Harsh Kashiwal
2003085
C21

DATE
PAGE NO.

Experiment 7

Aim: Implement Circular linked list

Theory:

Circular linked list: linked list where all nodes form a circle.

→ Useful for implementation of a queue. We don't need to maintain two pointers for front and rear if we use the linked list.

→ Any node can be starting point. We can traverse the list by starting from any point. We just need to stop when the first visited node is visited again.

→ Different Operations:

1) Insert in beginning:

Algorithm: Insert_beg(val)

```
1. create new node.
2. newnode -> data = val
   if (head == null){
   head = newnode
   newnode -> next = head.
   return }.
  temp = head.
  while (temp -> next) = null)
     temp = temp -> next.
   new node -> next = head.
  temp -> next = newnode
   head = newnode }.
```

Harsh Kushwaha
2003085

2> Insert at end:

Algorithm: A node is inserted at the end of the circular linked list

```
insert_end(val){
Create newnode
newnode->data=val
if(head==null){
   head=newnode
   newnode->next=head
   return
}
temp=head
while(temp->next!=head)
   temp=temp->next
newnode->next=head
temp->next=newnode
}
```

3> Delete from beginning:

Algorithm:

```
del_beg(){
if(head==null)
printf(underflow)
else if(head->next==head){
   temp=head
   head=null
   free(temp)}
else{
   temp=head
```

Harish Kashiwal

```
while (temp→next != head)
    temp = temp→next
temp→next = head→next
free(head)
head = temp→next
}
```

4. Delete from the End:

```
Algorithm: del_End(){
    if (head == null)
        print (underflow)
    else if (head→next == head){
        temp = head
        head = null
        free(temp)
    }
    temp1 = temp2 = head
    while (temp1→next != head){
        temp2 = temp1
        temp1 = temp1→next
    }
    temp2→next = temp1→next
    free(temp))
}
```

Harshkaul
2003085

5. Forward traversal (Display):

Algorithm: Display(start){
ptr = start.
while (Ptr → next != start){
print (Ptr → data)
ptr → ptr → next.
}
print (ptr → data);
return start.

6. Reverse traversal

Algorithm: back (start){
If (start → next == start)
print (start → data)
return 0.
back (start → next)
print (start → data).
return 0
}.

Conclusion: If the size of the list is fixed, it is much more efficient to use circular list. We can go to any node from any node in the circular list which was not possible in the singly list if we reached the last node.

PROGRAM:

Write a menu driven code to implement Circular Linked List.

Code:

```
#include <iostream>
#include <conio.h>
using namespace std;
struct node
{
    int data;
    struct node *next;
};
int flag = 0;
int count = 0;
struct node *start = NULL;
struct node *list(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 *);
void backtraversal(struct node *);
struct node *nodes(struct node *);
int main(int argc, char *argv[])
{
    int option;
    do
    {
        cout<<"\n1.LIST ";
        cout<<"\n2.ADD NODE IN THE BEGINING";
        cout<<"\n3.ADD NODE IN THE END";
        cout<<"\n4.DELETE A NODE FROM BEGINNING";
        cout<<"\n5.DELETE A NODE FROM END";
        cout<<"\n6.DISPLAY";
        cout<<"\n7.DISPLAY REVERSE";
        cout<<"\n8.COUNT THE NUMBER OF NODES ";
        cout<<"\n9.EXIT";
        cout<<"\n\nENTER YOUR OPTION : ";
        cin>>option;
        switch (option)
        {
        case 1:
            start = list(start);
            cout<<"\n--LINKED LIST CREATED--";
            break;
        case 2:
            start = insert_beg(start);
```

```
            break;
        case 3:
            start = insert_end(start);
            break;
        case 4:
            start = delete_beg(start);
            break;
        case 5:
            start = delete_end(start);
            break;
        case 6:
            start = display(start);
            break;
        case 7:
            backtraversal(start);
            break;
        case 8:
            start = nodes(start);
            cout<<"THE NUMBER OF NODES IN THE LINKED LIST ARE : "<<count;
            break;
        }
    } while (option != 9);
    getch();
    return 0;
}
struct node *list(struct node *start)
{
    struct node *new_node, *ptr;
    int num;
    cout<<"ENTER -1 TO END!"<<endl;
    cout<<"\nENTER THE DATA :";
    cin>>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; // new node ka address
        }
        else
        {
            ptr = start;
            while (ptr->next != start)
                ptr = ptr->next;
            ptr->next = new_node;
            new_node->next = start;
```

```
        }
        cout<<"\nENTER THE DATA : ";
        cin>>num;
    }
    return start;
}
struct node *display(struct node *start)
{
    if (flag == 1)
    {
        cout<<"\n\n  EMPTY LIST  \n\n";
    }
    struct node *ptr;
    ptr = start;
    while (ptr->next != start)
    {
        cout<<"\t "<<ptr->data;
        ptr = ptr->next;
    }
    cout<<"\t "<<ptr->data;
    return start;
}
void backtraversal(struct node *ptr)
{
    if (ptr->next != start)
    {
        backtraversal(ptr->next);
    }
    cout<<" "<< ptr->data<<"\t";
}
struct node *insert_beg(struct node *start)
{
    struct node *new_node, *ptr;
    int num;
    cout<<"\nENTER THE DATA : ";
    cin>>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;
    cout<<"\nENTER THE DATA ";
    cin>>num;
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    new_node->next = start;
    ptr = start;
    while (ptr->next != start)
        ptr = ptr->next;
    ptr->next = new_node;
    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 *nodes(struct node *start)
{
    struct node *ptr;
    ptr = start->next;
    count = 1;
    while (ptr != start)
    {
        count = count + 1;
        ptr = 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;
```

```
}
```

OUTPUT:

```
PS D:\Harsh\SEM 3\DS\CODES> cd "d:\Harsh\SEM 3\DS\CODES\" ; if ($?) { g++ tempCodeRun

1.LIST
2.ADD NODE IN THE BEGINING
3.ADD NODE IN THE END
4.DELETE A NODE FROM BEGINNING
5.DELETE A NODE FROM END
6.DISPLAY
7.DISPLAY REVERSE
8.COUNT THE NUMBER OF NODES
9.EXIT

ENTER YOUR OPTION : 1
ENTER -1 TO END!

ENTER THE DATA :12

ENTER THE DATA : 13

ENTER THE DATA : 14

ENTER THE DATA : -1

--LINKED LIST CREATED--
1.LIST
2.ADD NODE IN THE BEGINING
3.ADD NODE IN THE END
4.DELETE A NODE FROM BEGINNING
5.DELETE A NODE FROM END
6.DISPLAY
7.DISPLAY REVERSE
8.COUNT THE NUMBER OF NODES
9.EXIT

ENTER YOUR OPTION : 2

ENTER THE DATA : 15

1.LIST
2.ADD NODE IN THE BEGINING
3.ADD NODE IN THE END
4.DELETE A NODE FROM BEGINNING
5.DELETE A NODE FROM END
```

```
6.DISPLAY
7.DISPLAY REVERSE
8.COUNT THE NUMBER OF NODES
9.EXIT

ENTER YOUR OPTION : 3

ENTER THE DATA 16

1.LIST
2.ADD NODE IN THE BEGINING
3.ADD NODE IN THE END
4.DELETE A NODE FROM BEGINNING
5.DELETE A NODE FROM END
6.DISPLAY
7.DISPLAY REVERSE
8.COUNT THE NUMBER OF NODES
9.EXIT

ENTER YOUR OPTION : 6
        15      12      13      14      16
1.LIST
2.ADD NODE IN THE BEGINING
3.ADD NODE IN THE END
4.DELETE A NODE FROM BEGINNING
5.DELETE A NODE FROM END
6.DISPLAY
7.DISPLAY REVERSE
8.COUNT THE NUMBER OF NODES
9.EXIT

ENTER YOUR OPTION : 4

1.LIST
2.ADD NODE IN THE BEGINING
3.ADD NODE IN THE END
4.DELETE A NODE FROM BEGINNING
5.DELETE A NODE FROM END
6.DISPLAY
7.DISPLAY REVERSE
8.COUNT THE NUMBER OF NODES
9.EXIT

ENTER YOUR OPTION : 5

1.LIST
2.ADD NODE IN THE BEGINING
```

```
3.ADD NODE IN THE END
4.DELETE A NODE FROM BEGINNING
5.DELETE A NODE FROM END
6.DISPLAY
7.DISPLAY REVERSE
8.COUNT THE NUMBER OF NODES
9.EXIT

ENTER YOUR OPTION : 6
         12      13      14
1.LIST
2.ADD NODE IN THE BEGINING
3.ADD NODE IN THE END
4.DELETE A NODE FROM BEGINNING
5.DELETE A NODE FROM END
6.DISPLAY
7.DISPLAY REVERSE
8.COUNT THE NUMBER OF NODES
9.EXIT

ENTER YOUR OPTION : 7
 14      13      12
1.LIST
2.ADD NODE IN THE BEGINING
3.ADD NODE IN THE END
4.DELETE A NODE FROM BEGINNING
5.DELETE A NODE FROM END
6.DISPLAY
7.DISPLAY REVERSE
8.COUNT THE NUMBER OF NODES
9.EXIT

ENTER YOUR OPTION : 8
THE NUMBER OF NODES IN THE LINKED LIST ARE : 3
1.LIST
2.ADD NODE IN THE BEGINING
3.ADD NODE IN THE END
4.DELETE A NODE FROM BEGINNING
5.DELETE A NODE FROM END
6.DISPLAY
7.DISPLAY REVERSE
8.COUNT THE NUMBER OF NODES
9.EXIT

ENTER YOUR OPTION : 9
```