<table>
<thead>
<tr>
<th>Pay 5</th>
<th>Pay 4</th>
<th>HW 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>8/15</td>
<td>8/15</td>
<td></td>
</tr>
<tr>
<td>Final</td>
<td></td>
<td>HW #3, Problem Sets, Project 5, Papers, Project 5 Slide Show, Multi-Team Project, End of Project 5</td>
</tr>
</tbody>
</table>
Insert in sorted order

must end without a tail point

must at beginning

Insert

Linear Linked List

Node

Data

Next Pointer

Head

Tail

1

[Diagram of a linked list structure]
2. Insert at beginning

Before

(head is null)

Empty List
```plaintext
if (head == NULL) { // case 1
} else { // case 2
}

head = new node;
head->next = NULL;
// store the data
head->movie->read();
// tail = head;

struct node
{
    video movie;
    node * next;
};
node * head = NULL;

node * temp;
temp = new node;
// store the data
temp->movie->read();
// temp->next = head;
temp->next = head;
head = temp; // Grab the address in head's memory
```
Read before:

After:  

Add at the end (No tailpur)
if (head == NULL) { // case 1.
head = new node;
}
else { // case 2.
node = current = head;
while (current->next != NULL) { // while current != NULL
  current = current->next;
}
if (current == NULL) { // case 3.
  current = node; // current = new node
  while (current->next != NULL) { // while current->next != NULL
    current = current->next;
  }
  if (current == NULL) { // case 4.
    current = node; // current = new node
  }
}
else { // case 2.
  node = current = head;
  while (current->next != NULL) { // while current->next != NULL
    current = current->next;
  }
  if (current == NULL) { // case 5.
    current = node; // current = new node
  }
}
Assume head being null

Add: X
Kings: Terminator

Example

3
else
{
    temp = temp -> next;
}

if (temp -> data > Head) // if (Head < datanode) // Temp <- Temp's more is less Head's more
    Head = temp;
else if (Head -> more < Temp) // Temp = Temp's more compar (Temp = more)
    Head = Temp;

if (Head = NULL) // Empty list
    case 1

if (temp = NULL) // Value = Null
    temp = move; // Read();

node * temp = new node; // Adding a node;

/* Instruct (node *heed) */

void
while (current &amp;&amp; current-&gt;next) {
    temp = current-&gt;next;
    current-&gt;next = null;
    previous = current;
    current = current-&gt;next;
}

node = previous;
node-&gt;data = 28;
node = previous;
node-&gt;next = temp;
node-&gt;data = 38;
node-&gt;next = temp->next;
node-&gt;data = head->data;
node = previous;
node-&gt;next = temp->next;
node-&gt;data = head->data;
node = previous;
null
insert at end with a tail pointer.

1. current = current->next; // save current
2. if (current) {
   node = new node;
   current->next = node;
   node->next = current;
   if (node) {
      list->next = node;
      list = list->next;
   }
}
- Remove all
- Remove something in the middle
- Remove the last item
- Remove the first item

Remove all dates

Final LLL
Remove first item

1. node = temp
2. head = head -> next
3. delete temp

head

node = temp

temp

node = head

head = head -> next

delete head
```java
if (head == null) {
    delete count = null;
} else {
    if (head.next == null) {
        Head = null;
    } else {
        if (head.next.next == null) {
            previous = current;
            current = current.next;
            while (current.next != null) {
                previous.next = current;
                current = current.next;
            }
            previous.next = null;
        } else {
            delete count = null;
        }
    }
}
Remove last item
```
if (head == data == match) {
    current = head;
    head = head->next;
    delete current;
} else {
    node * current = head;
    node * previous = NULL;
    while (current && current->data != match) {
        previous = current;
        current = current->next;
    }
    if (current == NULL) {
        cout << "NO Match";
    } else {
        previous->next = current->next;
        delete current;
    }
}
{ 
  while (head) 
  { 
    head = head -> next; 
    temp = head;
  }

Remove All