

CSE222 - Homework4 – Question3

Report

Nevzat Seferoglu

171044024

Problem Solution Approach :

* Problem Definition :

Define each of the following problem recursively:

- Identify the base case (or base cases) that stops the recursion
- Define the smaller problem (or problems)
- Explain how to combine solutions of smaller problems to get the solution of original problem

Write recursive methods for the problems.

1. Reversing a string. For example, if the input is “this function writes the sentence in reverse”, then the output should be “reverse in sentence the writes function this”.
2. Determining whether a word is elfish or not. A word is considered elfish if it contains the letters: e, l, and f in it, in any order. For example, white leaf, tasteful, unfriendly, and waffles are some elfish words.
3. Sorting an array of elements using selection sort algorithm.
4. Evaluating a Prefix expression
5. Evaluating a Postfix expression
6. Printing the elements of an array on the screen as in the example below.

Input:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20

Output: 1 2 3 4 8 12 16 20 19 18 17 13 9 5 6 7 11 15 14 10

* Approach :

Part1:

For reversing given string in according to its sentence , there is an **indexOf()** methods which can determine the first occurrence of space character and **substring** that can return a new string denoted by given index. Combining two of them with recursion call.

```
public static String reverseSentence( String str )  
Reverse the string according to definition.
```

Base Cases : Check the index of space character.

Part2:

Elfish:

A word is considered elfish if it contains the letters: e, l, and f in it, in any order.

```
public static boolean isElfish( String str )  
Method for customer calling.
```

```
private static boolean elfishWrapper( String str , String in, boolean[] arrBool )
```

Main wrapper class of elfish checking method. It is a wrapper method , so all other method used as an internal function in this wrapper.

Base Cases :

- 1 - Check str is null , or length zero or in is null .
- 2 - When boolean array is filled return true.(shortcut)

```
private static boolean isCover( boolean[] arrBool , int index )
```

Check the given boolean array contains all true or not.

Base Cases :

- 1 - Check the index whether equals given boolean array size.
- 2 - Check the current index whether is true.(shortcut)

```
private static void update( String str , String in, boolean[] arrBool , int index )
```

Update the current status of the elfish determination process.

Base Cases :

- 1 - Check whether index is equivalent to 'in' length.

Part3:

Selection-Sort Definition :

The algorithm divides the input list into two parts: a sorted sublist of items which is built up from left to right at the front (left) of the list and a sublist of the remaining unsorted items that occupy the rest of the list. Initially, the sorted sublist is empty and the unsorted sublist is the entire input list. The algorithm proceeds by finding the smallest (or largest, depending on sorting order) element in the unsorted sublist, exchanging (swapping) it with the leftmost unsorted element (putting it in sorted order), and moving the sublist boundaries one element to the right.

There are four function for different purpose ;

```
public static void selectionSort( Integer[] arr )  
Method for customer calling.
```

```
private static void sortWithSelectionSort( Integer[] arr , int index )
```

Main wrapper method that manipulates swap and minIndex functions as well. User-called function only contains this function. As a result , it is meaning as main wrapper method for recursive selection-sort.

Base Cases :

1 – Check whether index is grader than length of arr minus one.

```
private static void swap(Integer[] arr , int index1 , int index2 )
```

Swaps two numbers which are in given indexes with each other. Combining with minIndex Function will be good result.

```
private static int minIndex( Integer[] arr , int index1 , int index2 )
```

Determines the index of minimum number over the range of given indexes. When the minimum number index finds , it will be using for swap method.

Base Cases :

1 – Check given indexes whether equal.
2 – Index value comparison.

Part4:

Evaluation of prefix expression :

Prefix expressions can be evaluated faster than an infix expression. This is because we don't need to process any brackets or follow operator precedence rule. In prefix expressions which ever operator comes before will be evaluated first, irrespective of its priority. Also, there are no brackets in these expressions. As long as we can guarantee that a valid prefix expression is used, it can be evaluated with correctness.

```
public static double prefixEvaluate( String input )  
Customer calling function.  
Combine all of sub methods and takes an input expression as String.
```

```
private static void prefixWrapper(String input , Stack<String> stack )  
Main wrapper method that contains all other function and combine them with each other. Also use a stack  
its internal implementation. Stack keeps operands in evaluation process.
```

Base Cases :

1 - Check index value and length of the input string.

```
private static class SyntaxErrorException extends Exception
```

Handling for exception case , creating a new class about syntax exception will be more readable because , if stack throws an exception it corresponds that input string is not valid , if stack would have been thrown and catch it will be meaningless for customer. Therefore, I catch an exception that is thrown by stack or other code component.

```
private static String reverse( String str )
```

Reverse given string according to its sentence. In evaluation process given string is needed to be reversed.

Base Cases :

1 - Check the index of space character.

```
private static boolean isNumeric( String str )
```

Check whether given string has numeric structure. According to result , it may throw an exception , function that call that methods handle this exception.

Part5:

Evaluation of postfix expression :

Postfix expressions can be evaluated faster than an infix expression. This is because we don't need to process any brackets or follow operator precedence rule. In Postfix expressions which ever operator comes before will be evaluated first, irrespective of its priority. Also, there are no brackets in these expressions. As long as we can guarantee that a valid Postfix expression is used, it can be evaluated with correctness.

```
public static double postfixEvaluate( String input )
```

Combine all of sub methods and takes an input expression as String.

```
private static void postfixWrapper(String input , Stack<String> stack )
```

Main wrapper method that contains all other function and combine them with each other. Also use a stack its internal implementation. Stack keeps operands in evaluation process.

Base Cases :

1 - Check index value and length of the input string.

```
private static class SyntaxErrorException extends Exception
```

Handling for exception case , creating a new class about syntax exception will be more readable because , if stack throws an exception it corresponds that input string is not valid , if stack would have been thrown and catch it will be meaningless for customer. Therefore, I catch an exception that is thrown by stack or other code component.

```
private static boolean isNumeric( String str )
```

Check whether given string has numeric structure. According to result , it may throw an exception , function that call that methods handle this exception.

Part6:

Snailing output of string :

```
public static <E> String snail( E[][] arr )
```

Function that is presented to customer.

```
private static boolean isTrackFilled( boolean[][] arr )
```

Check the given boolean array has a false element in it , it corresponds to unreached element of given two-dimensional string array. It is used for checking the end of the snailing , solely base case of another recursive function.

Base Cases :

- 1 - Given indexes element whether true.
- 2 - Given index row coordinate whether equals to given array length minus one.

```
private static<E> String snailWrapper( E[][] arr , int xInitial , int xFinal , int yInitial ,  
int yFinal , int taskCode , boolean[][] checkArr , StringBuilder str )
```

Main wrapper recursive function that goes over all element in given array in a sense that given task.

There are number of different tasks that iterate on x-rows and y-columns coordinate. These tasks carry out a significant task for going over the two-dimensional string.

Base Cases :

- 1 - isTrackFilled function return value checking.

Test Cases:

Part 1:

Test ID	Scenario	Test Data	Expected Result	Actual Result	Pass	Fail
T01	Given sentence : "this function writes the sentence in reverse"	String given by the user.	Result : reverse in sentence the writes function this	Expected result occurred.	✓	
T02	Given sentence : "I am a student at Gebze Technical University"	String given by the user.	Result : University Technical Gebze at student a am I	Expected result occurred.	✓	
T03	Given sentence : "024 044 171"	String given by the user.	Result : 171 044 024	Expected result occurred.	✓	

Part 2:

Test ID	Scenario	Test Data	Expected Result	Actual Result	Pass	Fail
T01	Test : elfish	elfish	true	Expected result occurred.	✓	
T02	Test : car	car	false	Expected result occurred.	✓	
T03	Test : whiteleaf	whiteleaf	true	Expected result occurred.	✓	
T04	Test : GTU	GTU	false	Expected result occurred.	✓	
T05	Test : tasteful	tasteful	true	Expected result occurred.	✓	
T06	Test : computer	computer	false	Expected result occurred.	✓	
T07	Test : unfriendly	unfriendly	true	Expected result occurred.	✓	
T08	Test : waffles	waffles	true	Expected result occurred.	✓	

Part 3:

Test ID	Scenario	Test Data	Expected Result	Actual Result	Pass	Fail
T01	Unsorted form : -1 2 -3 4 6 9	-1 2 -3 4 6 9	Sorted form : -3 -1 2 4 6 9	Expected result occurred.	✓	
T02	Unsorted form : 1 7 1 0 4 4 0 2 4	1 7 1 0 4 4 0 2 4	Sorted form : 0 0 1 1 2 4 4 4 7	Expected result occurred.	✓	
T03	Unsorted form : 9 -1 8 -2 7 -3 6 -4 5	9 -1 8 -2 7 -3 6 -4 5	Sorted form : -4 -3 -2 -1 5 6 7 8 9	Expected result occurred.	✓	
T04	Unsorted form : 1 5 2 3 4 -6 7 -1 -4 8 2 1 23 1 33 93 -65	1 5 2 3 4 -6 7 -1 -4 8 2 1 23 1 33 93 -65	Sorted form : -65 -6 -4 -1 1 1 1 2 2 3 4 5 7 8 23 33 93	Expected result occurred.	✓	







Part 4:

Test ID	Scenario	Test Data	Expected Result	Actual Result	Pass	Fail
T01	Prefix Expression : - + + 5 / - 4 * 2 1 2 3 / 7 1	- + + 5 / - 4 * 2 1 2 3 / 7 1	Evaluation Result : 2.0	Expected result occurred.	✓	
T02	Prefix Expression : + / + 2 * 3 2 4 2	+ / + 2 * 3 2 4 2	Evaluation Result : 4.0	Expected result occurred.	✓	
T03	Prefix Expression : + / + 2 * 3 - 2 4 2	+ / + 2 * 3 - 2 4 2	Evaluation Result : 1.0	Expected result occurred.	✓	
T04	Prefix Expression : / + / * - 1 3 2 2 5 - 3	+ / + 2 * 3 - 2 4 2	Evaluation Result : -1.0	Expected result occurred.	✓	

Part 5:

Test ID	Scenario	Test Data	Expected Result	Actual Result	Pass	Fail
T01	Postfix Expression : 5 4 2 1 * - 2 / + 3 + 7 1 / -	5 4 2 1 * - 2 / + 3 + 7 1 / -	Evaluation Result : 2.0	Expected result occurred.	✓	
T02	Postfix Expression : 1 3 + 2 * 8 / 1 2 + -	1 3 + 2 * 8 / 1 2 + -	Evaluation Result : -2.0	Expected result occurred.	✓	
T03	Postfix Expression : 10 5 + 3 / 2 3 - 4 * +	10 5 + 3 / 2 3 - 4 * +	Evaluation Result : 1.0	Expected result occurred.	✓	
T04	Postfix Expression : 1 4 - 3 * 10 +	1 4 - 3 * 10 +	Evaluation Result : 1.0	Expected result occurred.	✓	

Part 6:

Test ID	Scenario	Test Data	Expected Result	Actual Result	Pass	Fail
T01	Input array : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20	Snail output : 1 2 3 4 8 12 16 20 19 18 17 13 9 5 6 7 11 15 14 10	Expected result occurred.		
T02	Input array : 1 2 3 4 5 6 7 8	1 2 3 4 5 6 7 8	Snail output : 1 2 3 4 8 7 6 5	Expected result occurred.		
T03	Input array : 1 2 3 4 5 6 7 8 9 10 11 12	1 2 3 4 5 6 7 8 9 10 11 12	Snail output : 1 2 4 6 8 10 12 11 9 7 5 3	Expected result occurred.		
T04	Input array : a b c d e f g h i j k l m n o p q r s t u v w x y z ~	a b c d e f g h i j k l m n o p q r s t u v w x y z ~	Snail output : a b c f i l o r u x ~ z y v s p m j g d e h k n q t w	Expected result occurred.		
T05	Input array : 1 2 3 4 5 * 7 8 G a b T c * U	1 2 3 4 5 * 7 8 G a b T c * U	Snail output : 1 2 3 * G T U * c a 7 4 5 8 b	Expected result occurred.		
T06	Input array : a b c d e f g h i j k l	a b c d e f g h i j k l	Snail output : a b c d h l k j i e f g	Expected result occurred.		

Class Diagram :

Class diagram is inserted to the homework file.

Running Commend and Result:

Running command and result stage is inserted to the homework file.