Experiment No: 5.

Aim:

A double-ended queue (deque) is a linear list in which addition and deletions may be made at either end. Obtain a data representation mapping a deque into a one-dimensional array. Write C++ program to stimulate deque with function to add & delete elements from ~~pre-requ~~ either end of deque.

Pre-requiste:

Knowledge of Queue.

Types of Queue.

Knowledge of double-ended queue & different operations that can be performed on it.

Objectives:

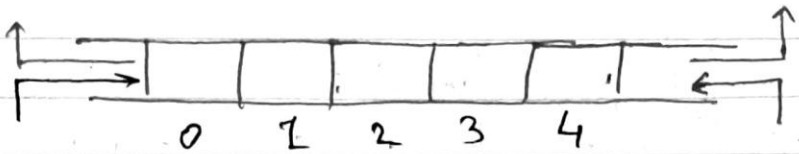
To stimulate deque with functions to add and delete elements from either end of deque.

Theory:

Double-ended deque.

Deque.

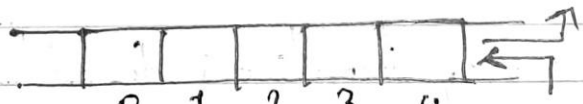
The deque stands for Double-ended deque. in this insertion takes place from one end while the deletion takes place from another end. The end at which the insertion occurs is known as rear end where as the end at which the deletion occurs is known as front end.



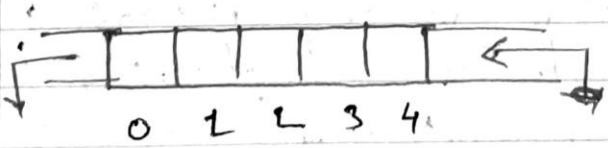
Deque is linear data structure in which the insertion and deletion operations are performed from both ends. We can say that deque is generalised version of ~~deque~~ queue.

Deque can used both as stack and queue as it allows all insertion & deletion operation on both ends.

In deque insertion and deletion operation can be performed from one side. The stacks flows the LIFO rule in which both the insertion & deletion can be performed only from one end. So, we conclude that deque can be considered as stack.



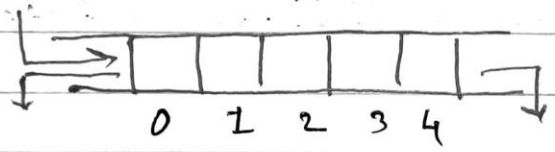
In deque insertion can perform on one end & deletion can be done on another end. The queue follows the FIFO rule in which element is inserted on one end and deletion from another end. So, we conclude that deque can also be considered as queue.



There are two types of Queues.

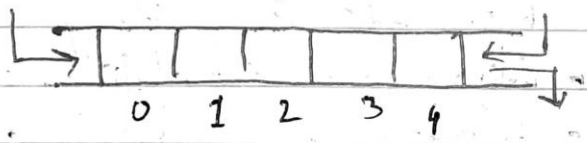
1] Input - restricted queue:

It means that some restrictions are applied to insertion, in this the insertion is applied to one end while the deletion is applied from both ends.



2] Output - restricted queue:

It means that some restrictions are applied to deletion operation, in this the deletion is applied only from one end, whereas insertion is possible from both ends.



Operations on Deque:

Following are operations applied on deque;

- Insert at front
- Delete from end
- Insert at rear
- Delete from rear

Other than insertion & deletion, we can also perform peek operation in deque. Through peek operation, we can get the front & rear element of dequeue.

We can perform two more operations on dequeue;

isfull() operation = This function returns true value if stack is full; otherwise, it return a false value.

isempty() = This function returns a true value if stack is empty; otherwise, it return a false value.

Applications of Deque:

The deque can be used as stack and queue, it can perform both redo and undo operations.

It can be used as palindrome checker means that if we read the string from both ends then string would be same.

It can be used for multiprocessor scheduling. Suppose we have two processors & each processor can have one process to execute. Each processor is assigned with a process or a job & each process contain multiple threads. Each processor maintain a deque that contains threads that are ready to execute. The processor excute a process & if process create a child process then that process will be inserted at front of the deque of parent process. Suppose the processor P_2 will take thread from front end to complete all the execution of threads then it steals from rear end of processor P_1 & add to front end of processor P_2 then P_2 will take thread from front end. So, deletion takes place from both ends i.e. front end & rear end. This is known as A-steal algorithm for scheduling.

Algorithm for insertion at rear end;

Step I: [check for overflow] if (rear == MAX)
Print ("Queue is overflow");
return;

Step II: [Insert element] else
rear = rear + 1;
q[rear] = no;
[set rear and front pointer]
if rear = 0
rear = 1; if front = 0
front = 1;

Step III: return.

Algorithm for insertion at front end:

Step I: [check for front position] if (front <= 1)
Print ("Can't add item at front end");
return;

Step II: [Insert at front] else
front = front - 1;
q[front] = no;

Step III: return;

Algorithm for Deletion from rear end;

Step I: [check from rear pointer] if rear=0
print("Can't delete value at rear end");
return;

Step II: [perform deletion] else
no = q[rear];
[check for front & rear pointer] if front = rear
front=0; rear=0 else
rear = rear-1;
print("Deleted element is ;", no);

Step III: Return.

Algorithm for Deletion from front end:

Step I: [check from front pointer] if front=0
print("Queue is underflow"); return;

Step II: [Perform deletion] else
no=q[front];
print("Deleted element is", no); [Set front & rear pointer]
if front= rear; front=0; rear=0
else front = front+1

Step III: Return.

Program Code:

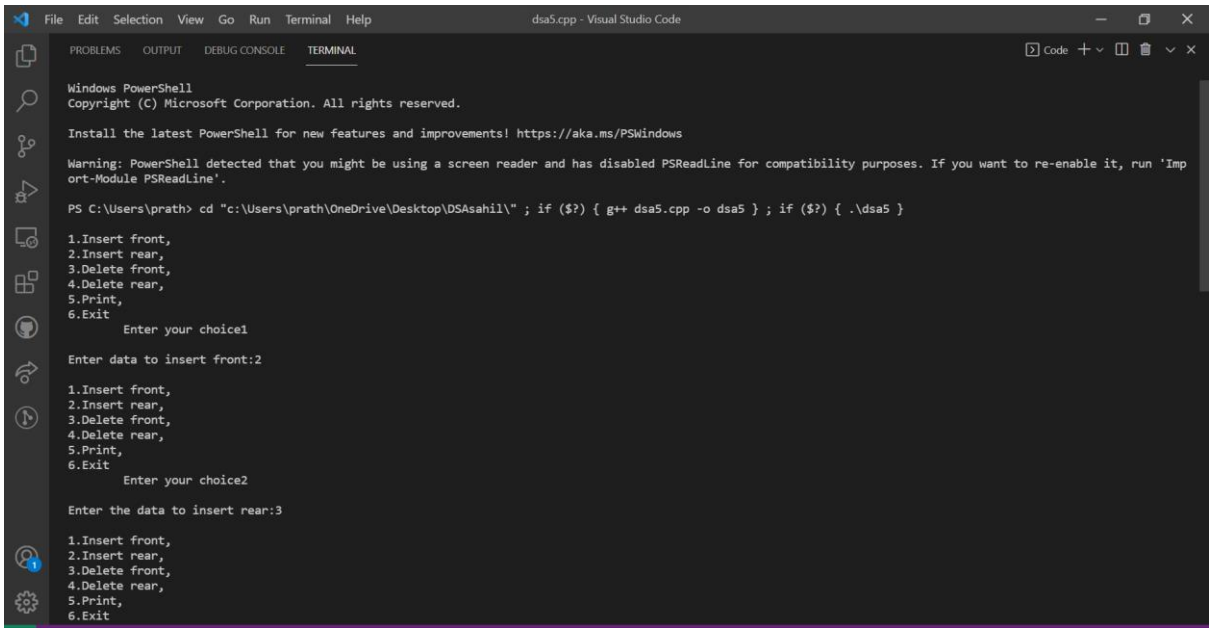
```
#include <iostream>
#include <stdio.h>
#define MAX 10
using namespace std;

struct que
{
   int arr[MAX];
   int front, rear;
};

void init(struct que *q)
{
   q->front = -1;
   q->rear = -1;
}

void print(struct que q)
{
   int i;
   i = q.front;
   while (i != q.rear)
   {
      cout << "\t" << q.arr[i];
      i = (i + 1) % MAX;
   }
   cout << "\t" << q.arr[q.rear];
}

int isempty(struct que q)
{
   return q.rear == -1 ? 1 : 0;
}

int isfull(struct que q)
{
   return (q.rear + 1) % MAX == q.front ? 1 : 0;
}

void addf(struct que *q, int data)
{
   if (isempty(*q))
   {
      q->front = q->rear = 0;
      q->arr[q->front] = data;
   }
```

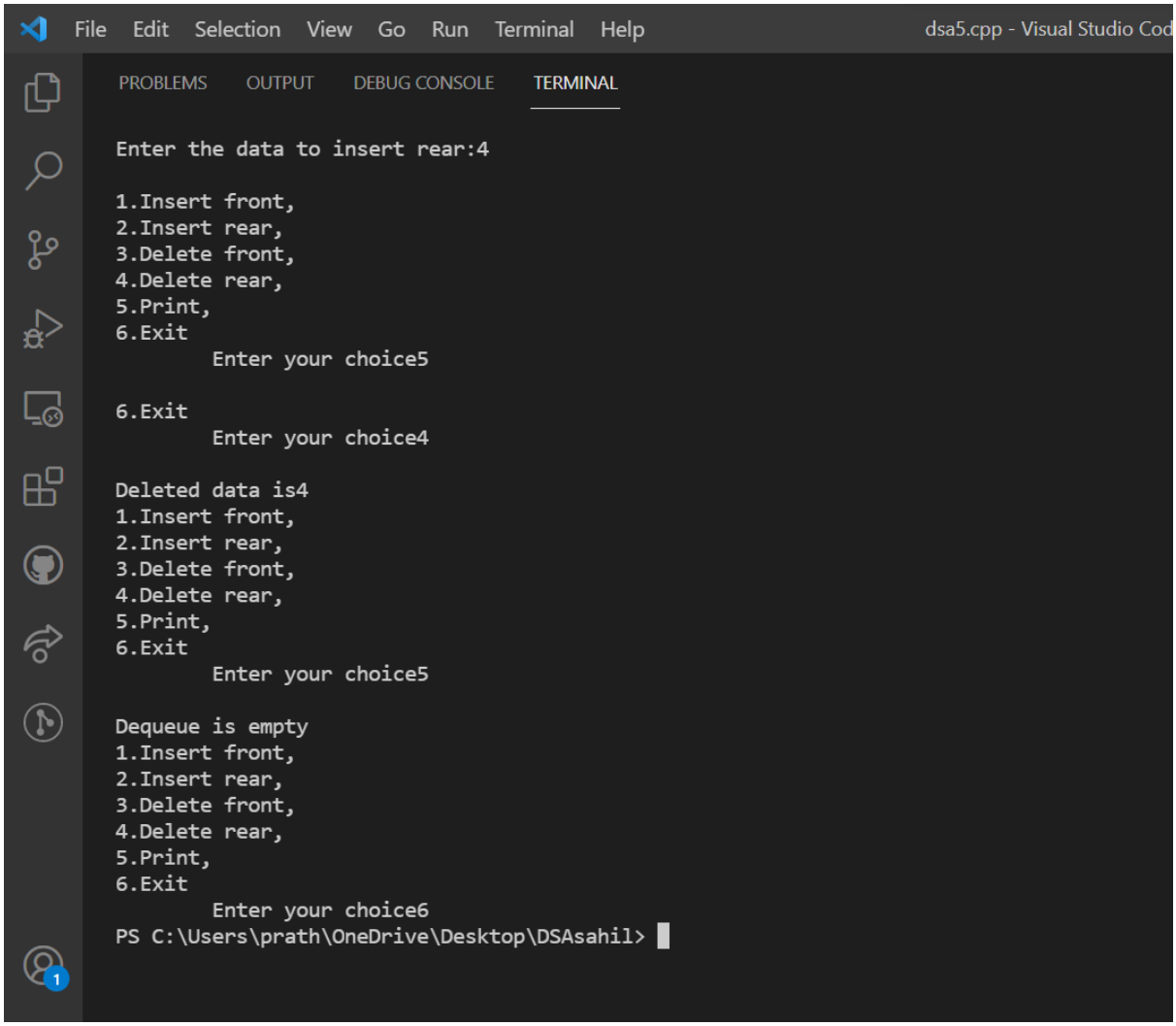
```
    else
    {
        q->front = (q->front - 1 + MAX) % MAX;
        q->arr[q->front] = data;
    }
}

void addr(struct que *q, int data)
{
    if (isempty(*q))
    {
        q->front = q->rear = 0;
        q->arr[q->rear] = data;
    }
    else
    {
        q->rear = (q->rear + 1) % MAX;
        q->arr[q->rear] = data;
    }
}

int delf(struct que *q)
{
    int data1;
    data1 = q->arr[q->front];
    if (q->front == q->rear)
        init(q);
    else
        q->front = (q->front + 1) % MAX;
    return data1;
}

int delr(struct que *q)
{
    int data1;
    data1 = q->arr[q->rear];
    if (q->front == q->rear)
        init(q);
    else
        q->rear = (q->rear - 1 + MAX) % MAX;
    return data1;
}

int main()
{
    struct que q;
    int data, ch;
    init(&q);
    while (ch != 6)
    {
```

```
cout << "\t\n1.Insert front,"
        "\t\n2.Insert rear,"
        "\t\n3.Delete front,"
        "\t\n4.Delete rear,"
        "\t\n5.Print,"
        "\t\n6.Exit";
cout << "\n\tEnter your choice";
cin >> ch;
switch (ch)
{
case 1:
   cout << "\nEnter data to insert front:";
   cin >> data;
   addf(&q, data);
   break;

case 2:
   cout << "\nEnter the data to insert rear:";
   cin >> data;
   addr(&q, data);
   break;

case 3:
   if (isempty(q))
      cout << "\nDequeue is empty";
   else
   {
      data = delf(&q);
      cout << "\nDeleted data is" << data;
   }
   break;

case 4:
   if (isempty(q))
      cout << "\nDequeue is empty";
   else
   {
      data = delr(&q);
      cout << "\nDeleted data is" << data;
   }
   break;

case 5:
   if (isempty(q))
      cout << "\nDequeue is empty";
   else
   {
      cout << "\nDequeue elements are:";
      print(q);
   }
```

```
            break;
        }
    }
    return 0;
}
```

Program Output:




File Edit Selection View Go Run Terminal Help
dsa5.cpp - Visual Studio Cod
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Enter the data to insert rear:4
1.Insert front,
2.Insert rear,
3.Delete front,
4.Delete rear,
5.Print,
6.Exit
Enter your choice5
6.Exit
Enter your choice4
Deleted data is4
1.Insert front,
2.Insert rear,
3.Delete front,
4.Delete rear,
5.Print,
6.Exit
Enter your choice5
Dequeue is empty
1.Insert front,
2.Insert rear,
3.Delete front,
4.Delete rear,
5.Print,
6.Exit
Enter your choice6
PS C:\Users\prath\OneDrive\Desktop\DSAsahil>
1

Conclusion:

By this way we can perform operations on double ended queue.