NAME: AYUSH PRASHANT KALASKAR
CLASS: S.E. COMP-I
ROLL NO.: 69 PRN.: F22111074
SEMESTER: SEM-III (2023-2024)
SUBJECT: DATA STRUCTURE LABORATORY (DSL)

ASSIGNMENT: $\boxed{\text{E-31}}$:-

□ ALGORITHM FOR THE PROGRAM CODE.

STEP:1:- START

STEP:2:- Create a class with class_name 'Dequeue' having private datamembers : an integer array 'a' to store dequeue elements.

STEP:3:- Also, declare 'Front' and 'rear' to track the front and rear indices, and 'count' to keep track of the number of elements.

STEP:4:- Implement a default constructor to intialize 'Front,' 'rear' and 'count' to -1.

STEP:5:- Implement the 'add Begin(item: Integer) method to add an element to the Front of the double-ended queue (deque). if the deque is empty, intialize Front and rear and add the element. If the deque is not empty and not full, shift elements to make space for the new item and add it.

STEP:6:- Implement the 'add End (item: Integer)' method to add an element to the end of the deque.. If the deque is empty, intialize 'Front' and 'rear' and add the Element. If the deque is not empty and not Full, simply add the element to the rear.

STEP:7:- Implement the 'delete Front()' method to remove an element from the Front of the deque. check if the deque is empty and display an error message.

If not, delete the element from the front and adjust the 'front' index. If the deque becomes empty, reset front and rear.

STEP: 8:- Implement the 'delete End()' method to remove an element from the end of the deque. Check if the deque is empty and display an error message. If not delete the element from the end and adjust the 'rear' index. If the deque becomes empty, reset 'front' and 'rear'

STEP: 9:- Implement the 'display()' method to display the elements of the deque from front to rear

STEP: 10:- In the 'main' function create an instance (object) of the class 'Dequeve' named as 'd1' and use a 'do-while' loop to display a menu for various operations

- 1. Insert an Element at the beginning.
- 2. Insert an element at the End.
- 3. Display the ~~Dequeve~~.
- 4. Delete an Element from the front.
- 5. Delete an Element from the End.
- 6. Exit.

STEP: 11:- Based on the users choice call the corresponding methods of the 'Dequeve' class to perform the desired Operation.

STEP: 12:- Repeat the loop until the user chooses to Exit.

STEP: 13:- Stop.

D PSEUDOCODE :-

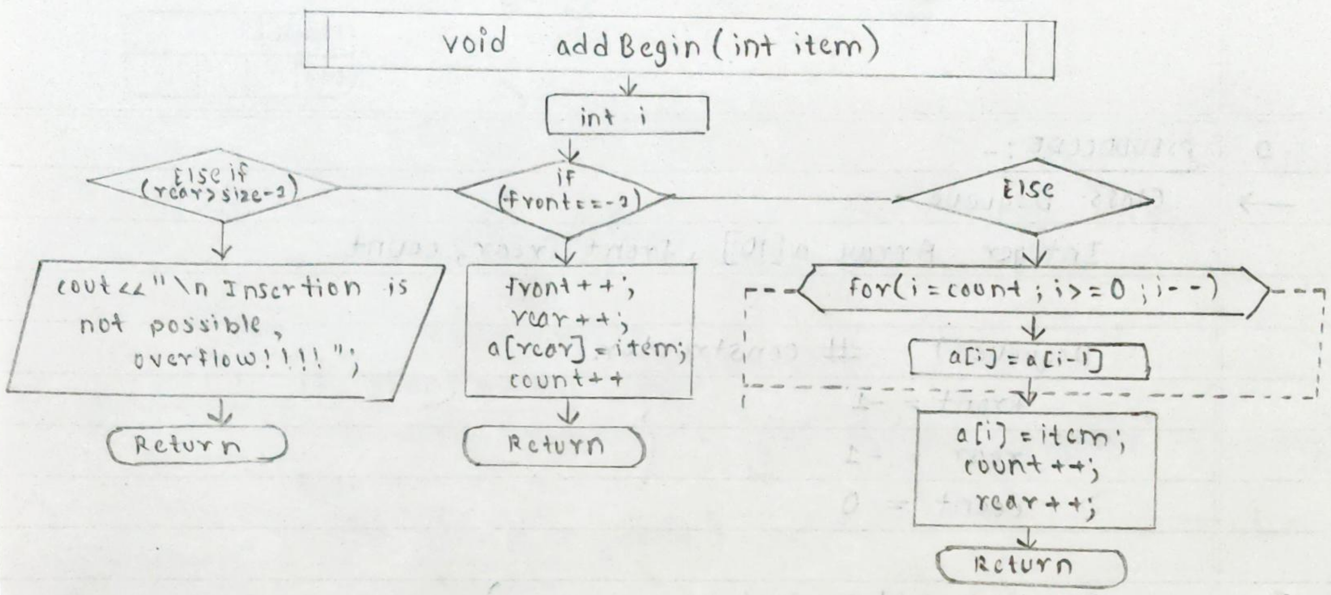
```
→ Class Dequeue
    Integer Array a[10] , front, rear, count

    Dequeue()      # constructor.
        Front = -1
        rear  = -1
        count = 0

    Function add Begin (item: Integer)
        Integer i
        If front == -1  then
            front ++
            rear ++
            a[rear] = item
            count = count + 1
        Else if rear >= (SIZE-1)  then
            Display " Insertion is not possible, overflow !!!!"
        Else
            for (i = count ; i > 0 ; i --)
                a[i] = a[i-1]
            a[i] = item
            count ++
            reor ++

    Function add End ( item: Integer)
        If Front == -1  then
            Front ++
            rear ++
            a[rear] = item
            count ++
        Else if rear >= (SIZE -1) then
```

Page No.
Date

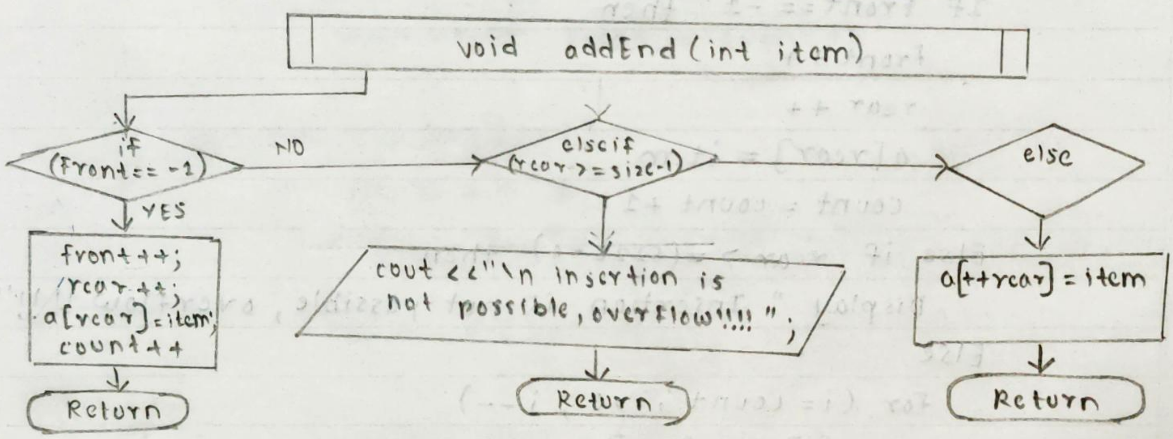
```
        Display " Insertion is not possible , overflow !!! "
    Else
        a[++rear] = item

Function deleteFront()
    if front == -1 then
        Display " Deletion is not possible : Dequeue is empty "
    Else
        if (front = rear) then
            front = -1
            rear = -1
        Display "The Deleted Element" << a[~~rear~~ front]
        ~~rear = rear~~ front = front +1

Function deleteEnd()
    if front == -1
        Display " Deletion is not possible : Dequeue is Empty "
    Else
        If front = rear ~~then~~
            front = -1
            rear = -1
        Display " The Deleted Element " << a[rear
        rear = rear - 1.

Function Display
    For (int i = front ; i <= rear ; i++)
        Display a[i] + "   "

Function main()
    Integer c , item
    Dequeue d1
    do
```

```
Display " ***** DEQUEUE OPERATION ***** "
Display " 1 - Insert at Beginning "
Display " 2 - Insert at End. "
Display " 3 - Display."
Display " 4 - Deletion from Front."
Display " 5 - Deletion from End."
Display " 6 - EXIT. "
Display " Enter your choice <1-6> :"
Read c.

Switch c
    Case 1:
        Display " Enter the Element to be inserted : "
        Read item
        Call d1.addBegin(item)
    Case 2:
        Display " Enter the Element to be inserted : "
        Read item
        Call d1.addEnd(item)
    Case 3:
        Call d1.display()
    case 4:
        Call d1.deleteFront()
    Case 5:
        Call d1.deleteEnd()
    Case 6:
        Exit the Program
    Default:
        Display" Invalid Choice"
while (c! =7)
Return 0
```

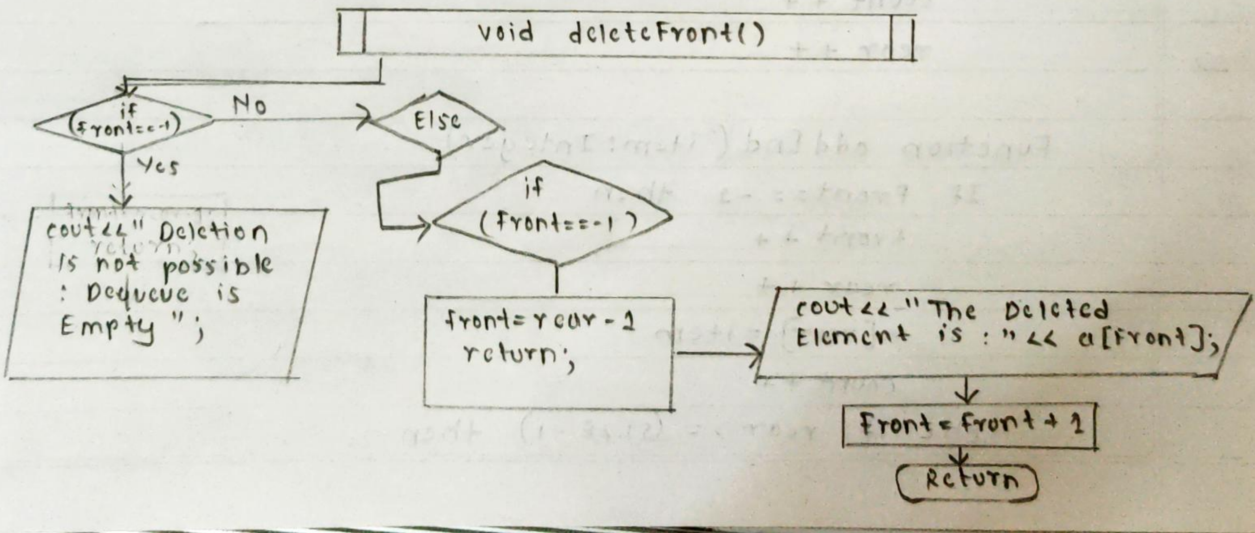
(1.) FLOWCHART FOR THE FUNCTION : addBegin (int item)



(2.) FLOWCHART FOR THE FUNCTION:- addEnd (int item)

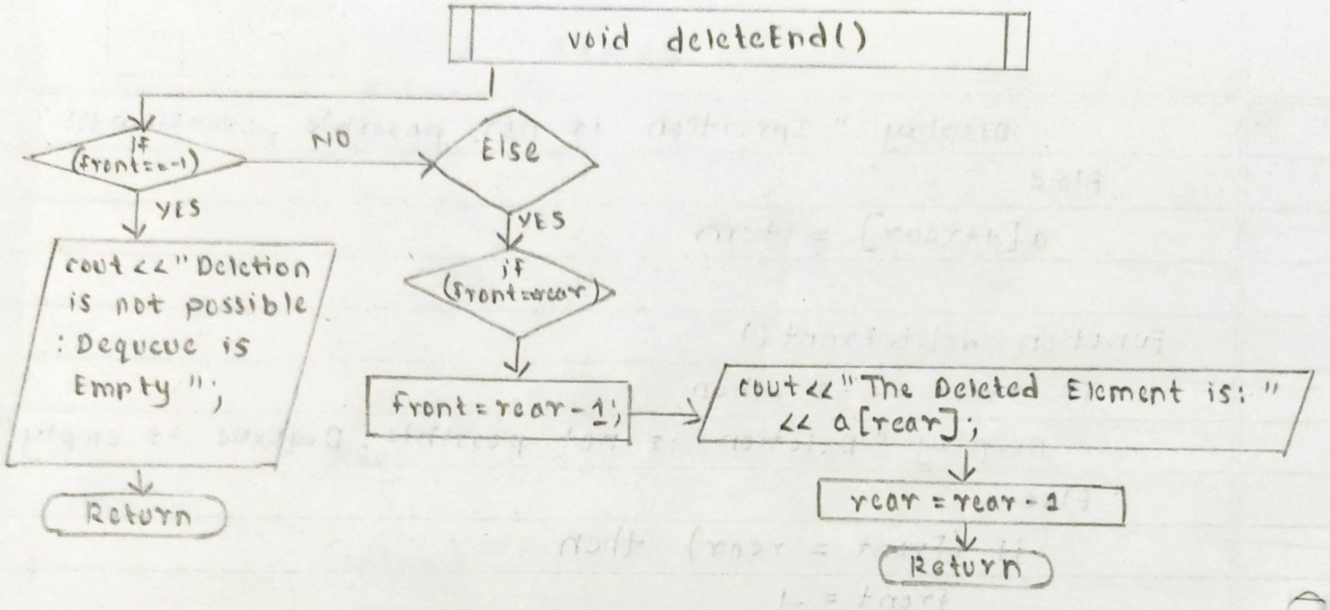


(3.) FLOWCHART FOR THE FUNCTION:- deleteFront ()

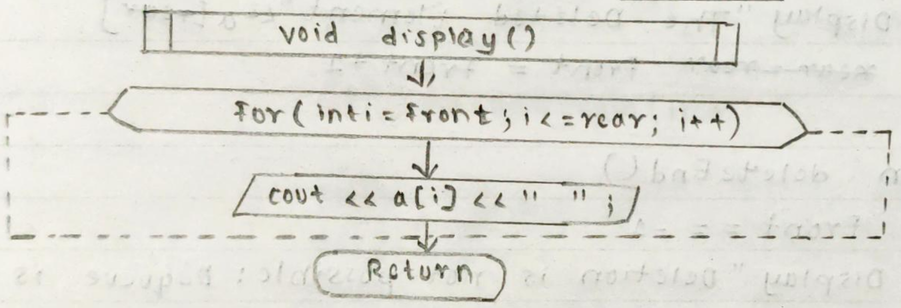


void deleteFront()
if (front==-1)
No
Else
Yes
cout<<" Deletion is not possible : Dequeue is Empty ";
if (front==-1)
front=rear-1
return;
cout<<" The Deleted Element is : "<< a[front];
Front = front + 1
Return

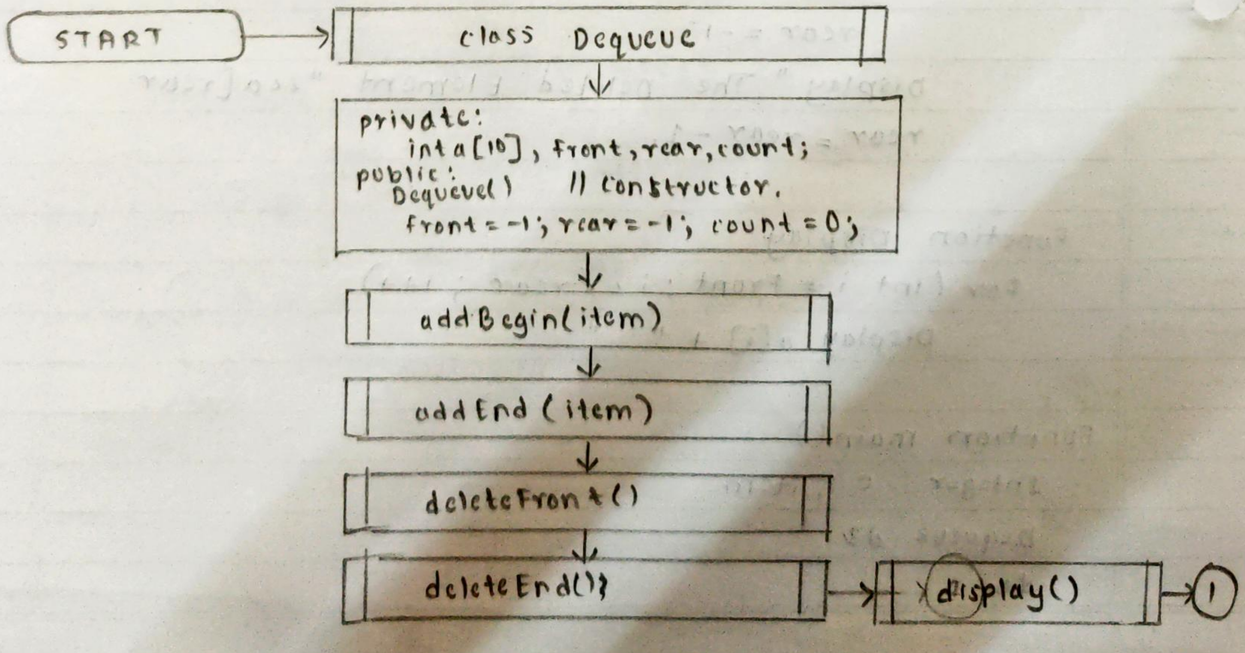
(4) FLOWCHART FOR THE FUNCTION :- deleteEnd()

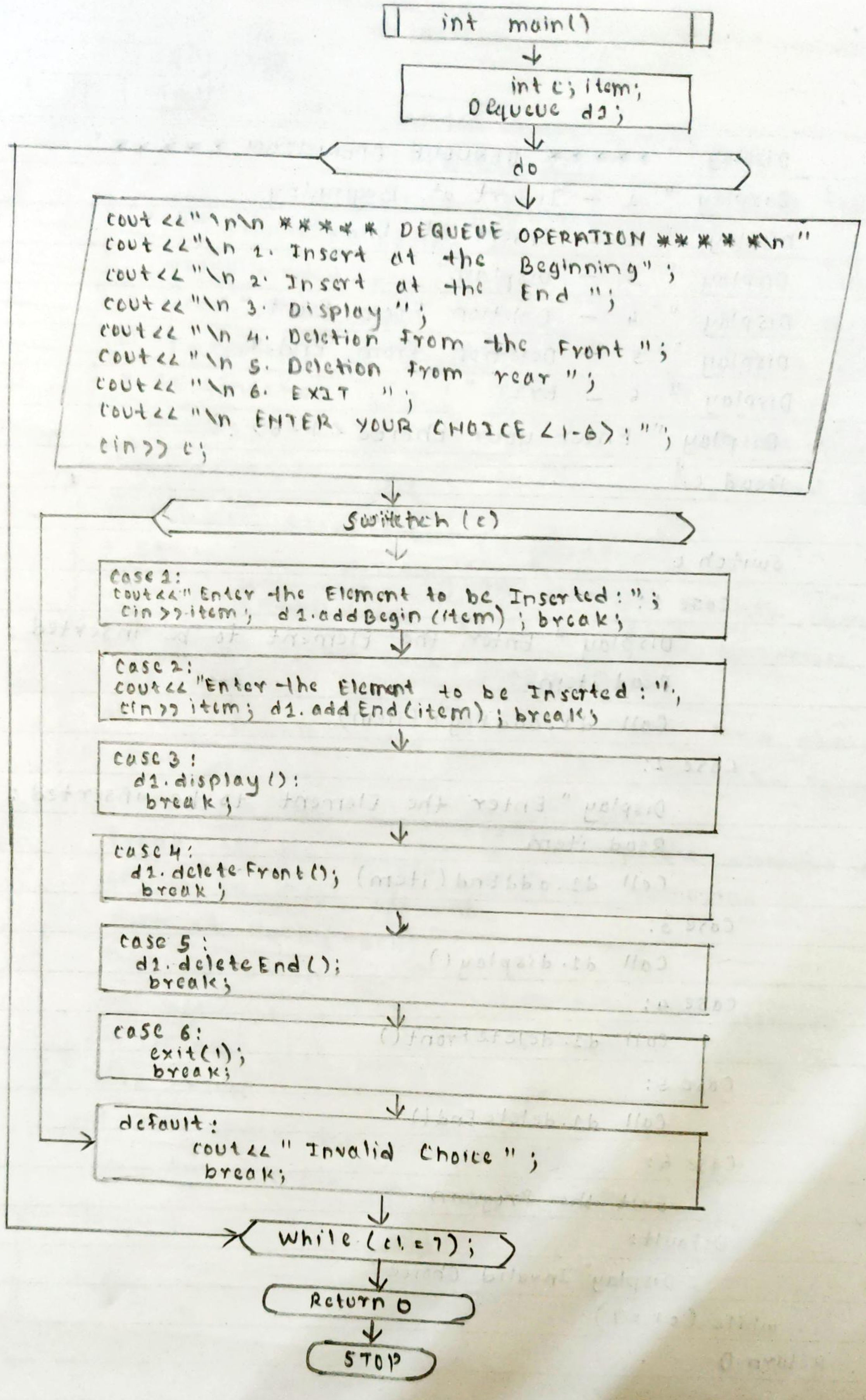

void deleteEnd()
if (front==-1)
NO
Else
YES
cout<<"Deletion is not possible : Dequeue is Empty";
Return
YES
if (front==rear)
front=rear-1;
cout<<"The Deleted Element is: " << a[rear];
rear=rear-1
Return

(5) FLOWCHART FOR THE FUNCTION :- display()



(6) FLOWCHART FOR THE MAIN PROGRAM :-





```
int main()

int c; item;
Dequeue d1;

do

cout<<"\n\n ***** DEQUEUE OPERATION *****\n"
cout<<"\n 1. Insert at the Beginning";
cout<<"\n 2. Insert at the End ";
cout<<"\n 3. Display";
cout<<"\n 4. Deletion from the Front ";
cout<<"\n 5. Deletion from rear ";
cout<<"\n 6. EXIT ";
cout<<"\n ENTER YOUR CHOICE <1-6>: "";
cin>> c;

Swititch (c)

case 1:
cout<<"Enter the Element to be Inserted: ";
cin>>item; d1.addBegin(item); break;

case 2:
cout<<"Enter the Element to be Inserted: ",
cin>> item; d1.add End(item); break;

case 3:
  d1.display():
  break;

case 4:
  d1.deleteFront();
  break;

case 5:
  d1.deleteEnd();
  break;

case 6:
  exit(1);
  break;

default:
      cout<<" Invalid Choice ";
      break;

While (c!=7);

Return 0

STOP
```

ASSIGNMENT: $\boxed{E-31}$:-

D QUESTIONS :-

Q1.) Describe Double-ended Queue Operations.

ANS. The Double-Ended Queue Operations are as follows:-

(1.) Insertion at Front (enqueue Front)

- It involves adding the element at the front of double-ended queue and can be performed in constant time, O(1), without shifting of existing elements.

(2.) Insertion at Back (enqueue Back)

- It involves adding an element at the back of the double-ended queue and can be performed in constant time, O(1).

(3.) Deletion from front (dequeue Front)

- It involves deleting an element at the front of the double ended queue. It is an O(1) operation.

(4.) Deletion from Back (dequeue Back)

- It involves removing an element from the back of the Double-ended queue. It is an O(1) Operation.

(5.) Peek at Front (front)

- This operation involves examining the element at the front of the dequeue without removing it.
- It is an O(1) operation

(6.) Peek at Back (back)

- This Operation involves examining the element at the back of the dequeue without removing it.
- It is an O(1) operation.

(7.) Size check

- We can check the size or number of elements in the double ended queue using this operation.

Q2.] How can we process one-dimensional array using double ended queue?

ANS. Common tasks to perocess a 1-D array using double ended queue are :-

1. Sliding Window problems.
2. Efficient Element Removal.
3. Queue and stack operations.
4. Circular Array Operations.

For Eg:- Python program.

```
from collections import deque

# Somple 1-D Array.
arr = [1, 2, 3, 4, 5, 6, 7]
d = deque()   # Intializing double ended queue.
for element in arr:   # Push Elemments to the
    d.append(element)   # back of the deque.

# Process elements from the front of double ended queue
while d:
    front_element = d.popleft()
    # Perform some processing on front_element

# Process elements from the back of the double
# ended queue
for element in reversed(d):
    back_element = element
    # Perform some processing on back element.
```

Q3.] What are the Advantages of Double-Ended Queue over single/simple Queue?

ANS. The Key Advantages of using a Double ended queue over a simple queue are:-

1. Bidirectional Insertion and Deletion (Removal) of elements.
2. Efficient Stack and Queue Operations
3. Double ended Queues are particularly used for solving sliding window problems.
4. They can simulate circular array operations, such as rotation efficiently.
5. Double-ended queue allows efficient removal of elements from both front and back ends in constant time
6. Useful in implementing Algorithms like Breadth-First search (BFS) and Depth-First search (DFS)
7. Useful to efficiently maintain the minimum or maximum elements in the window, which is challenging to do with a simple queue.
8. It is Dynamic Data structure allowing to handle varying workloads.
9. It ~~is~~ provides simpler i.e. cleaner and more readable code than simple queue.