CIRCULAR 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 can have circular singly linked list as well as circular doubly linked list.
- We traverse a circular singly linked list until we reach the same node where we started.
- The circular singly liked list has no beginning and no ending.
- There is no null value present in the next part of any of the nodes.
- Circular linked list are mostly used in task maintenance in operating systems. Fig. 3.14 shows a circular singly linked list.

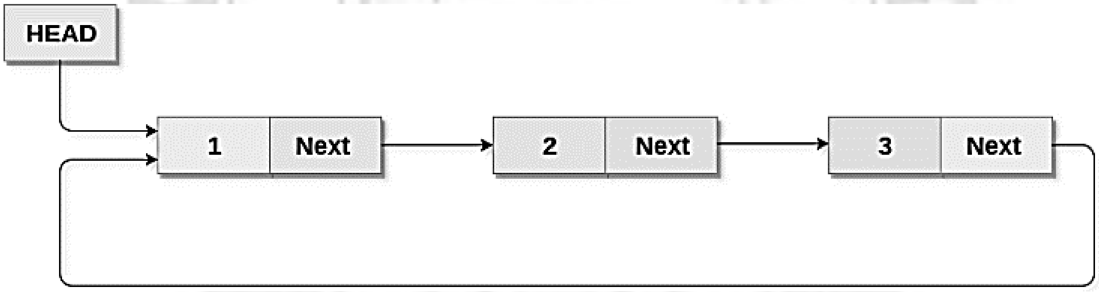

Fig. 3.14: Circular Singly Linked List.

Memory Representation of circular linked list

- Fig 3.15 shows the memory representation of a circular linked list containing marks of a student in 4 subjects. However, the image shows a glimpse of how the circular list is being stored in the memory. The start or head of the list is pointing to the element with the index 1 and containing 13 marks in the data part and 4 in the next part. Which means that it is linked with the node that is being stored at 4th index of the list.

- However, due to the fact that we are considering circular linked list in the memory therefore the last node of the list contains the address of the first node of the list.

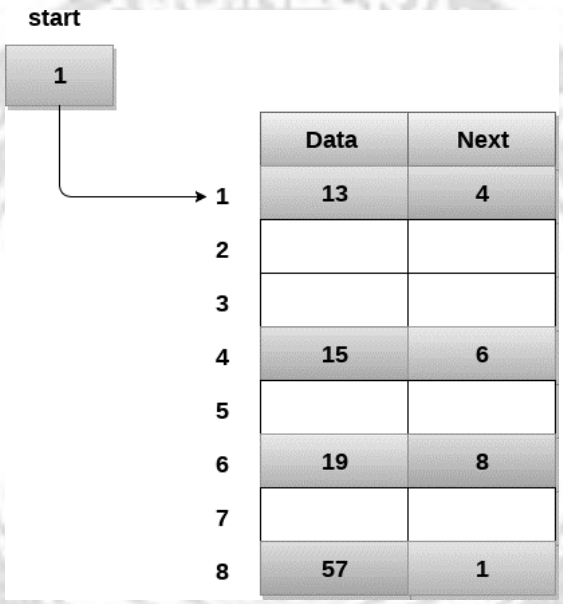

Fig. 3.15: Memory Representation of Circular Linked List

Operations on Circular Singly linked list

Table 3.2 describes all the operations performed on a Doubly Linked List

Table 3.2: Operations on Doubly Linked List

Sl. No.	Operation	Description
1.	Insertion at beginning	Adding a node into circular singly linked list at the beginning.
2.	Insertion at end	Adding a node into circular singly linked list at the end.
3.	Deletion at beginning	Removing the node from circular singly linked list at the beginning.
4.	Searching	Compare each element of the node with the given item and return the location at which the item is present in the list otherwise return null.

5.	Traversing	Visiting each element of the list at least once in order to perform some specific operation.

Insertion at the beginning

- There are two scenario in which a node can be inserted in circular singly linked list at beginning. Either the node will be inserted in an empty list or the node is to be inserted in an already filled list.
- Firstly, allocate the memory space for the new node by using the malloc method of C language.
- In the first scenario, the condition head == NULL will be true. Since, the list in which, we are inserting the node is a circular singly linked list, therefore the onlynode of the list (which is just inserted into the list) will point to itself only.
- Also need to make the head pointer point to this node.
- In the second scenario, the condition head == NULL will become false which means that the list contains at least one node.
- In this case, traverse the list in order to reach the last node of the list.
- At the end of the loop, the pointer temp would point to the last node of the list.
- Make the next pointer of the last node point to the head node of the list and the new node which is being inserted into the list will be the new head node of the list
- Therefore the next pointer of temp will point to the new node ptr.

Algorithm 3.9

```
Step 1: IF PTR = NULL
            Write OVERFLOW
            Go to Step 11
            [END OF IF]
Step 2: SET NEW_NODE = PTR
Step 3: SET PTR = PTR -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET TEMP = HEAD
Step 6: Repeat Step 8 while TEMP -> NEXT != HEAD
Step 7: SET TEMP = TEMP -> NEXT
```

```
        [END OF LOOP]
Step 8: SET NEW_NODE -> NEXT =
HEAD Step 9: SET TEMP → NEXT =
NEW_NODE Step 10: SET HEAD =
NEW_NODE
Step 11: EXIT
```

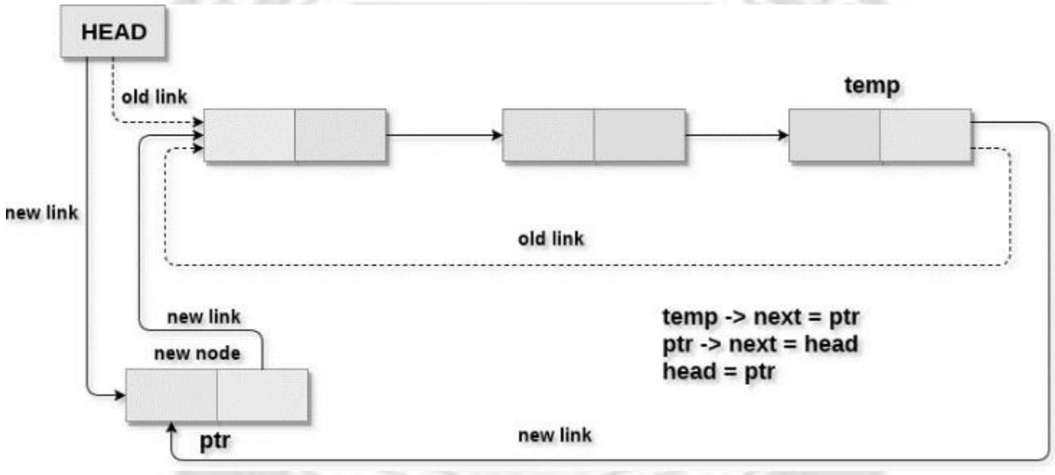

Fig. 3.16: Insertion into a Circular Linked List at the beginning

Insertion at the end

- Allocate the memory space for the new node by using the malloc method of C language.
- In the first scenario, the condition head == NULL will be true.
- Also make the head pointer point to this node.
- In the second scenario, the condition head == NULL will become false which means that the list contains at least one node.
- In this case, traverse the list in order to reach the last node of the list.
- At the end of the loop, the pointer temp would point to the last node of the list.
- The existing last node i.e. temp must point to the new node ptr

Algorithm 3.10

```
Step 1: IF PTR = NULL
            Write OVERFLOW
```

```
        Go to Step 1
        [END OF IF]
Step 2: SET NEW_NODE = PTR
Step 3: SET PTR = PTR -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = HEAD
Step 6: SET TEMP = HEAD
Step 7: Repeat Step 8 while TEMP -> NEXT != HEAD
Step 8: SET TEMP = TEMP -> NEXT
        [END OF LOOP]
Step 9: SET TEMP -> NEXT = NEW_NODE
Step 10: EXIT
```

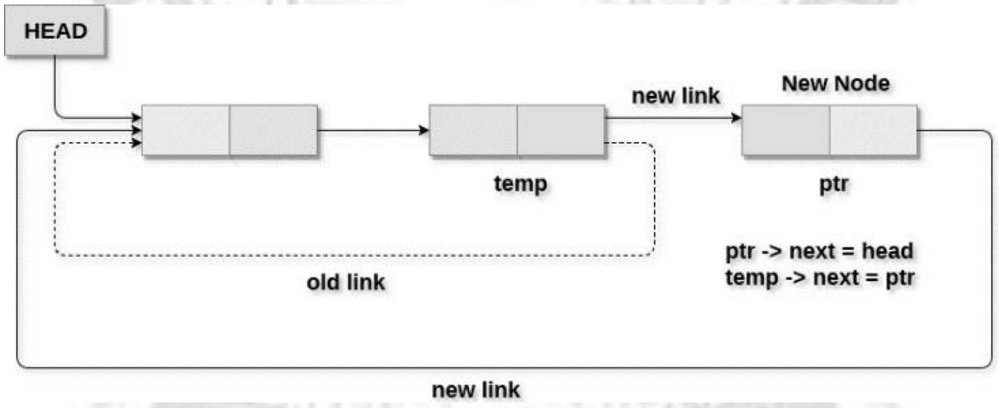

Fig. 3.17: Insertion into a Circular Linked List at the end

Deletion at the beginning

In order to delete a node in circular singly linked list, we need to make a few pointer adjustments. There are three scenarios of deleting a node from circular singly linked list at beginning.

- Scenario 1: (The list is Empty) - If the list is empty then the condition head == NULL will become true, in this case, just to print underflow on the screen and make exit.

- Scenario 2: (The list contains single node) - If the list contains single node then, the condition head → next == head will become true. In this case, delete the entire list and make the head pointer free.
- Scenario 3: (The list contains more than one node) - If the list contains more than one node then, in that case, traverse the list by using the pointer ptr to reach the last node of the list.
- At the end of the loop, the pointer ptr point to the last node of the list.
- The last node of the list will point to the next of the head node.
- Now, free the head pointer by using the free() method.
- Make the node pointed by the next of the last node, the new head of the list.

Algorithm 3.11

```
Step 1: IF HEAD = NULL
            Write UNDERFLOW
             Go to Step 8
            [END OF IF]
Step 2: SET PTR = HEAD
Step 3: Repeat Step 4 while PTR → NEXT != HEAD
Step 4: SET PTR = PTR → next
            [END OF LOOP]
Step 5: SET PTR → NEXT = HEAD → NEXT
Step 6: FREE HEAD
Step 7: SET HEAD = PTR → NEXT
Step 8: EXIT
```

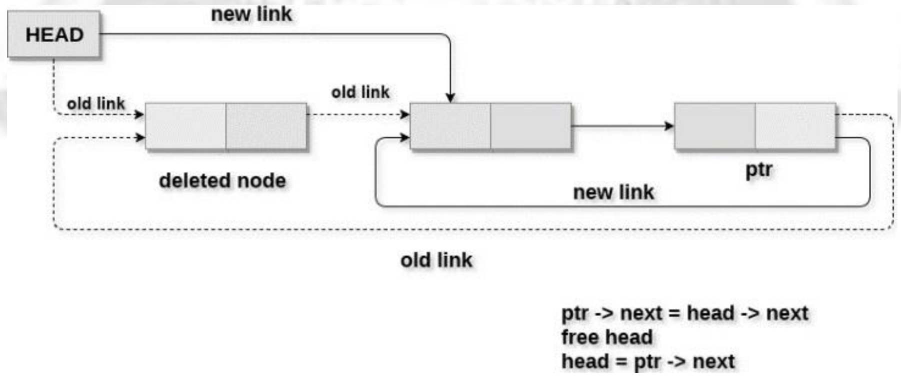

Fig. 3.18: Deletion in a Circular Linked List at beginning

Deletion at the end

- Scenario 1 (the list is empty) - If the list is empty then the condition head == NULL will become true, in this case, just to print underflow on the screen and make exit.
- Scenario 2(the list contains single element) - If the list contains single node then, the condition head → next == head will become true. In this case, delete the entire list and make the head pointer free.
- Scenario 3(the list contains more than one element) - If the list contains more than one element, then in order to delete the last element, reach the last node. Also keep track of the second last node of the list. For this purpose, the two pointers ptr andpreptr are defined.
- Make just one more pointer adjustment. We need to make the next pointer of preptr point to the next of ptr (i.e. head) and then make pointer ptr free.

Algorithm 3.12

```
Step 1: IF HEAD = NULL
            Write UNDERFLOW
             Go to Step 8
            [END OF IF]
Step 2: SET PTR = HEAD
Step 3: Repeat Steps 4 and 5 while PTR -> NEXT != HEAD
Step 4: SET PREPTR = PTR
Step 5: SET PTR = PTR -> NEXT
          [END OF LOOP]
Step 6: SET PREPTR -> NEXT = HEAD
Step 7: FREE PTR
Step 8: EXIT
```

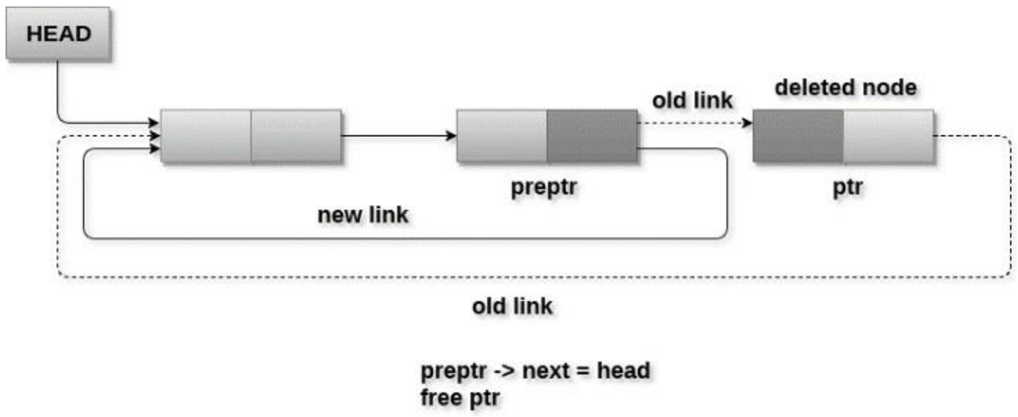

Fig. 3.19: Deletion in a Circular Linked List at the end

Searching

- Searching in circular singly linked list needs traversing across the list.
- The item which is to be searched in the list is matched with each node data of the list once.
- If the match found then the location of that item is returned otherwise -1 is returned.

Algorithm 3.13

```
STEP 1: SET PTR = HEAD
STEP 2: Set I = 0
STEP 3: IF PTR = NULL
        WRITE "EMPTY LIST"
        GOTO STEP 8
        END OF IF
STEP 4: IF HEAD → DATA = ITEM
        WRITE i+1 RETURN [END OF IF]
STEP 5: REPEAT STEP 5 TO 7 UNTIL PTR->next != head
STEP 6: if ptr → data = item
        write i+1
        RETURN
        End of IF
STEP 7: I = I + 1
STEP 8: PTR = PTR → NEXT
        [END OF LOOP]
STEP 9: EXIT
```

Searching

- Traversing in circular singly linked list can be done through a loop.
- Initialize the temporary pointer variable temp to head pointer and run the whileloop until the next pointer of temp becomes head.

Algorithm 3.14

```
STEP 1: SET PTR = HEAD
STEP 2: IF PTR = NULL
        WRITE "EMPTY LIST"
        GOTO STEP 8
        END OF IF
STEP 4: REPEAT STEP 5 AND 6 UNTIL PTR → NEXT != HEAD
STEP 5: PRINT PTR → DATA
STEP 6: PTR = PTR → NEXT
        [END OF LOOP]
STEP 7: PRINT PTR→ DATA
STEP 8: EXIT
```

Advantages of Circular Linked Lists

It is possible to traverse from the last node back to the first i.e. the head node.

- The starting node does not matter as we can traverse each and every node despite whatever node we keep as the starting node.
- The previous node can be easily identified.
- There is no need for a NULL function to code. The circular list never identifies a NULL identifier unless it is fully assigned.
- Circular linked lists are beneficial for end operations as start and finish coincide

Disadvantages of Circular Linked Lists

- If the circular linked list is not handled properly then it can lead to an infinite loop as it is circular in nature.
- In comparison with singly-linked lists, doubly linked lists are more complex in nature
- Direct accessing of elements is not possible.
- It is generally a complex task to reverse a circular linked list