Name: Shuqing Ye **UCI NET ID:** shuqiny2

1) (100 pts) given a list of **unique** strings, where each of them has a fixed length of L, write a C++ or Java function that finds the longest palindrome string made by concatenating the strings in the input list. The input strings consist of only lowercase letters. If there is no such string, it should return an empty string. if there are multiple answers, return just one of them. You may write multiple helper functions. Example, if the input is [“xyz”,“zyx”,”aaa”, “bcd”], then the output will be “xyzaaazyx”. Please write your code in two columns to give yourself maximum vertical space. **Please do not write #includes or imports.** please write **ONE line** of comment above each function. Reduce the size of your font to fit your code in one column to make it easier to read.

Test cases (including the edge cases): [], [“a”, ”b”], [“aa”,”a”], [“aba”, “ba”], [“aaa”, “bbb”], [“aaa”,”a”,”b”], [“xyz”,“zyx”,”aaa”, “bcd”]

```
// public method
public String getLongestPalindrome(List<String> L) {
        if (L.size() == 0)
                return “”;
        // stores a String that itself is a palindrome
        List<String> selfPalindrome = new ArrayList<>();

        // store a pair of strings that are palindromes with
each other
        List<String> pairPalindrome1 =
                                        new ArrayList<>();
        List<String> pairPalindrome2 =
                                        new ArrayList<>();



        for (int i = 0; i < L.size(); i++) {
                String s1= L.get(i);
                for (int j = i; j < L.size(); j++) {
                        String s2 = L.get(j);
                        if (isPalindrome(s1, s2)) {
                                pairPalindrome1.add(s1);
                                pairPalindrome2.add(s2);
                        else if (isPalindrome(s1 + s2))
                                selfPalindrome.add(s1+s2);
                        else if (i == j && isPalindrome(s1))
                                selfPalindrome.add(s1);
// check if a string can pair with others before check if
itself is a palindrome
                        }
                }  // end of inside for loop
        } // end of outside for loop

        return print(selfPalindrome,
                pairPalindrome1, pairPalindrome2);
}

// T(n) = o(n^2)
```

```
// judge whether s is a palindrome
private boolean isPalindrome(String s) {
        int i = 0, j = s.length() – 1;
        for ( ; i< s.length() && j >= 0; i++, j--){
                if (s.charAt(i) != s.charAt(j)) {
                        break;
                }
        }
        return (i == s1.length() && j == 0);
}


// judge whether s1, s2 are palindromes
private boolean isPalindrome(String s1, String s2) {
        if (s1.length() != s2.length())
                return false;

        int i = 0, j = s2.length() – 1;
        for ( ; i< s1.length() && j >= 0; i++, j--){
                if (s1.charAt(i) != s2.charAt(j)) {
                        break;
                }
        }
        return (i == s1.length() && j == 0);
}
```

```
// concatenation of palindromes (like a sandwich)
Private String print(List<String> selfPalindrome,
                    List<String> pairPalindrome1,
                    List<String> pairPalindrome2 ) {

        StringBuffer res = new StringBuffer();

        If (pairPalindrome1.size() == 0)
        for (String s: pairPalindrome1)
                res.append(s);

        // select the longest one from selfPalindrome
        String longestString = “”;
        For (String s: selfPalindrome) {
                If (s.length() > longestString.length())
                        longestString = s;
        }
        res.append(longestString);

        for (int  i = pairPalindrome2.size() – 1; i>=0 ; i++)
                res.append(pairPalindrome2.get(i));

        return res.toString();

}
```