Experiment no. 3

Objective: Implementation of circular and double linked list

Code: Circular Linked list:

```
#include <stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *head;

void insert_beg ();
void insert_end ();
void insert_random();
void delete_beg();
void delete_end();
void delete_random();
void display();
void search();
void main ()
{
   int choice =0;
   while(choice != 9)
   {
```

```
printf("\n**********Menu**********\n");
printf("\n1.Insert in begining\n");
printf("2.Insert at last\n");
printf("3.Insert at random\n");
printf("4.Delete from Beginning\n");
printf("5.Delete from last\n");
printf("6.Delete at random\n");
printf("7.Search for an element\n");
printf("8.Show\n");
printf("9.Exit\n");
printf("\nEnter your choice:\n");
scanf("\n%d",&choice);
switch(choice)
{
   case 1:
   insert_beg();
   break;
   case 2:
   insert_end();
   break;
   case 3:
   insert_random();
   break;
   case 4:
   delete_beg();
   break;
   case 5:
```

```
            delete_end();
            break;
            case 6:
            delete_random();
            case 7:
            search();
            break;
            case 8:
            display();
            break;
            case 9:
            exit(0);
            break;
            default:
            printf("Please enter valid choice:");
        }
    }
}
void insert_beg()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
```

```
    else
    {
        printf("\nEnter the node data:");
        scanf("%d",&item);
        ptr -> data = item;
        if(head == NULL)
        {
            head = ptr;
            ptr -> next = head;
        }
        else
        {
            temp = head;
            while(temp->next != head)
                temp = temp->next;
            ptr->next = head;
            temp -> next = ptr;
            head = ptr;
        }
        printf("\nnode inserted\n");
    }

}
void insert_end()
{
    struct node *ptr,*temp;
    int item;
```

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

   printf("\nnode inserted\n");
```

```
    }

}
void insert_random(int data, int position)
{
struct node *newnode,*current;
int i;
if(head == NULL)
{
printf("List is empty.\n");
}
else if(position == 0)
{
insert_beg(data);
}
else
{
newnode=(struct node*)malloc(sizeof(struct node));
newnode->data=data;
printf("Enter your data:");
scanf("%d",data);
printf("\nThe element inserted");
printf("\n");
current = head;
for(i=2; i<=position; i++)
{
current = current->next;
```

```
}
newnode->next=current->next;
current->next=newnode;
}
}

void delete_beg()
{
   struct node *ptr;
   if(head == NULL)
   {
      printf("\nUNDERFLOW");
   }
   else if(head->next == head)
   {
      head = NULL;
      free(head);
      printf("\nnode deleted\n");
   }

   else
   {   ptr = head;
      while(ptr -> next != head)
         ptr = ptr -> next;
      ptr->next = head->next;
      free(head);
      head = ptr->next;
```

```
        printf("\nnode deleted\n");

    }
}
void delete_end()
{
    struct node *ptr, *preptr;
    if(head==NULL)
    {
        printf("\nUNDERFLOW");
    }
    else if (head ->next == head)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");

    }
    else
    {
        ptr = head;
        while(ptr ->next != head)
        {
            preptr=ptr;
            ptr = ptr->next;
        }
        preptr->next = ptr -> next;
```

```
        free(ptr);
        printf("\nnode deleted\n");

    }
}
void delete_random()
{
struct node * temp, *s;
if (head == NULL)
printf("\nThe List is empty");
else
{
int count = 0, pos;
printf("\nEnter the position to be deleted : ");
scanf("%d", &pos);
temp = head;
while (count < pos)
{
s = temp;
temp = temp -> next;
count++;
}
printf("\nThe element is deleted");
printf("\n");
s -> next = temp -> next;
printf("\n");
free(temp);
```

```
}
}

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)
            {
```

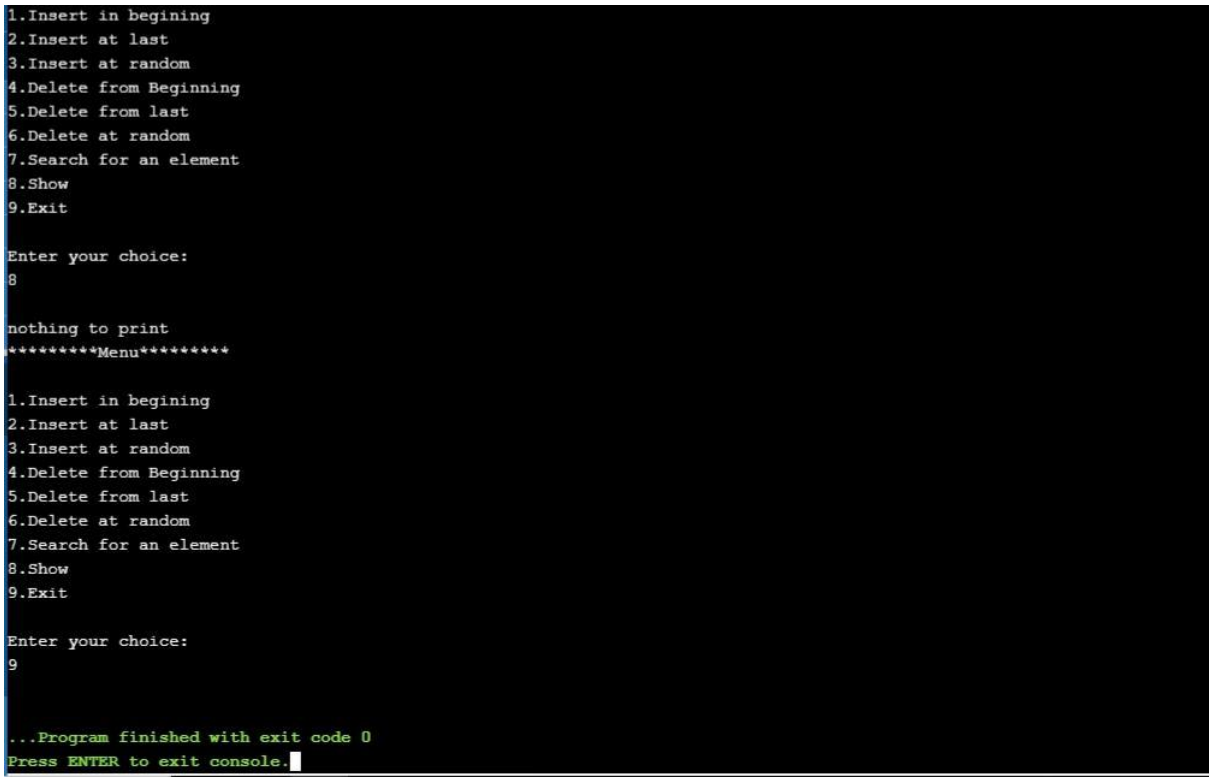
```
            printf("item found at location %d ",i+1);
            flag=0;
            break;
          }
          else
          {
            flag=1;
          }
          i++;
          ptr = ptr -> next;
        }
        }
        if(flag != 0)
        {
          printf("Item not found\n");
        }
    }

}

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);
    }

}
```

Output: Circular Linked list:

```
*********Menu*********

1.Insert in begining
2.Insert at last
3.Insert at random
4.Delete from Beginning
5.Delete from last
6.Delete at random
7.Search for an element
8.Show
9.Exit

Enter your choice:
1

Enter the node data:2

node inserted

*********Menu*********

1.Insert in begining
2.Insert at last
3.Insert at random
4.Delete from Beginning
5.Delete from last
6.Delete at random
7.Search for an element
8.Show
9.Exit
```

```
9.Exit

Enter your choice:
2

Enter Data:3

node inserted

*********Menu*********

1.Insert in begining
2.Insert at last
3.Insert at random
4.Delete from Beginning
5.Delete from last
6.Delete at random
7.Search for an element
8.Show
9.Exit

Enter your choice:
4

node deleted

*********Menu*********

1.Insert in begining
2.Insert at last
3.Insert at random
4.Delete from Beginning
5.Delete from last
6.Delete at random
7.Search for an element
8.Show
9.Exit

Enter your choice:
5

node deleted

*********Menu*********

1.Insert in begining
2.Insert at last
3.Insert at random
4.Delete from Beginning
5.Delete from last
6.Delete at random
7.Search for an element
8.Show
9.Exit

Enter your choice:
6

The List is empty
Empty List

*********Menu*********

1.Insert in begining
```



Code: Double Linked list:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   struct node *prev;
   struct node *next;
   int data;
};
struct node *head;
void insert_beg();
void insert_end();
void insert_random();
```

```
void delete_beg();
void delete_end();
void delete_random();
void display();
void search();
void main ()
{
int choice =0;
   while(choice != 9)
   {
      printf("\n*********Menu*********\n");
      printf("\n1.Insert in begining\n");
      printf("2.Insert at end\n");
      printf("3.Insert at any random\n");
      printf("4.Delete from Beginning\n");
      printf("5.Delete from end\n");
      printf("6.Delete at random\n");
      printf("7.Search\n");
      printf("8.Show\n");
      printf("9.Exit\n");
      printf("\nEnter your choice:\n");
      scanf("\n%d",&choice);
      switch(choice)
      {
         case 1:
         insert_beg();
         break;
```

```
    case 2:
    insert_end();
    break;
    case 3:
    insert_random();
    break;
    case 4:
    delete_beg();
    break;
    case 5:
    delete_end();
    break;
    case 6:
    delete_random();
    break;
    case 7:
    search();
    break;
    case 8:
    display();
    break;
    case 9:
    exit(0);
    break;
    default:
    printf("Please enter valid choice:");
}
```

```
    }
}
void insert_beg()
{
   struct node *ptr;
   int item;
   ptr = (struct node *)malloc(sizeof(struct node));
   if(ptr == NULL)
   {
      printf("\nOVERFLOW");
   }
   else
   {
    printf("\nEnter Item value:");
    scanf("%d",&item);

   if(head==NULL)
   {
      ptr->next = NULL;
      ptr->prev=NULL;
      ptr->data=item;
      head=ptr;
   }
   else
   {
      ptr->data=item;
      ptr->prev=NULL;
```

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

}
void insert_end()
{
    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)
        {
            ptr->next = NULL;
            ptr->prev = NULL;
            head = ptr;
```

```
        }
        else
        {
          temp = head;
          while(temp->next!=NULL)
          {
             temp = temp->next;
          }
          temp->next = ptr;
          ptr ->prev=temp;
          ptr->next = NULL;
          }

        }
      printf("\nnode inserted\n");
     }
void insert_random()
{
   struct node *ptr,*temp;
   int item,loc,i;
   ptr = (struct node *)malloc(sizeof(struct node));
   if(ptr == NULL)
   {
      printf("\n OVERFLOW");
   }
   else
   {
```

```
      temp=head;
      printf("Enter the location:");
      scanf("%d",&loc);
      for(i=0;i<loc;i++)
      {
         temp = temp->next;
         if(temp == NULL)
         {
            printf("\n There are less than %d elements", loc);
            return;
         }
      }
      printf("Enter value:");
      scanf("%d",&item);
      ptr->data = item;
      ptr->next = temp->next;
      ptr -> prev = temp;
      temp->next = ptr;
      temp->next->prev=ptr;
      printf("\nnode inserted\n");
   }
}
void delete_beg()
{
   struct node *ptr;
   if(head == NULL)
   {
```

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

}
void delete_end()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\n UNDERFLOW");
    }
    else if(head->next == NULL)
    {
```

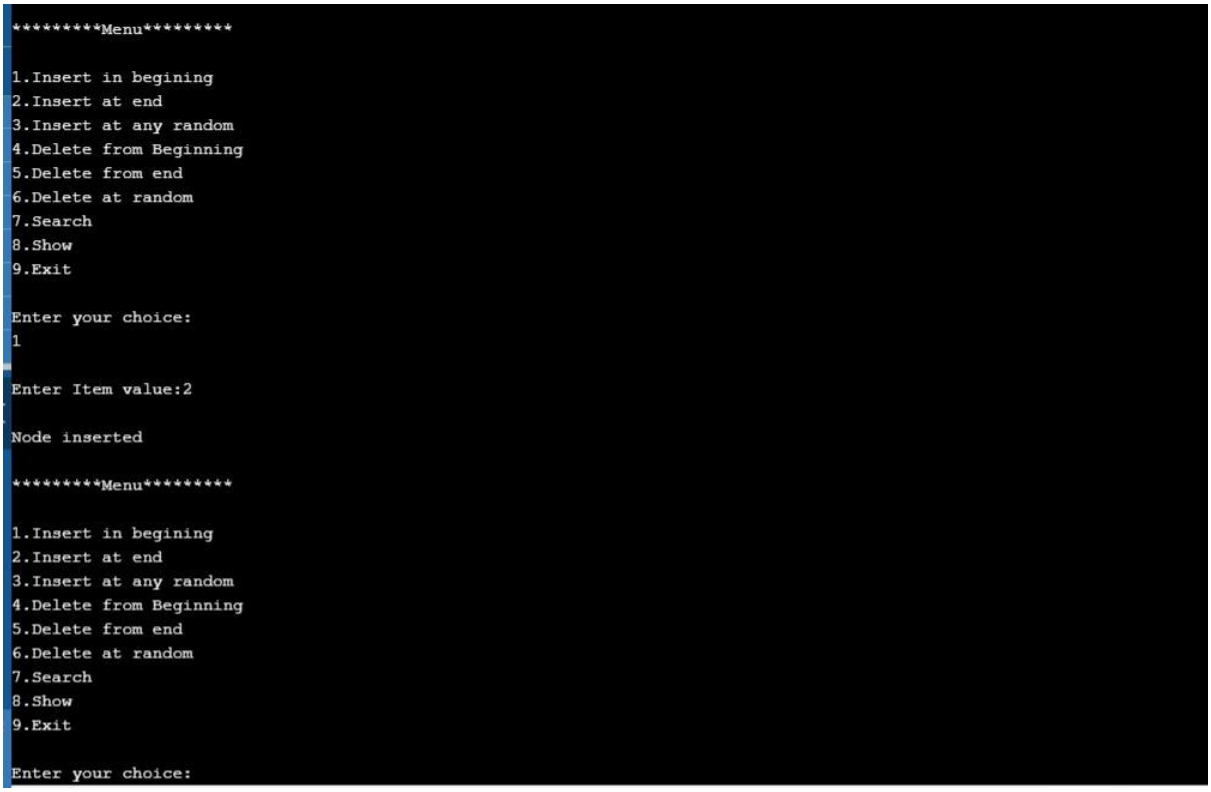
```
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
    else
    {
        ptr = head;
        if(ptr->next != NULL)
        {
            ptr = ptr -> next;
        }
        ptr -> prev -> next = NULL;
        free(ptr);
        printf("\nnode deleted\n");
    }
}
void delete_random()
{
    struct node *ptr, *temp;
    int val;
    printf("\n Enter the data to be deleted : ");
    scanf("%d", &val);
    ptr = head;
    while(ptr -> data != val)
    ptr = ptr -> next;
    if(ptr -> next == NULL)
    {
```

```
        printf("\nCan't delete\n");
    }
    else if(ptr -> next -> next == NULL)
    {
        ptr ->next = NULL;
    }
    else
    {
        temp = ptr -> next;
        ptr -> next = temp -> next;
        temp -> next -> prev = ptr;
        free(temp);
        printf("\nnode deleted\n");
    }
}
void display()
{
    struct node *ptr;
    printf("\n printing values...\n");
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d\n",ptr->data);
        ptr=ptr->next;
    }
}
void search()
```

```
{
    struct node *ptr;
    int item,i=0,flag;
    ptr = head;
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        printf("\nEnter item to search:\n");
        scanf("%d",&item);
        while (ptr!=NULL)
        {
            if(ptr->data == item)
            {
                printf("\nitem found at location %d ",i+1);
                flag=0;
                break;
            }
            else
            {
                flag=1;
            }
            i++;
            ptr = ptr -> next;
        }
```

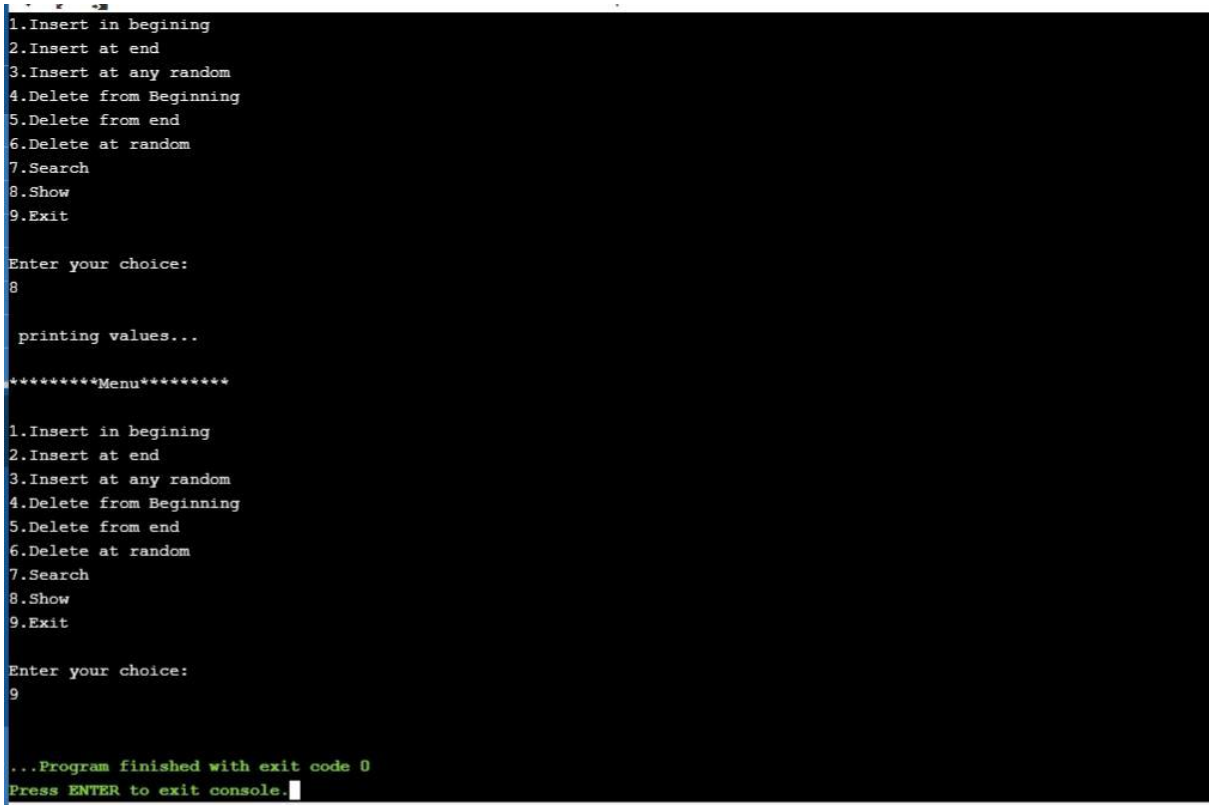
```
    if(flag==1)
    {
        printf("\nItem not found\n");
    }
}
```

Output: Double Linked list:



```
8.Show
9.Exit

Enter your choice:
4

node deleted

*********Menu*********

1.Insert in begining
2.Insert at end
3.Insert at any random
4.Delete from Beginning
5.Delete from end
6.Delete at random
7.Search
8.Show
9.Exit

Enter your choice:
5

node deleted

*********Menu*********

1.Insert in begining
2.Insert at end
3.Insert at any random
4.Delete from Beginning
5.Delete from end
6.Delete at random
```

```
4.Delete from Beginning
5.Delete from end
6.Delete at random
7.Search
8.Show
9.Exit

Enter your choice:
7

Empty List

*********Menu*********

1.Insert in begining
2.Insert at end
3.Insert at any random
4.Delete from Beginning
5.Delete from end
6.Delete at random
7.Search
8.Show
9.Exit

Enter your choice:
8

 printing values...

*********Menu*********

1.Insert in begining
2.Insert at end
```



Presented by: Gelle Hruthesh reddy (20BCB7031)