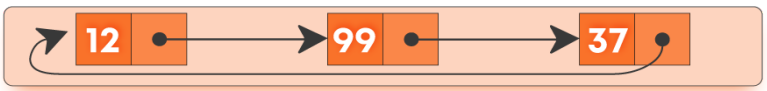
What are circular linked lists?

Circular linked lists are just linked lists where the next pointer of the last node is pointing to the first node of the list. There is no head node.

Why circular linked lists?

The advantage of using a circular linked list is that when we want to traverse in only one direction and move to the first node cyclically, we don't need to store additional pointers to mark the first and the last node. Typical usage of this concept is to implement queues using one pointer.



Operations on circular linked lists

- **insertAtBeginning(data)**: Inserting a node in front of the head of a linked list.
- **deleteFromBeginning**: Deleting a node from the front of a linked list.
- **display**: Display the contents of the linked list

Implementation of circular linked list

Circular Linked Lists will have one pointer: **head**
Each node in a circular-linked list has two properties: **data, next(pointer to the next node).**

- **Insert at beginning**

```
function insertAtBeginning(data)
        /*
                create a new node : newNode
                set newNode's data to data
         */

        newNode.data =  data

        //   If list is empty, set head as newNode
        if head is null
                head = newNode
                head.next = head
                return head
```

```
// traverse to the end of the circular list

temp = head
while temp.next != head
        temp = temp.next

// set the next of temp as newNode and return head

newNode.next = head
temp.next = newNode
head = newNode
return head
```

- **Delete from beginning**

```
function deleteFromBeginning()

    //  if the head is null, return
    if head is null
        print "Linked List is Empty"
        return head
    if head.next == head
        head = NULL
        return head

    //  traverse to the end of the circular list

    temp = head
    while temp.next != head
        temp = temp.next

    //  set the next of temp as next of head

    temp.next = head.next
    delete head

    head = temp.next
    return head
```

- **Display**

```
function display()
        /*
                create a new node : newNode
                set newNode's data to data
         */

        //   If the list is empty, print nothing
        if head is null
                print "list empty"
                return

         // traverse to the end of the circular list until head is encountered

         temp = head
         while temp.next != head
                 print (temp.data)
                 temp = temp.next
         print(temp.data)

         return
```

Time Complexity of various operations

Let 'n' be the number of elements in the linked lists. The complexities of linked list operations with this representation can be given as:

Operations	Time Complexity
insertAtBeginning(data)	O(n)
deleteFromBeginning()	O(n)
display()	O(n)

Applications of Circular Linked Lists

- They are used for implementations of data structures like Fibonacci heap where a circular doubly linked list is used.
- They can be used to implement circular queues that have applications in CPU scheduling algorithms like round-robin scheduling.
- They are used to switch between players in multiplayer games