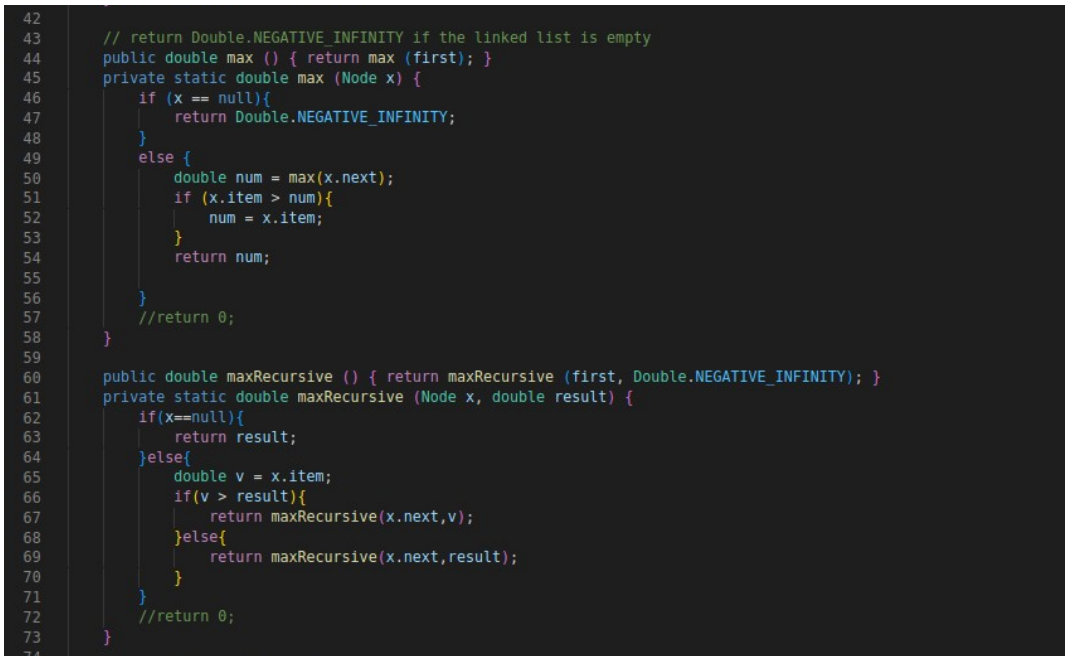
Max and Recursive Max Function:

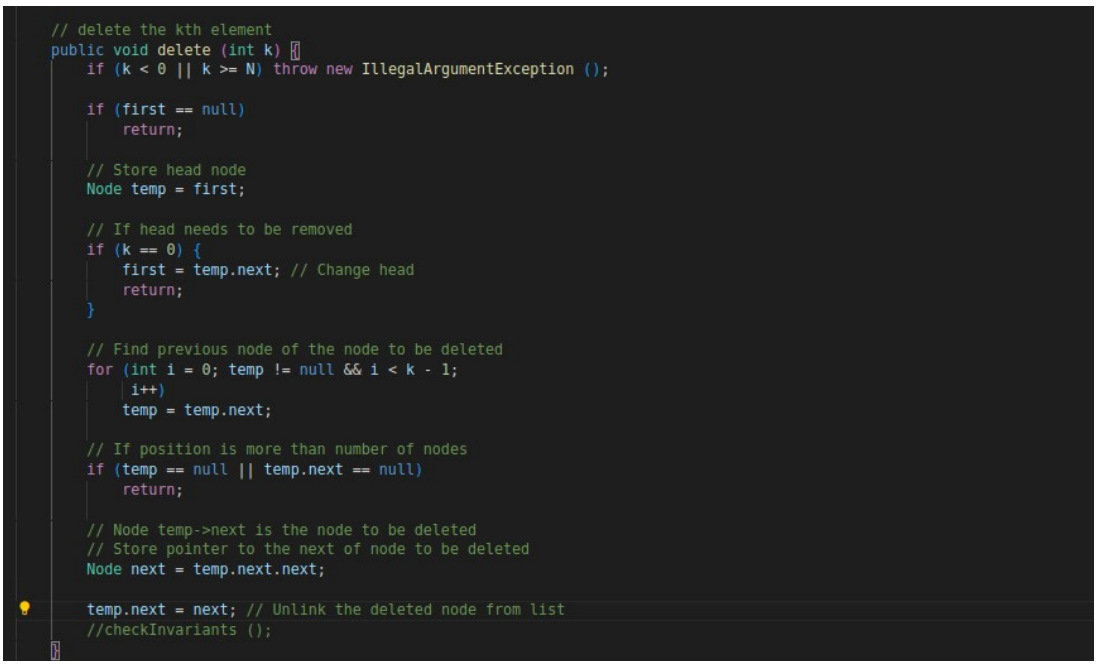
Max function finds the maximum item in the linked list by iterating over the linked list
Max Recursive finds maximum of item in linked list by recursively calling the function base condition returns the maximum of the linked list.



```
42
43      // return Double.NEGATIVE_INFINITY if the linked list is empty
44      public double max () { return max (first); }
45      private static double max (Node x) {
46          if (x == null){
47              return Double.NEGATIVE_INFINITY;
48          }
49          else {
50              double num = max(x.next);
51              if (x.item > num){
52                  num = x.item;
53              }
54              return num;
55
56          }
57          //return 0;
58      }
59
60      public double maxRecursive () { return maxRecursive (first, Double.NEGATIVE_INFINITY); }
61      private static double maxRecursive (Node x, double result) {
62          if(x==null){
63              return result;
64          }else{
65              double v = x.item;
66              if(v > result){
67                  return maxRecursive(x.next,v);
68              }else{
69                  return maxRecursive(x.next,result);
70              }
71          }
72          //return 0;
73      }
```

Delete Kth Element:

An item from linked list is deleted by the index of that item in the linked list, the function finds the position of item to be deleted and delete its pointer to the next node.



```
// delete the kth element
public void delete (int k) {
    if (k < 0 || k >= N) throw new IllegalArgumentException ();

    if (first == null)
        return;

    // Store head node
    Node temp = first;

    // If head needs to be removed
    if (k == 0) {
        first = temp.next; // Change head
        return;
    }

    // Find previous node of the node to be deleted
    for (int i = 0; temp != null && i < k - 1;
         i++)
        temp = temp.next;

    // If position is more than number of nodes
    if (temp == null || temp.next == null)
        return;

    // Node temp->next is the node to be deleted
    // Store pointer to the next of node to be deleted
    Node next = temp.next.next;

    temp.next = next; // Unlink the deleted node from list
    //checkInvariants ();
}
```

Reverse Linked List:

This function inverts the pointer direction of each node, for example the node pointing to next item will point to the previous item.

```
// reverse the list "in place"... without creating any new nodes
public void reverse () {
    Node previous = null;
    Node current = first;
    Node next;
    while (current != null) {
        next = current.next;
        current.next = previous;
        previous = current;
        current = next;
    }
    first = previous;
    checkInvariants ();
}
```

Remove item:

This function removes the nodes by the occurrence of item passed in function.

```
// remove
public void remove (double item) {

    Node temp = first, prev = null;

    // If head node itself holds the key
    // or multiple occurrences of key
    while (temp != null && temp.item == item)
    {
        first = temp.next; // Changed head
        temp = first; // Change Temp
    }

    // Delete occurrences other than head
    while (temp != null)
    {
        // Search for the key to be deleted,
        // keep track of the previous node
        // as we need to change 'prev->next'
        while (temp != null && temp.item != item)
        {
            prev = temp;
            temp = temp.next;
        }

        // If key was not present in linked list
        if (temp == null)
            return;

        // Unlink the node from linked list
        prev.next = temp.next;

        // Update Temp for next iteration of outer loop
        temp = prev.next;
    }
    //checkInvariants ();
}
```

Output:

```
[Running] cd "/home/wada/Documents/MyLinked/src/" && javac MyLinked.java && java MyLinked
empty: : -Infinity
singleton: -1.0 : -1.0
at end: -4.0 -3.0 -2.0 -1.0 : -1.0
at beginning: 5.0 -4.0 -3.0 -2.0 -1.0 : 5.0
in the middle: 4.0 2.0 3.0 5.0 -4.0 -3.0 -2.0 -1.0 : 5.0
empty: : -Infinity
singleton: -1.0 : -1.0
at end: -4.0 -3.0 -2.0 -1.0 : -1.0
at beginning: 5.0 -4.0 -3.0 -2.0 -1.0 : 5.0
in the middle: 4.0 2.0 3.0 5.0 -4.0 -3.0 -2.0 -1.0 : 5.0
singleton: 1.0
deleted:
bigger list: 12.0 11.0 10.0 9.0 8.0 7.0 6.0 5.0 4.0 3.0 2.0 1.0
deleted at beginning: 11.0 10.0 9.0 8.0 7.0 6.0 5.0 4.0 3.0 2.0 1.0
deleted at end: 11.0 10.0 9.0 8.0 7.0 6.0 5.0 4.0 3.0 2.0
deleted in middle: 11.0 10.0 9.0 8.0 6.0 5.0 4.0 3.0 2.0
reverse empty:
singleton: 1.0
reverse singleton: 1.0
two: 2.0 1.0
reverse two: 1.0 2.0
reverse again: 2.0 1.0
bigger list: 6.0 6.0 5.0 5.0 4.0 4.0 3.0 3.0 2.0 1.0
reversed: 1.0 2.0 3.0 3.0 4.0 4.0 5.0 5.0 6.0 6.0
removed 4 from empty:
removed 4 from singelton: 1.0
removed 1 from singelton:
longer list: 4.0 4.0 4.0 4.0 4.0 3.0 3.0 3.0 3.0 3.0 2.0 2.0 2.0 2.0 2.0 1.0 1.0 1.0 1.0 1.0 4.0 4.0 3.0 3.0 2.0 2.0 1.0 1.0
removed all 9s: 4.0 4.0 4.0 4.0 4.0 3.0 3.0 3.0 3.0 3.0 2.0 2.0 2.0 2.0 2.0 1.0 1.0 1.0 1.0 1.0 4.0 4.0 3.0 3.0 2.0 2.0 1.0 1.0
removed all 3s: 4.0 4.0 4.0 4.0 4.0 2.0 2.0 2.0 2.0 2.0 1.0 1.0 1.0 1.0 1.0 4.0 4.0 2.0 2.0 1.0 1.0
removed all 1s: 4.0 4.0 4.0 4.0 4.0 2.0 2.0 2.0 2.0 2.0 4.0 4.0 2.0 2.0
removed all 4s: 2.0 2.0 2.0 2.0 2.0 2.0 2.0
removed all 2s:
```