CSE101 Stack Recursion Test Questions: Eschew self-reference

- All code must be written in C/C++.
- Please be careful about using built-in libraries or data structures. The instructions will tell you what is acceptable, and what is not. If you have any doubts, please ask the instructors or TAs.

1 Setting

You already have recursive and stack-based code that prints all subsequences of an input string.

The I/O format has been fixed. The input and output files are given as command line arguments. The first line of the input file is read into a string. This is processed by a function to produce a list of strings as output. This list is sorted and then printed to a file. This is exactly the setting you will see in your test.

Here are some example test questions for the next test. You may get one of these questions, or some variant of these questions. Pay attention to the declaration in these examples. *You cannot use recursion to implement these functions. You must use a stack based implementation.*

1.1 Example test questions

For your test, you will be given one function to code up. In all cases, you can assume that input is purely alphanumeric and that the argument $k \geq 0$. *You can also assume that no characters are repeated.* You can generate the output in any order, but each output string should only appear once.

- `List allAnagrams(string input)`. An *anagram* of a string is just a permutation of the string. You must write a function that produces all anagrams of the input string in a list. The order does not matter. For example, if the input string was "abc", the function should output a list containing the following strings: abc, acb, bac, bca, cab, cba.
- `List language(string input, int k)`. You will see a lot about *languages* when you study Computational Models. Just think of the input string

as a set of characters Σ (formally called an *alphabet*). The language generated by Σ is the set of all strings comprising of characters (symbols) in Σ. This function generates a list of all strings in the language, whose length is at most k. Note that empty string is also part of the language. The order of the list does not matter. For example, if the input is "abc" and $k = 2$, then the output list has the following strings: (the empty string), a, b, c, aa, bb, cc, ab, ac, ba, bc, ca, cb. You can assume that input does not have repeated characters.

• `List stretch(string input, int k)`. A *stretch* of the input string is generated by repeating each character in order *up to* k times (and at least once). For input string "abc" and $k = 2$, the output list should have: abc, aabc, abbc, abcc, aabbc, aabcc, abbcc, aabbcc. (Again, the order does not matter.)

• `List bubbling(string input, int k)`. A *bubbling* of the input string is generated as follows. Iterate over every index (except the last index) of the string: in the ith iteration, you can swap the ith and $(i+1)$th characters. (You can also choose to not swap.) Any sequence of such swaps leads to a string, called a *bubbling* of the original string. For example, suppose the input string is "abc". Here are the bubblings: abc, acb, bac, bca. (These were achieved, respectively, by: no swaps at all, no swap and swap, swap and no swap, swap and swap.) The order of the output does not matter.

1.2 How it works

You will get a stackrecursion.cpp file with the right function declaration. Your job is to simply code up the function (that you will only get one as a test question).

You will also get a small test input and test output. To make life easier, I will actually give you access to my grading scripts (that you can run directly through a shell script). If it catches a bug in your code, it will give you a test input where your code file. So you will get to see your score *before* you submit.

The test is completely closed book. You cannot bring any written material, but you can get one blank sheet of scratch paper. ou will get and can only open the specific Codio box for the test. You cannot refer to any other codes, or even open any other Codio box. If you do so, it will be considered cheating and you will get -10 points.

1.3 What should you do

Start by writing recursive code for these function. You should be able to do so within 20 lines of code (probably less). Then, convert this recursive code into code that uses a stack. The stack will store objects corresponding to each recursive call. Note that you may want to construct some extra objects (like the Pair used in the current code), though it is possible to avoid that. (Instead of creating a stack of Pairs, I could have used a stack of strings. Each push/pop should actually be two pushes/pops, to get both the in_str and the fixed_str. Think about it.)

This test uses a bit more C++. Firstly, you need to manipulate the strings. If you find that painful, you can copy it into a character array, and then process this array. (You can see the function that does this in my listwrapper code, just before calling `strtok`.) Also, you can use the inbuilt C++ stack, so you need to see how to create a stack of any desired object. This is called *templating*. This is nothing too fancy, you simply specify the type in the stack definition. Look at my code to understand, and play around with it.

1.4 Other questions

A few other questions, that might be good practice. (And great interview practice.)

- For convenience, we assumed that no characters in the input string are repeated. What if there are repeated characters?
- For each of the functions, can you write the formula for the *size* of the output (or the length of the output list)?
- Can you change the order of the stack pushes? If you do, what happens?
- Suppose you wanted to store the anagrams in lexicographic order directly. How would you do that?
- Can you use a queue instead of a stack? (In some cases you might luck out and get the right output, but it is not a faithful simulation of recursion.)
- If you really wanted more coding practice, you could write your own stack. Actually, you can just modify the existing list code to simulate your stack. As we discussed in class, the insert is already like a push.