NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES
CS 201 – DATA STRUCTURES LAB

Instructors: Faizan Yousuf & Abdul Aziz

LAB SESSION # 03

Outline

- Linked List Basics
- Simply Linked List
- Doubly Linked List
- Circular Linked List
- Exercise

Linked List Basics

A linked list is a sequence of data structures, which are connected together via links.

Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.

- **Link** − Each link of a linked list can store a data called an element.
- **Next** − Each link of a linked list contains a link to the next link called Next.
- **LinkedList** − A Linked List contains the connection link to the first link called First.

Types of Linked List

Following are the various types of linked list.

- **Simple Linked List** − Item navigation is forward only.
- **Doubly Linked List** − Items can be navigated forward and backward.
- **Circular Linked List** − Last item contains link of the first element as next and the first element has a link to the last element as previous.

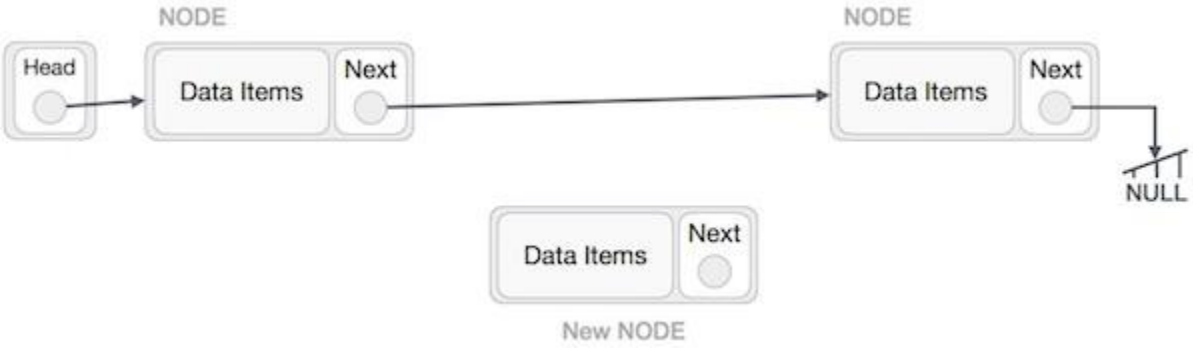
Single Linked List:

Representation

Linked list can be visualized as a chain of nodes, where every node points to the next node.

Insertion Operation

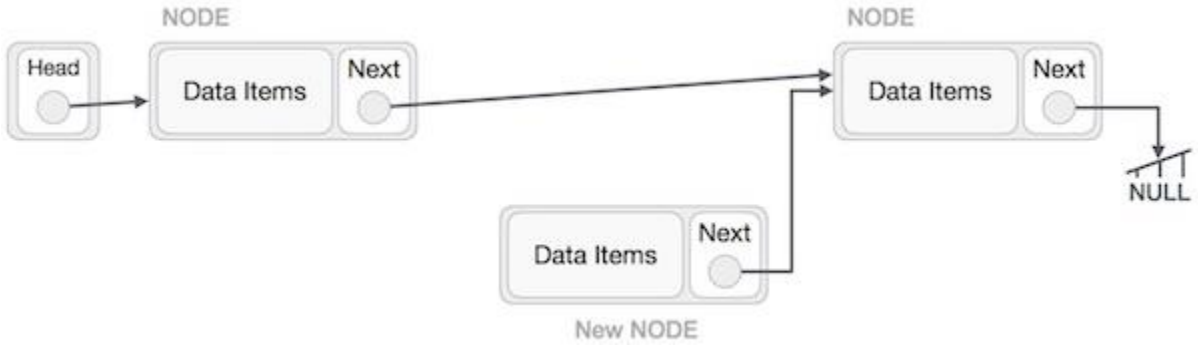
Adding a new node in linked list is a more than one step activity. We shall learn this with diagrams here. First, create a node using the same structure and find the location where it has to be inserted.



Imagine that we are inserting a node **B** (NewNode), between **A** (LeftNode) and **C** (RightNode). Then point B.next to C −

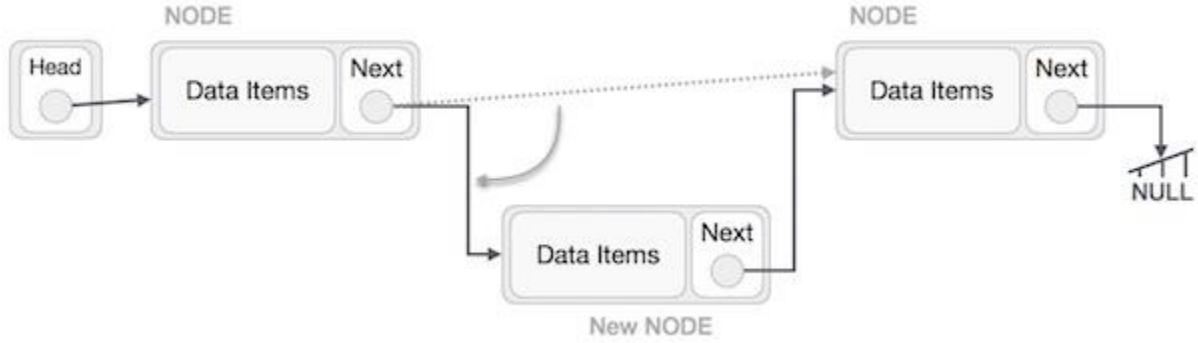
```
NewNode.next −> RightNode;
```

It should look like this −



Now, the next node at the left should point to the new node.

```
LeftNode.next −> NewNode;
```



This will put the new node in the middle of the two. The new list should look like this −

Similar steps should be taken if the node is being inserted at the beginning of the list. While inserting it at the end, the second last node of the list should point to the new node and the new node will point to NULL.

Deletion Operation

Deletion is also a more than one step process. We shall learn with pictorial representation. First, locate the target node to be removed, by using searching algorithms.

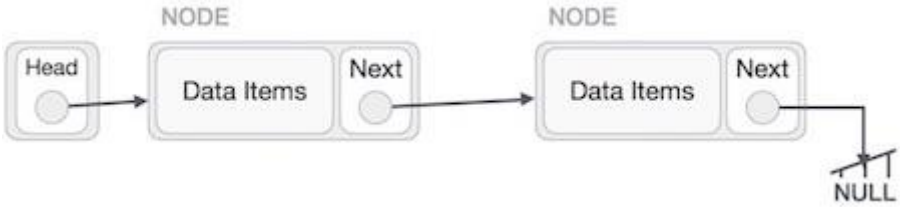
The left (previous) node of the target node now should point to the next node of the target node −

```
LeftNode.next −> TargetNode.next;
```

This will remove the link that was pointing to the target node. Now, using the following code, we will remove what the target node is pointing at.

```
TargetNode.next −> NULL;
```

We need to use the deleted node. We can keep that in memory otherwise we can simply deallocate memory and wipe off the target node completely.

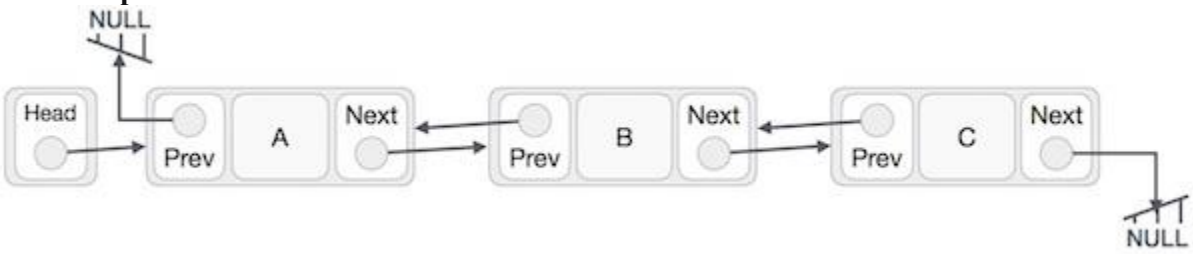


Doubly Linked List:

Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward and backward easily as compared to Single Linked List. Following is the important term to understand the concept of doubly linked list.

- **Prev** − Each link of a linked list contains a link to the previous link called Prev.

Doubly Linked List Representation



Insertion Operation

Following code demonstrates the insertion operation at the beginning of a doubly linked list.

Example

```
public void insert_first(int n)
     {
          Double_LinkList node = new Double_LinkList();
          if(head == null)
          {
               node.data=n;
               node.next=null;
               node.pre=null;//constant null
               head=node;
               temp=node;
          }
          else
          {
             node.data = n;
             node.pre = null;
             node.next = null;
             temp = head;
             temp.pre = node;
             node.next = temp;
             head = node;

          }
     }
```

Deletion Operation

Removing a node w from a DLList is easy. We only need to adjust pointers at w.next and w.prev so that they skip over w. Again, the use of the dummy node eliminates the need to consider any special cases:

Example

```
void remove(Node w)
{
        w.prev.next = w.next;
        w.next.prev = w.prev;
        n--;
}
```

Now the remove(i) operation is trivial. We find the node with index i and remove it:

Example

```
T remove(int i)
{
        Node w = getNode(i);
        remove(w);
```

```
        return w.x;
}
```

Insertion at the End of an Operation

Following code demonstrates the insertion operation at the last position of a doubly linked list.

Example

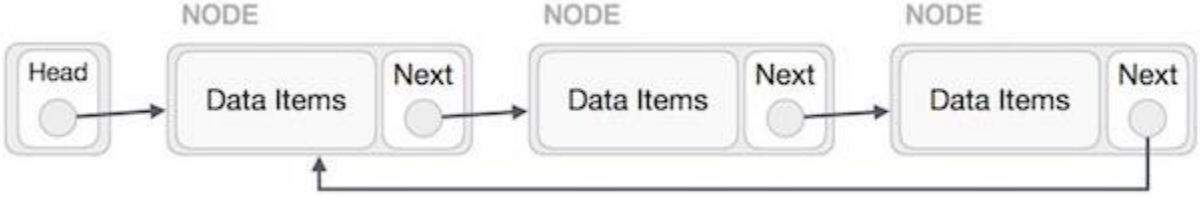
```
//insert link at the last location
public void insert_last(int n)
    {
       Double_LinkList node = new Double_LinkList();
       if(head == null)
       {
          node.data=n;
          node.next=null;
          node.pre=null;//constant null
          head=node;
          temp=node;
       }
       else
       {
          temp = head;
          while(temp.next!=null)
          {
            temp = temp.next;
          }
          node.data=n;
          node.next=null;
          temp.next = node;
          node.pre = temp;
          temp=node;
       }
    }
```

Circular Linked List:

Circular Linked List is a variation of Linked list in which the first element points to the last element and the last element points to the first element. Both Singly Linked List and Doubly Linked List can be made into a circular linked list.

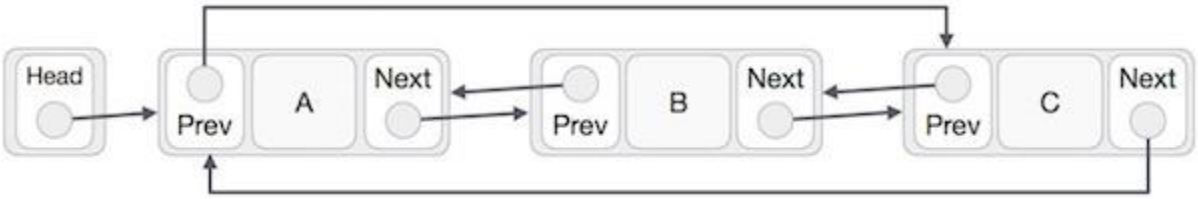
Singly Linked List as Circular

In singly linked list, the next pointer of the last node points to the first node.



Doubly Linked List as Circular

In doubly linked list, the next pointer of the last node points to the first node and the previous pointer of the first node points to the last node making the circular in both directions.



Insertion Operation

Following code demonstrates the insertion operation in a circular linked list based on single linked list.

Example

```
//insert link at the first location

public void insertAtStart(int val)
   {
      Node nptr = new Node(val,null);
      nptr.setLink(start);
      if(start == null)
      {
         start = nptr;
         nptr.setLink(start);
         end = start;
      }
      else
      {
         end.setLink(nptr);
         start = nptr;
      }
   }
```

Deletion Operation

Following code demonstrates the deletion operation in a circular linked list based on single linked list.

```
public void deleteAtPos(int pos)
   {
      if (size == 1 && pos == 1)
      {
         start = null;
         end = null;
         size = 0;
         return ;
      }
      if (pos == 1)
      {
         start = start.getLink();
         end.setLink(start);
         size--;
         return ;
      }
      if (pos == size)
      {
         Node s = start;
         Node t = start;
         while (s != end)
         {
            t = s;
            s = s.getLink();
         }
         end = t;
         end.setLink(start);
         size --;
         return;
      }
      Node ptr = start;
      pos = pos - 1 ;
      for (int i = 1; i < size - 1; i++)
      {
         if (i == pos)
         {
            Node tmp = ptr.getLink();
            tmp = tmp.getLink();
            ptr.setLink(tmp);
            break;
         }
         ptr = ptr.getLink();
      }
  }
```

Display List Operation

Following code demonstrates the display list operation in a circular linked list.

```
public void display()
```

```
    {
        System.out.print("\nCircular Singly Linked List = ");
        Node ptr = start;
        if (size == 0)
        {
            System.out.print("empty\n");
            return;
        }
        if (start.getLink() == start)
        {
            System.out.print(start.getData()+ "->"+ptr.getData()+ "\n");
            return;
        }
        System.out.print(start.getData()+ "->");
        ptr = start.getLink();
        while (ptr.getLink() != start)
        {
            System.out.print(ptr.getData()+ "->");
            ptr = ptr.getLink();
        }
        System.out.print(ptr.getData()+ "->");
        ptr = ptr.getLink();
        System.out.print(ptr.getData()+ "\n");
    }
}
```

Exercise:

Question No. 1:

Write a Count() function that counts the number of times a given int occurs in a list. The code for this has the classic list raversal structure as demonstrated in Length().

```
void CountTest() {
List myList = BuildOneTwoThree(); // build {1, 2, 3}
int count = Count(myList, 2); // returns 1 since there's 1 '2' in the list
}
/*
 Given a list and an int, return the number of times that int occurs
 in the list.
*/
int Count(struct node* head, int searchFor) {
// Your code
```

Question No. 2:

Write a GetNth() function that takes a linked list and an integer index and returns the data value stored in the node at that index position. GetNth() uses the C numbering convention that the first node is index 0, the second is index 1, ... and so on. So for the list {42, 13, 666} GetNth() with index 1 should return 13. The index should be in the range [0..length-1]. If it is not, GetNth() should assert() fail (or you could implement some other error case strategy).

```
void GetNthTest() {
struct node* myList = BuildOneTwoThree(); // build {1, 2, 3}
```

```
int lastNode = GetNth(myList, 2); // returns the value 3
}
```
Essentially, GetNth() is similar to an array[i] operation — the client can ask for elements by index number. However, GetNth() no a list is much slower than [ ] on an array. The advantage of the linked list is its much more flexible memory management — we can Push() at any time to add more elements and the memory is allocated as needed.

```
// Given a list and an index, return the data
// in the nth node of the list. The nodes are numbered from 0.
// Assert fails if the index is invalid (outside 0..lengh-1).
int GetNth(struct node* head, int index) {
// Your code
```

Question No. 3:

Your friend is an Intelligence officer at Pakistan Railway, his colleague gave him a news about Karachi Express incident.
Incident: A group of People Hijack a Cabin of a Train, and one of their member is hidden somewhere in train.
Your task is to implement the scenario using double linked list to help your friend.
Step1: Find a Hijacked Cabin
Step2: Than Go back to the Engine and start finding the Last member

Question No. 4:

You are a Network Manager; your head asks you to implement a Series but Circular Networking between devices in admin office.
Your task is to implement the Scenario using Circular linked list.
The Network have 1 Router and 6 End Devices.

Question No. 5:

Write a RemoveDuplicates() function which takes a list sorted in increasing order and deletes any duplicate nodes from the list. Ideally, the list should only be traversed once.

Question No. 6:

Write and test a method public void swapNodes(Node n) to search for a node containing a given data (as search keyword) and then swap this node with the next node. E.g. if the list 2->5->7->9 the call of the method swapNodes(5) will rearrange the list as 2->7->5->9, Note that each digits presents a complete node NOT the data saved inside.

Question No. 7:

Write and test a method public void reverse() to reverse the order of the nodes in the linked list. E.g. if the list a->b->c->d the call of the method reverse() will rearrange the list as d->c->b->a..