**Data Structures and Algorithms**

**Lab 05**

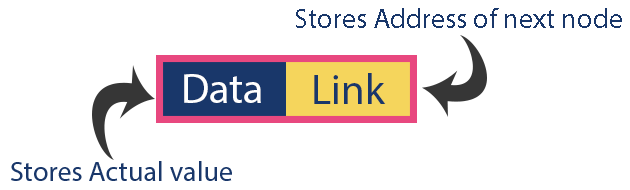
**Name:** Ahmad Amjad Mughal

**Reg No:** 121672

**Class:** BSCS-6C

Task 1:

**Understand the concept of Linked List and write it down in detail along with Visual Linked List Representation. Write down the pseudo code**

****

**Pseudo code**

* Structure Node
* Integer data
* Pointer next of node type

Now, we need a class which will contain the functions to handle the nodes. This class should have two important pointers, i.e. head and tail. The constructer will make them NULL to avoid any garbage value.

**Class list (making of Linked list):**

* We make a class **list**
* We declare pointers Head and Tail of Node type privately.
* We ensure that both Head = NULL and tail = NULL.

**Node Creation:**

**Pseudo Code:**

* Create function createnode
* Pointer temporary of node type
* Temporary🡪 data = value
* Temporary🡪 address = NULL
* Check whether the (Head == NULL)
* If true (If it is Empty)
* Head is Temporary
* Temporary is NULL
* Else (if non Empty)
* Temporary is pointing Tail node

**Display Function:**

If we want to see that what is placed in our linked list then we will have to make a display function. he logic behind this function is that we make a temporary node and pass the address of the head node to it. So we need a loop which runs as many times as nodes exist. Every node contains the address of the next node so the temporary node walks through the whole linked list. If the temporary node becomes equal to NULL then the loop would be terminated.

**Pseudo Code:**

* Create a display function to display a node
* Pointer Temporary points it to the Head node
* Loop for non Empty node (Head != NULL)
* Temporary 🡪data
* Temporary is equals to Temporary 🡪 next

**Insertion:**

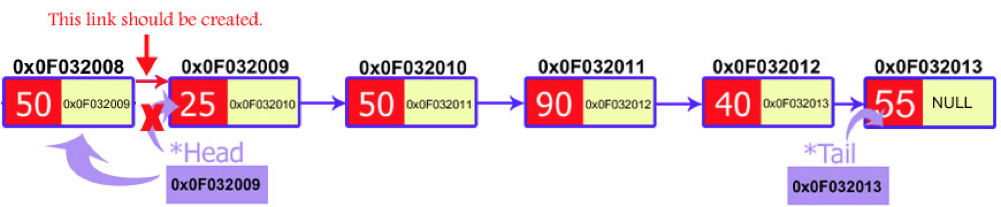
Inserting a new node in the linked list is called insertion.A new node is created and inserted in the linked list.There are three cases considered while inserting a node:

1. Insertion at the start
2. Insertion at the end
3. Insertion at a particular position

**Insertion at the beginning:**

New node should be connected to the first node, which means the head. This can be achieved by putting the address of the head in the next field of the new node.

New node should be considered as a head. It can be achieved by declaring head equals to a new node.

****

**Pseudo Code:**

* Create a newNode with given value.
* Check whether list is Empty (head == NULL)
* If it is Empty then, set newNode→next = NULL and head = newNode.
* If it is Not Empty then, set newNode→next = head and head = newNode.

**Insertion at the end:**

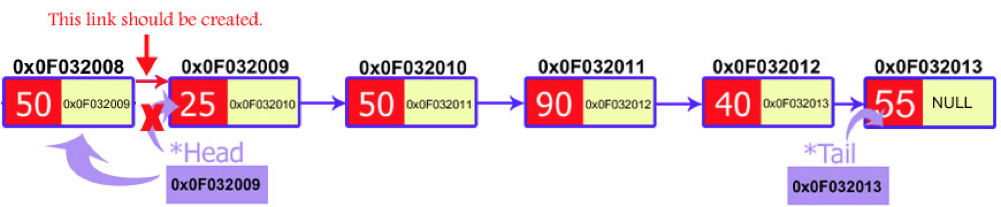
The insertion of a node at the end of a linked list is the same as we have done in node creation function. If you noticed then, we inserted the newly created node at the end of the linked list. So this process is the same.

**Pseudo Code:**

* Create a newNode with given value and newNode → next as NULL.
* Check whether list is Empty (head == NULL).
* If it is Empty then, set head = newNode.
* If it is Not Empty then, define a node pointer temp and initialize with head.
* Keep moving the temp to its next node until it reaches to the last node in the list (until temp → next is equal to NULL).
* Set temp → next = newNode.

**Insertion at a particular point:**

In this case, we don’t disturb the head and tail nodes. Rather, a new node is inserted between two consecutive nodes. We call one node as current and the other as previous, and the new node is placed between them.

****

**Pseudo Code:**

* Create a newNode with given value.
* Check whether list is Empty (head == NULL)
* If it is Empty then, set newNode → next = NULL and head = newNode.
* If it is Not Empty then, define a node pointer temp and initialize with head.
* Keep moving the temp to its next node until it reaches to the node after which we want to insert the newNode (until temp1 → data is equal to location, here location is the node value after which we want to insert the newNode).
* Every time check whether temp is reached to last node or not. If it is reached to last node then display 'Given node is not found in the list!!! Insertion not possible!!!' and terminate the function. Otherwise move the temp to next node.
* Finally, Set 'newNode → next = temp → next' and 'temp → next = newNode'

**Deletion:**

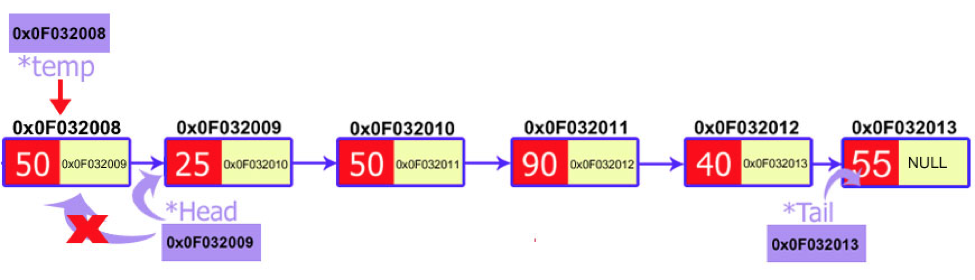
Linked lists provide us the great feature of deleting a node. The process of deletion is also easy to implement. The basic structure is to declare a temporary pointer which points the node to be deleted.

1. Deletion at the start
2. Deletion at the end
3. Deletion at a particular position

**Deletion at the start:**

In this case, the first node of the linked list is deleted

* Declare a temp pointer and pass the address of the first node, i.e. head to this pointer.
* Declare the second node of the list as head as it will be the first node of linked list after deletion.
* Delete the temp node.

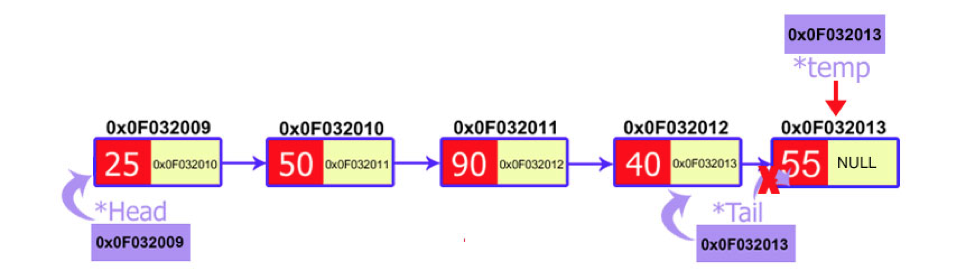
****

**Pseudo Code:**

* Check whether list is Empty (head == NULL)
* If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.
* If it is Not Empty then, define a Node pointer 'temp' and initialize with head.
* Check whether list is having only one node (temp → next == NULL)
* If it is TRUE then set head = NULL and delete temp (Setting Empty list conditions)
* If it is FALSE then set head = temp → next, and delete temp.

**Deletion at the end:**

In the case of the first node, you just need access to the head and you can delete it. But in the case of the last node, you also need access to the second to the last node of the linked list as you will delete the last node and make the previous node as the tail of linked list.

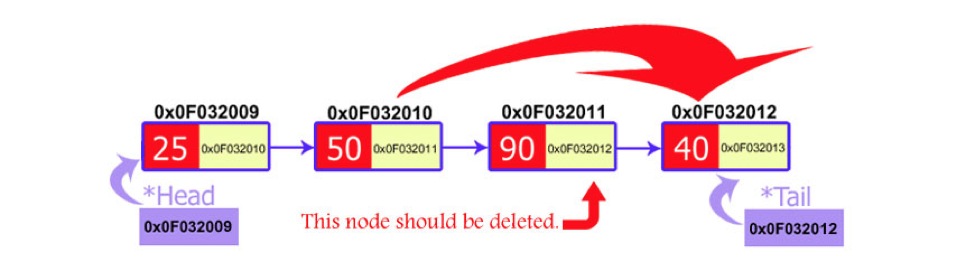
****

**Pseudo Code:**

* Check whether list is Empty (head == NULL)
* If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.
* If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1' with head.
* Check whether list has only one Node (temp1 → next == NULL)
* If it is TRUE. Then, set head = NULL and delete temp1. And terminate the function. (Setting Empty list condition)
* If it is FALSE. Then, set 'temp2 = temp1 ' and move temp1 to its next node. Repeat the same until it reaches to the last node in the list. (until temp1 → next == NULL)
* Finally, Set temp2 → next = NULL and delete temp1.

**Deletion at the particular point:**

In linked list, we can delete a specific node. We ask the user to input the position of the node to be deleted. After that, we just move two temporary pointers through the linked list until we reach our specific node. Now, we delete our current node and pass the address of the node after it to the previous pointer.

****

**Pseudo Code:**

* Check whether list is Empty (head == NULL)
* If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.
* If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1' with head.
* Keep moving the temp1 until it reaches to the exact node to be deleted or to the last node. And every time set 'temp2 = temp1' before moving the 'temp1' to its next node.
* If it is reached to the last node then display 'Given node not found in the list! Deletion not possible!!!'. And terminate the function.
* If it is reached to the exact node which we want to delete, then check whether list is having only one node or not
* If list has only one node and that is the node to be deleted, then set head = NULL and delete temp1 (free(temp1)).
* If list contains multiple nodes, then check whether temp1 is the first node in the list (temp1 == head).
* If temp1 is the first node then move the head to the next node (head = head → next) and delete temp1.
* If temp1 is not first node then check whether it is last node in the list (temp1 → next == NULL).
* If temp1 is last node then set temp2 → next = NULL and delete temp1 (free(temp1)).
* If temp1 is not first node and not last node then set temp2 → next = temp1 → next and delete temp1 (free(temp1)).

**Counter Function:**

Function for counting the singly linked nodes is very similar to display(), Only difference is that instead of printing data we are incrementing length variable.

**Pseudo Code:**

* We create a function count
* Define Pointer of node type temporary
* Initilialize the length variable to 0
* Temporary is equal to Start
* While Temporary is not pointing to NULL
* Length is incremented
* Temporary is equal to Temporary 🡪 next
* Print the length of linked list