**Name of Student : Anushka Ganeshsing Rajput**

**Expt. No.: 03 Roll. No.: 60 Batch: B1**

**Title of Experiment: WAP to implement of double ended queue menu driven program.**

**Date of Performance: Date of Submission:**

**Marks: Sign:**

**AIM:** Implementation of double ended queue menu driven program.

# THEORY:

A **deque**, also known as a double-ended queue, is an ordered collection of items similar to the queue. It has two ends, a front and a rear, and the items remain positioned in the collection. What makes a deque different is the unrestrictive nature of adding and removing items. New items can be added at either the front or the rear. Likewise, existing items can be removed from either end. In a sense, this hybrid linear structure provides all the capabilities of stacks and queues in a single data structure. Figure 1 shows a deque of data objects.

It is important to note that even though the deque can assume many of the characteristics of stacks and queues, it does not require the LIFO and FIFO orderings that are enforced by those data structures. It is up to you to make consistent use of the addition and removal operations.



Figure 1. Deque data object

Double Ended Queue is also a Queue data structure in which the insertion and deletion operations are performed at both the ends (**front** and **rear**). That means, we can insert at both front and rear positions and can delete from both front and rear positions.

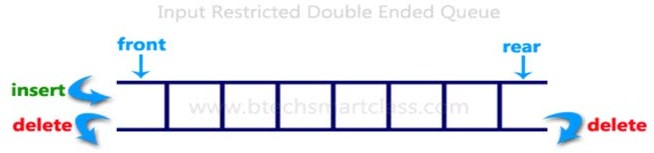


Double Ended Queue can be represented in TWO ways, those are as follows...

1. Input Restricted Double Ended Queue
2. Output Restricted Double Ended Queue

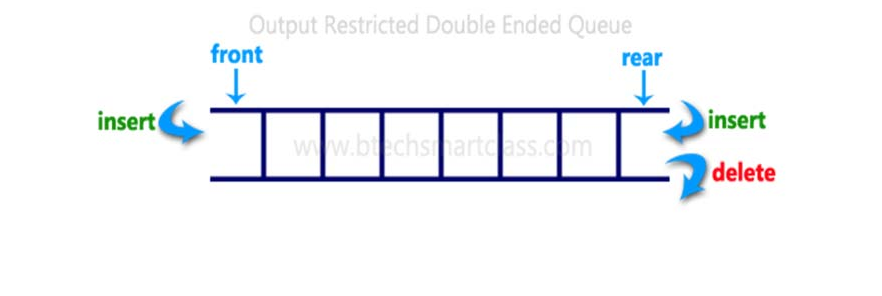
# Input Restricted Double Ended Queue

In input restricted double ended queue, the insertion operation is performed at only one end and deletion operation is performed at both the ends.



# Output Restricted Double Ended Queue

In output restricted double ended queue, the deletion operation is performed at only one end and insertion operation is performed at both the ends.



# CODE:

**#include <stdio.h> #include <conio.h> #include <stdlib.h> #define size 5**

# int main()

**{**

int arr[size], R = -1, F = 0, te = 0, ch, n, i, x;

for(;;) // An infinite loop

{

printf("F= %d R= %d\n\n", F, R); printf("1. Add Rear\n");

printf("2. Delete Rear\n"); printf("3. Add Front\n"); printf("4. Delete Front\n"); printf("5. Display\n"); printf("6. Exit\n"); printf("Enter Your Choice: "); scanf("%d", &ch);

switch(ch)

{

case 1:

if(te == size)

{

printf("Queue is full");

getch(); // pause the loop to see the message

}

else

{

printf("Enter a number "); scanf("%d",&n);

R = (R + 1) % size; arr[R] = n;

te = te+1;

}

break;

case 2:

if(te == 0)

{

printf("Queue is empty");

getch(); // pause the loop to see the message

}

else

{

if(R == -1)

{

R = size - 1;

}

printf("Number Deleted From Rear End = %d", arr[R]); R = R-1;

te = te-1;

getch(); // pause the loop to see the number

}

break;

case 3:

if(te==size)

{

printf("Queue is full");

getch(); // pause the loop to see the message

}

else

{

printf("Enter a number "); scanf("%d",&n);

if(F == 0)

{

F = size-1;

}

else

{

F = F-1;

}

arr[F] = n; te = te + 1;

}

break;

case 4:

if(te == 0)

{

printf("Queue is empty");

getch(); // pause the loop to see the message

}

else

{

printf("Number Deleted From Front End = %d",arr[F]); F = (F + 1) % size;

te = te - 1;

getch(); // pause the loop to see the number

}

break;

case 5:

if(te == 0)

{

printf("Queue is empty");

getch(); // pause the loop to see the message

}

else

{

x = F;

for(i = 1; i <= te; i++)

{

printf("%d ", arr[x]); x = (x + 1) % size;

}

getch(); // pause the loop to see the numbers

}

break;

case 6:

exit(0); break;

default:

printf("Wrong Choice Selected");

getch(); // pause the loop to see the message

}

}

return 0;

}

# OUTPUT:

# 

# CONCLUSION: