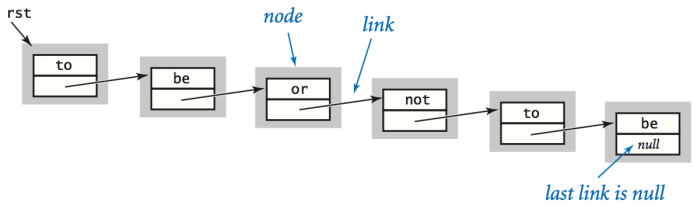
4.3 Linked Lists

Linked lists. A *singly linked list* comprises a sequence of *nodes*, with each node containing a reference (or *link*) to its successor. By convention, the link in the last node is *null*, to indicate that it terminates the list. With object-oriented programming, implementing linked lists is not difficult. We define a class for the node abstraction that is *recursive* in nature:

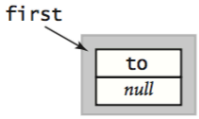
```
class Node {
      String item;
      Node next;
}
```



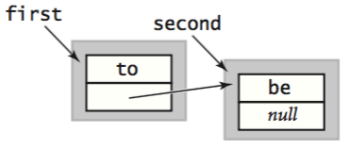
A Node object has two instance variables: a String and a Node. The String is a placeholder in this example for any data that we might want to structure with a linked list (we can use any set of instance variables); the instance variable of type Node characterizes the linked nature of the data structure.

- *Linking together a linked list.* For example, to build a linked list that contains the items “to", "be", and "or", we create a Node for each item:

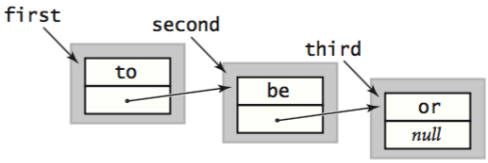
```
Node first  = new Node();
first.item  = "to";
```




```
Node second = new Node();
second.item = "be";
first.next  = second;
```



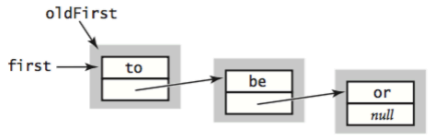

```
Node third  = new Node();
third.item  = "or";
second.next = third;
```



- *Insert.* Suppose that you want to insert a new node into a linked list. The easiest place to do so is at the beginning of the list. For example, to insert the string not at the beginning of a given linked list whose first node is first, we save first in a temporary variable oldFirst, assign to first a new Node, and assign its item field to not and its next field to oldFirst.

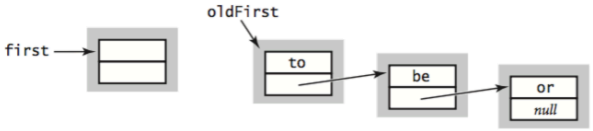
save a link to the first node in the linked list

```
Node oldFirst = first;
```



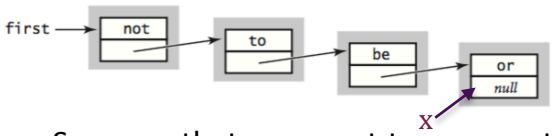
create a new node for the beginning

```
first = new Node();
```



set the instance variables in the new node

```
first.item = "not";
first.next = oldFirst;
```

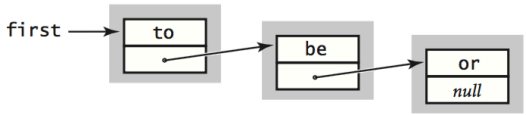


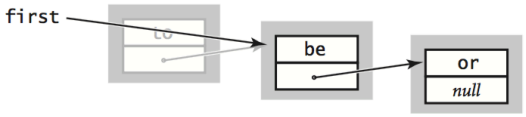

```
if (first != null) {
Node x = first;
while (x != null)
   x = x.next;
Node z  = new Node();
z.item = "be";
x.next = z; }
else first = z;
```

x

- *Remove.* Suppose that you want to remove the first node from a list. This operation is even easier: simply assign to first the value first.next.

```
if (first != null)
   first = first.next;
```





- *Traversal.* To examine every item in a linked list, we initialize a loop index variable x that references the the first Node of the linked list. Then, we find the value of the item associated with x by accessing x.item, and then update x to refer to the next Node in the linked list, assigning to it the value of x.next and repeating this process until x is null (which indicates that we have reached the end of the linked list). This process is known as *traversing* the list, and is succinctly expressed in this code fragment:

```
for (Node x = first; x != null; x = x.next)
    StdOut.println(x.item);
```

Last modified on August 13, 2017.