Syllabus :- UNIT-2

Linear Data structures : Queues - ADT Queues, Types of Queues: linear Queue, Circular Queue, Dequeue : operations on each type of Queues

Definition:- A queue is a linear list of elements in which deletions can take place only at one end, called the "front" and insertions can take place only at the other end called "rear"

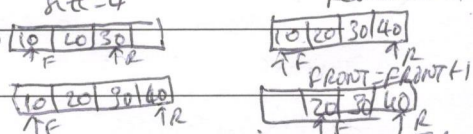
Queues are called first-in-first-out (FIFO) lists, since the first element in a queue will be the first element out of the queue. In other words, the order in which elements enter a queue is the order in which they leave. This contrasts with stacks which are LIFO lists.

Applications :- 1) People waiting for their transactions in bank
2) Time sharing operating system - Process execution

REAR ↓ (Insertions) FRONT ↑ (Deletions)

E	D	C	B	A

Primitive operations on Queues:- 1) Insertion
2) Deletion
3) Display



Representation of Queues:- Each linear array Queue is represented with two pointer variables

1) FRONT: Containing the location of the front element of the queue
2) REAR: Containing the location of the rear element of Queue

The condition FRONT = NULL will indicate that the queue is empty.

Whenever an element is deleted from the queue, the value of FRONT is increased by 1 ⇒ FRONT := FRONT+1

Similarly, whenever an element is added to the queue, the value of REAR is increased by 1 ⇒ REAR := REAR+1

Initial condition are FRONT = REAR = NULL.

Procedure Q INSERT (Queue, N, front, rear, item)

This procedure inserts an element "item" into a queue.

1. [Queue already filled?]
   If front = 1 and rear = N or if front = rear+1 then
   write : OVERFLOW and return
2. [Find new value of REAR]
   If FRONT := NULL then : [Queue initially empty]
   Set FRONT := 1 and REAR := 1
   else if REAR = N then set rear := 1
   else :
   set REAR := REAR+1
   [End of If structure]
3. Set Queue [REAR] := ITEM [This inserts new element]
4. Return.

Procedure QDELETE (QUEUE, N, front, rear, item)
This procedure deletes an element from a queue and assigns it to the variable "Item"

1. [Queue already empty?]
   If front = NULL then : write : UNDERFLOW and return
2. Set ITEM := QUEUE[FRONT]
3. [Find new value of FRONT]
   If FRONT = REAR, then [Queue has only one element to start]
   Set FRONT := NULL and REAR := NULL
   else if FRONT = N then set FRONT := 1
   else: Set FRONT := FRONT + 1
   [End of If structure]
4. Return

Queue representation:-

1. Static Representation: Using arrays int Q[10] can store (N+1) maximum of 10 elements where index start with 0 and ending 9
2. Dynamic Representation. Using linked lists

Conditions in Queue:-

1. Overflow condition:- If rear = MAXSIZE-1 then overflow / full (full)
2. ~~Underflow condition (empty) If: front == -1 Queue is empty.~~

Algorithm for Insertion:-

Step 1: Start
2: [Check the overflow condition in queue]
If (rear == (MAXSIZE-1)) then write "Queue is full"
else goto step 3
3: [Insert an element into queue after incrementing rear]
rear = rear + 1
Q[rear] = item
If (front == -1) then front = front + 1
4: stop

Algorithm for Deletion:-

1. Start
2. [Check the underflow condition]
If (front == -1) then print "queue is empty"
else goto step 3
3: [Deletion of an element]
print "The deleted Item is Q[front]
If (front == rear) then front = rear = -1
else front = front + 1;
4: Stop

Algorithm for Display:-

1. Start
2. [Check the condition of underflow]
If (front == -1) then write "The queue is empty"
else goto step 3
3 for i = front up to rear
print Q[i]
4 stop

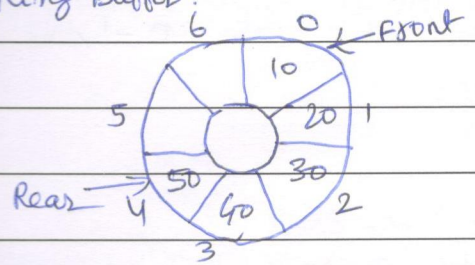
Circular Queue :- (special queue)

In classical queues, elements are inserted into the queue until the queues capacity is reached. However, once this occures a new element can only be inserted when all the elements are deleted from the queue.

Ex:- | 25 | 30 | 51 | 60 | 85 | 45 | | | | 60 | 85 | 45 |

Now if the first three elements are deleted from the front of the queue (left hand side) we get queue remains full but we can not insert a new element because, the back of the queue (right hand side) remains as it was before. This is the major limitation of classical queue i.e even if there is a space available at the front of the queue we can not use that space.

It is a linear data structure in which the operations are performed based on FIFO principle and the last position is connected back to the first position to make a circle. It is also called a "Ring Buffer".



A circular queue is type of queue in which no beginning and no end.

Operations in Circular Queue :-

1. Front : Get the front item from queue
2. Rear : Get the last item from queue
3. enQueue(value). Used to insert an element into the circular queue (CQ). In CQ, the new element always insuted at Rear position
4. deQueue() : Used to delete an element from CQ. Here the element always deleted from front position.

Applications :- 1. Memory Management :- The unused memory locations in the case of ordinary queues can be utilised in circular queues

2. Traffic System :- In computer controlled traffic system, CQ are used to switch on the traffic lights one by one repeatedly as per the time set.
3. CPU Scheduling :- OS often maintains a queue of processes that are ready to execute or that are waiting for a particular event to occur

Primitive Operations on CQ:

1. Insertion: set front = rear = -1

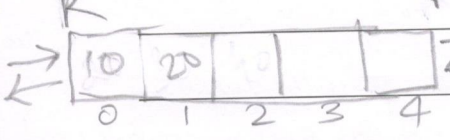
Step 1: Start
2: [Check the overflow condition]
if (rear == (size-1) && front == 0 || front = rear+1)
then print "The queue is full"
otherwise goto Step3
3. [Insert an element]
if (rear == -1) then front = rear = 0
else
rear = (rear+1) % size
Q[rear] = element
4 STOP

2 Deletion:-
Step 1: Start
2: [Check the underflow condition]
if (front == -1) then print "Queue is empty"
otherwise go to step3
3: [Delete an element]
print "The deleted element is Q[front]
if (front == rear) then front = rear = -1;
else front = (front+1) % size;
4- Stop

3 Display:
1. Start
2. [Check the emptyness of the queue]
if (front == -1) then print "Queue is empty"
otherwise goto step3
3. [Display the contents of queue]
for i in front upto rear
print Q[i]
4 Stop.

S=5

	F	f	R	IR	IF	DR	DF	x
		-1	-1	20	10			10
		0	0					20
			1					



Double-Ended Queue: DEQUEUE

It is also a ordered collection of elements in which insertion and deletion operations are performed from both ends. That is, we can insert elements from the rear end or from the front end, the deletion can be made either from front end or from the rear end. It is also referred as DEQUEUE.

We can perform four operations on dequeue namely:

1) Insert at rear-end
2) Insert at front-end
3) Delete at front-end
4) Delete at Rear-End.

Algorithm to perform Insert an element from rear end

Step 1: If (rear == (MAXSIZE-1)) then
print "DQ is full"
else goto step 2

2: [Insert an element]
if (rear == -1) then front = rear = 0
else
rear = rear + 1
DQ[rear] = item;

3 Stop.

Algorithm to Perform Insert an element from front-end.

Step 1: If (front == 0) then print "DQ is full from front-end"
else goto step 2

2: [Insert an element]
If (front == -1) then front = rear = 0
else front = front - 1
DQ[front] = item

3 Stop.

Algorithm to perform Deletion from front-end.

Step 1: If (front == -1) then print "DQ is empty" else goto step 2

2: item = DQ[front];
If (front == rear) then front = rear = -1
else front = front + 1
print "The deleted item from DQ is" item".

3: Stop.

Algorithm to perform Deletion from Rear End.

Step 1: If (rear == -1) then print "DQ is empty"
else goto step 2

2: Item = DQ[rear]
If (front == rear) then front = rear = -1
else
rear = rear - 1
print "The deleted item from DQ is: "Item"

3 Stop.

Algorithm to display content of DQ:

Step 1: if (front == -1 || rear == -1) then print "DQ is empty."
else goto step 2

2. for i in front to rear
print DQ[i]

Reverse of a string using stack

```
#include<stdio.h>
#include<conio.h>
int top=-1
char stack[30];
void push(char item)
{ top++;
  stack[top]=item;
}
char pop()
{ return(stack[top--]);
}
void main()
{ char ch;
  clrscr();
  printf("\n Enter a string ");
  do
  {
  ch=getchar();
  push(ch);
  } while(ch!='\n');
  printf("\n The Reverse of the string ");
  while(top>=0)
     printf("%c", pop());
}
```

String palindrome using input restricted DQ

```
#include<stdio.h>
#include<conio.h>
int front=-1, rear=-1;
char DQ[20];
void rinsert(char c)
{ if(rear==-1)
     rear=front=0;
  else
     rear++;
  DQ[rear]=c;
}
char fdel()
{ return(DQ[front++]);
}
char rdel()
{ return(DQ[rear--]);
}
void main()
{ char DQ[20], c;
  int
  clrscr();
  printf("\n Enter the string");
  while(1)
  {
    c=getchar();
    if(c=='\n')
       break;
    else
       rinsert(c);
  }
  rinsert('\0');
  c=rdel();
  while(front<=rear)
  {
    if(fdel()!=rdel())
       break;
  }if(front<rear)
     printf("\n String is not a palindrome");
  else
     printf("\n String is a palindrome");
  getch();
}
```

Implementation of DEQUEUE using arrays

```
#include<stdio.h><conio.h>
#define max 5
void finsert(); fdelete(), rinsert(),
  rdelete(), display();
int DQ[max], front=-1, rear=-1;
void main()
{int option;
 clrscr();
 do{
   printf("\n Double-Ended Queue operations");
   printf("\n 1. Front-Insert 2 Front-delete
   3. Rear-Insert 4 Rear-Delete 5 Display 6 Exit");
   printf("\n Select your option");
   scanf("%d", &option);
   switch(option)
   {case1: finsert(); break;
    case 2: fdelete(); break;
    case 3: rinsert(); break;
    case 4: rdelete(); break;
```

```
        case 6 : return;
    }
} while (option != 6);
getch();
}
void finsert()
{
int item;
if (front == 0)
{
printf("\n DQ is full from front-End");
return;
}
printf("\n Enter an item to insert");
scanf("%d", &item);
if (front == -1)
{
front = rear = 0;
}
else
    front --;
    DQ[front] = item;
}
void rinsert()
{
int item;
if (rear == (max-1))
{
printf("\n DQ is full from rear-end");
return;
}
printf("\n Enter an item to insert");
scanf("%d", &item);
if (rear == -1)
{ front = rear = 0;
}
else
    rear ++;
    DQ[rear] = item;
}
void fdelete()
{
int item;
if (front == -1)
{
printf("\n DQ is empty");
return;
}
item = DQ[front];
if (front == rear)
{
front = rear = -1;
}
else
    front ++;
printf("\n The deleted item from front-end is: %d", item);
}
void rdelete()
{
int item;
if (rear == -1)
{
printf("\n DQ is empty");
return;
}
item = DQ[rear];
if (front == rear)
{
front = rear = -1;
}
else
    rear --;
printf("\n The deleted item from rear-end is %d", item);
}
Void display()
{
int i;
if (front == -1 || rear == -1)
{
printf("\n DQ is empty");
return;
}
for (i = front; i <= rear; i++)
    printf("%5d", DQ[i]);
}
```