Linked list

COMP SCI / SFWR ENG 2S03

Department of Computing and Software
McMaster University

November 10, 2013

Array vs Linked-List

Array

- Fixed size
- Wastes memory because it might not be fully populated
- To insert or delete a new element into an array requires a new array to be created with a new size

Linked-list

- Dynamic structure
- Create new data "on demand"
- Data manipulation can be easily done by changing references (no need to create new structure)
- Extra field to store reference

What is a linked-list?

What is a linked-list?

- List of items, called nodes
- Contains two field variable called head and tail
- Head points to the first node in the list
- Tail points to the last node in the list
- Every node contains an address to the next node(exception for the last node) (hence the name "linked" list)

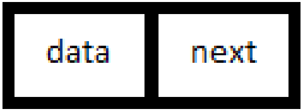
Node

Node

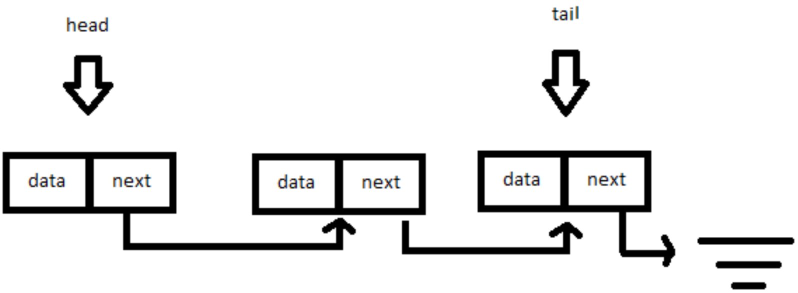
- Record that consists of an object and a reference to the next node
- Field variables are private (requires public mutator method to manipulate fields)
- E is a generic type in java, it means it can be any type (except for primitives, so no int, double, float, etc.)

```
public class Node<E> {
    private E data;
    private Node next;
    ...
    ...
}
```

Visual representation of a "node"



Visual representation of a "Linked-List"

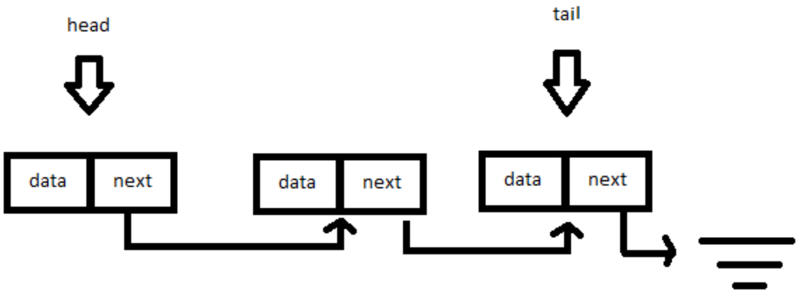


Basic operations

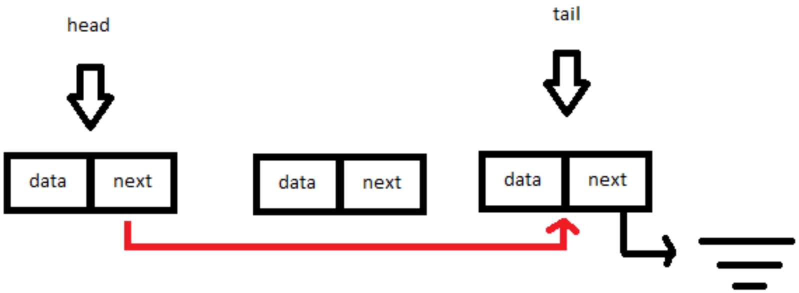
Basic operations:

- add - add new node into the linked-list
- remove - delete a node from the linked-list
- clear - make linked-list empty
- contains - search if a node is within the linked-list

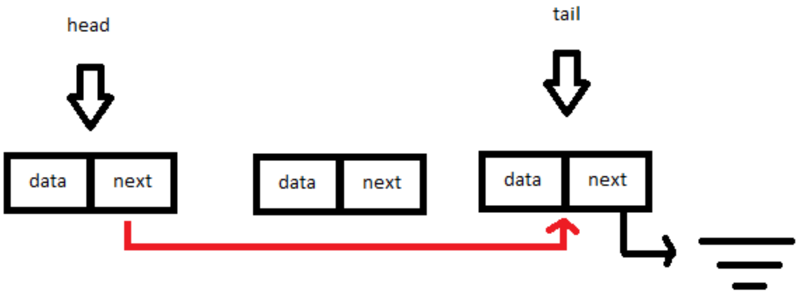
How deletion works:"



How deletion works:"



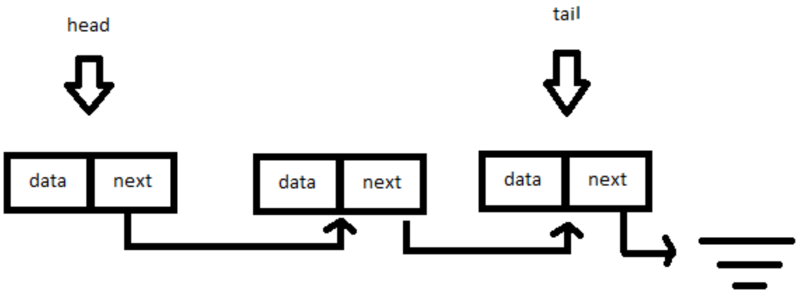
How deletion works:"



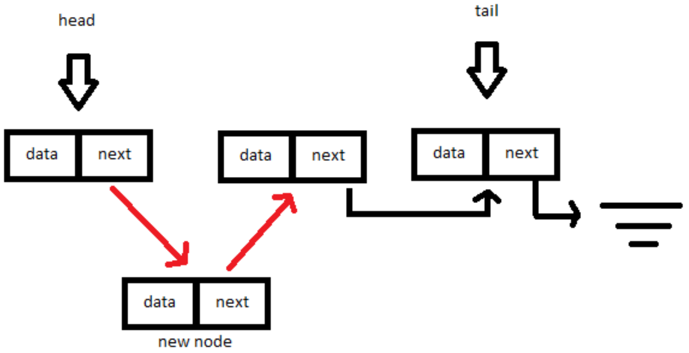
pseudo code

```
nodeToDelete
currentNode
if (currentNode.next == nodeToDelete) {
    currentNode.next = nodeToDelete.next
}
```

How insertion works:"



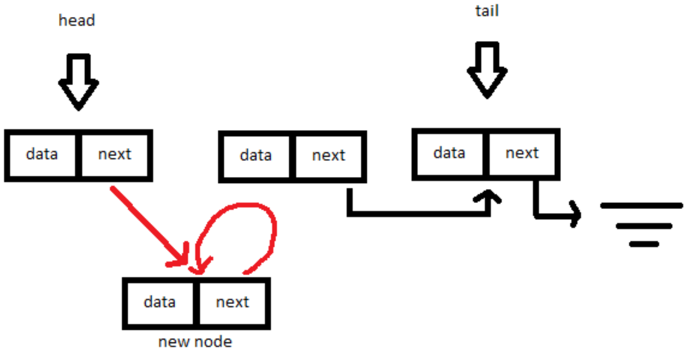
How insertion works:"



pseudo code

```
nodeToAdd
currentNode
//order matters!!!
(1) nodeToAdd.next=currentNode.next
(2) currentNode.next=nodeToAdd
```

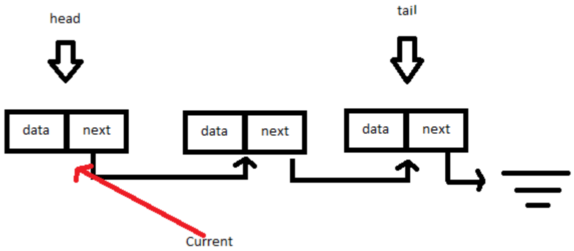
How insertion works:"



pseudo code

```
nodeToAdd
currentNode
//this is wrong
(1) currentNode.next=nodeToAdd
(2) nodeToAdd.next=currentNode.next
```

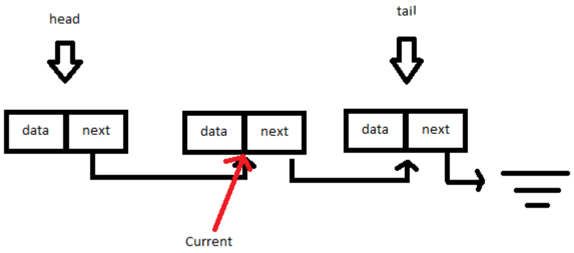
How searching works:"



pseudo code

```
nodeToFind
currentNode
while (currentNode!=null){
    if (currentNode==nodeToFind)
        return true;
    currentNode=currentNode.next;
}
return false;
```

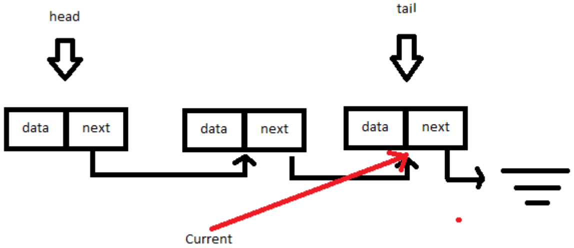
How searching works:"



pseudo code

```
nodeToFind
currentNode
while (currentNode!=null){
    if (currentNode==nodeToFind)
        return true;
    currentNode=currentNode.next;
}
return false;
```

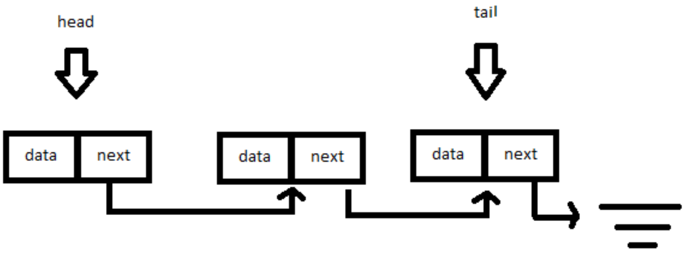
How searching works:"



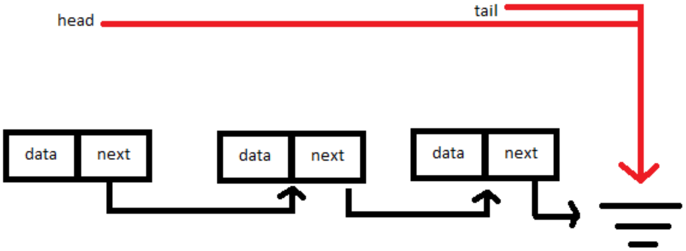
pseudo code

```
nodeToFind
currentNode
while (currentNode!=null){
    if (currentNode==nodeToFind)
        return true;
    currentNode=currentNode.next;
}
return false;
```

How clearing works:"



How clearing works:"



```
head=null;
tail=null;
```

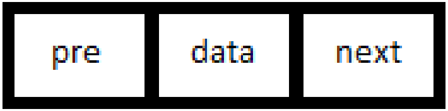
Doubly Linked List

Doubly Linked List

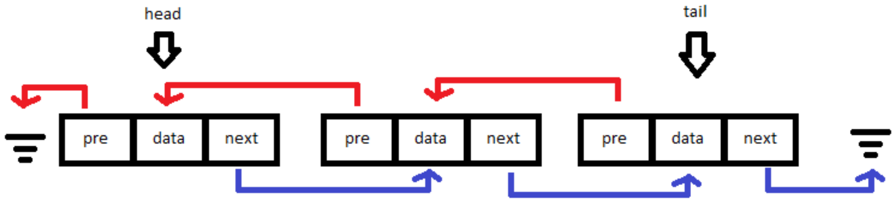
- Retain all properties of singly linked list
- An extra reference variable in the node that points to the predecessor node
- Allows traversing from the back

```
public class DoublyNode<E> {
    private Node pre;
    private E data;
    private Node next;
    ...
    ...
}
```

Visual representation of a doubly linked list node



Visual representation of a Doubly Linked List



Built-in Java LinkedList

Built-in Java LinkedList

- List of functions can be found at http://docs.oracle.com/javase/6/docs/api/java/util/LinkedList.html

LinkedList<E> list = new LinkedList<E>();

```
//E can be any object type
LinkedList<String> list = new LinkedList<String>();
LinkedList<Integer> list = new LinkedList<Integer>();
```

Useful functions

- add(E element)
- add(int index, E element)

```
LinkedList<String> list = new LinkedList<String>();
list.add("item1");
list.add("item2");
list.add(1,"item3");
System.out.println(list.toString());
```

↓

```
[item1, item3, item2]
```

Useful functions

- contains(Object o)

```
LinkedList<String> list = new LinkedList<String>();
list.add("item1");
list.add("item2");
list.add(1,"item3");
System.out.println(list.toString());
System.out.println("Is␣" item4"␣in␣the␣list?␣:"+list.contains("iter
System.out.println("Is␣" item3"␣in␣the␣list?␣:"+list.contains("iter
```

↓

```
[item1, item3, item2]
Is "item4" in the list? :false
Is "item3" in the list? :true
```

Useful functions

- get(in index)

```
LinkedList<String> list = new LinkedList<String>();
list.add("item1");
list.add("item2");
list.add(1,"item3");
System.out.println(list.toString());
System.out.println("object␣at␣index␣0␣is␣:" + list.get(0));
```

$\downarrow$

```
[item1, item3, item2]
object at index 0 is :item1
```

Useful functions

- indexOf(Object o)

```
LinkedList<String> list = new LinkedList<String>();
list.add("item1");
list.add("item2");
list.add(1,"item3");
System.out.println(list.toString());
System.out.println("index␣of␣item3␣is:" + list.indexOf("item3"));
System.out.println("index␣of␣item6␣is:" + list.indexOf("item6"));
```

↓

```
[item1, item3, item2]
index of item3 is:1
index of item6 is:-1
```

Useful functions

- remove(int index)

```
LinkedList<String> list = new LinkedList<String>();
list.add("item1");
list.add("item2");
list.add(1,"item3");
System.out.println(list.toString());
list.remove();
System.out.println(list.toString());
list.remove(1);
System.out.println(list.toString());
```

↓

```
[item1, item3, item2]
[item3, item2]
[item3]
```

Useful functions

- clear();

```
LinkedList<String> list = new LinkedList<String>();
list.add("item1");
list.add("item2");
list.add(1,"item3");
System.out.println(list.toString());
list.clear();
System.out.println(list.toString());
```

↓

```
[item1, item3, item2]
[]
```

Looping though linked lists

ListIterator

- use hasNext() to move to next node

```
LinkedList<String> list = new LinkedList<String>();
//iterates from first node
ListIterator<E> iter = list.iterator()

or

//iterates from "index" node
ListIterator<E> iter = listIterator(int index)
```

```
while (iter.hasNext()) {
    System.out.println(iter.next());
}
```

Looping though linked lists

For loop

```
LinkedList<String> list = new LinkedList<String>();
for (E s: list){
    System.out.println(s);
}

or

for (int i=0; i<list.size(); i++){
    System.out.println(list.get(i));
}
```

Exercise # 1

- Create a record for student which contain their student #, name, and grade. Use a linked list to store the students

Exercise # 2

- Implement a simple Node structure for singly linked list (ie. only has reference to the next node)

Exercise # 3

Implement a simple singly linked list that uses the node structure previously defined

- addBack()
- deleteBack()
- display()

Exercise # 4

- re-implement exercise 1 using the linked list that you've created