DSA

DIGITAL ASSESSSMENT

NAME: TISHA BANSAL

REG NO: 24BBS0142

1)Write a menu driven program to implement the following operations on stack. a. PUSH() b. POP() c. Display()

Pseudo Code:

```
Initialize queue with size MAX

Set front = -1, rear = -1

Function Enqueue():
  If rear == MAX - 1:
    Print "Queue is Full!"
  Else:
    If front == -1:
      Set front = 0
    Print "Enter value to enqueue:"
    Read value
    Increment rear
    Set queue[rear] = value


Function Dequeue():
  If front == -1 or front > rear:
    Print "Queue is Empty!"
  Else:
    Print queue[front], "dequeued"
    Increment front


Function Display():
  If front == -1 or front > rear:
    Print "Queue is Empty!"
  Else:
    For i from front to rear:
```

Print queue[i]

Print a new line

Main:

Call Enqueue()

Call Enqueue()

Call Dequeue()

Call Dequeue()

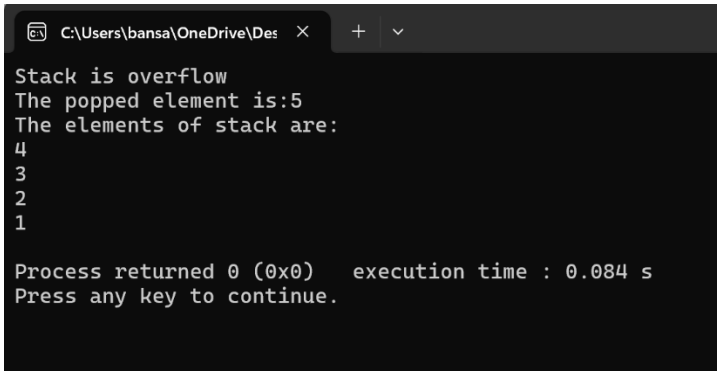
Call Display()

Program:

Testcase 1)

```
#include<stdio.h>
#define n 5
int top=-1;
int stack[n];
void push(int data){
if(top==n-1)
printf("Stack is overflow\n");
else{
top++;
stack[top]=data;
}
}
void pop(){
if(top==-1)
printf("Stack is underflow\n");
else
```

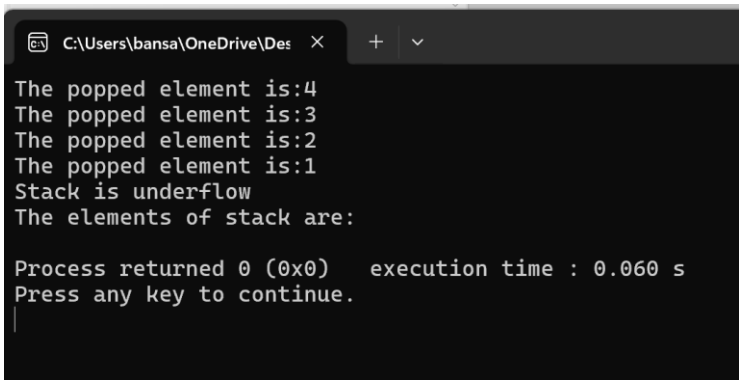
```
printf("The popped element is:%d\n",stack[top]);
 top--;
}
void display(){
 for(int i=top;i>=0;i--)
printf("%d\n",stack[i]);
 }
 int main()
 {
push(1);
 push(2);
push(3);
push(4);
 push(5);
 push(6);
pop();
display();
 return 0;
}
```



Testcase 2)

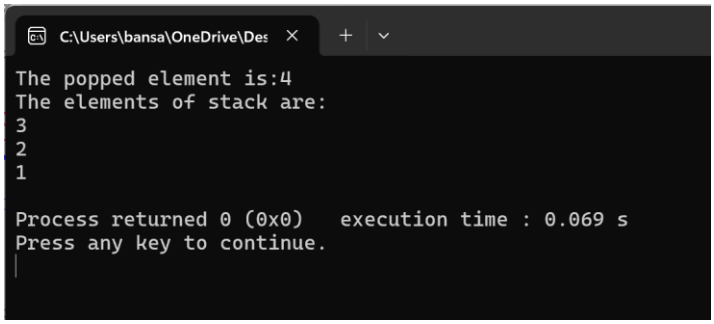
```
int main()
{
```

```
push(1);
push(2);
push(3);
push(4);
pop();
pop();
pop();
pop();
pop();
display();
return 0;
}
```



Testcase 3)

```
int main()
{
push(1);
push(2);
push(3);
push(4);
pop();
display();
return 0; }
```



2) Write a menu driven program to implement the following operations on Queue: a. Enqueue() b. Dequeue() c. Display()

Pseudo Code:

Initialize queue with size MAX

Set front = -1, rear = -1

Function Enqueue():

  If rear == MAX - 1:

    Print "Queue is Full!"

  Else:

    If front == -1:

      Set front = 0

    Print "Enter value to enqueue:"

    Read value

    Increment rear

    Set queue[rear] = value

Function Dequeue():

  If front == -1 or front > rear:

    Print "Queue is Empty!"

  Else:

    Print queue[front], "dequeued"

    Increment front

```
Function Display():
  If front == -1 or front > rear:
    Print "Queue is Empty!"
  Else:
    For i from front to rear:
      Print queue[i]
    Print a new line
Main:
  Call Enqueue()
  Call Enqueue()
  Call Dequeue()
  Call Dequeue()
  Call Display()
```

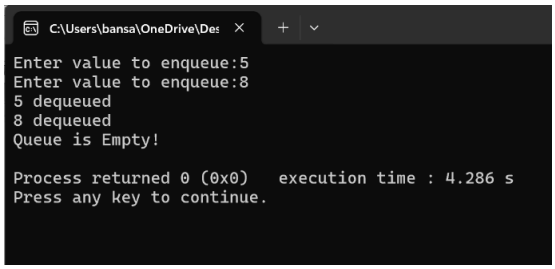
Program:

Testcase 1)

```
#include <stdio.h>
#define MAX 5

int queue[MAX];
int front = -1, rear = -1;

void Enqueue() {
  int value;
  if (rear == MAX - 1) {
    printf("Queue is Full!\n");
  } else {
```

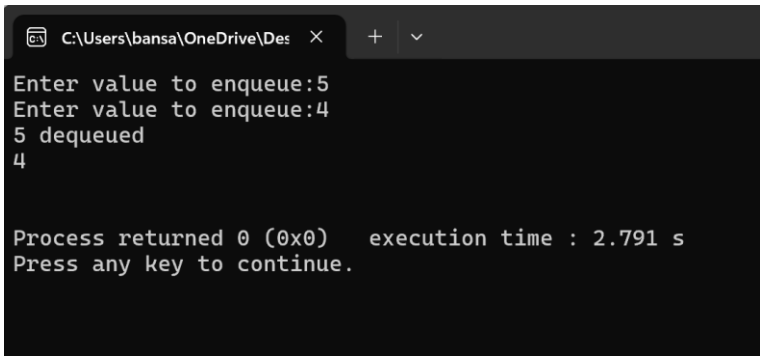
```
        if (front == -1) front = 0;
        printf("Enter value to enqueue:");
        scanf("%d", &value);
        rear++;
        queue[rear] = value;
    }
}

void Dequeue() {
    if (front == -1 || front > rear) {
        printf("Queue is Empty!\n");
    } else {
        printf("%d dequeued\n", queue[front]);
        front++;
    }
}

void Display() {
    if (front == -1 || front > rear) {
        printf("Queue is Empty!\n");
    } else {
        for (int i = front; i <= rear; i++) {
            printf("%d\n", queue[i]);
        }
        printf("\n");
    }
}
```

```
int main() {
   Enqueue();
   Enqueue();
   Dequeue();
   Dequeue();
   Display();
   return 0;
}
```



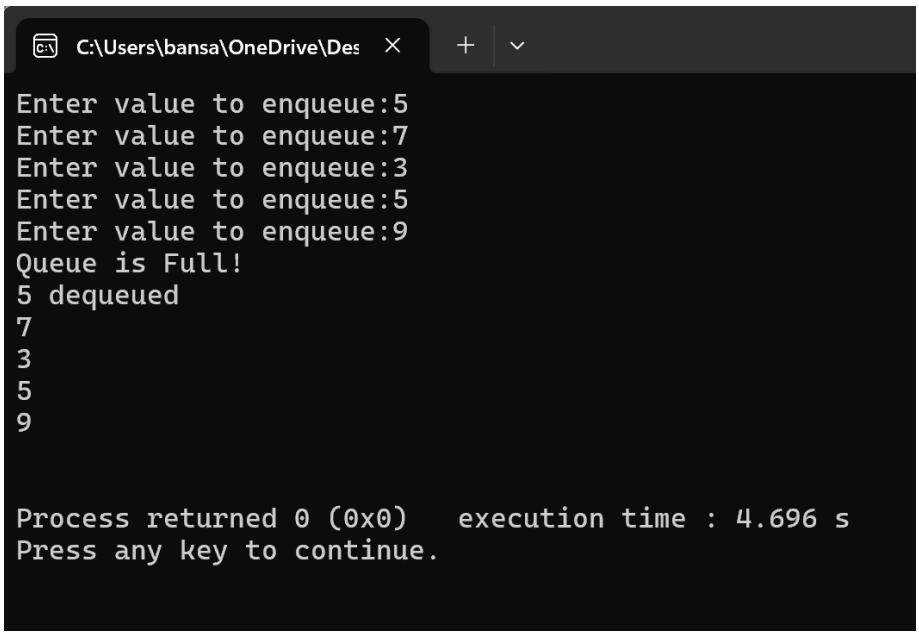
Testcase 2)

```
int main() {
   Enqueue();
   Enqueue();
   Dequeue();
   Display();

   return 0;
}
```



Testcase 3)

```
int main() {
   Enqueue();
   Enqueue();
   Enqueue();
   Enqueue();
   Enqueue();
   Enqueue();
   Dequeue();
   Display();

   return 0;
}
```



3) Write a menu driven program to implement the following operations on circular Queue: a. Enqueue() b. Dequeue() c. Display()

Pseudo code:

```
Initialize queue with size MAX

Set front = -1, rear = -1


Function Enqueue(value):
  If (rear + 1) % MAX == front:
    Print "Queue is Full!"
  Else:
    If front == -1:
      Set front = 0
    Increment rear circularly: rear = (rear + 1) % MAX
    Set queue[rear] = value
    Print "Enqueued value:", value


Function Dequeue():
  If front == -1:
    Print "Queue is Empty!"
  Else:
    Print "Dequeued value:", queue[front]
    If front == rear:
      Set front = rear = -1
    Else:
      Increment front circularly: front = (front + 1) % MAX


Function Display():
  If front == -1:
```

```
    Print "Queue is Empty!"
  Else:
    Set i = front
    While i != rear:
      Print queue[i]
      Increment i circularly: i = (i + 1) % MAX
    Print queue[rear]

Main:
  Call Enqueue(10)
  Call Enqueue(20)
  Call Enqueue(30)
  Call Enqueue(40)
  Call Enqueue(50)
  Call Dequeue()
  Call Enqueue(60)
  Call Display()
```

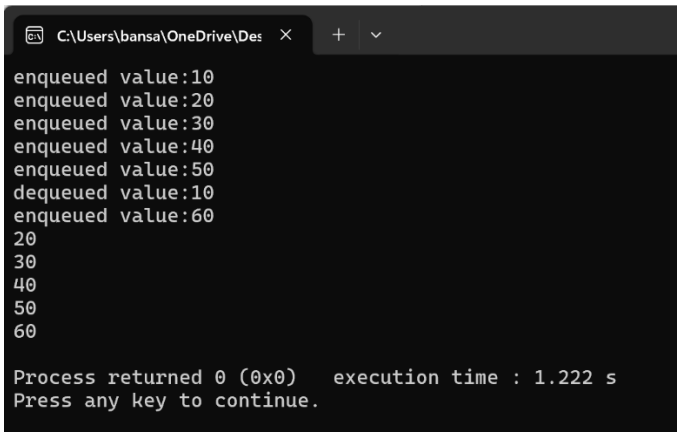
Program:

Testcase 1)

```
#include <stdio.h>
#define MAX 5

int queue[MAX];
int front = -1, rear = -1;

void Enqueue(int value) {
  if ((rear + 1) % MAX == front) {
```

```
        printf("Queue is Full!\n");
    } else {
        if (front == -1) front = 0;
        rear = (rear + 1) % MAX;
        queue[rear] = value;
        printf("enqueued value:%d\n", value);
    }
}

void Dequeue() {
    if (front == -1) {
        printf("Queue is Empty!\n");
    } else {
        printf("dequeued value:%d\n", queue[front]);
        if (front == rear) {
            front = rear = -1;
        } else {
            front = (front + 1) % MAX;
        }
    }
}

void Display() {
    if (front == -1) {
        printf("Queue is Empty!\n");
    } else {
        int i = front;
        while (i != rear) {
```

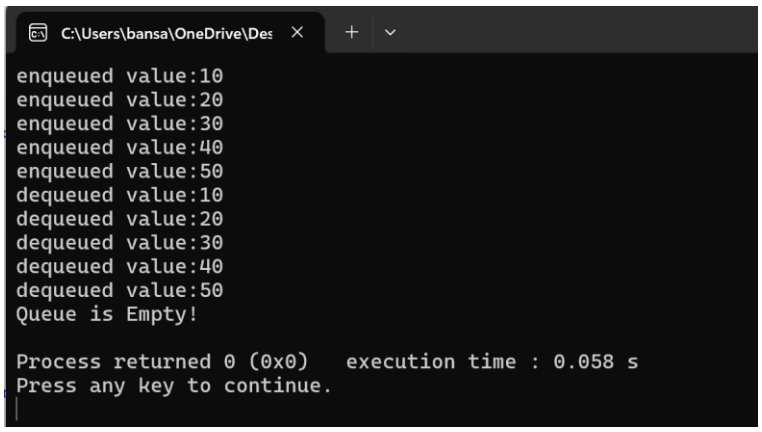
```
        printf("%d\n", queue[i]);
        i = (i + 1) % MAX;
      }
      printf("%d\n", queue[rear]);
   }
}

int main() {
   Enqueue(10);
   Enqueue(20);
   Enqueue(30);
   Enqueue(40);
   Enqueue(50);
   Dequeue();
   Enqueue(60);
   Display();
   return 0;
}
```



Testcase 2)

```
int main() {
   Enqueue(10);
```

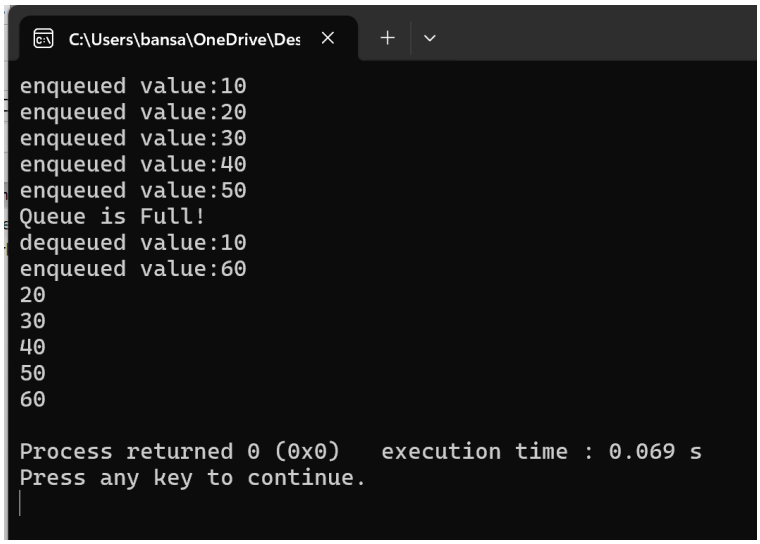
```
    Enqueue(20);
    Enqueue(30);
    Enqueue(40);
    Enqueue(50);
    Dequeue();
    Dequeue();
    Dequeue();
    Dequeue();
    Dequeue();
    Display();
    return 0;
}
```



Testcase 3)

```
int main() {
    Enqueue(10);
    Enqueue(20);
    Enqueue(30);
    Enqueue(40);
    Enqueue(50);
    Enqueue(60);
    Dequeue();
```

```
    Enqueue(60);
    Display();
    return 0;
}
```



4) Write a menu driven program to implement the following operations on singly linked list: a. Insertion() i. Beginning ii. End iii. At a given position b. Deletion() i. Beginning ii. End iii. At a given position c. Search(): search for the given element on the list

Pseudo Code:

```
Initialize queue with size MAX
Set front = -1, rear = -1

Function InsertBeginning(value):
  If rear == MAX - 1:
    Print "Queue is full"
  Else:
    If front == -1:
      Set front = 0
    For i from rear down to front:
      queue[i + 1] = queue[i]
    queue[front] = value
    Increment rear

Function InsertEnd(value):
  If rear == MAX - 1:
    Print "Queue is full"
  Else:
    If front == -1:
      Set front = 0
    Increment rear
    queue[rear] = value
```

```
Function DeleteBeginning():
  If front == -1 or front > rear:
    Print "Queue is empty"
  Else:
    Print "Deleted value:", queue[front]
    Increment front

Function DeleteEnd():
  If front == -1 or front > rear:
    Print "Queue is empty"
  Else:
    Print "Deleted value:", queue[rear]
    Decrement rear

Function Display():
  If front == -1 or front > rear:
    Print "Queue is empty"
  Else:
    For i from front to rear:
      Print queue[i]

Main:
  Call InsertBeginning(10)
  Call InsertEnd(20)
  Call Display()
  Call DeleteBeginning()
  Call Display()
  Call InsertEnd(30)
```

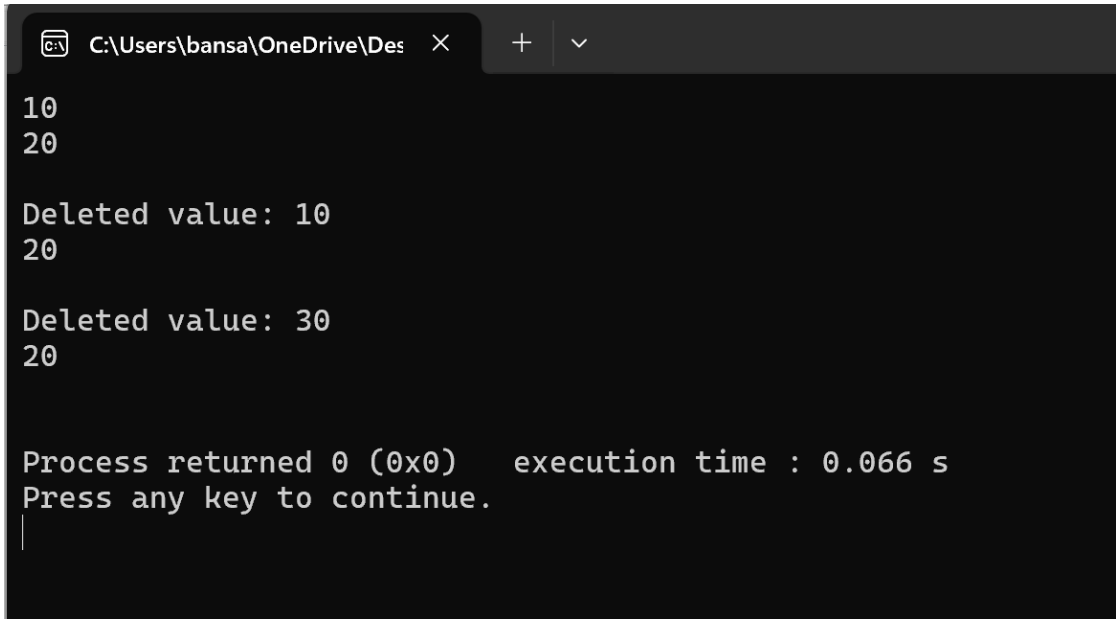
```
   Call DeleteEnd()
   Call Display()
```

Program:

```
#include <stdio.h>
#define MAX 5
int queue[MAX];
int front = -1, rear = -1;

// Insert at the beginning (simulated)
void InsertBeginning(int value) {
   if (rear == MAX - 1) {
      printf("Queue is full\n");
   } else {
      if (front == -1) front = 0;
      for (int i = rear; i >= front; i--) {
         queue[i + 1] = queue[i];
      }
      queue[front] = value;
      rear++;
   }
}

// Insert at the end (simulated)
void InsertEnd(int value) {
   if (rear == MAX - 1) {
      printf("Queue is full\n");
   } else {
```

```
        if (front == -1) front = 0;
        queue[rear + 1] = value;
        rear++;
    }
}

// Delete at the beginning (simulated)
void DeleteBeginning() {
    if (front == -1 || front > rear) {
        printf("Queue is empty\n");
    } else {
        printf("Deleted value: %d\n", queue[front]);
        front++;
    }
}

// Delete at the end (simulated)
void DeleteEnd() {
    if (front == -1 || front > rear) {
        printf("Queue is empty\n");
    } else {
        printf("Deleted value: %d\n", queue[rear]);
        rear--;
    }
}

// Display list
void Display() {
```

```
    if (front == -1 || front > rear) {
        printf("Queue is empty\n");
    } else {
        for (int i = front; i <= rear; i++) {
            printf("%d\n", queue[i]);
        }
        printf("\n");
    }
}

int main() {
    InsertBeginning(10);
    InsertEnd(20);
    Display();

    DeleteBeginning();
    Display();

    InsertEnd(30);
    DeleteEnd();
    Display();

    return 0;
}
```


C:\Users\bansa\OneDrive\Des
10
20
Deleted value: 10
20
Deleted value: 30
20
Process returned 0 (0x0)   execution time : 0.066 s
Press any key to continue.

5) Write a menu driven program to implement the following operations on Doubly linked list: a. Insertion() i. Beginning ii. End iii. At a given position b. Deletion() i. Beginning ii. End iii. At a given position c. Search(): search for the given element on the list

Pseudo Code:

```
Initialize list with MAX size
Set count = 0
Function Insert(value):
  If count is equal to MAX:
    Print "List is full"
    Return
  Set list[count][0] = value
  If count is 0:
    Set list[count][1] = -1
  Else:
    Set list[count][1] = count - 1
  Increment count

Function Delete():
  If count is 0:
    Print "List is empty"
    Return
  Decrement count

Function Display():
  If count is 0:
    Print "List is empty"
    Return
```

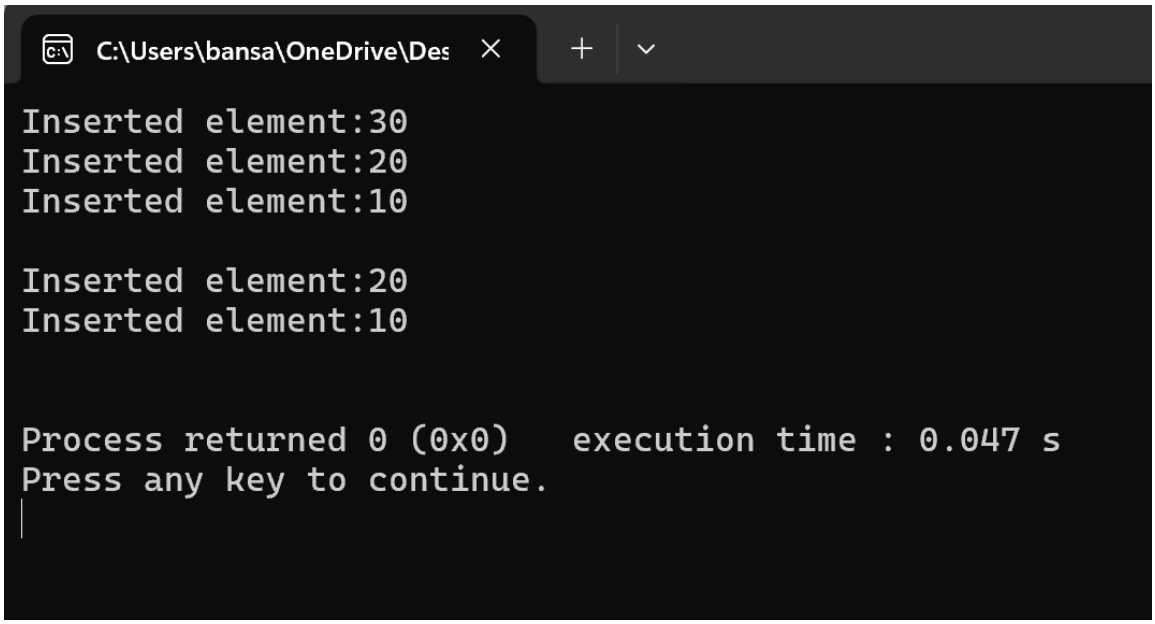
```
  For i from count - 1 down to 0:
    Print list[i][0]

Main:
  Call Insert(10)
  Call Insert(20)
  Call Insert(30)
  Call Display()
  Call Delete()
  Call Display()
```

Program:

```
#include <stdio.h>
#define MAX 5

int list[MAX][2];
int count = 0;

void Insert(int value) {
  if (count >= MAX) {
    printf("List is full\n");
    return;
  }
  list[count][0] = value;
  list[count][1] = (count == 0) ? -1 : count - 1;
  count++;
}
```

```
void Delete() {
   if (count == 0) {
      printf("List is empty\n");
      return;
   }
   count--;
}

void Display() {
   if (count == 0) {
      printf("List is empty\n");
      return;
   }
   for (int i = count - 1; i >= 0; i--) {
      printf("Inserted element:%d\n", list[i][0]);
   }
   printf("\n");
}

int main() {
   Insert(10);
   Insert(20);
   Insert(30);
   Display();
   Delete();
   Display();

   return 0;
```

}


C:\Users\bansa\OneDrive\Des
Inserted element:30
Inserted element:20
Inserted element:10
Inserted element:20
Inserted element:10
Process returned 0 (0x0) execution time : 0.047 s
Press any key to continue.