DS-LAB EXPERIMENT 9

NAME: YASH SARANG D6AD/47

Aim: - Implement Circular Linked List ADT

Theory: - In a circular linked list, the last node contains a pointer to the first node of the list. We can have a circular linked list as well as circular doubly linked list. While traversing a circular linked list, we can begin at any node and traverse the list in any direction, forward or backward, until we reach the same node where we started. Thus, a circular linked list has no ending and no beginning. The only downside of a circular linked list is the complexity of iteration. Note that there is no NULL value in the NEXT part of any of the nodes of list. Circular linked list is widely used in operating system for task maintenance.

Operations

In a circular linked list, we perform the following operations...

1. Insertion
2. Deletion
3. Display

Before we implement actual operations, first we need to setup empty list. First perform the following steps before implementing actual operations.

Step 1 - Include all the header files which are used in the program.
Step 2 - Declare all the user defined functions.
Step 3 - Define a Node structure with two members data and next
Step 4 - Define a Node pointer 'head' and set it to NULL.

Step 5 - Implement the main method by displaying operations menu and make suitable function calls in the main method to perform user selected operation.

Insertion

In a circular linked list, the insertion operation can be performed in three ways. They are as follows...

1. Inserting At Beginning of the list
2. Inserting At End of the list
3. Inserting At Specific location in the list

Inserting At Beginning of the list

We can use the following steps to insert a new node at beginning of the circular linked list...

Step 1 - Create a new Node with given value.
Step 2 - Check whether list is Empty (head == NULL)
Step 3 - If it is Empty then, set head = new Node and new Node→next = head .
Step 4 - If it is Not Empty then, define a Node pointer 'temp' and initialize with 'head'.
Step 5 - Keep moving the 'temp' to its next node until it reaches to the last node (until 'temp →
next == head').
Step 6 - Set 'new Node → next =head', 'head = new Node' and 'temp → next = head'.

Inserting At End of the list

We can use the following steps to insert a new node at end of the circular linked list...

Step 1 - Create a new Node with given value.
Step 2 - Check whether list is Empty (head == NULL).
Step 3 - If it is Empty then, set head = new Node and new Node → next = head.
Step 4 - If it is Not Empty then, define a node pointer temp and initialize with head.
Step 5 - Keep moving the temp to its next node until it reaches to the last node in the list
(until temp → next == head).

Step 6 - Set temp → next = new Node and new Node → next = head.

Deletion

In a circular linked list, the deletion operation can be performed in three ways those are as follows...

1. Deleting from Beginning of the list
2. Deleting from End of the list
3. Deleting a Specific Node

Deleting from Beginning of the list

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.

Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize both 'temp1' and 'temp2' with head.

Step 4 - Check whether list is having only one node (temp1 → next == head)

Step 5 - If it is TRUE then set head = NULL and delete temp1 (Setting Empty list conditions)

Step 6 - If it is FALSE move the temp1 until it reaches to the last node. (until temp1 → next == head)

Step 7 - Then set head = temp2 → next, temp1 → next = head and delete temp2.

Deleting from End of the list

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.

Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1' with head.

Step 4 - Check whether list has only one Node (temp1 → next == head)

Step 5 - If it is TRUE. Then, set head = NULL and delete temp1. And terminate from the function.

(Setting Empty list condition)
Step 6 - If it is FALSE. Then, set 'temp2 = temp1 ' and move temp1 to its next node.
Repeat the
same until temp1 reaches to the last node in the list. (until temp1 → next == head)
Step 7 - Set temp2 → next = head and delete temp1.

Program :-

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
struct node
{
  int data;
  struct node *next;
};
struct node *start = NULL;
struct node *create_cll(struct node *);
struct node *display(struct node *);
struct node *insert_beg(struct node *);
struct node *insert_end(struct node *);
struct node *delete_beg(struct node *);
struct node *delete_end(struct node *);
struct node *delete_after(struct node *);
struct node *delete_list(struct node *);
int main()
{
  int option;
  do
  {
    printf("\n\n***MAIN MENU***");
    printf("\n 1: Create a list");
    printf("\n 2: Display the list");
    printf("\n 3: Add a node at the beginning");
    printf("\n 4: Add a node at the end");
    printf("\n 5: Delete a node from the beginning");
    printf("\n 6: Delete a node from the end");
    printf("\n 7: Delete a node after a given node");
    printf("\n 8: Delete the entire list");
    printf("\n 9: EXIT");
    printf("\n\n Enter your option : ");
```

```
    scanf("%d", &option);
    switch(option)
    {
       case 1: start = create_cll(start);
       printf("\n CIRCULAR LINKED LIST CREATED");
       break;
       case 2: start = display(start);
       break;
       case 3: start = insert_beg(start);
       break;
       case 4: start = insert_end(start);
       break;
       case 5: start = delete_beg(start);
       break;
       case 6: start = delete_end(start);
       break;
       case 7: start = delete_after(start);
       break;
       case 8: start = delete_list(start);
       printf("\n CIRCULAR LINKED LIST DELETED");
       break;
    }
  }while(option !=9);
  getch();
  return 0;
}
struct node *create_cll(struct node *start)
{
    struct node*new_node, *ptr;
    int num;
    printf("\n Enter -1 to end");
    printf("\n Enter the data : ");
    scanf("%d", &num);
    while(num!=-1)
    {
       new_node = (struct node*)malloc(sizeof(struct node));
       new_node->data = num;
       if(start == NULL)
       {
          new_node->next = new_node;
          start = new_node;
       }
       else
       {
          ptr = start;
          while(ptr->next != start);
```

```
            ptr = ptr->next;
          ptr->next = new_node;
          new_node->next = start;
       }
       printf("\n Enter the data : ");
       scanf("%d", &num);
    }
    return start;
}
struct node *display(struct node *start)
{
    struct node *ptr;
    ptr=start;
    while(ptr->next != start)
    {
       printf("\t %d", ptr->data);
       ptr = ptr->next;
    }
    printf("\t %d", ptr->data);
    return start;
}
struct node *insert_beg(struct node *start)
{
    struct node *new_node, *ptr;
    int num;
    printf("\n Enter the data : ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    ptr = start;
    while(ptr->next != start)
      ptr = ptr->next;
    ptr->next = new_node;
    new_node->next = start;
    start = new_node;
    return start;
}
struct node *insert_end(struct node *start)
{
    struct node *ptr, *new_node;
    int num;
    printf("\n Enter the data : ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    ptr = start;
```

```
    while(ptr->next != start)
       ptr = ptr->next;
    ptr->next = new_node;
    new_node->next = start;
    return start;
}
struct node *delete_beg(struct node *start)
{
    struct node *ptr;
    ptr = start;
    while(ptr->next != start)
       ptr = ptr->next;
    ptr->next = start->next;
    free(start);
    start = ptr->next;
    return start;
}
struct node *delete_end(struct node *start)
{
    struct node *ptr, *preptr;
    ptr = start;
    while(ptr->next != start)
    {
       preptr = ptr;
       ptr = ptr->next;
    }
    preptr->next = ptr->next;
    free(ptr);
    return start;
}
struct node *delete_after(struct node *start)
{
    struct node *ptr, *preptr;
    int val;
    printf("\n Enter the value after which the node has to deleted : ");
    scanf("%d", &val);
    ptr = start;
    preptr = ptr;
    while(preptr->data != val)
    {
       preptr = ptr;
       ptr = ptr->next;
    }
    preptr->next = ptr->next;
    if(ptr == start)
       start = preptr->next;
```

```
    free(ptr);
    return start;
}
struct node *delete_list(struct node *start)
{
   struct node *ptr;
   ptr = start;
   while(ptr->next != start)
      start = delete_end(start);
   free(start);
   return start;
}
```

Output:

```
***MAIN MENU***
 1: Create a list
 2: Display the list
 3: Add a node at the beginning
 4: Add a node at the end
 5: Delete a node from the beginning
 6: Delete a node from the end
 7: Delete a node after a given node
 8: Delete the entire list
 9: EXIT

 Enter your option : 1

 Enter -1 to end
 Enter the data : 1

 Enter the data : 2

 Enter the data : -1

 CIRCULAR LINKED LIST CREATED

***MAIN MENU***
 1: Create a list
 2: Display the list
 3: Add a node at the beginning
 4: Add a node at the end
 5: Delete a node from the beginning
 6: Delete a node from the end
 7: Delete a node after a given node
 8: Delete the entire list
 9: EXIT
```

```
Enter your option : 3

Enter the data : 5


***MAIN MENU***
1: Create a list
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Delete a node from the beginning
6: Delete a node from the end
7: Delete a node after a given node
8: Delete the entire list
9: EXIT

Enter your option : 2
        5       1       2

***MAIN MENU***
1: Create a list
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Delete a node from the beginning
6: Delete a node from the end
7: Delete a node after a given node
8: Delete the entire list
9: EXIT

Enter your option : 9

Process returned 0 (0x0)   execution time : 33.477 s
Press any key to continue.
```