CS 61B Spring 2018

Linked Lists & Arrays

Discussion 3: January 30, 2018

1 More Practice with Linked Lists

```
1  public class SLList {
2      private class IntNode {
3          public int item;
4          public IntNode next;
5          public IntNode(int item, IntNode next) {
6              this.item = item;
7              this.next = next;
8          }
9      }
10
11     private IntNode first;
12
13     public void addFirst(int x) {
14         first = new IntNode(x, first);
15     }
16 }
```

1.1 Implement `SLList.insert` which takes in an integer `x` and inserts it at the given position. If the position is after the end of the list, insert the new node at the end.

For example, if the `SLList` is $5 \to 6 \to 2$, `insert(10, 1)` results in $5 \to 10 \to 6 \to 2$.

```
1  public void insert(int item, int position) {
```

```
1      if (first == null || position == 0) {
2          addFirst(item);
3          return;
4      }
5      IntNode currentNode = first;
6      while (position > 1 && currentNode.next != null) {
7          position--;
8          currentNode = currentNode.next;
9      }
10     IntNode newNode = new IntNode(item, currentNode.next);
11     currentNode.next = newNode;
```

currentNode会正好在需要插入位置的前面

1.2 Add another method to the `SLList` class that reverses the elements. Do this using the existing `IntNodes` (you should not use **`new`**).

```
public void reverse() {
```

```
    IntNode frontOfReversed = null;
    IntNode nextNodeToAdd = first;
    while (nextNodeToAdd != null) {
        IntNode remainderOfOriginal = nextNodeToAdd.next;
        nextNodeToAdd.next = frontOfReversed;
        frontOfReversed = nextNodeToAdd;
        nextNodeToAdd = remainderOfOriginal;
    }
    first = frontOfReversed;
```

1.3 *Extra*: If you wrote `reverse` iteratively, write a second version that uses recursion (you may need a helper method). If you wrote it recursively, write it iteratively.

```
public void reverse() {
    first = reverseRecursiveHelper(first);
}

private IntNode reverseRecursiveHelper(IntNode front) {
    if (front == null || front.next == null) {
        return front;
    } else {
        IntNode reversed = reverseRecursiveHelper(front.next);
        front.next.next = front;
        front.next = null;
        return reversed;
    }
}
```

2 Arrays

2.1 Consider a method that inserts `item` into array `arr` at the given position. The method should return the resulting array. For example, if `x = [5, 9, 14, 15]`, `item = 6`, and `position = 2`, then the method should return `[5, 9, 6, 14, 15]`. If `position` is past the end of the array, insert `item` at the end of the array.

Is it possible to write a version of this method that returns void and changes `arr` in place (i.e., destructively)?

No, because arrays have a fixed size, so to add an element, you need to create a new array.

Extra: Write the described method:

```
public static int[] insert(int[] arr, int item, int position) {

    int[] result = new int[arr.length + 1];
    position = Math.min(arr.length, position);
    for (int i = 0; i < position; i++) {
        result[i] = arr[i];
    }
    result[position] = item;
    for (int i = position; i < arr.length; i++) {
        result[i + 1] = arr[i];
    }
    return result;
}
```

2.2 Consider a method that destructively reverses the items in `arr`. For example calling `reverse` on an array `[1, 2, 3]` should change the array to be `[3, 2, 1]`.

What is the fewest number of iteration steps you need? What is the fewest number of additional variables you need?

Half the length of the array. You can swap the two paired indices at the same step. One additional variable as a temporary buffer during the swap; one index for the iteration. More may make your code neater.

Extra: Write the method:

```
public static void reverse(int[] arr) {

    for (int i = 0; i < arr.length / 2; i++) {
        int j = arr.length - i - 1;
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}
```

2.3 *Extra*: Write a non-destructive method `replicate(int[] arr)` that replaces the number at index `i` with `arr[i]` copies of itself. For example, `replicate([3, 2, 1])` would return `[3, 3, 3, 2, 2, 1]`.

```
public static int[] replicate(int[] arr) {
```

```
    int total = 0;
    for (int item : arr) {
        total += item;
    }
    int[] result = new int[total];
    int i = 0;
    for (int item : arr) {
        for (int counter = 0; counter < item; counter++) {
            result[i] = item;
            i++;
        }
    }
    return result;
}
```