Homework 3 Progress

Luke Mattfeld
CSCD300
05/28/18

**Tester class methods:**

public int countSpace(String str) {

1) Check if the passed in string contains a space:
    a) If Not, return 0;
    b) If so, return a recursive call with the string after the first space, plus 1;

}

public boolean myContains(String s1, String s2) {

1) If either string is null or empty, or if string 2 is longer than string 1, return false;
2) If the the first (length of s2) characters of s1 are equal to s2, return true;
3) Else, return a recursive call mContains of s1 without the first character;

}

public int div(int m, int n) throws IllegalArgumentException {

1) If n is greater than m, return 0;
2) Else, return a recursive call div(m-n, n) + 1;

}

private boolean isSum24(int arr[], int targetSum) {

1) If the array length is 0, return false;
2) If the array size is 1, check if that item is equal to the targetSum;
3) Else
    a) Create a new array containing all but the first element of arr;
    b) Return a recursive call isSum24 calling the new array as the array input, and the target sum minus the value of the first element in arr.

}

private void reverseArray(int a[], int low, int high) {

1) If the low index is not equal to the high index, or the low index is not greater than the high index:
    a) Create a temp int to hold a[high]
    b) Swap a[high] and a[low], using temp
    c) Do a recursive call reverseArray, passing in the original array, low+1, and high-1.
2) The idea is that low and high will get closer and closer with each recursive call, each time switching places. Once they are on the same value, or have passed each other, the array has been reversed;

}

Luke Mattfeld
CSCD300
05/28/18

private void recursiveSelectionSort(int a[], int low, int high) {

1) If high and low equal each other, return (exit the method call)
2) Else, create an int small to hold the index of the smallest value of the passed in array;
3) Use a for loop to find the smallest value, and swap it with the a[low] element;
4) Do a recursive call, passing in the array, low+1, and high;
5) Basically, each time the section of the array scanned will get smaller, and by the time low = high, all of the smallest elements in each sub-scan have been put in order, and the array is sorted.

}

**MyLinkedList class methods:**

//Recursive method for reverse1()
private MyLinkedList reverse(ListNode node) {

1) Create a new MyLinkedList (ex. Called my)
2) If the passed in node is not null:
    a) Set my to the return of a recursive call passing in node.next
    b) Add the current node to the end of my (helper method addLast required)
3) Return my;

}

//Recursive method for reverse2()
private ListNode reverse(ListNode prev, ListNode subHead) {

1) Create a ListNode (ex. Named bob)
2) Check if subHead.next is equal to null
    a) If so, set bob to subHead;
    b) If not, set bob to the return of a recursive call, passing in subHead as prev, and subHead.next as subHead;
3) Set subHead.next to prev
4) Set prev.next to null (so we don't get a circular list);
5) Return bob;

}