Lab Report

Of

Data Structures And Algorithms

Banke Bageshwori Campus

(Affiliated to TU)

Nepalgunj Banke, Surkhet road

**Submitted by:**

Name: ………………….

Program: B.Sc. CSIT

Batch: 2079

Semester: 3rd

**Submitted to**:

Name: Lautan Tharu

Signature:………….

**LAB 1:**

//array implementation of circular queue

#include<stdio.h>

#include<stdlib.h>

#define SIZE 5

struct cqueue

{

int item[SIZE];

int rear;

int front;

};

typedef struct cqueue qu;

void insert(qu\*);

void delete(qu\*);

void display(qu\*);

int main()

{

int ch;

qu q;

q.rear = SIZE-1;

q.front = SIZE-1;

printf("Menu for program:\n");

printf("1:Enqueue\n2:Dequeue\n3:Display\n4:Exit\n");

do

{

printf("Enter your choice\n");

scanf("%d",&ch);

switch (ch)

{

case 1:

insert(&q);

break;

case 2:

delete(&q);

break;

case 3:

display(&q);

break;

case 4:

exit(1);

break;

default:

printf("INVALID CHOICE\n");

break;

}

} while (ch!=4);

return 0;

}

void insert(qu\* q)

{

int d;

if ((q->rear+1)%SIZE==q->front)

{

printf("Queue is full\n");

}

else

{

q->rear=(q->rear+1)%SIZE;

printf("Enter data to be inserted:");

scanf("%d",&d);

q->item[q->rear]=d;

}

}

void delete(qu\* q)

{

if (q->rear==q->front)

{

printf("Queue is empty\n");

}

else

{

q->front=(q->front+1)%SIZE;

printf("Deleted item is:");

printf("%d\n",q->item[q->front]);

}

}

void display(qu \*q)

{

int i;

if (q->rear==q->front)

{

printf("Queue is empty\n");

}

else{

printf("Items of queue are:\n");

for (int i = (q->front+1)%SIZE; i!=(q->rear+1); i=(i+1)%SIZE)

{

printf("%d\t",q->item[i]);

}

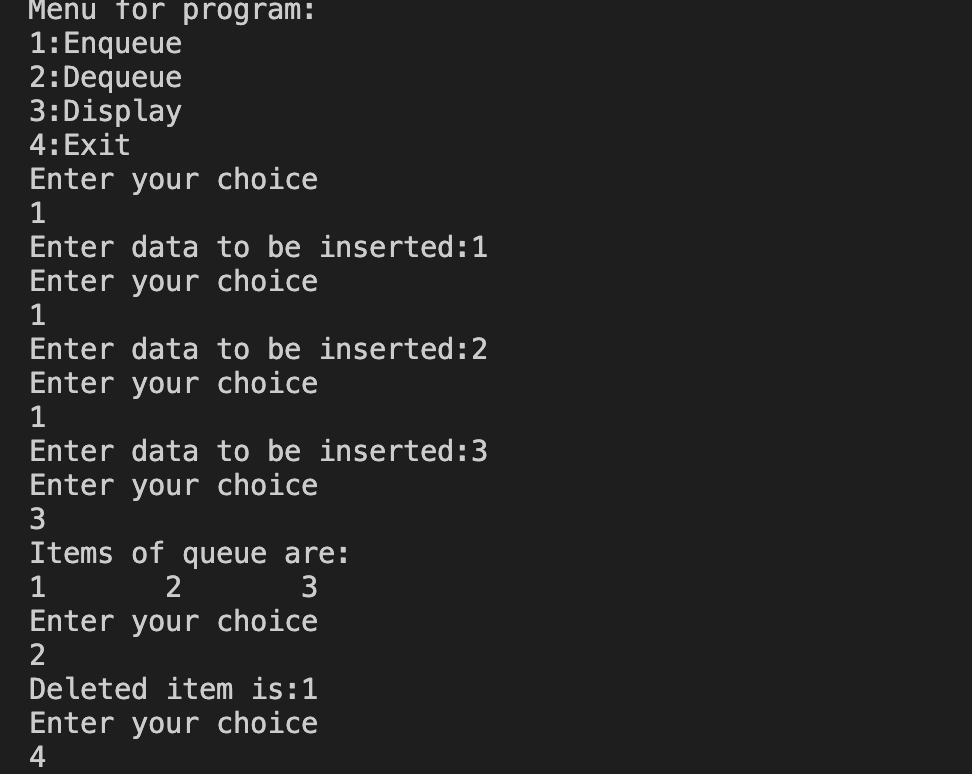
printf("\n");

}

}

**OUTPUT :**

**Circular queue**

****

**LAB 2:**

//program to implement the operations of stack

#include<stdio.h>

#include<stdlib.h>

int stack[100],n,i,j;

int top=-1;

void push()

{

int item;

if (top==n)

{

printf("Stack Overflow\n");

}

else{

printf("Enter the data to be pushed:");

scanf("%d",&item);

top=top+1;

stack[top]=item;

}

}

void pop()

{

int item;

if (top<0)

{

printf("Stack is Empty\n");

}

else{

item=stack[top];

top=top-1;

printf("the poped item is:%d\n",item);

}

}

void display()

{

if (top<0)

{

printf("Stack is Empty\n");

}

else{

i=top;

while (i>=0)

{

printf("%d\t",stack[i]);

i=i-1;

}

printf("\n");

}

}

int main()

{

int choice,size;

printf("Menu driven program to show stack operations\n");

printf("Enter the size of stack:");

scanf("%d",&size);

n=size-1;

printf("choose\n1.push\n2.pop\n3.display\n4.exit\n");

do

{

printf("\nEnter you choice:");

scanf("%d",&choice);

switch (choice)

{

case 1:

printf("you choose to push\n");

push();

break;

case 2:

printf("you choose to pop\n");

pop();

break;

case 3:

printf("you choose to display stack\n");

display();

break;

case 4:

printf("Exitinig....\n");

break;

default:

printf("Invalid choice!!!!\n");

break;

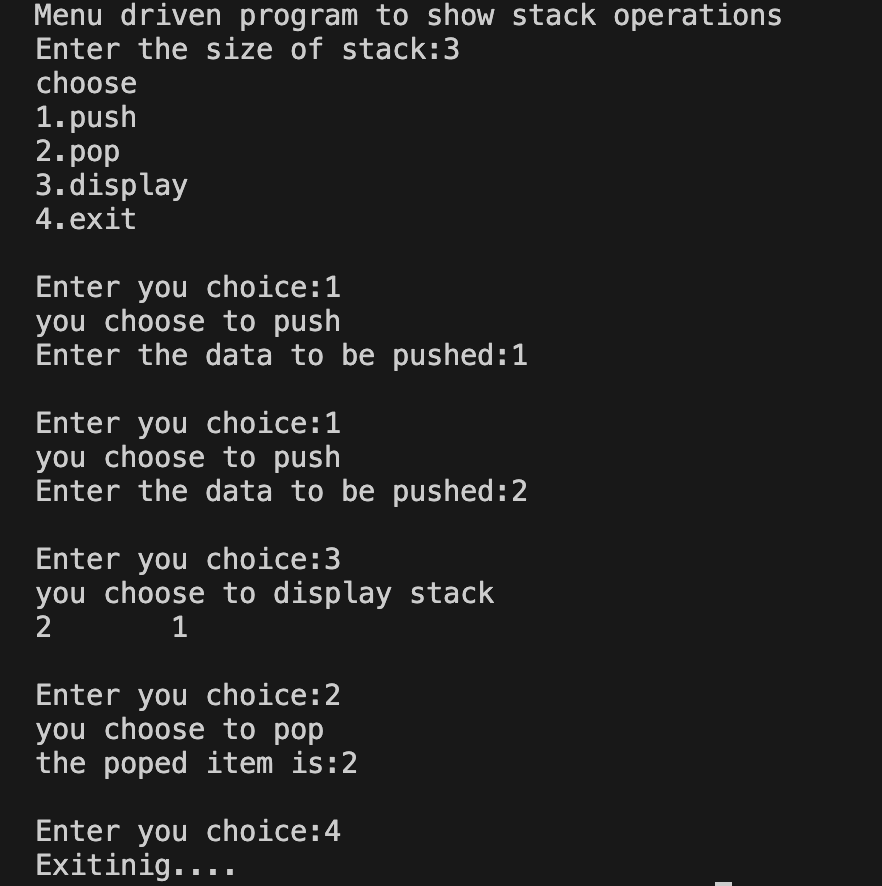
}

} while (choice!=4);

}

**OUTPUT:**

**Stack**

****

**LAB 3:**

//recursive solution of Tower of Hanoi

#include<stdio.h>

void TOH(int n,char A,char B,char C){

if (n>0)

{

TOH(n-1,A,C,B);

printf("Move disk %d from %c to %c\n",n,A,B);

TOH(n-1,C,B,A);

}

}

int main()

{

int n;

printf("Enter the number of disks:");

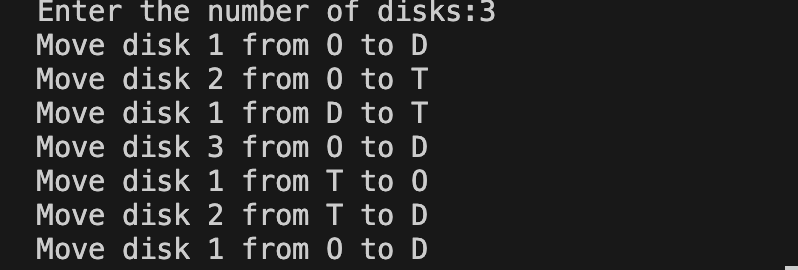
scanf("%d",&n);

TOH(n,'O','D','T');

return 0;

}

**OUTPUT:**

**Tower of Hanoi**

**LAB 4:**

//program to implement circular double linked list

#include<stdio.h>

#include<stdlib.h>

struct Node{

int info;

struct Node \*prev;

struct Node \*next;

};

struct Node \*first=NULL;

struct Node \*last=NULL;

void insert\_first()

{

int item;

struct Node \*Newnode;

Newnode=(struct Node\*)malloc(sizeof(struct Node));

printf("Enter the info:");

scanf("%d",&item);

Newnode->info=item;

if (first==NULL)

{

first=last=Newnode;

first->next=last;

first->prev=last;

}

else{

first->prev=Newnode;

Newnode->next=first;

first=Newnode;

Newnode->prev=last;

last->next=first;

}

}

void insert\_end()

{

int item;

struct Node \*Newnode;

Newnode=(struct Node\*)malloc(sizeof(struct Node));

printf("Enter the info:");

scanf("%d",&item);

Newnode->info=item;

if (first==NULL)

{

first=last=Newnode;

first->prev=last;

first->next=last;

}

else{

last->next=Newnode;

Newnode->prev=last;

Newnode->next=first;

first->prev=Newnode;

last=Newnode;

}

}

void delete\_first()

{

struct Node \*temp;

if (first==NULL)

{

printf("The linked list is Empty\n");

}

else if (first==last)

{

temp=first;

first=last=NULL;

printf("%d is successfully deleted from first\n",temp->info);

free(temp);

}

else{

temp=first;

first=first->next;

first->prev=last;

last->next=first;

printf("%d is successfully deleted from first\n",temp->info);

free(temp);

}

}

void delete\_last()

{

struct Node \*temp;

if (first==NULL)

{

printf("The linked list is Empty\n");

}

else if (first==last)

{

temp=last;

first=last=NULL;

printf("%d is successfully deleted from last\n",temp->info);

free(temp);

}

else{

temp=last;

last=last->prev;

last->next=first;

first->prev=last;

printf("%d is successfully deleted from last\n",temp->info);

free(temp);

}

}

void display()

{

struct Node \*temp;

if (first==NULL)

{

printf("The linked list is empty\n");

}

else{

temp=first;

while (temp!=last)

{

printf("%d\t",temp->info);

temp=temp->next;

}

printf("%d\t\n",last->info);

}

}

int main()

{

int choice;

printf("Menu driven program to show the operations in circular doubly linked list\n");

printf("choose\n1.insert first\n2.insert last\n3.delete first\n4.delete last\n5.display\n6.exit\n");

do

{

printf("\nEnter your choice:");

scanf("%d",&choice);

switch (choice)

{

case 1:

insert\_first();

break;

case 2:

insert\_end();

break;

case 3:

delete\_first();

break;

case 4:

delete\_last();

break;

case 5:

display();

break;

case 6:

printf("Exiting...\n");

break;

default:

printf("Invalid choice!!!\n");

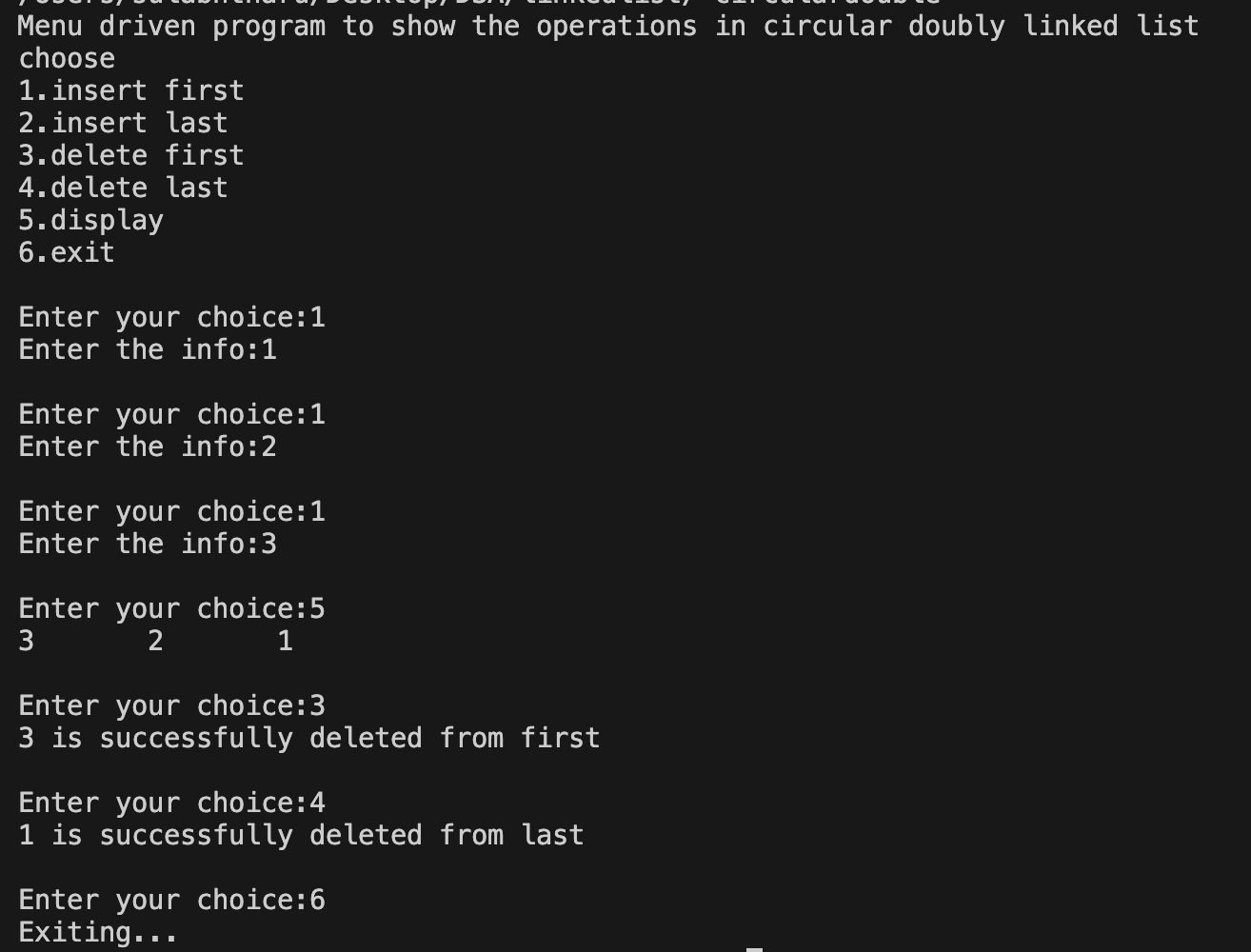
break;

}

} while (choice!=6);

}

**OUTPUT:**

**Circular doubly linked list**

**LAB 5:**

//program to implement operations of linked list as stack

#include<stdio.h>

#include<stdlib.h>

struct stacklist

{

int info;

struct stacklist \*next;

};

struct stacklist \*top=NULL;

void push()

{

int item;

struct stacklist \*Newnode;

Newnode=(struct stacklist\*)malloc(sizeof(struct stacklist));

printf("Enter The item to be pushed:");

scanf("%d",&item);

Newnode->info=item;

if (top==NULL)

{

Newnode->next=NULL;

top=Newnode;

}

else{

Newnode->next=top;

top=Newnode;

}

}

void pop()

{

struct stacklist \*temp;

if (top==NULL)

{

printf("Stack is empty\n");

}

else{

temp=top;

top=top->next;

printf("%d is deleted from top\n",temp->info);

free(temp);

}

}

void display()

{

struct stacklist \*temp;

temp=top;

if (top==NULL)

{

printf("Stack list is empty\n");

}

else{

while (temp!=NULL)

{

printf("%d\t",temp->info);

temp=temp->next;

}

printf("\n");

}

}

int main()

{

int choice;

printf("Menu driven program to show operations of Linked list as stack\n");

printf("Choose\n1.Push\n2.Pop\n3.Display\n4.Exit\n");

do

{

printf("\nEnter your choice");

scanf("%d",&choice);

switch (choice)

{

case 1:

push();

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

printf("Exiting...\n");

break;

default:

printf("Invalid Choice!!\n");

break;

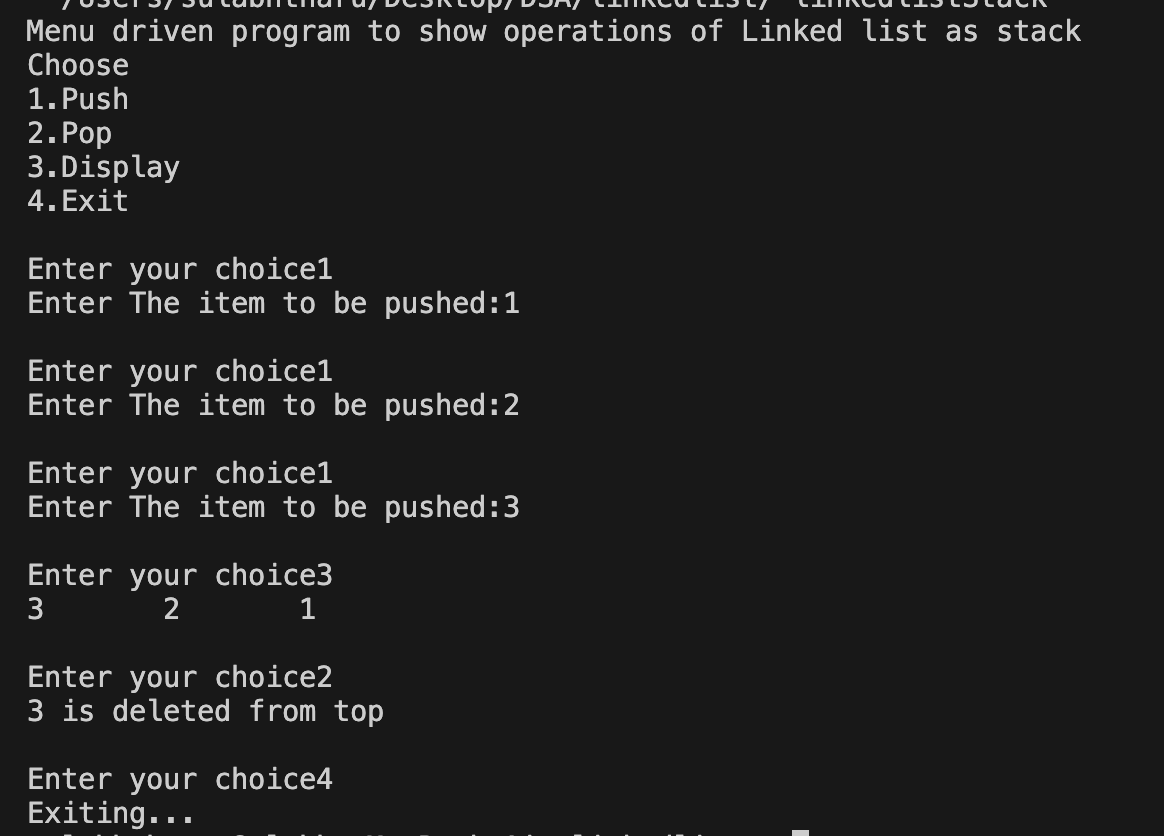
}

} while (choice!=4);

}

**OUTPUT:**

**Linked list as Stack**

****

**LAB 6:**

//program to implement the operations of linked list in linear queue

#include<stdio.h>

#include<stdlib.h>

struct queuelist

{

int info;

struct queuelist \*next;

};

struct queuelist \*front=NULL;

struct queuelist \*rear=NULL;

void enqueue()

{

int item;

struct queuelist \*Newnode;

Newnode=(struct queuelist\*)malloc(sizeof(struct queuelist));

printf("Enter the item to be inserted:");

scanf("%d",&item);

Newnode->info=item;

if (front==NULL)

{

Newnode->next=NULL;

rear=front=Newnode;

}

else

{

Newnode->next=NULL;

rear->next=Newnode;

rear=Newnode;

}

}

void dequeue()

{

struct queuelist \*temp;

temp=front;

if (front==NULL)

{

printf("The queue list is Empty\n");

}

else if (front==rear)

{

front=rear=NULL;

printf("%d item is deleted from front\n",temp->info);

free(temp);

}

else{

front=front->next;

printf("%d item is deleted from front\n",temp->info);

free(temp);

}

}

void display()

{

struct queuelist \*temp;

temp=front;

if (front==NULL)

{

printf("The queue list is empty\n");

}

else{

while (temp!=rear)

{

printf("%d\t",temp->info);

temp=temp->next;

}

printf("%d\n",rear->info);

}

}

int main()

{

int choice;

printf("Menu driven program to show operations of linked list as queue\n");

printf("choose\n1.Enqueue\n2.Dequeue\n3.Show\n4.Exit\n");

do

{

printf("\nEnter your choice:");

scanf("%d",&choice);

switch (choice)

{

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

printf("Exiting...\n");

break;

default:

printf("Invaid Choice!!\n");

break;

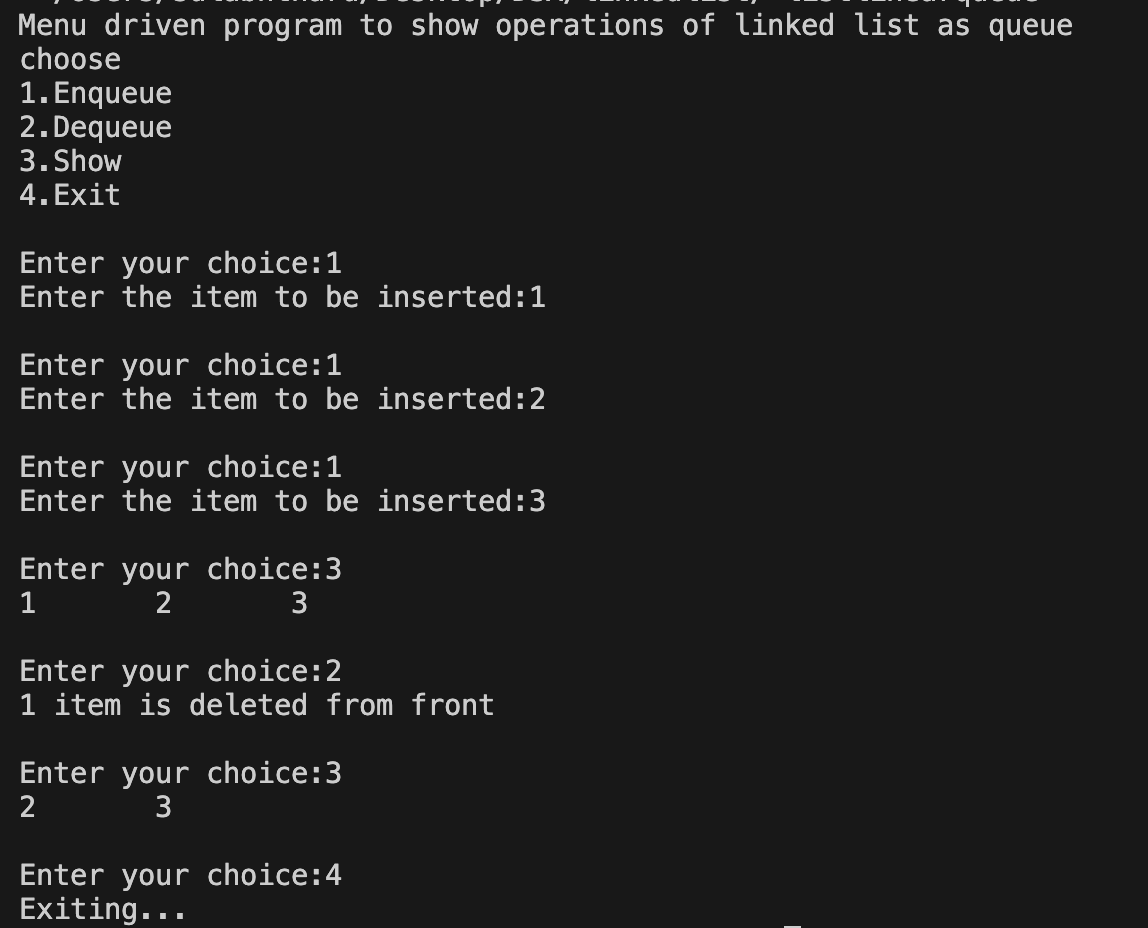
}

} while (choice!=4);

}

**OUTPUT:**

**Linked list as Queue**

****

**LAB 7:**

//program for bubble sort

#include<stdio.h>

void bubble\_sort(int a[],int n)

{

int temp;

for (int i = 0; i < n-1; i++)

{

for (int j = 0; j < n-i-1; j++)

{

if (a[j]>a[j+1])

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

}

}

int main()

{

int n;

int a[100];

printf("Enter the number of elements in array:");

scanf("%d",&n);

printf("Enter the elements of the array:\n");

for (int i = 0; i < n; i++)

{

scanf("%d",&a[i]);

}

printf("The elements of array before sorting are:\n");

for (int i = 0; i < n; i++)

{

printf("%d\t",a[i]);

}

printf("\n");

printf("Sorting elements of array\n");

bubble\_sort(a,n);

printf("The elements of array after sorting are:\n");

for (int i = 0; i < n; i++)

{

printf("%d\t",a[i]);

}

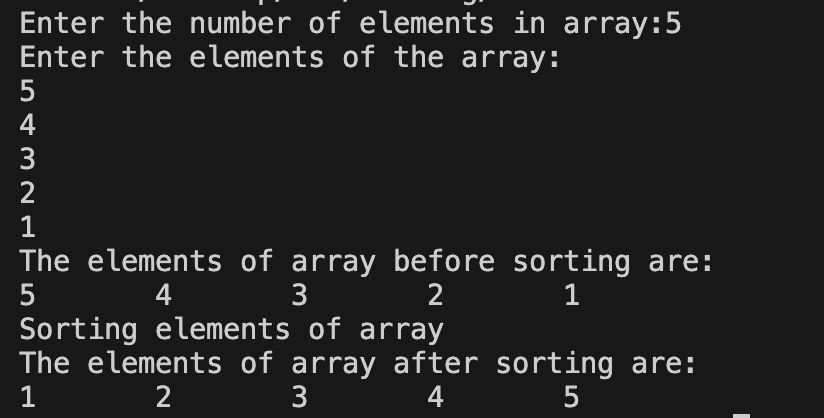
printf("\n");

return 0;

}

**OUTPUT:**

**Bubble Sort**

****

**LAB 8:**

//program to implement quick sort

#include<stdio.h>

int partition(int a[],int l, int r)

{

int low=l;

int high=r;

int pivot=a[l],temp;

while (low<high)

{

while (a[low]<=pivot)

{

low++;

}

while (a[high]>pivot)

{

high--;

}

if (low<high)

{

temp=a[low];

a[low]=a[high];

a[high]=temp;

}

}

a[l]=a[high];

a[high]=pivot;

return high;

}

void quicksort(int a[],int l, int r)

{

int p;

if (l<r)

{

p=partition(a,l,r);

quicksort(a,l,p-1);

quicksort(a,p+1,r);

}

}

void display(int a[],int n)

{

for (int i = 0; i < n; i++)

{

printf("%d\t",a[i]);

}

printf("\n");

}

int main()

{

int a[100],n,l,r;

printf("\*\*\*\*\*\*Quick Sort\*\*\*\*\*\*\nEnter the no. of elements:");

scanf("%d",&n);

printf("Enter the elements in array:\n");

for (int i = 0; i < n; i++)

{

scanf("%d",&a[i]);

}

printf("\nBefore sorting.....\n");

display(a,n);

printf("\n.....Using Quick Sort....\n ");

l=0;

r=n-1;

quicksort(a,l,r);

printf("\nAfter sorting......\n");

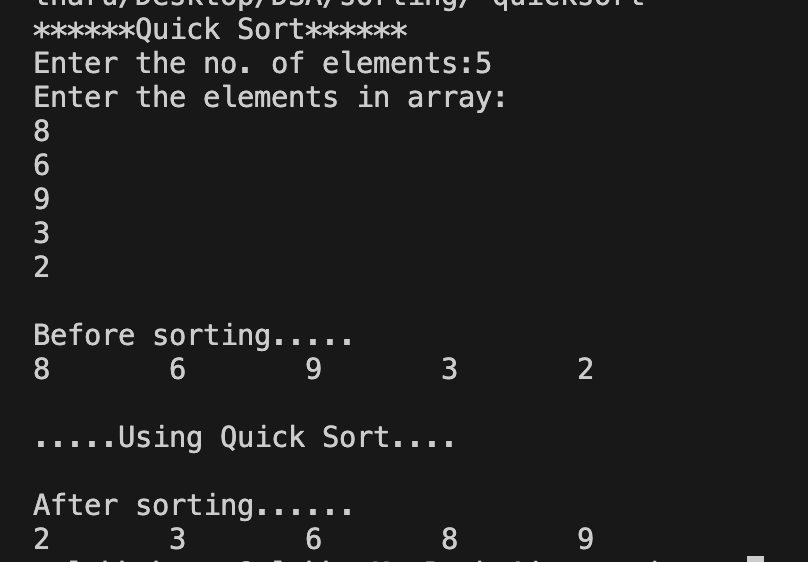
display(a,n);

return 0;

}

**OUTPUT:**

**Quick Sort**

****

**LAB 9:**

//program to implement merge sort

#include<stdio.h>

void merge(int a[],int l,int m,int r)

{

int n1=m-l+1;

int n2=r-m;

int left[n1],right[n2];

int k=l;

for (int i = 0; i < n1; i++)

{

left[i]=a[l+i];

}

for (int j = 0; j < n2; j++)

{

right[j]=a[m+1+j];

}

int i=0,j=0;

while (i<n1 && j<n2)

{

if (left[i]<=right[j])

{

a[k]=left[i];

i++;

k++;

}

else{

a[k]=right[j];

k++;

j++;

}

}

while (i<n1)

{

a[k]=left[i];

i++;

k++;

}

while (j<n2)

{

a[k]=right[j];

j++;

k++;

}

}

void merge\_sort(int a[],int l,int r)

{

if (l<r)

{

int mid=(l+r)/2;

merge\_sort(a,l,mid);

merge\_sort(a,mid+1,r);

merge(a,l,mid,r);

}

}

void display(int a[],int n)

{

for (int i = 0; i < n; i++)

{

printf("%d\t",a[i]);

}

printf("\n");

}

int main()

{

int a[100],n;

printf("\*\*\*\*\*\*Merge Sort\*\*\*\*\*\*\nEnter the number of elements:");

scanf("%d",&n);

printf("Enter the elements:\n");

for (int i = 0; i < n; i++)

{

scanf("%d",&a[i]);

}

printf("\nBefore Sorting...\n");

display(a,n);

printf("\nUsing Merge Sort....\n");

merge\_sort(a,0,n-1);

printf("\nAfter Sorting....\n");

display(a,n);

return 0;

}

**OUTPUT:**

**Merge Sort**

****

**LAB 10:**

//program to sort using shell sort

#include<stdio.h>

void shell\_sort(int a[],int n)

{

int gap,temp,i,j;

for (int gap = n/2; gap > 0; gap=gap/2)

{

for ( i = gap; i < n; i++)

{

temp = a[i];

for ( j = i; j >= gap && a[j-gap]>=temp; j=j-gap)

{

a[j]=a[j-gap];

}

a[j]=temp;

}

}

}

void display(int a[],int n)

{

for (int i = 0; i < n; i++)

{

printf("%d\t",a[i]);

}

printf("\n");

}

int main()

{

int a[100],n;

printf("\*\*\*\*\*\*Shell Sort\*\*\*\*\*\*\nEnter the number of elements:");

scanf("%d",&n);

printf("Enter the elements:\n");

for (int i = 0; i < n; i++)

{

scanf("%d",&a[i]);

}

printf("Before Sorting....\n");

display(a,n);

shell\_sort(a,n);

printf("After Sorting....\n");

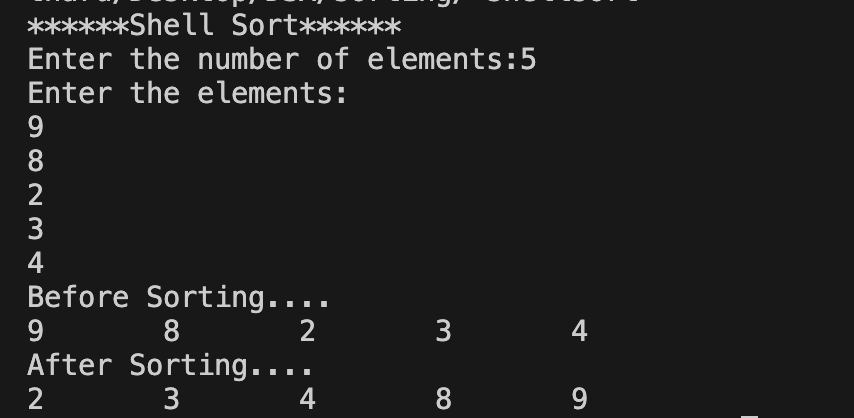
display(a,n);

return 0;

}

**OUTPUT:**

**Shell Sort**

****