Data Structures : Algorithms and Applications(Lab)

EXPERIMENT No: 8

AIM:The aim of this experiment is to implement and understand the operations of a Doubly Circular Linked List using the C programming language.

THEORY:

Doubly Circular Linked List

A Doubly Circular Linked List is a data structure that consists of nodes where each node contains three components:

1. Data: The value stored in the node.
2. Next Pointer: A pointer to the next node in the sequence.
3. Previous Pointer: A pointer to the previous node in the sequence.

Characteristics:

- Circular Structure: The last node's next pointer points back to the head of the list, creating a circular arrangement. Similarly, the head's prev pointer points to the last node, allowing traversal in both directions.
- Doubly Linked: Each node contains pointers to both the next and previous nodes, allowing easy traversal in both directions (forwards and backwards).

Advantages

- Bidirectional Traversal: Unlike singly linked lists, where traversal is only possible in one direction, doubly circular linked lists allow for traversal in both directions, enhancing flexibility.
- Ease of Insertion/Deletion: Insertion and deletion operations can be performed more efficiently compared to singly linked lists, as there is no need to maintain a pointer to the previous node when deleting a node (because each node has a pointer to its previous node).
- Circular Nature: The circular structure eliminates the need to check for null pointers when traversing the list, making operations like displaying the list simpler and more efficient.

Operations

1. **Insertion:**
    - At Beginning: A new node is created and added to the front of the list. The head pointer is updated to point to the new node.
    - At Last: A new node is appended to the end of the list. The pointers of the existing last node and the new node are updated to maintain the circular structure.

2. **Deletion:**
    - From Beginning: The node at the head of the list is removed. The head pointer is updated to point to the next node in the list.
    - From Last: The last node is removed, and the pointers are updated to ensure the circular nature of the list is preserved.
3. **Search:** A specific item can be searched within the list. If found, the position of the item is returned.
4. **Display:** The contents of the list can be printed, showcasing all the elements stored in the nodes.

CODE:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   struct node *prev;
   struct node *next;
   int data;
};
struct node *head;
void insertion_beginning();
void insertion_last();
void deletion_beginning();
void deletion_last();
void display();
void search();
void main ()
{
int choice =0;
   while(choice != 9)
   {
      printf("\n*********Main Menu*********\n");
      printf("\nChoose one option from the following list ...\n");
      printf("\n===============================================\n");
          printf("\n1.Insert in Beginning\n2.Insert at last\n3.Delete from Beginning\n4.Delete from
last\n5.Search\n6.Show\n7.Exit\n");
      printf("\nEnter your choice?\n");
      scanf("\n%d",&choice);
      switch(choice)
      {
         case 1:
         insertion_beginning();
         break;
```

```
            case 2:
                  insertion_last();
            break;
            case 3:
            deletion_beginning();
            break;
            case 4:
            deletion_last();
            break;
            case 5:
            search();
            break;
            case 6:
            display();
            break;
            case 7:
            exit(0);
            break;
            default:
            printf("Please enter valid choice..");
        }
    }
}
void insertion_beginning()
{
  struct node *ptr,*temp;
  int item;
  ptr = (struct node *)malloc(sizeof(struct node));
  if(ptr == NULL)
  {
     printf("\nOVERFLOW");
  }
  else
  {
   printf("\nEnter Item value");
   scanf("%d",&item);
   ptr->data=item;
  if(head==NULL)
  {
    head = ptr;
    ptr -> next = head;
    ptr -> prev = head;
  }
```

```
  else
  {
     temp = head;
   while(temp -> next != head)
   {
      temp = temp -> next;
   }
   temp -> next = ptr;
   ptr -> prev = temp;
   head -> prev = ptr;
   ptr -> next = head;
   head = ptr;
  }
  printf("\nNode inserted\n");
}

}
void insertion_last()
{
  struct node *ptr,*temp;
  int item;
  ptr = (struct node *) malloc(sizeof(struct node));
  if(ptr == NULL)
  {
     printf("\nOVERFLOW");
  }
  else
  {
     printf("\nEnter value");
     scanf("%d",&item);
      ptr->data=item;
     if(head == NULL)
     {
        head = ptr;
        ptr -> next = head;
        ptr -> prev = head;
     }
     else
     {
       temp = head;
       while(temp->next !=head)
       {
          temp = temp->next;
```

```
        }
        temp->next = ptr;
        ptr ->prev=temp;
        head -> prev = ptr;
     ptr -> next = head;
      }
  }
    printf("\nnode inserted\n");
}

void deletion_beginning()
{
   struct node *temp;
   if(head == NULL)
   {
      printf("\n UNDERFLOW");
   }
   else if(head->next == head)
   {
      head = NULL;
      free(head);
      printf("\nnode deleted\n");
   }
   else
   {
      temp = head;
      while(temp -> next != head)
      {
         temp = temp -> next;
      }
      temp -> next = head -> next;
      head -> next -> prev = temp;
      free(head);
      head = temp -> next;
   }

}
void deletion_last()
{
   struct node *ptr;
   if(head == NULL)
   {
      printf("\n UNDERFLOW");
```

```
    }
    else if(head->next == head)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
    else
    {
        ptr = head;
        if(ptr->next != head)
        {
            ptr = ptr -> next;
        }
        ptr -> prev -> next = head;
        head -> prev = ptr -> prev;
        free(ptr);
        printf("\nnode deleted\n");
    }
}

void display()
{
    struct node *ptr;
    ptr=head;
    if(head == NULL)
    {
        printf("\nnothing to print");
    }
    else
    {
        printf("\n printing values ... \n");

        while(ptr -> next != head)
        {

            printf("%d\n", ptr -> data);
            ptr = ptr -> next;
        }
        printf("%d\n", ptr -> data);
    }

}
```

```
void search()
{
   struct node *ptr;
   int item,i=0,flag=1;
   ptr = head;
   if(ptr == NULL)
   {
      printf("\nEmpty List\n");
   }
   else
   {
      printf("\nEnter item which you want to search?\n");
      scanf("%d",&item);
      if(head ->data == item)
      {
      printf("item found at location %d",i+1);
      flag=0;
      }
      else
      {
      while (ptr->next != head)
      {
         if(ptr->data == item)
         {
            printf("item found at location %d ",i+1);
            flag=0;
            break;
         }
         else
         {
            flag=1;
         }
         i++;
         ptr = ptr -> next;
      }
      }
      if(flag != 0)
      {
         printf("Item not found\n");
      }
   }

}
```

OUTPUT:

```
********* Main Menu *********

Choose one option from the following list ...

================================================

1. Insert in Beginning
2. Insert at Last
3. Insert at Random Position
4. Delete from Beginning
5. Delete from Last
6. Delete from Random Position
7. Search
8. Show
9. Exit

Enter your choice?
1

Enter Item value: 45

Node inserted
```

```
********* Main Menu *********

Choose one option from the following list ...

================================================

1. Insert in Beginning
2. Insert at Last
3. Insert at Random Position
4. Delete from Beginning
5. Delete from Last
6. Delete from Random Position
7. Search
8. Show
9. Exit

Enter your choice?
3

Enter Item value: 67
Enter the position (0 to 1) to insert: 1

Node inserted at position 1
```

```
********* Main Menu *********

Choose one option from the following list ...

================================================

1. Insert in Beginning
2. Insert at Last
3. Insert at Random Position
4. Delete from Beginning
5. Delete from Last
6. Delete from Random Position
7. Search
8. Show
9. Exit

Enter your choice?
3

Enter Item value: 90
Enter the position (0 to 2) to insert: 2

Node inserted at position 2
```

```
Choose one option from the following list ...

================================================

1. Insert in Beginning
2. Insert at Last
3. Insert at Random Position
4. Delete from Beginning
5. Delete from Last
6. Delete from Random Position
7. Search
8. Show
9. Exit

Enter your choice?
2

Enter value: 62

Node inserted
```

```
Choose one option from the following list ...

================================================

1. Insert in Beginning
2. Insert at Last
3. Insert at Random Position
4. Delete from Beginning
5. Delete from Last
6. Delete from Random Position
7. Search
8. Show
9. Exit

Enter your choice?
7

Enter item which you want to search?
90
Item found at location 3
```

```
Choose one option from the following list ...

================================================

1. Insert in Beginning
2. Insert at Last
3. Insert at Random Position
4. Delete from Beginning
5. Delete from Last
6. Delete from Random Position
7. Search
8. Show
9. Exit

Enter your choice?
8

Printing values ...
45
67
90
62
```

```
Choose one option from the following list ...

================================================

1. Insert in Beginning
2. Insert at Last
3. Insert at Random Position
4. Delete from Beginning
5. Delete from Last
6. Delete from Random Position
7. Search
8. Show
9. Exit

Enter your choice?
4

Node deleted
```

```
Choose one option from the following list ...

================================================

1. Insert in Beginning
2. Insert at Last
3. Insert at Random Position
4. Delete from Beginning
5. Delete from Last
6. Delete from Random Position
7. Search
8. Show
9. Exit

Enter your choice?
5

Node deleted
```

```
Choose one option from the following list ...

================================================

1. Insert in Beginning
2. Insert at Last
3. Insert at Random Position
4. Delete from Beginning
5. Delete from Last
6. Delete from Random Position
7. Search
8. Show
9. Exit

Enter your choice?
6
Enter the position (0 to 1) to delete: 1

Node deleted from position 1
```

```
Choose one option from the following list ...

================================================

1. Insert in Beginning
2. Insert at Last
3. Insert at Random Position
4. Delete from Beginning
5. Delete from Last
6. Delete from Random Position
7. Search
8. Show
9. Exit

Enter your choice?
8

Printing values ...
67
```

```
Choose one option from the following list ...

================================================

1. Insert in Beginning
2. Insert at Last
3. Insert at Random Position
4. Delete from Beginning
5. Delete from Last
6. Delete from Random Position
7. Search
8. Show
9. Exit

Enter your choice?
9


=== Code Execution Successful ===
```

CONCLUSION:

A Doubly Circular Linked List offers efficient bidirectional traversal and simplifies operations like insertion and deletion at both ends. Its circular structure removes the need for null pointer checks, making it ideal for applications requiring continuous data processing, such as circular buffers and navigation systems.