NAME: GAURAV KISHOR PATIL

DIV: 2 ROLL NO:54

BATCH: C

Experiment No.7
Implement Circular Linked List ADT
Date of Performance:
Date of Submission:

Experiment No. 7: Circular Linked List Operations

Aim: Implementation of Circular Linked List ADT

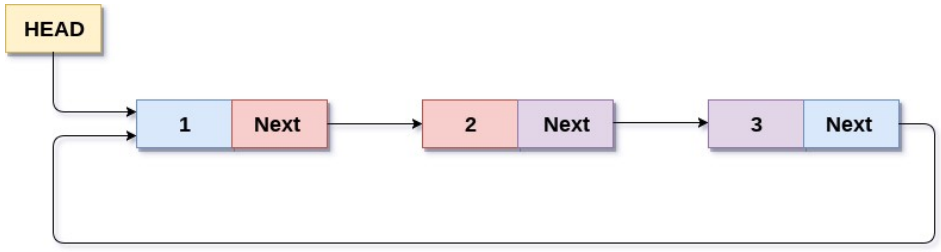
Objective: Circular Linked Lists can be used to manage the computing resources of the computer. Data structures such as stacks and queues are implemented with the help of the circular linked lists

Theory :

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.

The following image shows a circular singly linked list.



Circular Singly Linked List

Algorithm

Algorithm :

- **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

Code:

```
#include<stdio.h>
#include<stdlib.h>

struct node {
   int data;
   struct node *next;
};

struct node *head = NULL;
```

```
void display();
void insert_at_start(int n);
void insert_at_end(int n);
void insert_in_middle(int n);
void delete_at_start();
void delete_at_end();
void deleteFromMid();
void main() {
   int choice, ele, position;

   do {
      printf("Circular Linked List \n");
      printf("1) Insert at Start\n");
      printf("2) Insert at End\n");
      printf("3) Insert in Middle\n");
      printf("4) Delete at Start\n");
      printf("5) Delete at End\n");
          printf("6) Delete at Middle\n");
      printf("7) Display\n");
      printf("8) Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
```

```
    printf("Enter the element you want to insert: ");
    scanf("%d", &ele);
    insert_at_start(ele);
    break;
case 2:
    printf("Enter the element you want to insert: ");
    scanf("%d", &ele);
    insert_at_end(ele);
    break;
case 3:
    printf("Enter the element you want to insert: ");
    scanf("%d", &ele);
   insert_in_middle(ele);
   break;
case 4:
    delete_at_start();
    break;
case 5:
    delete_at_end();
    break;
case 6:
    deleteFromMid();
    break;
case 7:
display();
    break;
```

```
            case 8:printf("Program Closed\n");
            break;

         default:
            printf("Invalid Choice\n");
            break;
      }

   } while (choice != 7);
}

void display() {
   if (head == NULL) {
      printf("List is empty.\n");
      return;
   }
   struct node* temp = head;
   do {
      printf("%d ", temp->data);
      temp = temp->next;
   } while(temp != head);
   printf("\n");
}

void insert_at_start(int n) {
   struct node *newnode = (struct node*)malloc(sizeof(struct node));
```

```
    struct node *temp = head;
    newnode->data = n;
    newnode->next = head;

    if (head != NULL) {
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = newnode;
    } else {
        newnode->next = newnode; // For the first node
    }

    head = newnode;
}

void insert_at_end(int n) {
    struct node *newnode = (struct node*)malloc(sizeof(struct node));
    struct node *temp = head;

    newnode->data = n;
    newnode->next = head;

    if (head != NULL) {
        while (temp->next != head) {
            temp = temp->next;
```

```
        }
        temp->next = newnode;
    } else {
        head = newnode;
        newnode->next = newnode; // For the first node
    }
}

void deleteFromMid() {
    int size;
    struct node *temp, *current,*ct;


    if(head == NULL) {
        printf("List is empty \n");
        return;
    }
    else {
          ct=head;
          while(ct!=NULL){
            ct=ct->next;
            size++;
          }

        int count = (size % 2 == 0) ? (size/2) : ((size+1)/2);
```

```
if( head != NULL ) {

   temp = head;
   current = NULL;



   for(int i = 0; i < count-1; i++){
      current = temp;
      temp = temp->next;
   }
   if(current != NULL) {

      current->next = temp->next;

      temp = NULL;
   }

   else {
      head = temp->next;

      temp = NULL;
   }
}

else {
```

```
            head = NULL;
        }
    }
    size--;
}

void delete_at_start() {
    if (head == NULL) {
        printf("Underflow\n");
    } else {
        struct node *temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        struct node *nextNode = head->next;
        printf("The element deleted: %d\n", head->data);
        free(head);
        if (nextNode == head) {
            head = NULL;
        } else {
            temp->next = nextNode;
            head = nextNode;
        }
    }
}
```

```
void delete_at_end() {
    if (head == NULL) {
        printf("Underflow\n");
    } else if (head->next == head) {
        printf("The element deleted: %d\n", head->data);
        free(head);
        head = NULL;
    } else {
        struct node *temp = head;
        while (temp->next->next != head) {
            temp = temp->next;
        }
        struct node *nextNode = temp->next;
        printf("The element deleted: %d\n", nextNode->data);
        free(nextNode);
        temp->next = head;
    }
}

void insert_in_middle(int n)
{
    struct node *ct;
    int size;

    struct node *newNode = (struct node*)malloc(sizeof(struct node));
    newNode->data = n;
```

```
newNode->next = NULL;



if(head == NULL) {

   head = newNode;

}
else {
    ct=head;
     while(ct!=NULL){
       ct=ct->next;
       size++;
     }

   struct node *temp, *current;

   int count = (size % 2 == 0) ? (size/2) : ((size+1)/2);

   temp = head;
   current = NULL;



   for(int i = 0; i < count; i++) {

      current = temp;
```

```
            temp = temp->next;
        }



        current->next = newNode;


        newNode->next = temp;
    }
    size++;
}
```

Output:

```
ft-MIEngine-Out-ah0jslqo.5ha' '--stderr=Microsoft-
gine-Pid-urvf1bnr.t5z' '--dbgExe=C:\TDM-GCC-64\bin
Circular Linked List
1) Insert at Start
2) Insert at End
3) Insert in Middle
4) Delete at Start
5) Delete at End
6) Delete at Middle
7) Display
8) Exit
Enter your choice: 1
Enter the element you want to insert: 45
Circular Linked List
1) Insert at Start
2) Insert at End
3) Insert in Middle
4) Delete at Start
5) Delete at End
6) Delete at Middle
7) Display
8) Exit
Enter your choice: 1
Enter the element you want to insert: 90
Circular Linked List
1) Insert at Start
2) Insert at End
3) Insert in Middle
4) Delete at Start
5) Delete at End
6) Delete at Middle
7) Display
8) Exit
Enter your choice: 7
90 45
```

Conclusion:

A circular linked list in C is a dynamic data structure that has a cyclic structure, meaning the last node in the list points back to the first node, forming a circle. This unique structure allows for efficient operations at both ends of the list, making it particularly useful in scenarios where data needs to be continuously cycled or rotated. However, it requires careful handling of pointers during insertion and deletion operations to maintain the circular structure. Despite this complexity, circular linked lists are widely used due to their flexibility and efficiency in handling certain types of problems.