3 strings that has overlap, return common string
stringList = {"abc","bcd","cde"}
result = "abcde"

//provide 3 strings that don't have overlap, return concatenate of all 3 strings in any order
stringList = {"abc","def","ghi"}
return = "abcdefghi" or "abcghidef" or "defabcghi" or "defghiabc" or "ghiabcdef" or "ghidefabc"