

Week-7

The Doubly Linked List, Examples: Image viewer, music player list etc. (to be used to explain concept of list).

DLL node, List Operations – Create, appending nodes, delete, search.

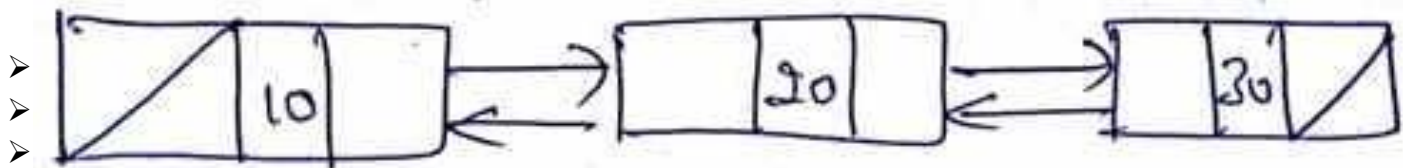
The Circular Linked List-Organization, List Operations – Appending nodes, delete, iterating circular list

The Doubly Linked List:

- A doubly linked list or a two-way linked list which contains a pointer to the next node as well as the previous node in the list.
- A Doubly Linked list is a list in which each node contains 2 link fields is called doubly Linked List.
- A Doubly Linked list is a sequence of data elements, which are connected together via 2 links.
- A Doubly Linked list is a collection of zero or more nodes where each node contains 3 fields, data, llink and rlink. (data, next, prev)
- Data field stores the actual information and llink field contains the address of the previous node and rlink contains the address of the next node.
- It allows us to traverse the list in the backward direction as well.
- The memory representation of a node is given below



- The memory representation of a doubly linked list is given below



- Left link field is a pointer which contains the address of the previous node.
- Last node's **rlink** field will be None and First node's **llink** field will be None.
- In this we can traverse in both directions.
- Every node in the doubly linked list points to the next element and previous elements in the linked list.

Applications of Doubly Linked list in real world:

1. **Image viewer** – Previous and next images are linked, hence can be accessed by next and previous button.
2. **Previous and next page in web browser** – We can access previous and next URL searched in web browser by pressing back and next button since, they are linked as linked list.
3. **Music Player** – Songs in music player are linked to previous and next song. We can play songs either from starting or ending of the list.
4. **Used to implement undo/redo operations.**
5. **In Navigation Systems for front and back navigation.**

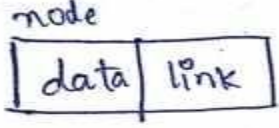
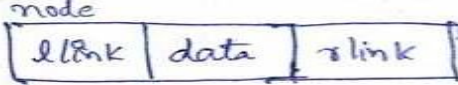
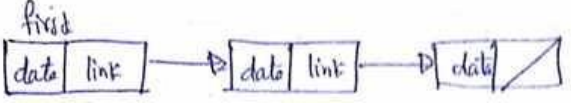

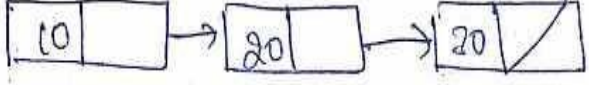
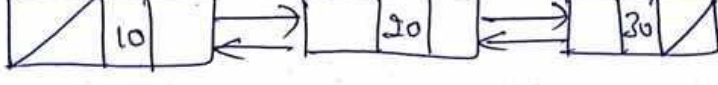
Advantages of DLL:

1. **Reversing a Doubly Linked List is very easy.**
2. **Bi-directional traversal is possible**
3. **Implementation of DLL is easy.**

Disadvantages of DLL:

1. **Uses extra memory for one extra link per node.**
2. **No random access to elements in DLL.**

Compare Singly Linked List with Doubly Linked List:

Singly Linked List	Doubly Linked List
Collection of nodes and each node contains of 2 fields i.e. data & link.	Collection of nodes & each node contains 3 fields i.e. data, left link and right link.
Node is represented as follows 	Node is represented as follows 
The elements can be accessed using link field only	The elements can be accessed using both left link and right link fields.
It contains only the address of the next node.	It contains address of both right node and left node.
It takes less memory	It takes more memory.
Less efficient access to elements	More efficient access to elements
Singly Linked List is represented as follows 	Doubly Linked List is represented as follows 
Ex:- 	Ex:- 

DLL Nodes:

- In Doubly Linked list is each node contains 3 fields, data, next and prev.
- The **data** field stores the actual information.
- The **next** field contains the address of the previous node.
- The **prev** field contains the address of the next node.
- The python code to create a new node of the doubly linked list is given below.

class Node:

def __init__(self, data = None):

self.data = data

self.next = None

self.prev = None

- **temp = Node(10)** will create a node with value 10.

List Operations - Creation of Doubly Linked List:

We can create a singly linked list in python by following the mentioned steps.

Step 1: Create a node class with 3 required variables.

Step 2: Create the Doubly Linked List class and declare the first node.

Step 3: Create a new node and assign it a value.

Step 4: Set the **next** reference of the newly created node to **None**.

Step 5: Set the **prev** reference of the newly created node to **None**.

Step 6: we will link first to this node by putting in its reference address

Step 7: Terminate.

The python code to create a new node of the doubly linked list is given below.

class Node:

def __init__(self, data = None):

self.data = data

self.next = None

self.prev = None

The python code to create a doubly linked list and initialize first reference.

class DoublyLinkedList:

def __init__(self):

self.first = None

dll = DoublyLinkedList()

List Operations - Appending Nodes:

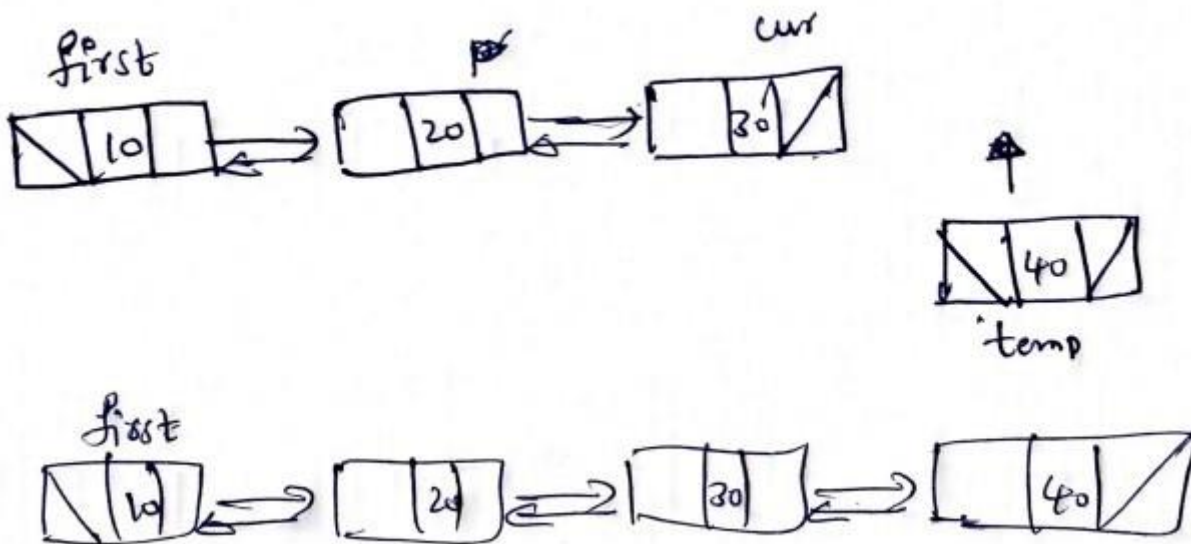
The following are the steps to be followed to append a new node to the doubly linked list.

Step 1:- Create a new node and assign a value to it.

Step 2:- If the list is empty then make new node as first. Go to step 4

Step 3:- Search for the last node. Once found set its **next** reference pointing to the new node and set the **prev** reference of the new node to the last node.

Step 4: Terminate.



The python code to append new node to the doubly linked list is given below.

```
def insertAtEnd(self, data):
    temp = Node(data)
    if(self.first == None):
        self.first=temp
    else:
        cur = self.first
        while(cur.next != None):
            cur = cur.next
        cur.next = temp
        temp.prev = cur
```

List Operations - Delete:

The following are the steps to be followed to remove node from the beginning of the list.

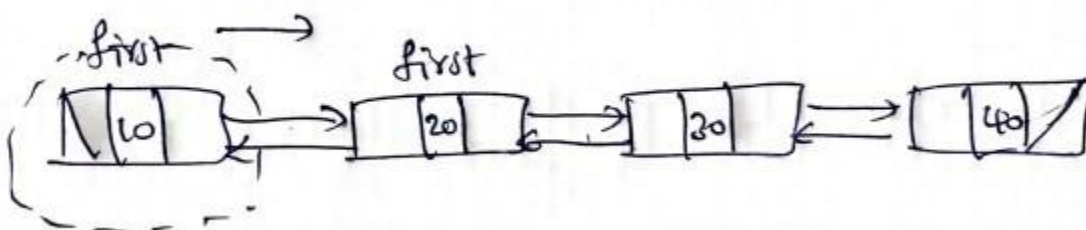
Step 1: If the linked list is empty, return and go to step 5.

Step 2: If there's only one element, delete that node and set first to **None**. Go to step 5.

Step 3: Set a "cur" node pointing at first node.

Step 4: Assign first as the next node. Set the **prev** reference of first node to **None**. Delete the cur node.

Step 5: Terminate



The python code to delete node from the linked list is given below.

```
def deleteFirst(self):
    if(self.first== None):
        print("list is empty")
    elif(self.first.next == None):
        print("the deleted item is",self.first.data)
        self.first = None
    else:
        cur=self.first
        self.first=self.first.next
        self.first.prev = None
        print("the deleted item is",cur.data)
```

List Operations - Traverse:

- A Doubly Linked list can only be traversed in forward direction from the first node to the last and in backward direction from last node to first.
- We get the value of the next data element by simply iterating with the help of the reference addresses **next** and **prev**.

- The following are the steps to be followed to traverse the doubly linked list.

Step 1: If the linked list is empty, display the message “List is empty” and move to step 4.

Step 2: Iterate over the linked list using the reference address **next** for each node until the last node.

Step 3: Print every node’s data.

Step 4: Terminate.

- The python code to traverse the nodes of doubly linked list is given below.

```
def display(self):
    if(self.first== None):
        print("list is empty")
        return
    current = self.first
    while(current):
        print(current.data, end = " ")
        current = current.next
```

List Operations - Searching for a Node:

- To find a node in a given doubly linked list, we use the technique of traversal. In this case as soon as we find the node, we will terminate the loop.

- Algorithm to search a given node from the linked list is given below

Step 1: If the linked list is empty, display the message “List is empty” and move to step 5.

Step 2: Iterate over the linked list using the reference address **next** for each node.

Step 3: Search every node for the given value.

Step 4: If the element is found, print the message “Element found” and return. If not, print the message “Element not found”.

Step 5: Terminate.

- The python code implementation of searching the nodes of linked list is given below.

```
def search(self,item):
    if(self.first== None):
        print("list is empty")
        return
    cur = self.first
    while cur != None:
        if cur.data == item:
            print("Item is present in the Linked list")
            return
        else:
            cur = cur.next
    print("Item is not present in the Linked list")
```

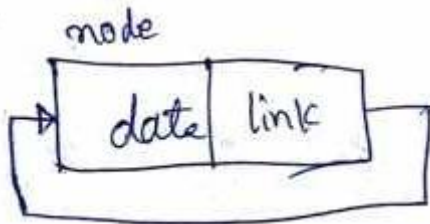
Implementation of Doubly linked list (Traversing the Nodes, searching for a Node, Appending Nodes, Deleting Nodes):

```
class Node:
    def __init__(self, data = None):
        self.data = data
        self.next = None
        self.prev = None
class DoublyLinkedList:
    def __init__(self):
        self.first = None
    def insertAtEnd(self, data):
        temp = Node(data)
        if(self.first == None):
            self.first=temp
        else:
            cur = self.first
            while(cur.next != None):
                cur = cur.next
            cur.next = temp
            temp.prev = cur
    def deleteFirst(self):
        if(self.first== None):
            print("list is empty")
        elif(self.first.next == None):
            print("the deleted item is",self.first.data)
            self.first = None
        else:
            cur=self.first
            self.first=self.first.next
```

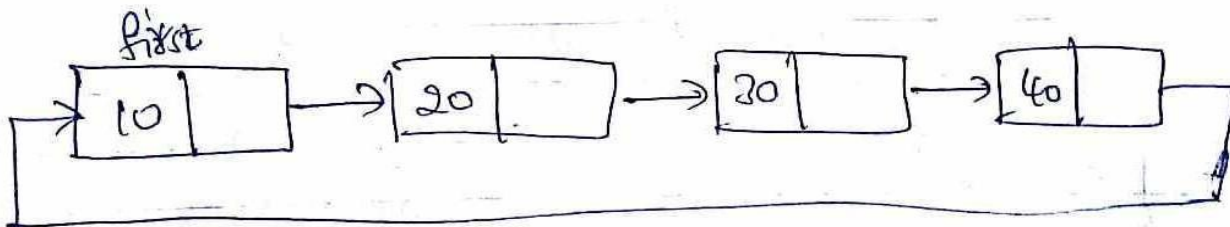
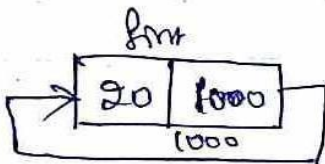
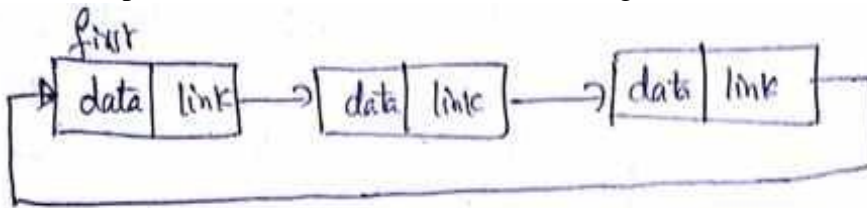
```
        self.first.prev = None
        print("the deleted item is",cur.data)
def display(self):
    if(self.first== None):
        print("list is empty")
        return
    cur = self.first
    while(cur):
        print(cur.data, end = " ")
        cur = cur.next
def search(self,item):
    if(self.first== None):
        print("list is empty")
        return
    cur = self.first
    while cur != None:
        if cur.data == item:
            print("Item is present in the Linked list")
            return
        else:
            cur = cur.next
    print("Item is not present in the Linked list")
#Doubly Linked List
dll = DoublyLinkedList()
while(True):
    ch = int(input("\nEnter your choice 1-insert 2-delete 3-search 4-display 5-exit :"))
    if(ch == 1):
        item = input("Enter the element to insert:")
        dll.insertAtEnd(item)
        dll.display()
    elif(ch == 2):
        dll.deleteFirst()
        dll.display()
    elif(ch == 3):
        item = input("Enter the element to search:")
        dll.search(item)
    elif(ch == 4):
        dll.display()
    else:
        break
```


Circular Linked List:

- A Circular Linked List is a list in which the link field of the last node contains the address of the first node of a list.
- While traversing a circular linked list, we can begin at any node and traverse the list until we reach the same node we started.
- A Singly Linked List can be converted into Circular Linked List by simply replacing the **None** value of the last node's link field to the address of the very first node of the Singly Linked List.
- In Circular Linked List the first node follows the last node therefore we have no first node or last node explicitly in a Circular Linked List.
- In a Circular Linked List from any given node we can move forward to the starting point.
- The Circular Linked List node is represented as follows.



- Each node contains two fields i.e. data and link.
- Pictorial representation of Circular Linked List is given below



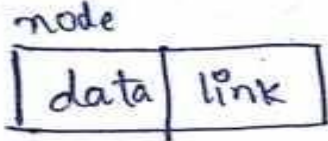
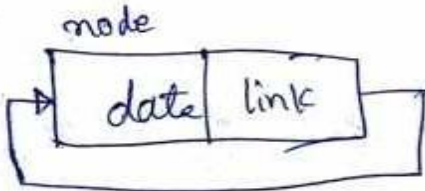
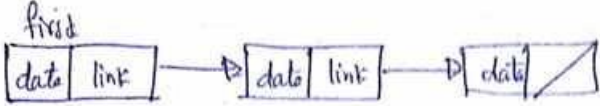
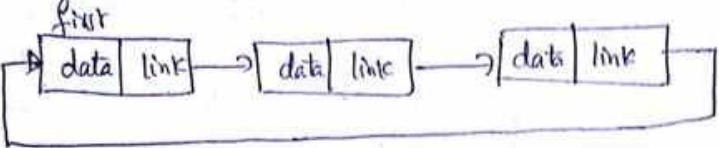

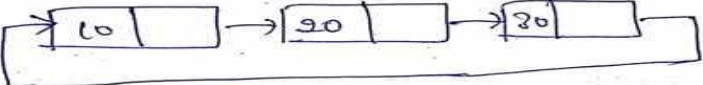
- A Circular Linked List Node is created with the following syntax.
- ```
class Node:
 def __init__(self, data = None):
 self.data = data
 self.next = None
```



where

- data field contains information
- next field contains address of the next node.

### Compare Singly Linked List with Circular Linked List:

| Singly Linked List                                                                                                                        | Circular Linked List                                                                                                                         |
|-------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| It is a Linear Linked List in which nodes are arranged linearly adjacent to each other.                                                   | It is a kind of Linked List in which the nodes are adjacent to each other but the first node appears next to the last node.                  |
| Link field of the last node is NULL                                                                                                       | Link field of last node contains the address of the first node.                                                                              |
| During traversal once we reach the last node it is not possible to visit the first node.                                                  | During traversal it is very easy to visit first node from the last node.                                                                     |
| In Singly Linked List node is represented as follows<br> | In Circular Linked List node is represented as follows<br> |
| Singly Linked List is represented as follows<br>       | Circular Linked List is represented as follows.<br>      |
| Ex:<br>                                                | Ex:-<br>                                                 |

### List Operations - Creation of Circular Linked List:

We can create a Circular linked list in python by following the mentioned steps.

Step 1: Create a node class with 2 required variables.

Step 2: Create the Circular Linked List class and declare the first node.

Step 3: Create a new node and assign it a value.

Step 4: Set the **next** reference of the newly created node to **itself**.

Step 5: we will link first to this node by putting in its reference address

Step 6: Terminate.

The python code to create a new node of the doubly linked list is given below.

**class Node:**

**def \_\_init\_\_(self, data = None):**

**self.data = data**

```
self.next = None
```

The python code to create a circular linked list and initialize first reference.

```
class CircularLinkedList:
```

```
 def __init__(self):
```

```
 self.first = None
```

```
cil = CircularLinkedList ()
```

### List Operations - Appending Nodes:

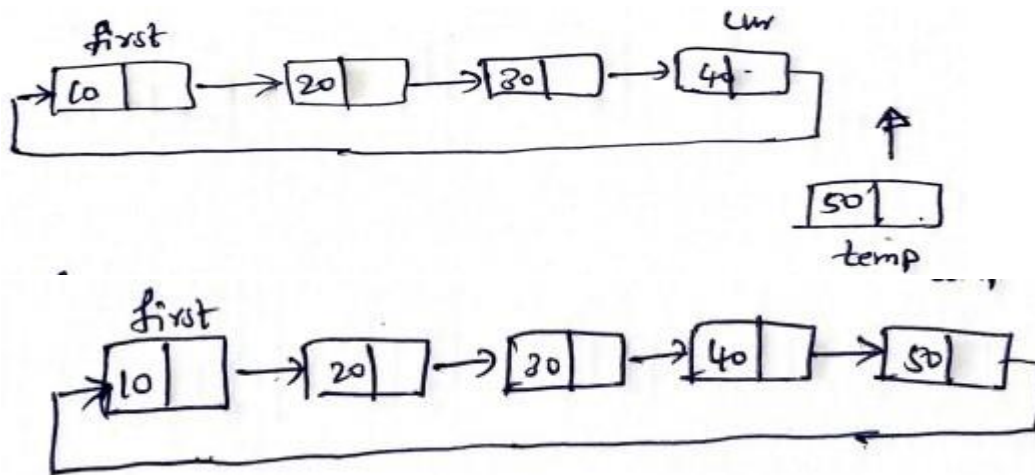
The following are the steps to be followed to append a new node to the circular linked list.

Step 1:- Create a new node and assign a value to it.

Step 2:- If the list is empty then make new node as first. Go to step 4

Step 3:- Search for the last node. Once found set its **next** reference to the new node and set the **next** reference of the new node to the first node.

Step 4: Terminate.



The python code to append new node to the circular linked list is given below.

```
def insertAtEnd(self, data):
```

```
 temp = Node(data)
```

```
 if(self.first == None):
```

```
 self.first = temp
```

```
 self.first.next = temp
```

```
 else:
```

```
 cur = self.first
```

```
 while(cur.next != self.first):
```

```
 cur = cur.next
```

```
 cur.next = temp
```

```
 temp.next = self.first
```

### List Operations - Delete:

The following are the steps to be followed to delete a node at the end of the list.

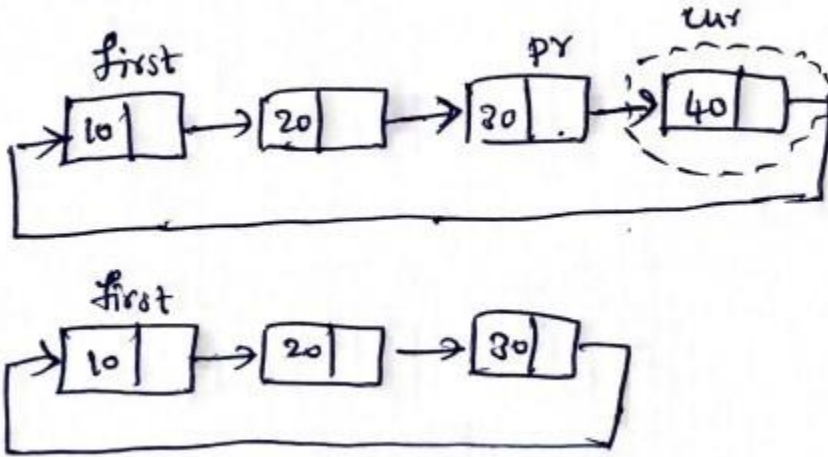
Step 1: If the linked list is empty, return and go to step 5.

Step 2: If there's only one element, delete that node and set first to **None**. Go to step 5.

Step 3: Set a "cur" node pointing at first node.

Step 4: Search for the last node and its previous node. Once found assign its previous node's next reference to first and delete the last node.

Step 5: Terminate



The python code to delete node from the circular linked list is given below.

```
def deleteAtEnd(self):
 if(self.first== None):
 print("list is empty")
 elif(self.first.next == self.first):
 print("the deleted item is",self.first.data)
 self.first = None
 else:
 cur=self.first
 while(cur.next != self.first):
 pr = cur
 cur = cur.next
 pr.next = self.first
 print("the deleted item is",cur.data)
```

### List Operations - Traverse:

- A Circular Linked list can only be traversed in forward direction from the first node to the last.
- We get the value of the next data element by simply iterating with the help of the reference address.
- The following are the steps to be followed to traverse the circular linked list.

Step 1: If the linked list is empty, display the message "List is empty" and move to step 4.

Step 2: Iterate over the linked list using the reference address **next** for each node until the last node.

Step 3: Print every node's data.

Step 4: Terminate.

- The python code to traversing the nodes of linked list is given below.

```
def display(self):
 if(self.first== None):
```

```

 print("list is empty")
 return
cur = self.first
while(True):
 print(cur.data, end = " ")
 cur = cur.next
 if(cur == self.first):
 break

```

### List Operations - Searching for a Node:

- To find a node in a given circular linked list, we use the technique of traversal. In this case as soon as we find the node, we will terminate the loop.
- Algorithm to search a given node from the linked list is given below
  - Step 1: If the linked list is empty, display the message “List is empty” and move to step 5.
  - Step 2: Iterate over the linked list using the reference address **next** for each node.
  - Step 3: Search every node for the given value.
  - Step 4: If the element is found, print the message “Element found” and return. If not, print the message “Element not found”.
  - Step 5: Terminate.
- The python code implementation of searching the nodes of linked list is given below.

```

def search(self,item):
 if(self.first== None):
 print("list is empty")
 return
 cur = self.first
 while cur.next != self.first:
 if cur.data == item:
 print("Item is present in the linked list")
 return
 else:
 cur = cur.next
 print("Item is not present in the linked list")

```

### Implementation of Circular linked list (Traversing the Nodes, searching for a Node, Appending Nodes, and Deleting Nodes):

```

class Node:
 def __init__(self, data = None):
 self.data = data
 self.next = None
class CircularLinkedList:
 def __init__(self):
 self.first = None
 def insertAtEnd(self, data):

```

```
temp = Node(data)
if(self.first == None):
 self.first = temp
 self.first.next = temp
else:
 cur = self.first
 while(cur.next != self.first):
 cur = cur.next
 cur.next = temp
 temp.next = self.first
def deleteAtEnd(self):
 if(self.first == None):
 print("list is empty")
 elif(self.first.next == self.first):
 print("the deleted item is",self.first.data)
 self.first = None
 else:
 cur=self.first
 while(cur.next != self.first):
 pr = cur
 cur = cur.next
 pr.next = self.first
 print("the deleted item is",cur.data)
def display(self):
 if(self.first == None):
 print("list is empty")
 return
 cur = self.first
 while(True):
 print(cur.data, end = " ")
 cur = cur.next
 if(cur == self.first):
 break
def search(self,item):
 if(self.first == None):
 print("list is empty")
 return
 cur = self.first
 while cur.next != self.first:
 if cur.data == item:
 print("Item is present in the linked list")
 return
 else:
```

```
 cur = cur.next
 print("Item is not present in the linked list")
#Circular Linked List
ccl = CircularLinkedList()
while(True):
 ch = int(input("\nEnter your choice 1-insert 2-delete 3-search 4-display 5-exit :"))
 if(ch == 1):
 item = input("Enter the element to insert:")
 ccl.insertAtEnd(item)
 ccl.display()
 elif(ch == 2):
 ccl.deleteAtEnd()
 ccl.display()
 elif(ch == 3):
 item = input("Enter the element to search:")
 ccl.search(item)
 elif(ch == 4):
 ccl.display()
 else:
 break
```