2.8 CIRCULAR LINKED LIST

Circular linked list is similar to normal linked list except that the last node contains a pointer to the first node of the list.

NODE DECLARATION

class Node:

```
class Node
{
public:
   int data;
   Node *next;
};
```

INSERTION

To insert a node at the beginning of a circular linked list, you need to follow these steps:

- Create a new node with the given data.
- If the list is empty, make the new node the head and point it to itself.
- Otherwise, set the next pointer of the new node to point to the current head.
- Update the head to point to the new node.
- Update the next pointer of the last node to point to the new head (to maintain the circular structure).

```
void insert(int item)
    {
        Node *newNode = new Node;
        newNode->data = item;
```

```
        if (last == NULL)
        {
            last = newNode;
            newNode->next = newNode;
        }
        else
        {
            newNode->next = last->next;
            last->next = newNode;
            last = newNode;
        }
    }
```

DELETION

To delete the node at the beginning of a circular linked list in Python, you need to follow these steps:

- Check if the list is empty. If it is empty, there is nothing to delete.
- If the list has only one node, set the head to None to delete the node.
- Otherwise, find the last node of the list (the node whose next pointer points to the head).
- Update the next pointer of the last node to point to the second node (head's next).
- Update the head to point to the second node.

```
    void del(int item)
        {
            if (last == NULL)
```

```
{
   cout << "\nList is empty";
   return;
}

Node *curr = last->next;
Node *prev = last;

do
{
   if (curr->data == item)
   {
      if (curr == last && curr->next == last)
      {
         last = NULL;
      }
      else
      {
         prev->next = curr->next;
         if (curr == last)
            last = prev;
      }
      delete curr;
```

```
            cout << "\nElement deleted";
            return;
        }
        prev = curr;
        curr = curr->next;
    } while (curr != last->next);

    cout << "\nElement not found";
}
```

Traversal of Circular Linked List:

- Traversing a circular linked list involves visiting each node of the list starting from the head node and continuing until the head node is encountered again.

```
void displaySearch(int item)
{
    if (last == NULL)
    {
        cout << "\nList is empty";
        return;
    }

    Node *temp = last->next;
    int pos = 1;
    int found = 0;

    cout << "\nCircular List: ";
```

```
        do
        {
            cout << temp->data << " ";

            if (temp->data == item)
            {
                found = 1;
                cout << "(Found at position " << pos << ") ";
            }

            temp = temp->next;
            pos++;
        } while (temp != last->next);

        if (!found)
            cout << "\nElement not found";
    }
```

Advantages:

- No NULL pointer. Supports continuous traversal without NULL.
- Continuous traversal possible
- Traversal can start from any node
- Efficient for cyclic operations
- Fast insertion with tail pointer

Disadvantages:

- Careless traversal may lead to infinite loops.
- More complex implementation than singly list
- Debugging is difficult
- Deletion logic is tricky
- Searching is slow (O(n))