Experiment No.7
Implement Circular Linked List ADT.
Name:Rohan Ganpat Mangaonkar
Roll No:27
Date of Performance:12/9/23
Date of Submission:26/9/23
Marks:
Sign:

Experiment No. 7: Circular Linked List Operations

Aim: Implementation of Circular Linked List ADT

Objective:

In circular linked list last node is connected to first node. On other hand circular linked list can be used to implement traversal along web pages.

Theory:

In a circular linked list, the last node contains a pointer to the first node of the list. We can have a circular singly linked list as well as a circular doubly linked list. While traversing a circular linked list, we can begin at any node and traverse the list in any one direction, forward or backward, until we reach the same node where we started. Thus, a circular linked list has no beginning and no ending.

Inserting a New Node in a Circular Linked List

Case 1: The new node is inserted at the beginning.

Case 2: The new node is inserted at the end.

Deleting a Node from a Circular Linked List

Case 1: The first node is deleted.

Case 2: The last node is deleted.

Insertion and Deletion after or before a given node is same as singly linked list.

Algorithm

Algorithm to insert a new node at the beginning

Step 1: IF AVAIL = NULL

Write OVERFLOW

Go to Step 9 [END OF IF]

Step 2: SET NEW_NODE = AVAIL

Step 3: SET AVAIL = AVAIL□NEXT

Step 4: SET NEW_NODE-->DATA = VAL

Step 5: SET PTR=START

Repeat Step 6 while PTR NEXT != START

Step 6: SET PTR = PTR NEXT [END OF LOOP]

Step 7: SET NEW_NODE--> NEXT= START

Step 8: SET PTR-->NEXT = START

Step 9: SET START = NEW_NODE

Step 10: EXIT

Algorithm to insert a new node at the end

Step 1: IF AVAIL = NULL

Write OVERFLOW

Go to Step 11 [END OF IF]

Step 2: SET NEW_NODE = AVAIL

Step 3: SET AVAIL = AVAIL--> NEXT

Step 4: SET NEW_NODE -->DATA = VAL

Step 5: SET NEW_NODE-->NEXT = START

Step 6: SET PTR = START

Step 7: Repeat Step 8 while PTR--> NEXT != START

Step 8: SET PTR = PTR -->NEXT [END OF LOOP]

Step 9: SET PTR -->NEXT = NEW_NODE

Step 10: EXIT

Algorithm to delete the first node

Step 1: IF START = NULL

Write UNDERFLOW

Go to Step 6 [END OF IF]

Step 2: SET PTR = START

Step 3: Repeat Step 4 while PTR--> NEXT != START

Step 4: SET PTR = PTR -->NEXT [END OF LOOP]

Step 4: SET PTR□NEXT = START -->NEXT

Step 5: FREE START

Step 6: EXIT

Algorithm to delete the last node

Step 1: IF START = NULL

Write UNDERFLOW

Go to Step 7 [END OF IF]

Step 2: SET PTR = START [END OF LOOP]

Step 3: Repeat Step 4 and Step 5 while PTR -->NEXT != START

Step 4: SET PREPTR = PTR

Step 5: SET PTR = PTR -->NEXT

Step 6: SET PREPTR-->NEXT = START

Step 7: FREE PTR

Step 8: EXIT

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node
{
int data;
struct node *next;
}node;
typedef struct ll
{
node *start;
}ll;
int count(ll*l)
{
  int c=0;
  node*p;
  p=l->start;
  while(p!= NULL)
```

```
    {
      c++;
      p=p->next;
    }
  return c;
}
void display(ll*l)
{
  if(l->start==NULL)
  {
    printf("\n ll is empty");
  }
  else
  {
  node*p;
  p=l->start;
  while(p!=NULL)
    {
      printf("\t %d",p->data);
      p=p->next;
    }
  }
}
void insertbeg(ll*l,int x)
{
  node*newrec;
```

```
 newrec=(node*)malloc(sizeof(node));//malloc fn always return void pointer so we used (node) to convert it into nodepointer
 newrec->data=x;
 newrec->next=NULL;
 if(l->start==NULL)
 {
  l->start=newrec;
 }
 else
 {
  newrec->next=l->start;
  l->start=newrec;
 }
}
void insertend(ll*l,int x)
{
 node*newrec,*p;
 newrec=(node*)malloc(sizeof(node));//malloc fn always return void pointer so we used (node) to convert it into nodepointer
 newrec->data=x;
 newrec->next=NULL;
 if(l->start==NULL)
 {
  printf("\nempty ll");
 }
 else
```

```
 {
  p=l->start;
  while(p->next!=NULL)
    {
     p=p->next;
    }
  p->next=newrec;
 }
}
void insertatpos(ll*l,int x,int pos)
{
 int i;
 node*newrec,*p;
 newrec=(node*)malloc(sizeof(node));
 newrec->data=x;
 newrec->next=NULL;
 if(pos>count(l)+1)
 {
  printf("\n invalid pos");
 }
 else if(pos==1)
  {
   newrec->next=l->start;
   l->start=newrec;
  }
 else
```

```
 {
  p=l->start;
  for(i=1;i<pos-1;i++)
    {
     p=p->next;
    }
  newrec->next=p->next;
  p->next=newrec;
 }
}
void deletebeg(ll*l)
{
 node*p;
 if(l->start==NULL)
 {
  printf("\nll is emty");
 }
 else
 {
  p=l->start;
  l->start=l->start->next;
  free(p);
 }
}
void deleteend(ll*l)
{
```

```
 node*p,*q;
 if(l->start==NULL)
 {
  printf("\nll is emty");
 }
 else if (l->start->next==NULL)
 {
  p=l->start;
  l->start=NULL;
  free(p);
 }
 else
 {
  q=l->start;

  while(q->next->next!=NULL)
    {
     q=q->next;
    }
  p=q->next;
  q->next=NULL;
  free(p);
 }
}
void deletepos(ll*l,int pos)
{
```

```
  int i;
  node *p,*q;
  if(pos>count(l))
    printf("\n invalide posn");
  else if (pos==1)
  {
    p=l->start;
    l->start=l->start->next;
    free(p);
  }
  else
  {
  q=l->start;
    for(i=1;i<pos-1;i++)
      {
        q=q->next;
      }
    p=q->next;
    q->next=p->next;
    free(p);
  }
}
int search (ll*l,int x)
{
  node *p;
  p=l->start;
```

```
  while(p!=NULL)
    {
     if(x==p->data)
       return 1;
      else
       p=p->next;
    }
  return 0;
}
void sort(ll *l)
{
  node *i,*j;
  int temp;
  for(i=l->start;i->next!=NULL;i=i->next)
    {
     for(j=l->start;j->next!=NULL;j=j->next)
       {
        if(j->data>j->next->data)
         {
          temp=j->data;
          j->data=j->next->data;
          j->next->data=temp;
         }
       }
    }
}
```

```
void reverse(ll*l)
{
  node *p,*q,*r;
  p=l->start;
  q= NULL;
  while(p!=NULL)
    {
      r=p->next;
      p->next=q;
      q=p;
      p=r;
    }
    l->start=q;

}
int main()
{
  int x,ch,pos,ele,f=1;
  ll l;
  l.start=NULL;
  while(1)
   {
     printf("\nMENU:\n1-insert beginning\n2-insert at end\n3-insert at posn\n4-delete at beginning\n5-delete at end\n6-delete at posn\n7-count\n8-search\n9-sort\n10-reverse\n11-display\n12-exit");
     scanf("%d",&ch);
```

```
if(ch==12)
  break;
switch(ch)
  {
    case 1:
      {
        printf("\nenter the element to be inserted");
        scanf("%d",&x);
        insertbeg(&l,x);
        display(&l);
      }break;
    case 2:
      {
        printf("\nenter the element to be inserted");
        scanf("%d",&x);
        insertend(&l,x);
        display(&l);
      }break;
    case 3:
      {
        printf("\nenter the element to be inserted and position");
        scanf("%d%d",&x,&pos);
        insertatpos(&l,x,pos);
        display(&l);
      }break;
    case 4:
```

```
  {
   deletebeg(&l);
   display(&l);
  }break;
case 5:
  {
   deleteend(&l);
   display(&l);
  }break;
case 6:
  {
   printf("\nenter the posn");
     scanf("%d",&pos);
   deletepos(&l,pos);
  }break;
case 7:
  {
   ele =count(&l);
   printf("no. of element:%d",ele);
  }break;
case 8:
  {
   printf("\nenter the element to be searhed ");
   scanf("%d",&x);
   ele=search(&l,x);
   if(ele==1)
```

```
        printf("\nelement present");
      else
        printf("\nelement  not present");
    }break;
  case 9:
    {
     sort(&l);
     display(&l);
   }break;
  case 10:
    {
     reverse(&l);
     display(&l);
    }break;
  case 11:
    {
     display(&l);
    }break;
  case 12:
    {
     f=0;
    }break;
  default:
  printf("\n invalid choice");
 }
if(f==0)
```

```
    {
      break;
    }
   }
}
```

Output:

MENU:

1-insert beginning

2-insert at end

3-insert at posn

4-delete at beginning

5-delete at end

6-delete at posn

7-count

8-search

9-sort

10-reverse

11-display

12-exit1

enter the element to be inserted1

1

MENU:

1-insert beginning

2-insert at end

3-insert at posn

4-delete at beginning

5-delete at end

6-delete at posn

7-count

8-search

9-sort

10-reverse

11-display

12-exit1

enter the element to be inserted0

0 1

MENU:

1-insert beginning

2-insert at end

3-insert at posn

4-delete at beginning

5-delete at end

6-delete at posn

7-count

8-search

9-sort

10-reverse

11-display

12-exit2

enter the element to be inserted2

0 1 2

MENU:

1-insert beginning

2-insert at end

3-insert at posn

4-delete at beginning

5-delete at end

6-delete at posn

7-count

8-search

9-sort

10-reverse

11-display

12-exit12

Conclusion:

Q1 Write an example of insertion and deletion in the circular linked list while traversing the web pages?

The provided code is a C program that implements a menu-driven system for creating, displaying, inserting, and deleting nodes in a circular linked list. It provides various options to manipulate the circular linked list.

However, there is an issue with the code related to the 'clrscr()' and 'getch()' functions, which appear to be from the old Borland C compiler and may not be compatible with modern

compilers or environments. Additionally, the code uses the `<conio.h>` and `<malloc.h>` headers, which are not part of the standard C library and are specific to certain compilers.

Regarding your question about a circular linked list example related to web page traversal, here's a conceptual example:

Assume you want to implement a browser history using a circular linked list. Each node in the list represents a web page, and the 'next' pointer connects the current page to the next page you visit.

1. *Insertion*: When you visit a new web page, you would insert it into the circular linked list. The 'insert_end' function in the code could be used for this purpose. It appends the new page to the end of the list, making it the next page in the browsing history.

2. *Deletion*: When you want to go back to the previous page, you can delete the last page you visited. The 'delete_end' function in the code could be used for this. It removes the last page, effectively taking you back in your browsing history.

By using this circular linked list, you can easily navigate forward and backward through the web pages you've visited, making it a suitable data structure for managing browser history. However, in a real web browser, you would need to store more information about each web page, such as the page URL, title, etc. The code you provided mainly deals with the 'data' field in the nodes, which could be expanded to include more information for a practical implementation.