EECS2011 Fundamentals of Data Structures (Winter 2022)

Q&A - Week 3 Lecture

Thursday, February 3

Announcements

1. addLast
2. removeLast
3. exercises

Node<String> vs. Node<Integer>

- Lecture W4 released (SLL and generics review)
- Assignment 1 (on SLLs) released on Monday.
  - ↳ scheduled TA hours

```
int i = 68;

String s = i + "";
```

Counting the Number of Nodes without a Loop?

Exercise: getSize() in Node

```
public class SinglyLinkedList {
  private Node head = null;
  public void setHead(Node n) { head = n; }
  public int getSize() { ... }
  public Node getTail() { ... }
  public void addFirst(String e) { ... }
  public Node getNodeAt(int i) { ... }
  public void addAt(int i, String e) { ... }
  public void removeLast() { ... }
}
```

You can modify the Node class if you wish.

```
return head == null ? 0 : head.getSize();
```

version in Node

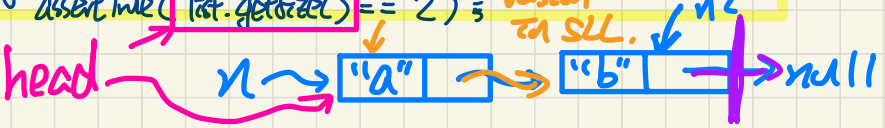
```
public class Node {
  private String element;
  private Node next;
  public Node(String e, Node n) { element = e; next = n; }
  public String getElement() { return element; }
  public void setElement(String e) { element = e; }
  public Node getNext() { return next; }
  public void setNext(Node n) { next = n; }
}
```

```
int getSize() {
  if(this.next == null){
    return 1;
  }
  else { return 1 + this.next.getSize(); }
}
```

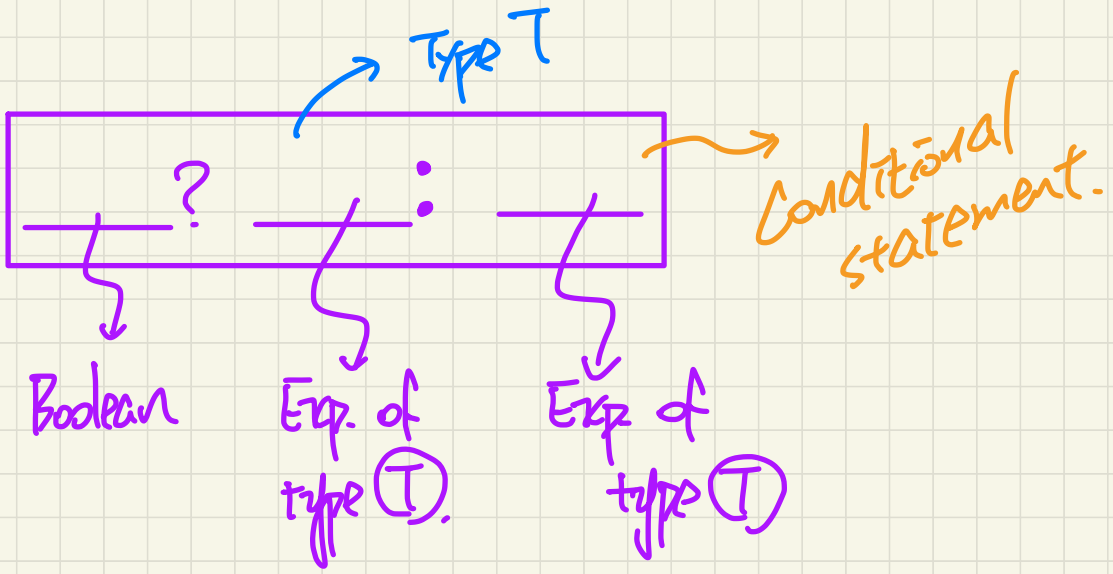
O(n)

```
Node n = new Node("a", new Node("b", null));
SinglyLinkedList list = new SLL(); list.setHead(n);
assertTrue(list.getSize() == 2);
```

version in SLL.

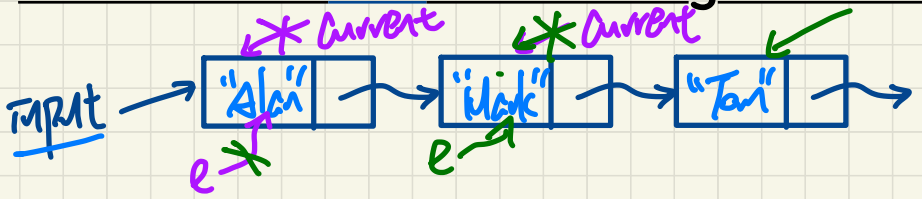


n2.getSize() 1
head.getS
list.getS 2
2

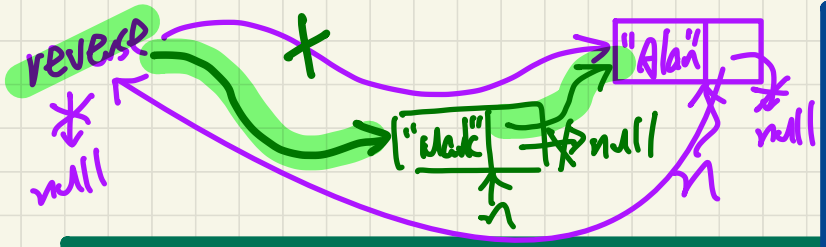


return x>y ? 23 : 46

Problem on SLL: Reversing a Chain of Nodes



public void reverseOf(Node n)



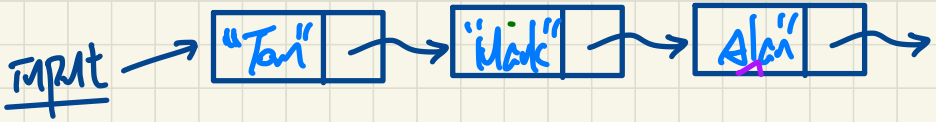

```java
public Node<String> reverseOf(Node<String> input) {
    Node<String> reverse = null;
    Node<String> current = input;
    while(current != null) {
        String e = current.getElement();
        Node<String> n = new Node<>(e, null);
        n.setNext(reverse);
        reverse = n;
        current = current.getNext();
    }
    return reverse;
}
```

```java
@Test
public void test() {
    ListUtilities util = new ListUtilities();

    Node<String> input = null;
    Node<String> output = util.reverseOf(input);
    assertNull(output);

    input = new Node<>("Alan", new Node<>("Mark", new Node<>("Tom", null)));
    output = util.reverseOf(input);
    assertEquals("Tom", output.getElement());
    assertEquals("Mark", output.getNext().getElement());
    assertEquals("Alan", output.getNext().getNext().getElement());
    assertNull(output.getNext().getNext().getNext());
}
```

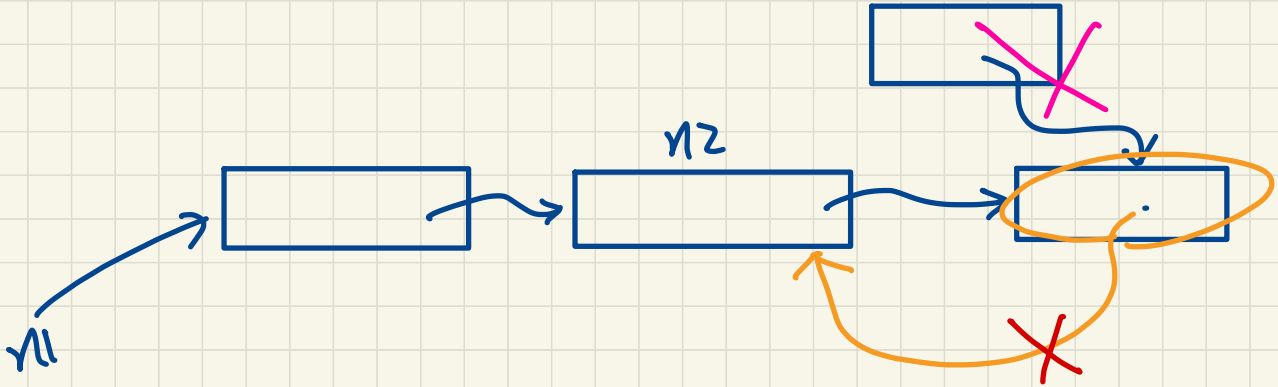
Problem on SLL: Reversing a Chain of Nodes




```
public void reverseOf(Node n)
  reverseOf(input);
```

input =

Exercise



1. extend the setNext method or constructor to check and catch such scenarios

2. Implement some method to detect such scenarios (e.g. already, cycle)