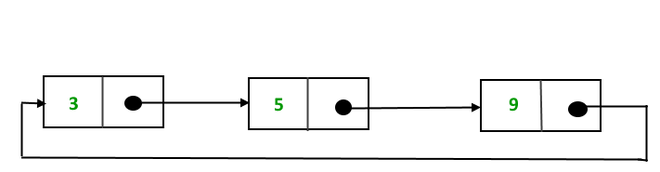
**Circular Linked List**

*The****circular linked list****is a linked list where all nodes are connected to form a circle. In a circular linked list, the first node and the last node are connected to each other which forms a circle. There is no NULL at the end.*

Circular Linked List

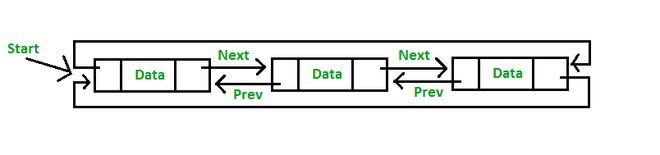
**There are generally two types of circular linked lists:**

* **Circular singly linked list:**In a circular Singly linked list, the last node of the list contains a pointer to the first node of the list. We traverse the circular singly linked list until we reach the same node where we started. The circular singly linked list has no beginning or end. No null value is present in the next part of any of the nodes.



*Representation of Circular singly linked list*

* **Circular Doubly linked list:**Circular Doubly Linked List has properties of both doubly linked list and circular linked list in which two consecutive elements are linked or connected by the previous and next pointer and the last node points to the first node by the next pointer and also the first node points to the last node by the previous pointer.



*Representation of circular doubly linked list*

**Note:**We will be using the singly circular linked list to represent the working of the circular linked list.

**Representation of circular linked list:**

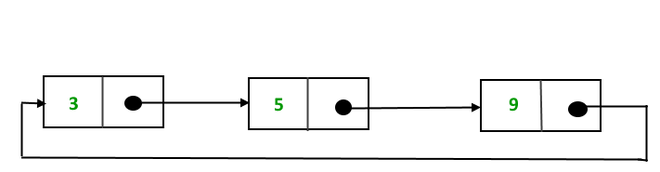
Circular linked lists are similar to single Linked Lists with the exception of connecting the last node to the first node.

Node representation of a Circular Linked List:

* C++
* Python3

|  |
| --- |
| // Class Node, similar to the linked list  **class** Node{  **int** value;      // Points to the next node.      Node next;  } |

Example of Circular singly linked list:



*Example of  circular linked list*

The above  Circular singly linked list can be represented as:

* C++
* Python3

|  |
| --- |
| // Initialize the Nodes.  Node one = **new** Node(3);  Node two = **new** Node(5);  Node three = **new** Node(9);    // Connect nodes  one.next = two;  two.next = three;  three.next = one; |

**Explanation:** In the above program one, two, and three are the node with values 3, 5, and 9 respectively which are connected in a circular manner as:

* **For Node One:**The Next pointer stores the address of Node two.
* **For Node Two:**The Next stores the address of Node three
* **For Node Three:** TheNext points to node one.

**Operations on the circular linked list:**

We can do some operations on the circular linked list similar to the singly linked list which are:

1. Insertion
2. Deletion

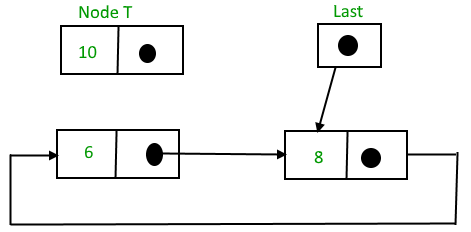
**1.**[Insertion in the circular linked list:](https://www.geeksforgeeks.org/circular-singly-linked-list-insertion/)

A node can be added in three ways:

1. Insertion at the beginning of the list
2. Insertion at the end of the list
3. Insertion in between the nodes

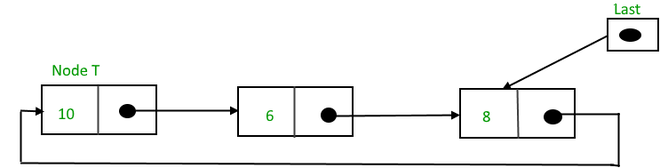
1) **Insertion at the beginning of the list:**To insert a node at the beginning of the list, follow these steps:

* Create a node, say T.
* Make T -> next = last -> next.
* last -> next = T.



*Circular linked list before insertion*

And then,



*Circular linked list after insertion*

Below is the code implementation to insert a node at the beginning of the list:

* C++
* Python3

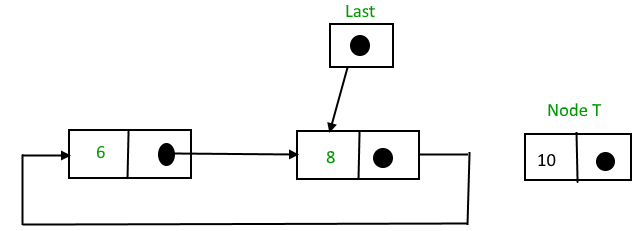
|  |
| --- |
| **struct** Node \*addBegin(**struct** Node \*last, **int** data)  {  **if** (last == NULL)  **return** addToEmpty(last, data);    // Creating a node dynamically.  **struct** Node \*temp          = (**struct** Node \*)**malloc**(**sizeof**(**struct** Node));    // Assigning the data.  temp -> data = data;    // Adjusting the links.  temp -> next = last -> next;  last -> next = temp;    **return** last;  } |

**Time complexity:**O(1) to insert a node at the beginning no need to traverse list it takes constant time   
**Auxiliary Space:** O(1)

2) **Insertion at the end of the list:**To insert a node at the end of the list, follow these steps:

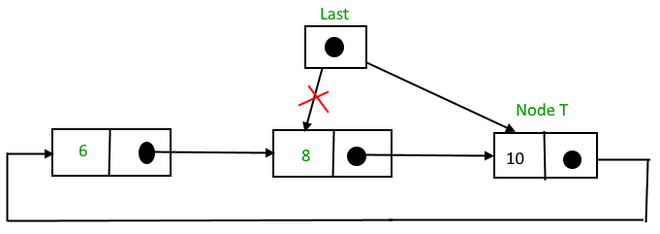
* Create a node, say T.
* Make T -> next = last -> next;
* last -> next = T.
* last = T.

Before insertion,



*Circular linked list before insertion of node at the end*

After insertion,



*Circular linked list after insertion of node at the end*

Below is the code implementation to insert a node at the beginning of the list:

* C++
* Python3

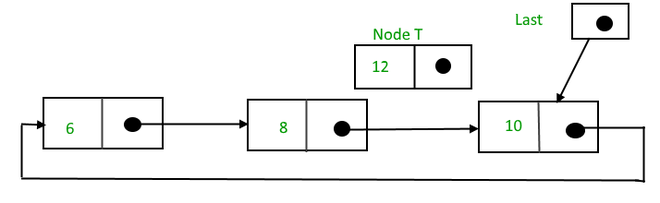
|  |
| --- |
| **struct** Node \*addEnd(**struct** Node \*last, **int** data)  {  **if** (last == NULL)  **return** addToEmpty(last, data);    // Creating a node dynamically.  **struct** Node \*temp =          (**struct** Node \*)**malloc**(**sizeof**(**struct** Node));    // Assigning the data.  temp -> data = data;    // Adjusting the links.  temp -> next = last -> next;  last -> next = temp;  last = temp;    **return** last;  } |

**Time Complexity:** O(1) to insert a node at the end of the list. No need to traverse the list as we are utilizing the last pointer, hence it takes constant time.  
**Auxiliary Space:** O(1)

3) **Insertion in between the nodes:**To insert a node in between the two nodes, follow these steps:

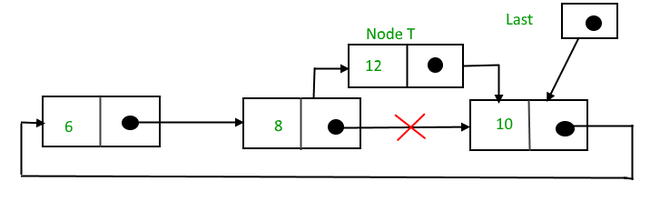
* Create a node, say T.
* Search for the node after which T needs to be inserted, say that node is P.
* Make T -> next = P -> next;
* P -> next = T.

Suppose 12 needs to be inserted after the node has the value 10,



*Circular linked list before insertion*

After searching and insertion,



*Circular linked list after  insertion*

Below is the code to insert a node at the specified position of the List:

* C++
* Python3

|  |
| --- |
| **struct** Node \*addAfter(**struct** Node \*last, **int** data, **int** item)  {  **if** (last == NULL)  **return** NULL;    **struct** Node \*temp, \*p;      p = last -> next;        // Searching the item.  **do**      {  **if** (p ->data == item)          {              // Creating a node dynamically.              temp = (**struct** Node \*)**malloc**(**sizeof**(**struct** Node));                // Assigning the data.              temp -> data = data;                // Adjusting the links.              temp -> next = p -> next;                // Adding newly allocated node after p.              p -> next = temp;                // Checking for the last node.  **if** (p == last)                  last = temp;    **return** last;          }          p = p -> next;      } **while** (p != last -> next);        cout << item << " not present in the list." << endl;  **return** last;  } |

**Time Complexity:** O(N)  
**Auxiliary Space:**O(1)

**2. Deletion in a circular linked list:**

**1) Delete the node only if it is the only node in the circular linked list:**

* Free the node’s memory
* The last value should be NULL A node always points to another node, so NULL assignment is not necessary.  
  Any node can be set as the starting point.  
  Nodes are traversed quickly from the first to the last.

**2) Deletion of the last node:**

* Locate the node before the last node (let it be temp)
* Keep the address of the node next to the last node in temp
* Delete the last memory
* Put temp at the end

**3) Delete any node from the circular linked list:**We will be given a node and our task is to delete that node from the circular linked list.

**Algorithm:**  
**Case 1**: List is empty.

* If the list is empty we will simply return.

**Case 2**:List is not empty

* If the list is not empty then we define two pointers **curr**and **prev** and initialize the pointer **curr** with the **head**node.
* Traverse the list using **curr** to find the node to be deleted and before moving to curr to the next node, every time set prev = curr.
* If the node is found, check if it is the only node in the list. If yes, set head = NULL and free(curr).
* If the list has more than one node, check if it is the first node of the list. Condition to check this( curr == head). If yes, then move prev until it reaches the last node. After prev reaches the last node, set head = head -> next and prev -> next = head. Delete curr.
* If curr is not the first node, we check if it is the last node in the list. Condition to check this is (curr -> next == head).
* If curr is the last node. Set prev -> next = head and delete the node curr by free(curr).
* If the node to be deleted is neither the first node nor the last node, then set prev -> next = curr -> next and delete curr.
* If the node is not present in the list return head and don’t do anything.

Below is the implementation for the above approach:

* C++
* Java
* Python3

|  |
| --- |
| // Java program to delete a given key from  // linked list.  **import** java.util.\*;  **import** java.io.\*;    **public** **class** GFG {      /\* structure for a node \*/  **static** **class** Node {  **int** data;          Node next;      };        /\* Function to insert a node at the beginning of  a Circular linked list \*/  **static** Node push(Node head\_ref, **int** data)      {          // Create a new node and make head as next          // of it.          Node ptr1 = **new** Node();          ptr1.data = data;          ptr1.next = head\_ref;            /\* If linked list is not null then set the  next of last node \*/  **if** (head\_ref != **null**) {              // Find the node before head and update              // next of it.              Node temp = head\_ref;  **while** (temp.next != head\_ref)                  temp = temp.next;              temp.next = ptr1;          }  **else**              ptr1.next = ptr1; /\*For the first node \*/            head\_ref = ptr1;  **return** head\_ref;      }        /\* Function to print nodes in a given  circular linked list \*/  **static** **void** printList(Node head)      {          Node temp = head;  **if** (head != **null**) {  **do** {                  System.out.printf("%d ", temp.data);                  temp = temp.next;              } **while** (temp != head);          }            System.out.printf("\n");      }        /\* Function to delete a given node from the list \*/  **static** Node deleteNode(Node head, **int** key)      {  **if** (head == **null**)  **return** **null**;  **int** flag = 0;          // Find the required node          Node curr = head, prev = **new** Node();  **while** (curr.data != key) {  **if** (curr.next == head) {                  System.out.printf(                      "Given node is not found in the list!!!\n");                  flag = 1;  **break**;              }                prev = curr;              curr = curr.next;          }            // Check if the element is not present in the list  **if**(flag == 1)  **return** head;            // Check if node is only node  **if** (curr == head && curr.next == head) {              head = **null**;  **return** head;          }            // If more than one node, check if          // it is first node  **if** (curr == head) {              prev = head;  **while** (prev.next != head)                  prev = prev.next;              head = curr.next;              prev.next = head;          }            // check if node is last node  **else** **if** (curr.next == head) {              prev.next = head;          }  **else** {              prev.next = curr.next;          }  **return** head;      }        /\* Driver code \*/  **public** **static** **void** main(String args[])      {          /\* Initialize lists as empty \*/          Node head = **null**;            /\* Created linked list will be 2.5.7.8.10 \*/          head = push(head, 2);          head = push(head, 5);          head = push(head, 7);          head = push(head, 8);          head = push(head, 10);            System.out.printf("List Before Deletion: ");          printList(head);            head = deleteNode(head, 7);            System.out.printf("List After Deletion: ");          printList(head);      }  }    // This code is contributed by Susobhan Akhuli |

**Output**

List Before Deletion: 10 8 7 5 2

List After Deletion: 10 8 5 2

**Time Complexity:** O(N), Worst case occurs when the element to be deleted is the last element and we need to move through the whole list.  
**Auxiliary Space:** O(1), As constant extra space is used.

**Advantages of Circular Linked Lists:**

* Any node can be a starting point. We can traverse the whole list by starting from any point. We just need to stop when the first visited node is visited again.
* Useful for implementation of a queue. Unlike [this](https://www.geeksforgeeks.org/queue-linked-list-implementation/)implementation, we don’t need to maintain two pointers for front and rear if we use a circular linked list. We can maintain a pointer to the last inserted node and the front can always be obtained as next of last.

* Circular lists are useful in applications to repeatedly go around the list. For example, when multiple applications are running on a PC, it is common for the operating system to put the running applications on a list and then cycle through them, giving each of them a slice of time to execute, and then making them wait while the CPU is given to another application. It is convenient for the operating system to use a circular list so that when it reaches the end of the list it can cycle around to the front of the list.
* Circular Doubly Linked Lists are used for the implementation of advanced data structures like the [Fibonacci Heap](http://en.wikipedia.org/wiki/Fibonacci_heap).

**Disadvantages of circular linked list:**

* Compared to singly linked lists, circular lists are more complex.
* Reversing a circular list is more complicated than singly or doubly reversing a circular list.
* It is possible for the code to go into an infinite loop if it is not handled carefully.
* It is harder to find the end of the list and control the loop.

**Applications of circular linked lists:**

* Multiplayer games use this to give each player a chance to play.
* A circular linked list can be used to organize multiple running applications on an operating system. These applications are iterated over by the OS.