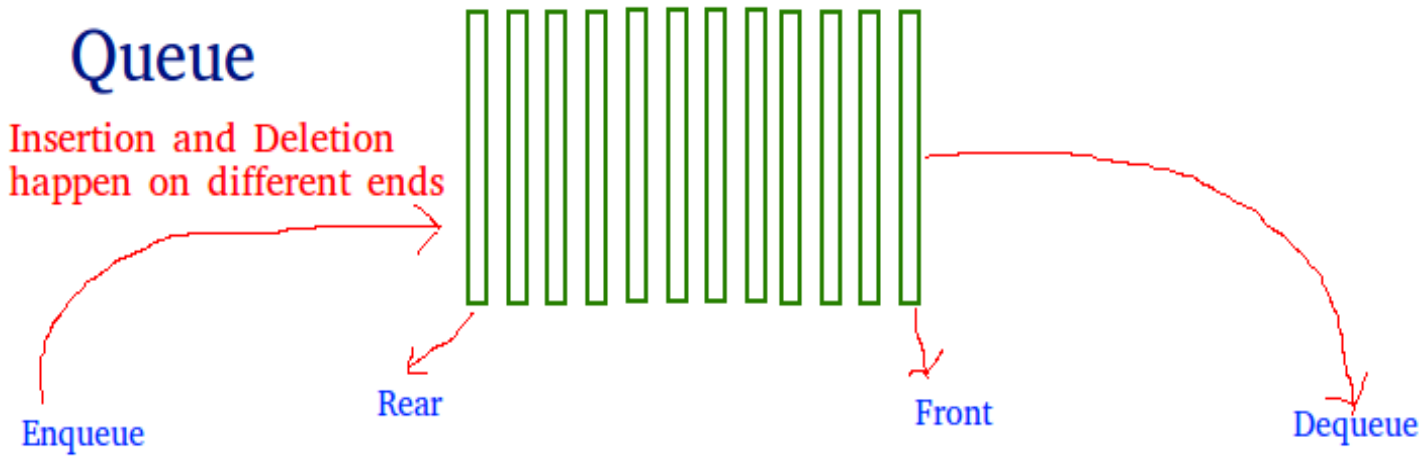
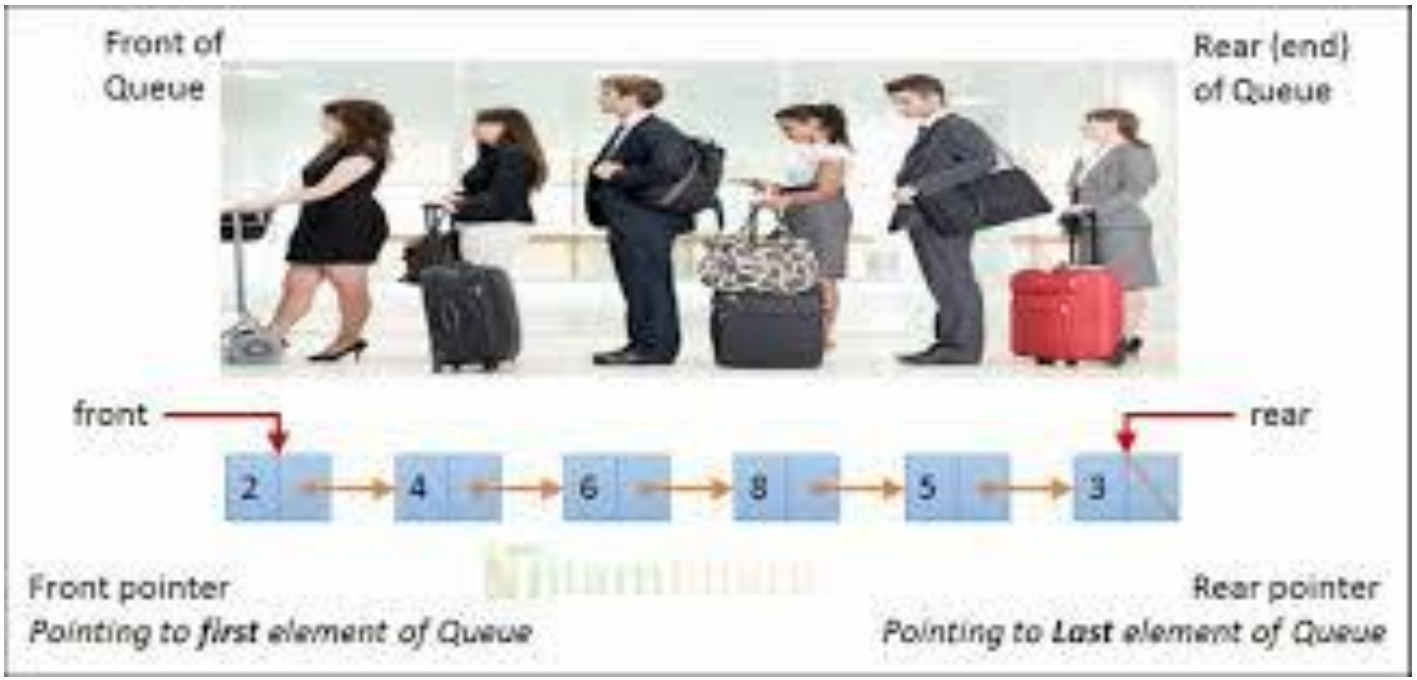
Guess??????

Queue



First in, first out

Queue


Front of Queue
Rear (end) of Queue
front
rear
2
4
6
8
5
3
Front pointer
Pointing to first element of Queue
Rear pointer
Pointing to Last element of Queue

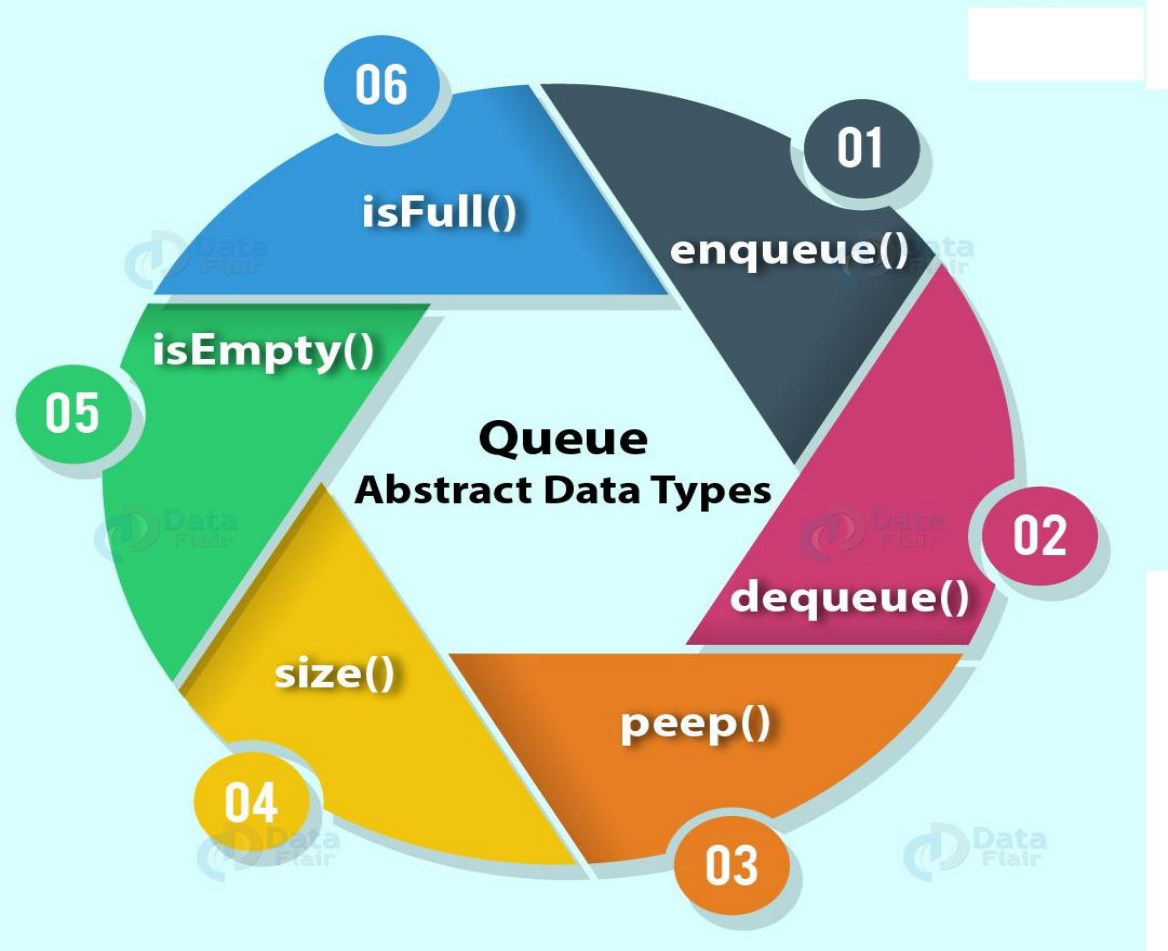
Operation

- Enqueue
- Dequeue

Implementation of Stack Using

- Linked List
- Array

Queue ADT


06
isFull()
01
enqueue()
isEmpty()
05
Queue
Abstract Data Types
02
dequeue()
size()
peep()
04
03

Implementation of Queue

Using

Array

Type Declaration For Queue using Array

Figure 3.57 Type declarations for queue—array implementation

```
#ifndef _Queue_h

struct QueueRecord;
typedef struct QueueRecord *Queue;

int IsEmpty( Queue Q );
int IsFull( Queue Q );
Queue CreateQueue( int MaxElements );
void DisposeQueue( Queue Q );
void MakeEmpty( Queue Q );
void Enqueue( ElementType X, Queue Q );
ElementType Front( Queue Q );
void Dequeue( Queue Q );
ElementType FrontAndDequeue( Queue Q );

#endif  /* _Queue_h */


/* Place in implementation file */
/* Queue implementation is a dynamically allocated array */
#define MinQueueSize ( 5 )

struct QueueRecord
{
    int Capacity;
    int Front;
    int Rear;
    int Size;
    ElementType *Array;
};
```

Checking Empty & Making Empty

```
int
IsEmpty( Queue Q )
{
        return Q->Size == 0;
}
```

Figure 3.58 Routine to test whether a queue is empty—array implementation

```
void
MakeEmpty( Queue Q )
{
        Q->Size = 0;
        Q->Front = 1;
        Q->Rear = 0;
}
```

Figure 3.59 Routine to make an empty queue—array implementation

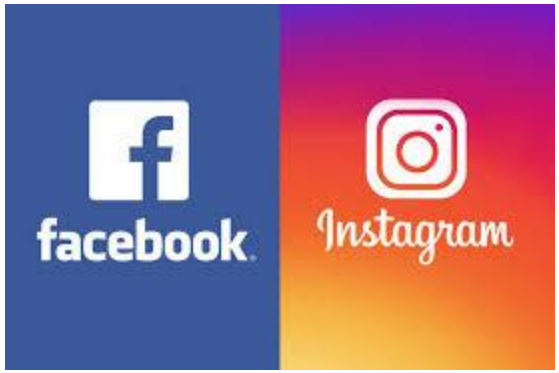
Enqueue

```
static int
Succ( int Value, Queue Q )
{
    if( ++Value == Q->Capacity )
        Value = 0;
    return Value;
}

void
Enqueue( ElementType X, Queue Q )
{
    if( IsFull( Q ) )
        Error( "Full queue" );
    else
    {
        Q->Size++;
        Q->Rear = Succ( Q->Rear, Q );
        Q->Array[ Q->Rear ] = X;
    }
}
```

Figure 3.60 Routines to enqueue—array implementation

Applications



Example for ADT Using Array