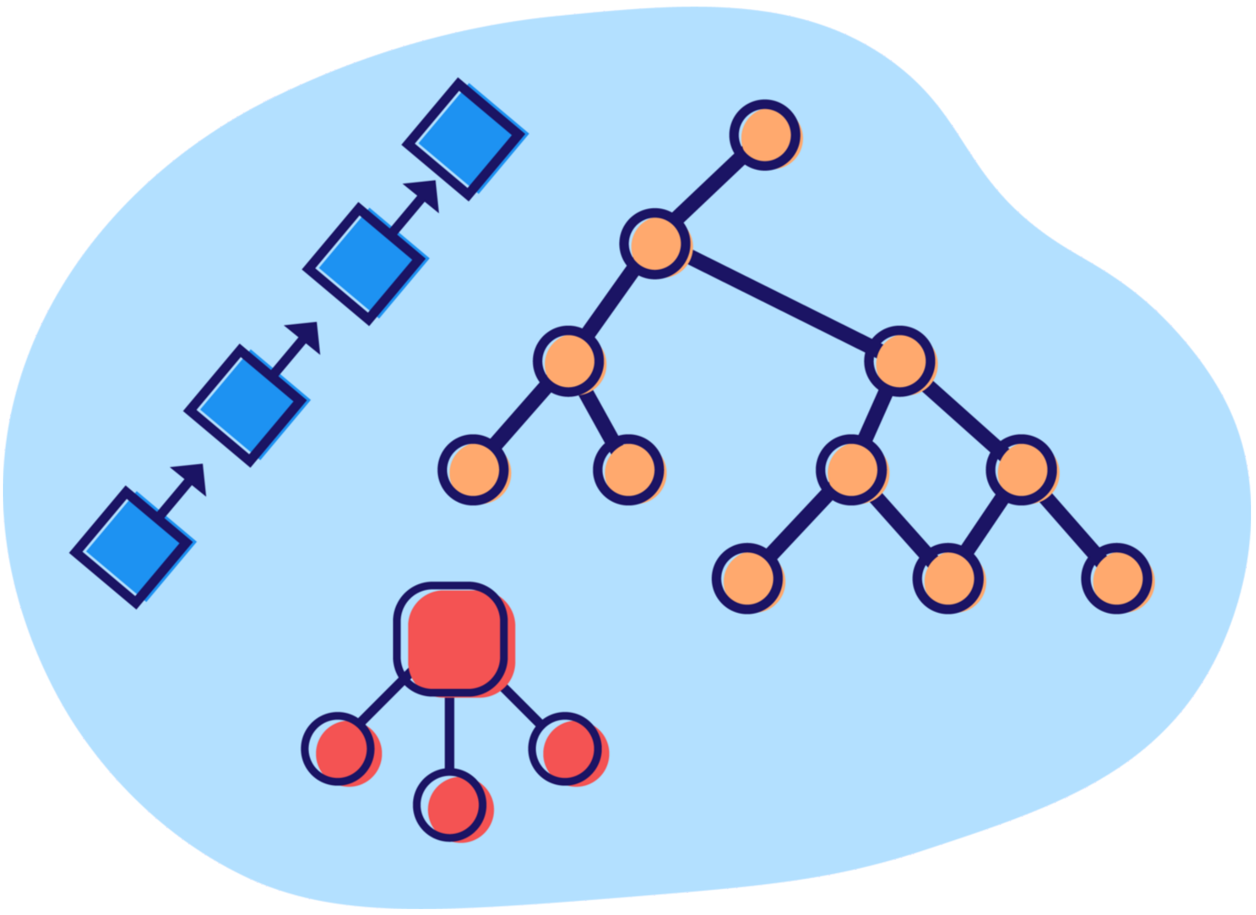


Data Structures & Algorithms

Chapter - 04.03

Circular Linked List

Rakin Mohammad Sifullah
Computer Science Engineer

Overview

In this chapter, we will cover Linked Lists.

We will cover the following -

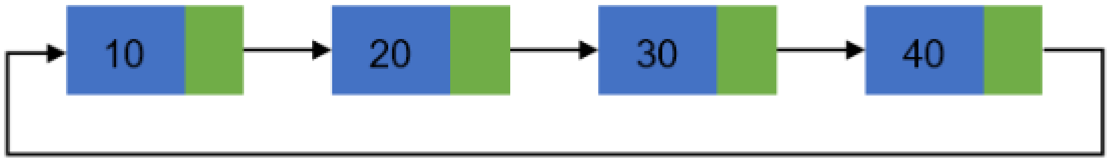
1. Introduction
2. Basic Operations
3. Advantages & Disadvantages of a Circular linked list
4. Applications/Uses of Circular linked list in real life
5. **Traverse Circular Linked List**
6. Insert

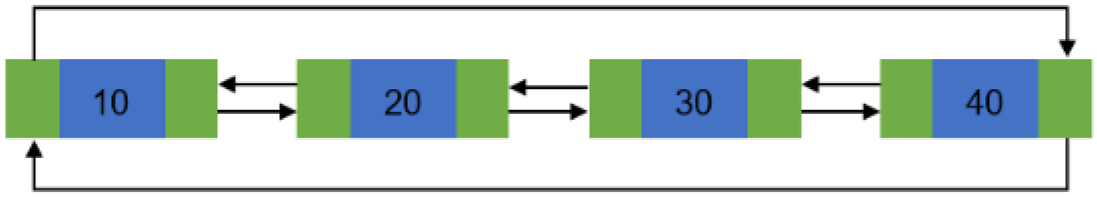
Introduction

A circular linked list is basically a linear linked list that may be singly or doubly. The only difference is that there is no any NULL value terminating the list. In fact in the list every node points to the next node and the last node points to the first node, thus forming a circle. Since it forms a circle with no end to stop, it is called a circular linked list.

In a circular linked list, there can be no starting or ending node, the whole node can be traversed from any node. In order to traverse the circular linked list only once we need to traverse the entire list until the starting node is not traversed again.

A circular linked list can be implemented using both a Singly Linked List and a Doubly Linked List. Here is the logical structure of a circular linked list.





As per the above illustration, the following are the important points to be considered.

1. The last link next points to the first link of the list in both cases of single as well as a doubly linked list.
2. The first link's previous points to the last of the list in case of a doubly linked list.

Basic Operations

Following are the essential operations supported by a circular list.

1. insert − Inserts an element at the start of the list.
2. delete − Deletes a part from the start of the list.
3. display − Displays the list.

Advantages & Disadvantages of a Circular linked list

Advantages:

1. The entire list can be traversed from any node.
2. Circular lists are the required data structure when we want a list to be accessed in a circle or loop.
3. Despite being a singly circular linked list we can easily traverse to its previous node, which is not possible in the singly linked list.

Disadvantages:

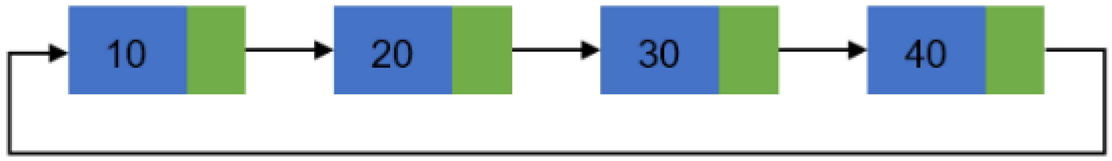
1. Circular lists are complex as compared to singly linked lists.
2. Reversing of the circular lists is complex as compared to singly or doubly lists.
3. If not traversed carefully, then we could end up in an infinite loop.
4. Like single and doubly lists circular linked lists also don't support direct access to elements.

Applications/Uses of Circular linked list in real life

1. Circular lists are used in applications where the entire list is accessed one by one in a loop. Example: Operating systems may use it to switch between various running applications in a circular loop.
2. It is also used by the Operating system to share time with different users, generally using a Round-Robin time sharing mechanism.
3. Multiplayer games use circular lists to swap between players in a loop.

Traverse Circular Linked List

Write a C program to create a circular linked list of n nodes and traverse the list how to create a circular linked list of n nodes and display all elements of the list in C. Algorithm to construct and traverse a Circular linked list.



Circular Linked List (Algorithm):

```
%%Input : $N$ {Total number of nodes to be created}
Being:
    alloc ($head$)
    read ($data$)
    $head.data \leftarrow data$;
    $head.next \leftarrow$ NULL;
    $prevNode \leftarrow head$;
    For $count \leftarrow 2$ to $N$ do
        alloc ($newNode$)
        read ($data$)
        $newNode.data \leftarrow data$;
        $newNode.next \leftarrow$ NULL;
        $prevNode.next \leftarrow newNode$;
        $prevNode \leftarrow newNode$;
    End for
    $prevNode.next \leftarrow head$;

End
```

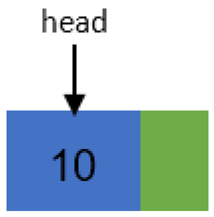
Traverse Circular Linked List (Algorithm):

```
%%Input : head {Pointer to the first node of the list}
Begin:
    If (head == NULL) then
        write ('List is empty')
    Else then
        current ← head;
        Do
            write ('Data =', current.data)
            current ← current.next;
        While (current != head)
    End if

End
```

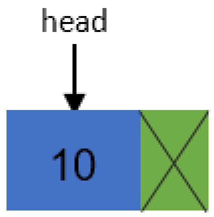
Steps to create circular linked list

The creation steps of a circular linked list are almost similar to those of a singly linked list. A circular linked list only differs in the last stage of its creation from a singly linked list.

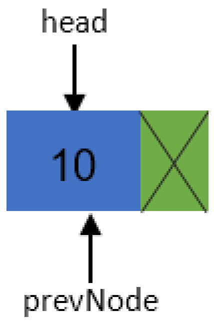
1. Create a head node and assign some data to its data field.



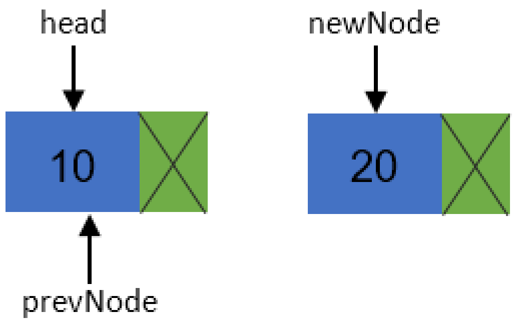
2. Make sure that the next pointer field of the head node should point to NULL.



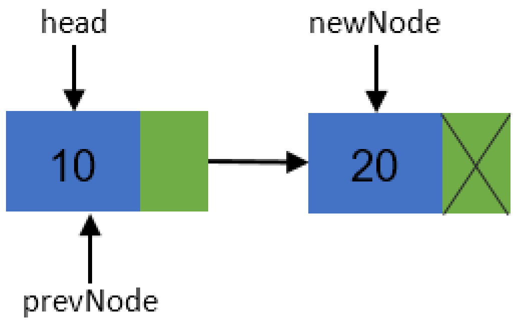
3. Now head node has been created that points to the first node of the list. Let's take another pointer that also points to the first node say provide a pointer.



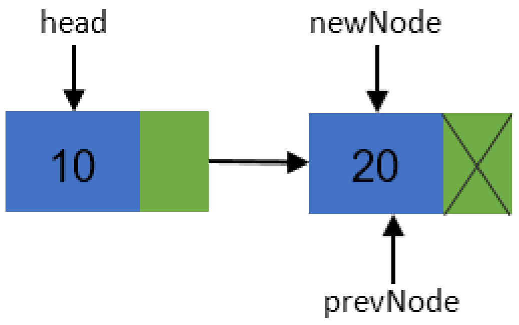
4. Create a newNode and assign some more value to its data field and also do sure that the next pointer field of the newNode should point to NULL.



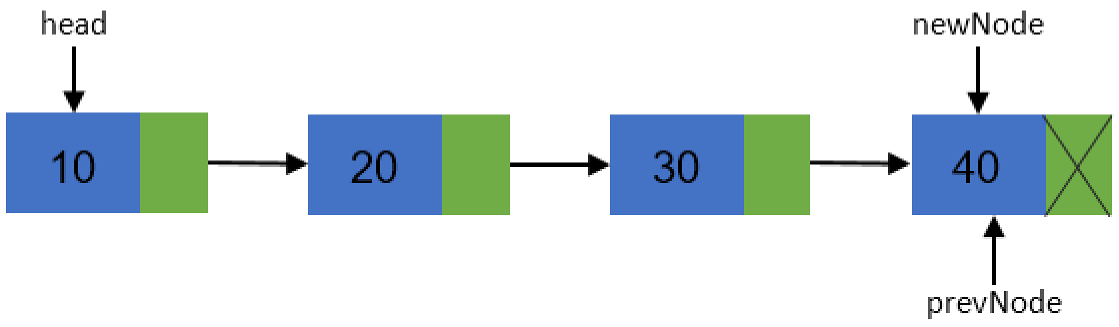
5. Now connect the previous node with the newly created node i.e. connect the next pointer field of provide with the newNode. So the next pointer field provides points to the newNode.



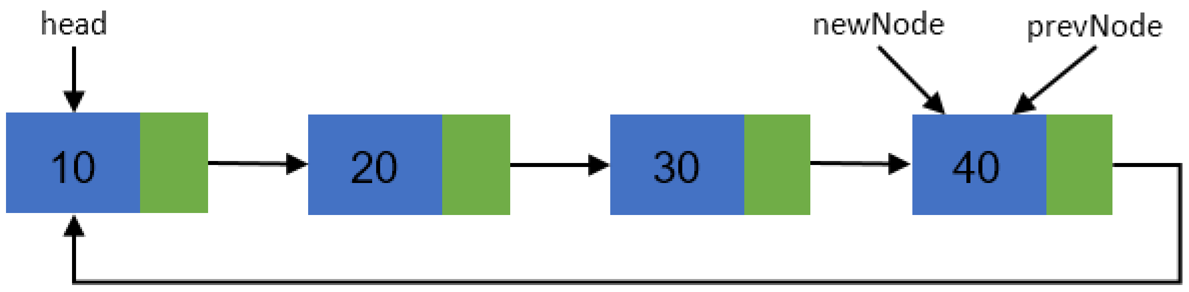
6. Move the prevNode ahead i.e. prevNode should point to newNode which can be done as prevNode = prevNode.next.



7. Repeat steps 4-6 till N (where N is the total number of nodes to be created).

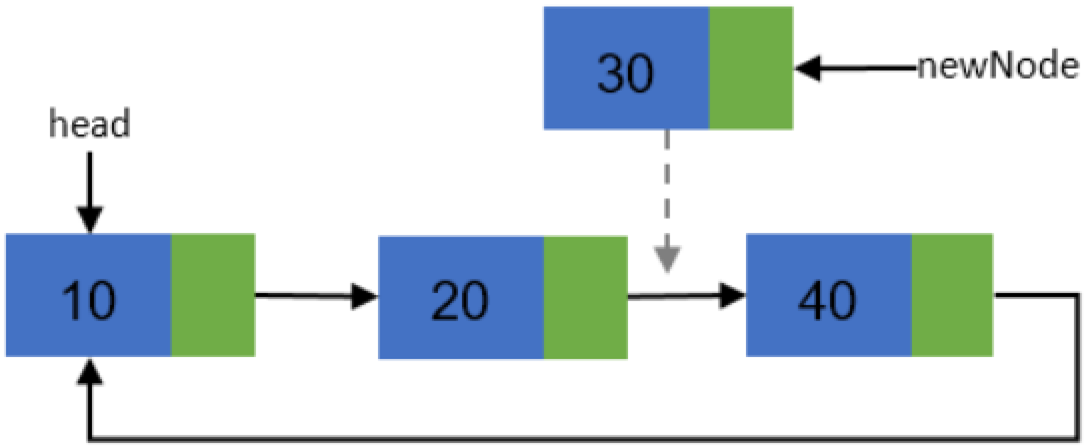


8. After all, nodes have been created. Now at the final stage, we have to connect the last node with the first node in order to make it circular. Therefore, now connect the next pointer field of provide with the head node i.e. provide. next = head (Since providing points to the last node of the list) and you are done.



Insert

Write a program to create a circular linked list and insert a new node at the beginning or at any position in the given list. How to insert a new node at the beginning of a circular linked list in C. How to insert a new node at any position in a circular linked list in C. Algorithm to insert a new node in a Circular linked list in the C program.



Algorithm to insert a new node at the beginning of the Circular linked list:

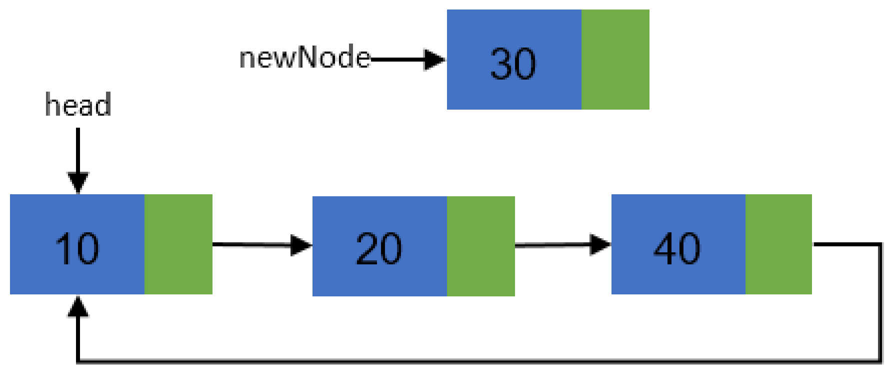
```
%%Input : head {Pointer to first node of the linked list}
Begin
    If (head == NULL) then
        write ('List is empty')
    End if
    Else then
        alloc (newNode)
        read (data)
        newNode.data ← data;
        newNode.next ← head;

        current ← head;
        While (current.next != head) do
            current ← current.next;
        End while
        current.next ← newNode;
    End if
End
```

Algorithm to insert a new node at any position of a circular linked list:

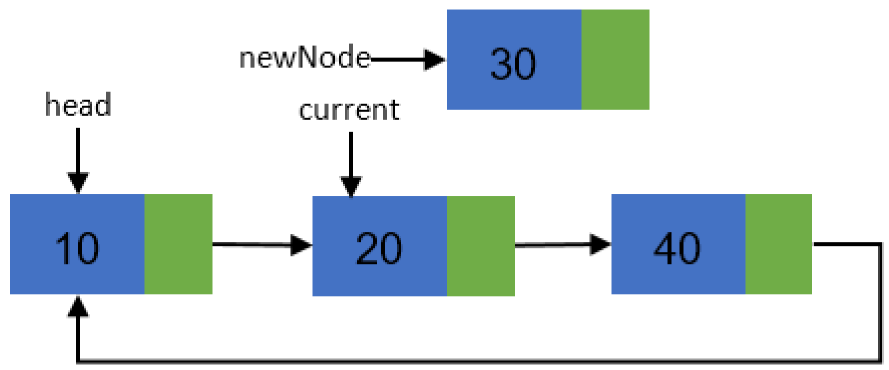
```
%%Input : head {Pointer to first node of the list}
        N {Position where to insert}
Begin
    If (head == NULL) then
        write ('List is empty')
    End if
    Else if (N == 1) then
        insertAtBeginning()
    End if
    Else then
        alloc (newNode)
        read (data)
        newNode.data ← data;

        current ← head;
        For count ← 2 to N-1 do
            current ← current.next;
        End for
        newNode.next ← current.next;
        current.next ← newNode;
    End if

End
```

Steps to insert new node in a Circular linked list

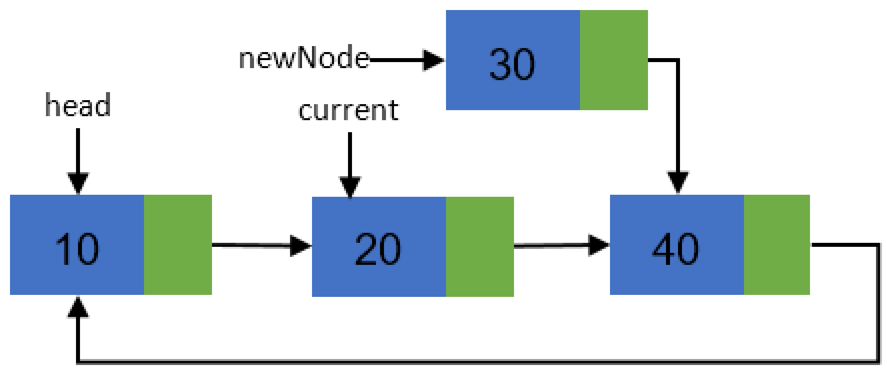
1. Create a newNode and assign some data to its data field.



2. Traverse to N-1 position in the list, in our case since we want to insert a node at 3rd position therefore we would traverse to 3-1 = 2nd position in the list. Say the current pointer points to N-1th node.

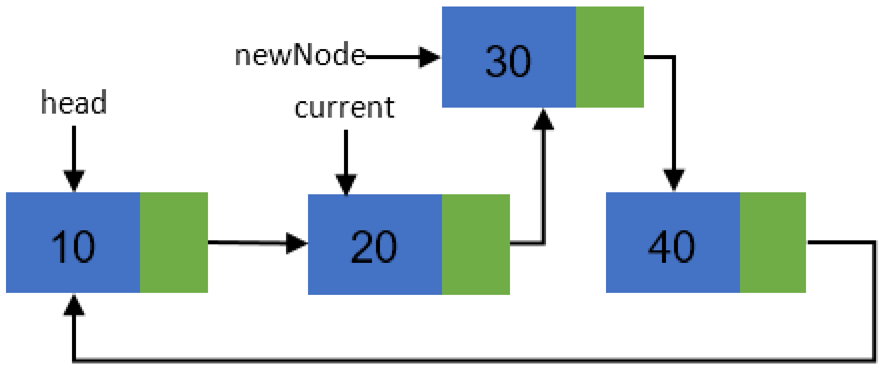


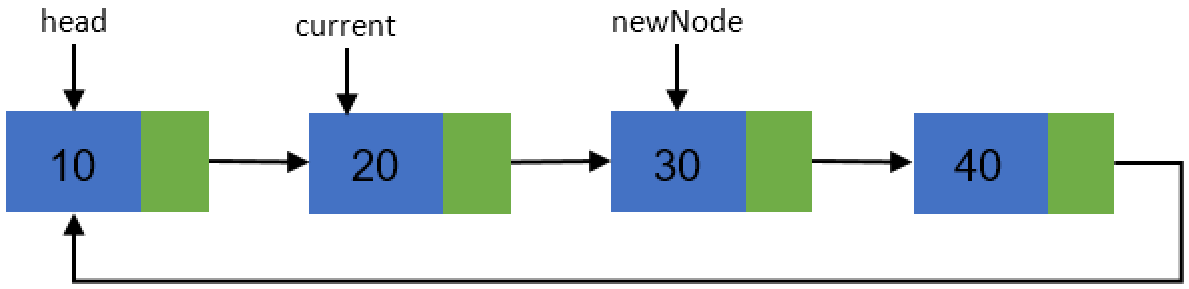
3. Link the next pointer field of newNode with the node pointed by the next pointer field of the current(N-1) node. Which means newNode.next = current. next.



4. Connect the next pointer field of the current node with the newly created node which means now the next pointer field of the current node will point to newNode and you are

done.


newNode
30
head
current
10
20
40


head
current
newNode
10
20
30
40