**Circular Linked List**

A circular linked list is a type of [linked list](https://www.programiz.com/dsa/linked-list) in which the first and the last nodes are also connected to each other to form a circle.

There are basically two types of circular linked list:

**1. Circular Singly Linked List**

Here, the address of the last node consists of the address of the first node.

|  |
| --- |
| Circular Linked List Representation |
| Circular Linked List Representation |

**2. Circular Doubly Linked List**

Here, in addition to the last node storing the address of the first node, the first node will also store the address of the last node.

|  |
| --- |
| Circular Doubly Linked List Representation |
| Circular Doubly Linked List Representation |

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

## Representation of Circular Linked List

Let's see how we can represent a circular linked list on an algorithm/code. Suppose we have a linked list:

|  |
| --- |
| Initial circular linked list |
| Initial circular linked list |

Ad

**For node one**

* next stores the address of two (there is no node before it)

**For node two**

* next stores the address of three

**For node three**

* next stores NULL (there is no node after it)
* next points to node one

## Insertion on a Circular Linked List

We can insert elements at 3 different positions of a circular linked list:

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

Suppose we have a circular linked list with elements 1, 2, and 3.

|  |
| --- |
| Initial circular linked list |
| Initial circular linked list |

Let's add a node with value 6 at different positions of the circular linked list we made above. The first step is to create a new node.

* allocate memory for newNode
* assign the data to newNode

|  |
| --- |
| New node |
| New node |

### 1. Insertion at the Beginning

* store the address of the current first node in the newNode (i.e. pointing the newNode to the current first node)
* point the last node to newNode (i.e making newNode as head)

|  |
| --- |
| Insert at the beginning |
| Insert at the beginning |

### 2. Insertion in between two nodes

Let's insert newNode after the first node.

* travel to the node given (let this node be p)
* point the next of newNode to the node next to p
* store the address of newNode at next of p

|  |
| --- |
| Insertion at a node |
| Insertion at a node |

### 3. Insertion at the end

* store the address of the head node to next of newNode (making newNode the last node)
* point the current last node to newNode
* make newNode as the last node

|  |
| --- |
| Insert at the end |
| Insert at the end |

## Deletion on a Circular Linked List

Suppose we have a double-linked list with elements 1, 2, and 3.

|  |
| --- |
| Initial circular linked list |
| Initial circular linked list |

### 1. If the node to be deleted is the only node

* free the memory occupied by the node
* store NULL in last

### 2. If last node is to be deleted

* find the node before the last node (let it be temp)
* store the address of the node next to the last node in temp
* free the memory of last
* make temp as the last node

|  |
| --- |
| Delete the last node |
| Delete the last node |

### 3. If any other nodes are to be deleted

* travel to the node to be deleted (here we are deleting node 2)
* let the node before node 2 be temp
* store the address of the node next to 2 in temp
* free the memory of 2

|  |
| --- |
| Delete a specific node |
| Delete a specific node |

## Circular Linked List Complexity

|  |  |  |
| --- | --- | --- |
| Circular Linked List Complexity | **Time Complexity** | **Space Complexity** |
| **Insertion Operation** | O(1) or O(n) | O(1) |
| **Deletion Operation** | O(1) | O(1) |

**1. Complexity of Insertion Operation**

* The insertion operations that do not require traversal have the time complexity of O(1).
* And, an insertion that requires traversal has a time complexity of O(n).
* The space complexity is O(1).

**2. Complexity of Deletion Operation**

* All deletion operations run with a time complexity of O(1).
* And, the space complexity is O(1).

## Why Circular Linked List?

1. The NULL assignment is not required because a node always points to another node.
2. The starting point can be set to any node.
3. Traversal from the first node to the last node is quick.

## Circular Linked List Applications

* It is used in multiplayer games to give a chance to each player to play the game.
* Multiple running applications can be placed in a circular linked list on an operating system. The os keeps on iterating over these applications