CS 61B
Spring 2018

Linked Lists & Arrays

Discussion 3: January 30, 2018

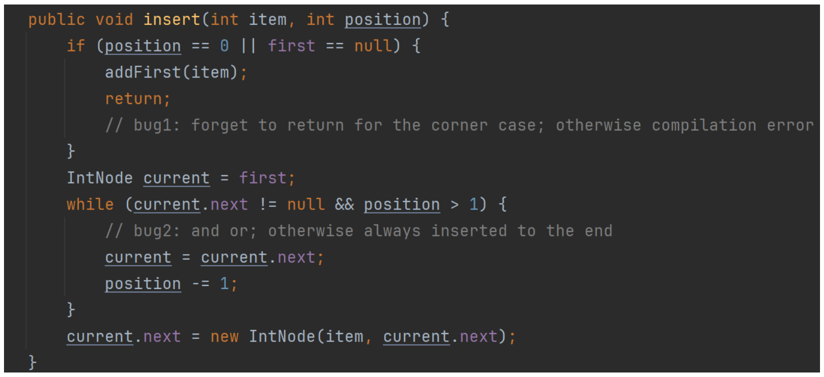
1 More Practice with Linked Lists

```
public class SLList {
    private class IntNode {
        public int item;
        public IntNode next;
        public IntNode(int item, IntNode next) {
            this.item = item;
            this.next = next;
        }
    }

    private IntNode first;

    public void addFirst(int x) {
        first = new IntNode(x, first);
    }
}
```

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 $5 \to 10 \to 6 \to 2$.

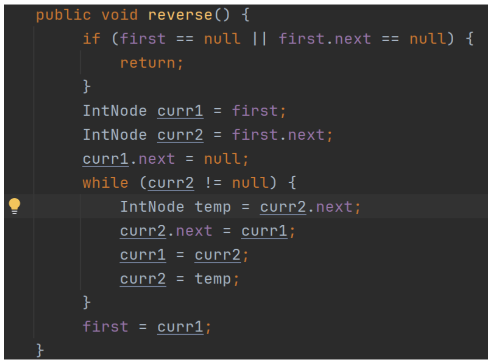
```
public void insert(int item, int position) {
```



```
public void insert(int item, int position) {
    if (position == 0 || first == null) {
        addFirst(item);
        return;
        // bug1: forget to return for the corner case; otherwise compilation error
    }
    IntNode current = first;
    while (current.next != null && position > 1) {
        // bug2: and or; otherwise always inserted to the end
        current = current.next;
        position -= 1;
    }
    current.next = new IntNode(item, current.next);
}
```

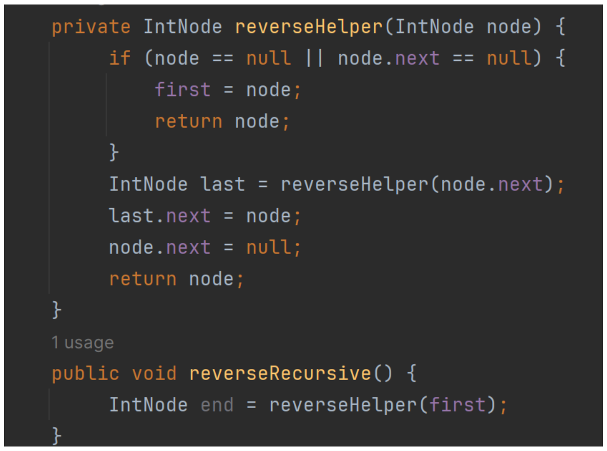
1.2 Add another method to the `SLList` class that reverses the elements. Do this using the existing `IntNodes` (you should not use **`new`**).

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



```java
public void reverse() {
    if (first == null || first.next == null) {
        return;
    }
    IntNode curr1 = first;
    IntNode curr2 = first.next;
    curr1.next = null;
    while (curr2 != null) {
        IntNode temp = curr2.next;
        curr2.next = curr1;
        curr1 = curr2;
        curr2 = temp;
    }
    first = curr1;
}
```

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.



```java
private IntNode reverseHelper(IntNode node) {
    if (node == null || node.next == null) {
        first = node;
        return node;
    }
    IntNode last = reverseHelper(node.next);
    last.next = node;
    node.next = null;
    return node;
}
1 usage
public void reverseRecursive() {
    IntNode end = reverseHelper(first);
}
```

2 Arrays

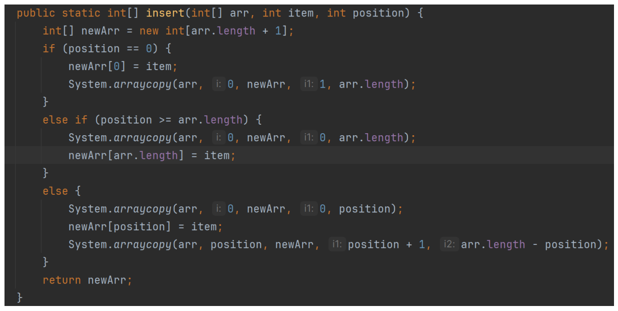
2.1 Consider a method that inserts `item` into array `arr` at the given position. The method should return the resulting array. For example, if `x = [5, 9, 14, 15]`, `item = 6`, and `position = 2`, then the method should return `[5, 9, 6, 14, 15]`. If `position` is past the end of the array, insert `item` 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, array cannot change size

Extra: Write the described method:

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



```java
public static int[] insert(int[] arr, int item, int position) {
    int[] newArr = new int[arr.length + 1];
    if (position == 0) {
        newArr[0] = item;
        System.arraycopy(arr, i: 0, newArr, i1: 1, arr.length);
    }
    else if (position >= arr.length) {
        System.arraycopy(arr, i: 0, newArr, i1: 0, arr.length);
        newArr[arr.length] = item;
    }
    else {
        System.arraycopy(arr, i: 0, newArr, i1: 0, position);
        newArr[position] = item;
        System.arraycopy(arr, position, newArr, i1: position + 1, i2: arr.length - position);
    }
    return newArr;
}
```

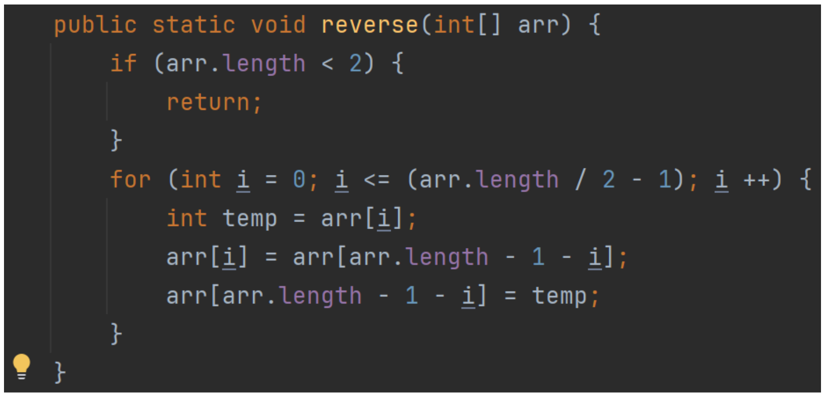
2.2 Consider a method that destructively reverses the items in `arr`. For example calling `reverse` on an array `[1, 2, 3]` should change the array to be `[3, 2, 1]`.

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

1 iteration loop
1 variable (temp) ← and index

Extra: Write the method:

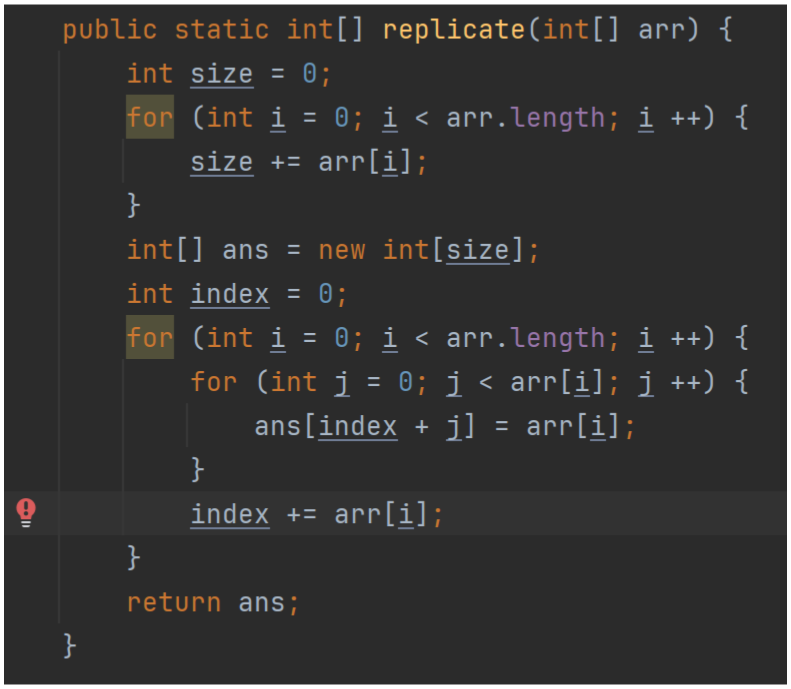
```java
public static void reverse(int[] arr) {
```



```java
public static void reverse(int[] arr) {
    if (arr.length < 2) {
        return;
    }
    for (int i = 0; i <= (arr.length / 2 - 1); i ++) {
        int temp = arr[i];
        arr[i] = arr[arr.length - 1 - i];
        arr[arr.length - 1 - i] = 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]`.

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



```java
public static int[] replicate(int[] arr) {
    int size = 0;
    for (int i = 0; i < arr.length; i ++) {
        size += arr[i];
    }
    int[] ans = new int[size];
    int index = 0;
    for (int i = 0; i < arr.length; i ++) {
        for (int j = 0; j < arr[i]; j ++) {
            ans[index + j] = arr[i];
        }
        index += arr[i];
    }
    return ans;
}
```