Name of Teacher: Miss Radhika M. Patil

Class: B.Sc. Computer Science (Entire)- II **Semester : 4**

Course Title: Introduction to Data Structure using C++

- **Queue:**

- Queue is a linear Data Structure in which elements can be **inserted** from one end called the **rear end** and **deleted** from other end called **front end**.
- **Front** points to the **beginning** of the queue and **Rear** points to the **end** of the queue.
- In a queue, one end is always used to insert data (enqueue) and the other is used to delete data (dequeue), because queue is open at both its ends.
- The enqueue() and dequeue() are two important functions used in a queue.
- The elements of queue are processed in the order in which they are inserted.
- Queue is based on the principle of **First In First Out (FIFO).**

- **Definition:**

A Queue is ordered collection of items from which items may be inserted at one end called rear and deleted from one end called front.

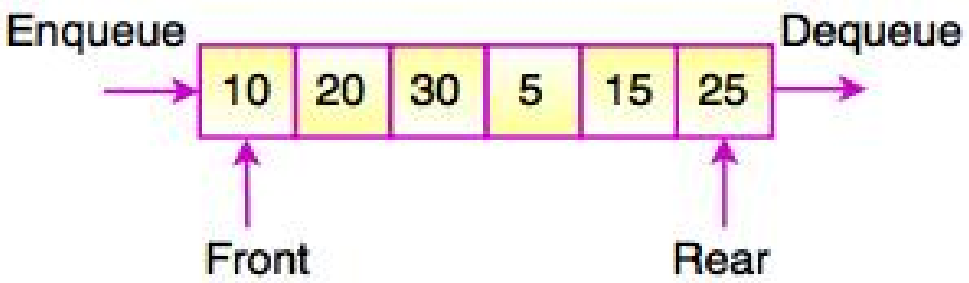
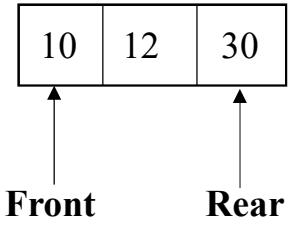
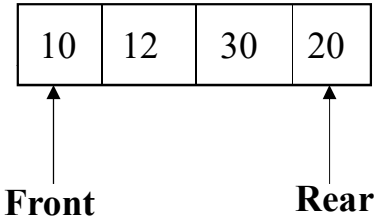


Fig. Queue

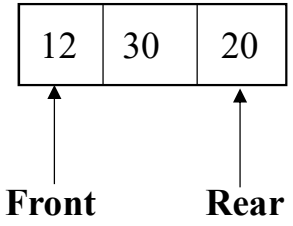
The following diagram shows a queue with elements 10, 12 and 30



A new element 20 will be added at the rear.



An element can be deleted only from the front. Thus 10 will be deleted from the queue.



- In case of insertion operation, rear should be incremented by first and the element should be stored at that position.
- In case of deletion, the element at front has to be returned and then front has to be incremented by 1.

- **Queue as an ADT:**

1. Set of values for Queue:

Queue is finite collection of elements having same data type and all operations are carried out at two ends called front and rear.

2. Properties of Queue:

The properties are related to inserting and deleting the element from the Queue. The operations are done in **FIFO** manner.

3. Operations on Queue:

a) Insert (Item x, Queue Q):

When Queue is not full the this function inserts an item x into rear of the queue and returns Queue Q' with rear pointing to the position of x.

b) remove (Queue Q):

If Queue is not empty then this function deletes the item x pointed by the front of the queue and returns new queue Q' with front is pointing to the item up to deleted item.

c) isEmpty(Queue Q):

This function returns TRUE when Queue is empty else returns FALSE.

d) isFull (Queue Q):

This function returns TRUE when Queue is full else returns FALSE.

- **Operations on a Queues**

The basic operations that can be performed on queue are :

1. To insert an element in a queue.
2. To delete an element from the queue.

1. To insert an element in a queue:

- **Algorithm**

Step 1: If Rear = MAX-1 then

Print “OVERFLOW” and Go to step 4

End if

Step 2: If Front = -1 and Rear = -1

Set

Front = Rear = 0

Else

Set Rear = Rear+1

End if

Step 3: Set Queue[Rear] = Num

Step 4: Return

- In this algorithm to insert an element in a queue.
- In Step 1, we first check for the overflow condition.
- In Step 2, we check if the queue is empty.
- In case the queue is empty, then both Front and Rear are set to zero, so that the new value can be stored at the 0th location.
- Otherwise, if the queue already has some values, then Rear is incremented so that it points to the next location in the array.
- In Step 3, the value is stored in the queue at the location pointed by Rear.

2. To delete an element from the queue.

- **Algorithm**

Step 1: If Front = -1 OR Front > Rear then

Print "UNDERFLOW"

Else

Set

Val = Queue[Front]

Front = Front+1

End if

Step 2: Return

- In this algorithm to delete an element from a queue.
- In Step 1, we check for underflow condition. An underflow occurs if Front = –1 or Front > Rear.
- However, if queue has some values, then Front is incremented so that it now points to the next value in the queue.

- **Static implementation of Queue:**

Program: Write an OOP to perform insert and delete operations on linear or simple queue.

```
int max=10;
class Queue
{
        int front, rear;
        int Q[20];
  public:
        Queue()
        {
          front =-1;
          rear = -1;
        }
        void insert();
        void del();
        void display();

};
```

```
void Queue::insert()
{
        int x;
        cout<<"\n Enter number to insert:";
        cin>>x;
        if(rear = = max-1)
        {
          cout<<"\n Queue is full";
        }
        else
        {
            if (front = = -1 && rear = = -1)
                {
                    front = 0;
                    rear = 0;
                }
        }
        else
        {
                    rear = rear + 1;
        }
        Q[rear] = x;
        display();
}
```

```
void Queue::del()
{
        int x;
        if(front = =-1 || front > rear)
        {
          cout<<"\n Queue is empty";
        }
        else
        {
          x=Q[front];
          cout<<"\n Deleted element is:"<<x;
          if (front = = rear)
            {
                    front = rear= -1;
            }
          else
            {
                    front = front + 1;
            }
          display();
        }
}
```

```
void Queue::display()
{
        if(front = =-1)
        {
          cout<<"\n Queue is empty";
        }
        else
        {
          cout<<"\n Queue is: ";
          for(int i= front; i<=rear ; i++)
          {
            cout<<" "<<Q[i];
          }
        }
}
```

```
void main()
{
   Queue q;
   int ch;
   clrscr();
            cout<<"\n 1. Insert";
            cout<<"\n 2. Delete";
            cout<<"\n 3. Exit";
   do
   {
            cout<<"\n Enter your choice:";
            cin>>ch;
            switch(ch)
            {
                     case 1 : q.insert();
                              break;
                     case 2 : q.del();
                              break;
                     case 3 : cout<<"\n Exit....";
                              break;
                     default: cout<<"\n Invalid choice";
            }
   }while(ch!=3);
   getch();
}
```

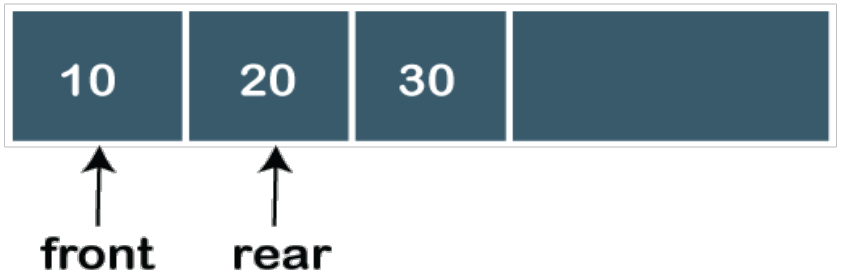
- **Types of Queue:**

There are four different types of queues:

1. Simple/ Linear Queue
2. Circular Queue
3. Priority Queue
4. Double Ended Queue

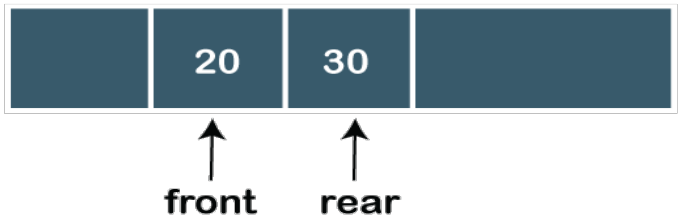
1. Linear Queue

- In Linear Queue, an insertion takes place from one end while the deletion occurs from another end.
- The end at which the insertion takes place is known as the rear end, and the end at which the deletion takes place is known as front end.
- It strictly follows the FIFO rule.
- The linear Queue can be represented, as shown in the below figure:



The above figure shows that the elements are inserted from the rear end, and if we insert more elements in a Queue, then the rear value gets incremented on every insertion.

If we want to show the deletion, then it can be represented as:



In the above figure, we can observe that the front pointer points to the next element, and the element which was previously pointed by the front was deleted.

- The major drawback of using a **linear Queue** is that insertion is done only from the rear end.
- If the first three elements are deleted from the Queue, we cannot insert more elements even though the space is available in a Linear Queue.
- In this case, the linear Queue shows the **overflow** condition as the rear is pointing to the last element of the Queue.

2. Circular Queue

- There was one limitation in the array implementation of Queue.
- If the rear reaches to the end position of the Queue then there might be possibility that some vacant spaces are left in the beginning which cannot be utilized.
- So, to overcome such limitations, the concept of the circular queue was introduced.

https://www.javatpoint.com/ds-types-of-queues

THANK YOU…