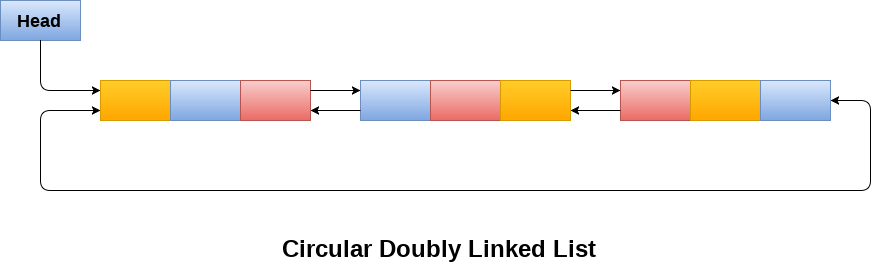
Circular Doubly Linked List

Circular doubly linked list is a more complexed type of data structure in which a node contain pointers to its previous node as well as the next node. Circular doubly linked list doesn't contain NULL in any of the node. The last node of the list contains the address of the first node of the list. The first node of the list also contain address of the last node in its previous pointer.

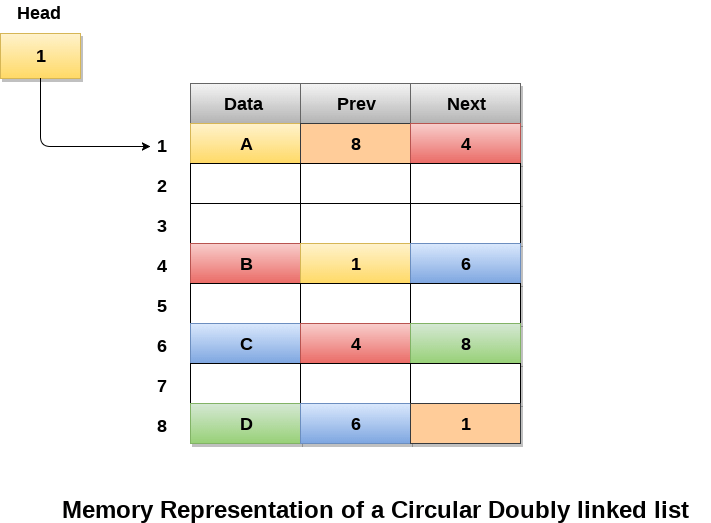
A circular doubly linked list is shown in the following figure.



Due to the fact that a circular doubly linked list contains three parts in its structure therefore, it demands more space per node and more expensive basic operations. However, a circular doubly linked list provides easy manipulation of the pointers and the searching becomes twice as efficient.

## **Memory Management of Circular Doubly linked list**

The following figure shows the way in which the memory is allocated for a circular doubly linked list. The variable head contains the address of the first element of the list i.e. 1 hence the starting node of the list contains data A is stored at address 1. Since, each node of the list is supposed to have three parts therefore, the starting node of the list contains address of the last node i.e. 8 and the next node i.e. 4. The last node of the list that is stored at address 8 and containing data as 6, contains address of the first node of the list as shown in the image i.e. 1. In circular doubly linked list, the last node is identified by the address of the first node which is stored in the next part of the last node therefore the node which contains the address of the first node, is actually the last node of the list.



## **Operations on circular doubly linked list :**

There are various operations which can be performed on circular doubly linked list. The node structure of a circular doubly linked list is similar to doubly linked list. However, the operations on circular doubly linked list is described in the following table.

|  |  |  |
| --- | --- | --- |
| **SN** | **Operation** | **Description** |
| 1 | [Insertion at beginning](https://www.javatpoint.com/insertion-in-circular-doubly-linked-list-at-beginning) | Adding a node in circular doubly linked list at the beginning. |
| 2 | [Insertion at end](https://www.javatpoint.com/insertion-in-circular-doubly-linked-list-at-end) | Adding a node in circular doubly linked list at the end. |
| 3 | [Deletion at beginning](https://www.javatpoint.com/deletion-in-circular-doubly-linked-list-at-beginning) | Removing a node in circular doubly linked list from beginning. |
| 4 | [Deletion at end](https://www.javatpoint.com/deletion-in-circular-doubly-linked-list-at-end) | Removing a node in circular doubly linked list at the end. |

Traversing and searching in circular doubly linked list is similar to that in the circular singly linked list.

|  |
| --- |
| # Python3 program to illustrate inserting  # a Node in a Cicular Doubly Linked list  # in begging, end and middle  start = None    # Structure of a Node  class Node:      def \_\_init\_\_(self, data):          self.data = data          self.next = None          self.prev = None    # Function to insert at the end  def insertEnd(value) :      global start        # If the list is empty, create a      # single node circular and doubly list      if (start == None) :            new\_node = Node(0)          new\_node.data = value          new\_node.next = new\_node.prev = new\_node          start = new\_node          return        # If list is not empty        # Find last node \*/      last = (start).prev        # Create Node dynamically      new\_node = Node(0)      new\_node.data = value        # Start is going to be next of new\_node      new\_node.next = start        # Make new node previous of start      (start).prev = new\_node        # Make last preivous of new node      new\_node.prev = last        # Make new node next of old last      last.next = new\_node    # Function to insert Node at the beginning  # of the List,  def insertBegin( value) :      global start        # Pointer points to last Node      last = (start).prev        new\_node = Node(0)      new\_node.data = value # Inserting the data        # setting up previous and      # next of new node      new\_node.next = start      new\_node.prev = last        # Update next and previous pointers      # of start and last.      last.next = (start).prev = new\_node        # Update start pointer      start = new\_node    # Function to insert node with value as value1.  # The new node is inserted after the node with  # with value2  def insertAfter(value1, value2) :      global start      new\_node = Node(0)      new\_node.data = value1 # Inserting the data        # Find node having value2 and      # next node of it      temp = start      while (temp.data != value2) :          temp = temp.next      next = temp.next        # insert new\_node between temp and next.      temp.next = new\_node      new\_node.prev = temp      new\_node.next = next      next.prev = new\_node    def display() :      global start      temp = start        print ("Traversal in forward direction:")      while (temp.next != start) :            print (temp.data, end = " ")          temp = temp.next        print (temp.data)        print ("Traversal in reverse direction:")      last = start.prev      temp = last      while (temp.prev != last) :            print (temp.data, end = " ")          temp = temp.prev        print (temp.data)    # Driver Code  if \_\_name\_\_ == '\_\_main\_\_':      global start        # Start with the empty list      start = None        # Insert 5. So linked list becomes 5.None      insertEnd(5)        # Insert 4 at the beginning. So linked      # list becomes 4.5      insertBegin(4)        # Insert 7 at the end. So linked list      # becomes 4.5.7      insertEnd(7)        # Insert 8 at the end. So linked list      # becomes 4.5.7.8      insertEnd(8)        # Insert 6, after 5. So linked list      # becomes 4.5.6.7.8      insertAfter(6, 5)        print ("Created circular doubly linked list is: ")      display() |

**Output:**

Created circular doubly linked list is:

Traversal in forward direction

4 5 6 7 8

Traversal in reverse direction

8 7 6 5 4