Deleting a node in SLL

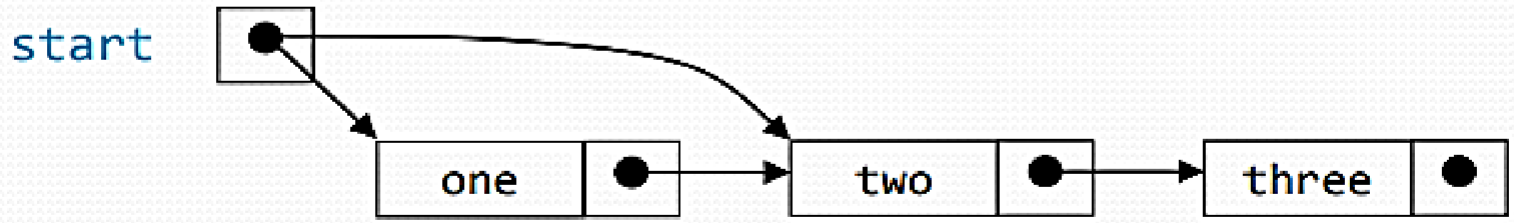
Here also we have three cases:-

- Deleting the first node
- Deleting the last node
- Deleting the intermediate node

Deleting the first node

Here we apply 2 steps:-

- Making the start pointer point towards the 2nd node
- Deleting the first node using delete keyword

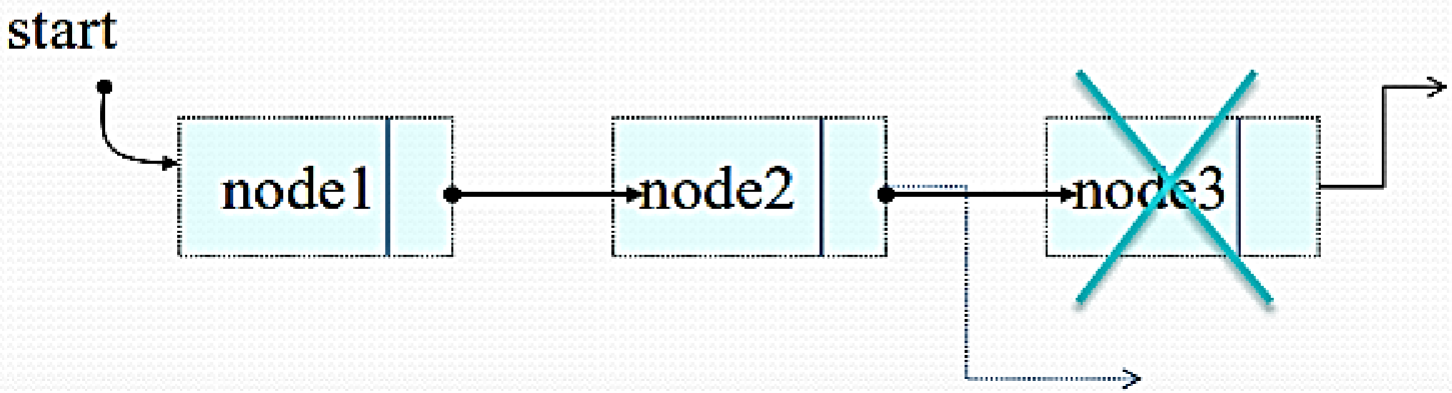


```
void del_first()
{
      if(start==NULL)
     cout<<"\nError.....List is empty\n";
     else
     {
       node* temp=start;
       start=temp->link;
       delete temp;
cout<<"\nFirst node deleted successfully....!!!";
     }
}
```

Deleting the last node

Here we apply 2 steps:-

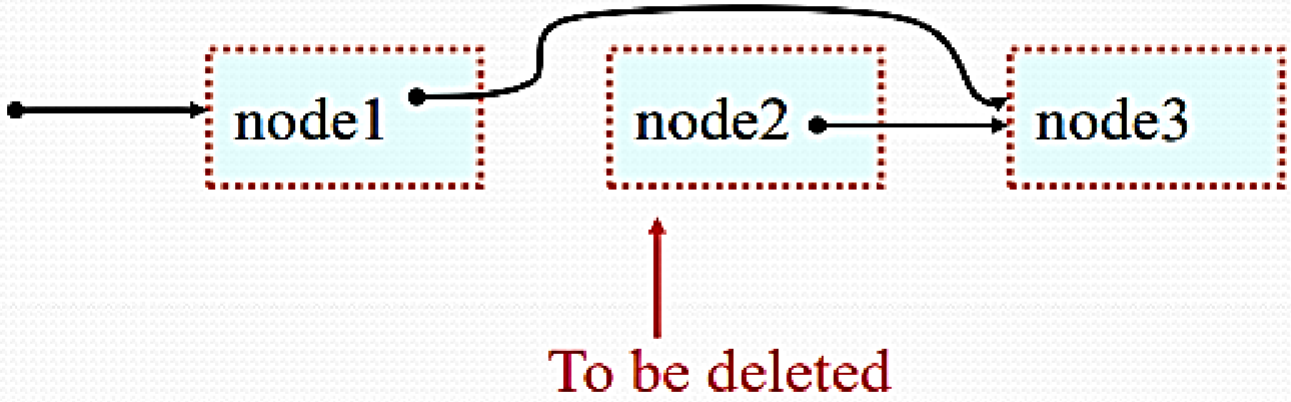
- Making the second last node's next pointer point to NULL
- Deleting the last node via delete keyword



```
void del_last()
{
        if(start==NULL)
        cout<<"\nError....List is empty";
        else
                {
                   node* q=start;
                            while(q->link->link!=NULL)
                             q=q->link;
                   node* temp=q->link;
                   q->link=NULL;
                   delete temp;
                   cout<<"\nDeleted successfully...";
                }
        }
}
```

Deleting a particular node

Here we make the next pointer of the node previous to the node being deleted ,point to the successor node of the node to be deleted and then delete the node using delete keyword



```
void del(int c)
{
node* q=start;
      for(int i=2;i<c;i++)
      {
       q=q->link;
             if(q==NULL)
             cout<<"\nNode not found\n";
      }
      if(i==c)
      {
       node* p=q->link;        //node to be deleted
      q->link=p->link;      //disconnecting the node p
      delete p;
      cout<<"Deleted Successfully";
      }
}
```

Searching a SLL

- Searching involves finding the required element in the list
- We can use various techniques of searching like linear search or binary search where binary search is more efficient in case of Arrays
- But in case of linked list since random access is not available it would become complex to do binary search in it
- We can perform simple linear search traversal

In linear search each node is traversed till the data in the node matches with the required value

```
void search(int x)
{
  node*temp=start;
    while(temp!=NULL)
      {
        if(temp->data==x)
         {
           cout<<"FOUND "<<temp->data;
           break;
         }
       temp=temp->next;
      }
}
```

Traversing a linked list

```
   struct Node {
       Object element;
       Node *next;
   };

Node *pWalker;
int count = 0;

cout <<"List contains:\n";

for (pWalker=pHead; pWalker!=NULL;
     pWalker = pWalker->next)
{
    count ++;
    cout << pWalker->element << endl;
}
```

Example 1

```
#include <iostream>
using namespace std;

Type defstruct node
{
   int data;
   node *next;
} Node;

Node *createNode(int n)
{
            Node *ptr = new Node();
            ptr->data = n;
            ptr->next = NULL;
            return ptr;
}
```

```
Node *appendNode(Node *node, int n)
{
        Node *cur = node;
        while(cur) {
                if(cur->next == NULL) {
                        cur->next = createNode(n);
                        return cur->next;
                }
                cur = cur->next;
        }
        return cur;
}

void printNodes(Node *head)
{
        Node *cur = head;
        while(cur) {
                cout << cur->data << " ";
                cur = cur->next;
        }
        cout << endl;
}
```

```
Node *delete Node(Node **head)
{
        Node *temp = *head;   // NOT Node *temp = head;
        *head = temp->next;   // NOT head = temp->next;
        delete temp;
        return *head;        // NOT return head;
}
```

```
Node *insertFront(Node **head, int n)
{
        Node *new Node = create Node(n);
        new Node->next = *head; // NOT newNode->next = head;
        *head = new Node;       // NOT head = newNode;
        return *head;          // NOT return head;
}

int main()
{
        Node *head = create Node(100);
        append Node(head, 200);
        append Node(head, 300);
        print Nodes(head);
        head = delete Node(&head;);
        print Nodes(head);
        head = insert Front(&head;, 100);
        print Nodes(head);

   return 0;
}
```

Example 2

```
/ Linked list implementation in C++

#include <bits/stdc++.h>
#include <iostream>
using namespace std;

// Creating a node
class Node {
   public:
  int value;
  Node* next;
};

int main() {
  Node* head;
  Node* one = NULL;
  Node* two = NULL;
  Node* three = NULL;
```

```
// allocate 3 nodes in the heap
  one = new Node();
  two = new Node();
  three = new Node();

  // Assign value values
  one->value = 1;
  two->value = 2;
  three->value = 3;

  // Connect nodes
  one->next = two;
  two->next = three;
  three->next = NULL;

  // print the linked list value
  head = one;
  while (head != NULL) {
    cout << head->value;
    head = head->next;
  }
}
```

Examples 3

```
#include <iostream>
using namespace std;

// Making a node struct containing an int data and a pointer
// to next node
struct Node {
  int data;
  Node *next;

  // Parameterised constructor with default argument
  Node(int val=0) :data(val),next(nullptr){}
  // Parameterise constructor
  Node(int val, Node *tempNext):data(val),next(tempNext){}
};
```

```
class LinkedList
{
  // Head pointer
  Node* head;

public:
  // default constructor. Initializing head pointer
  LinkedList():head(nullptr)
  {
  }

  // inserting elements (At start of the list)
  void insert(int val)
  {
    // make a new node
    Node* new_node = new Node(val);

    // If list is empty, make the new node, the head
    if (head == nullptr)
    {
      head = new_node;
    }
    // else, make the new_node the head and its next, the previous
```

```
else
  {
    new_node->next = head;
    head = new_node;
  }
}

// loop over the list. return true if element found
bool search(int val)
{
  Node* temp = head;
  while(temp != nullptr)
  {
    if (temp->data == val)
      return true;
    temp = temp->next;
  }
  return false;
}
```

```
  void remove(int val)
{
  Node* temp = head;
  // If the head is to be deleted
  if (temp != nullptr && temp->data == val)
  {
    head = temp->next;
    delete temp;
    return;
  }
  // Else loop over the list and search for the node to delete
  else
  {
    Node* curr = head;
    while(temp != nullptr && temp->data != val)
    {
    // When node is found, delete the node and modify the pointers
     curr = temp;
     temp = temp->next;
    }
```

```
// If values is not found in the linked list
      if(!temp)
      {
        cout << "Value not found" << endl;
        return;
      }

      curr->next = temp->next;
      delete temp;
    }
  }

  void display()
  {
    Node* temp = head;
    while(temp != nullptr)
    {
      cout << temp->data << " ";
      temp = temp->next;
    }
    cout << endl;
  }
};
```

```
int main() {

  Linked List l;
  // inserting elements
  l. insert(6);
  l. insert(9);
  l. insert(1);
  l. insert(3);
  l. insert(7);
  cout << "Current Linked List: ";
  l. display();

  cout << "Deleting 1: ";
  l.remove(1);
  l.display();

  cout << "Deleting 13: ";
  l.remove(13);

  cout << "Searching for 7: ";
  cout << l.search(7) << endl;

  cout << "Searching for 13: ";
  cout << l.search(13) << endl;
```

Output

Current Linked List: 7 3 1 9 6
Deleting 1: 7 3 9 6
Deleting 13: Value not found
Searching for 7: 1
Searching for 13: 0