Queue

1. A queue can be defined as an ordered list which enables insert operations to be performed at one end called **REAR** and delete operations to be performed at another end called **FRONT**.

2. Queue is referred to be as First In First Out list.

3. For example, people waiting in line for a rail ticket form a queue.

Applications of Queue

Due to the fact that queue performs actions on first in first out basis which is quite fair for the ordering of actions. There are various applications of queues discussed as below.

1. Queues are widely used as waiting lists for a single shared resource like printer, disk, CPU.
2. Queues are used in asynchronous transfer of data (where data is not being transferred at the same rate between two processes) for eg. pipes, file IO, sockets.
3. Queues are used as buffers in most of the applications like MP3 media player, CD player, etc.
4. Queue are used to maintain the play list in media players in order to add and remove the songs from the play-list.
5. Queues are used in operating systems for handling interrupts.

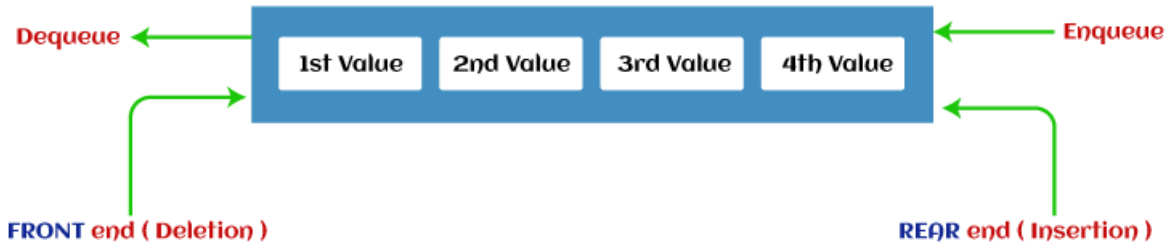
Types of Queue

What is a Queue? (Detailed Definition)

Queue is the data structure that is similar to the queue in the real world. A queue is a data structure in which whatever comes first will go out first, and it follows the FIFO (First-In-First-Out) policy. Queue can also be defined as the list or collection in which the insertion is done from one end known as the **rear end** or the **tail** of the queue, whereas the deletion is done from another end known as the **front end** or the **head** of the queue.

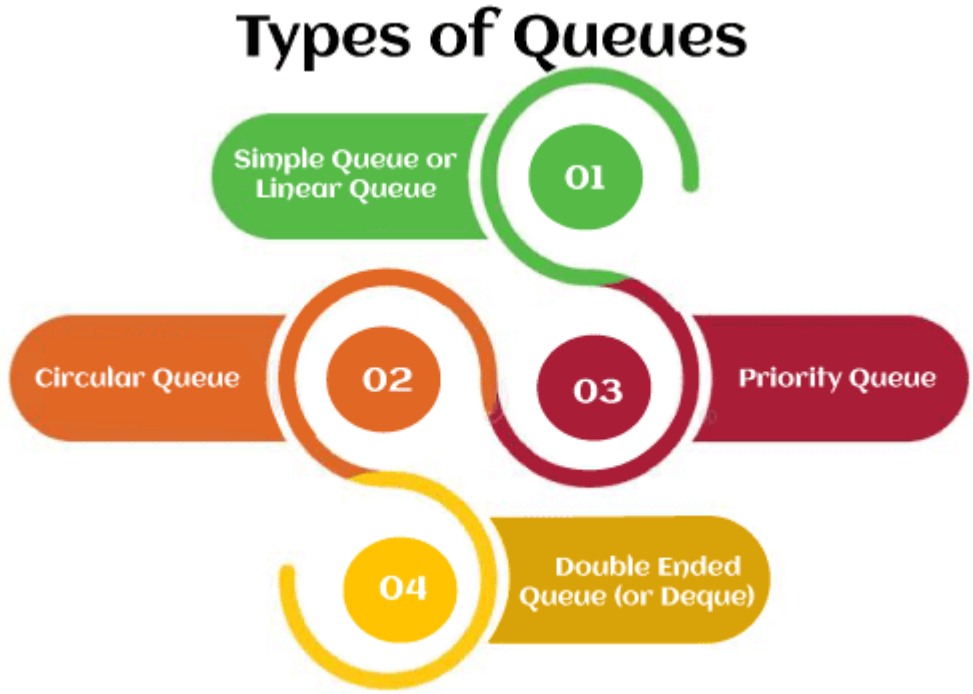
The real-world example of a queue is the ticket queue outside a cinema hall, where the person who enters first in the queue gets the ticket first, and the last person enters in the queue gets the ticket at last. Similar approach is followed in the queue in data structure.

The representation of the queue is shown in the below image -



Types of Queue

There are four different types of queue that are listed as follows -



- Simple Queue or Linear Queue
- Circular Queue
- Priority Queue
- Double Ended Queue (or Deque)

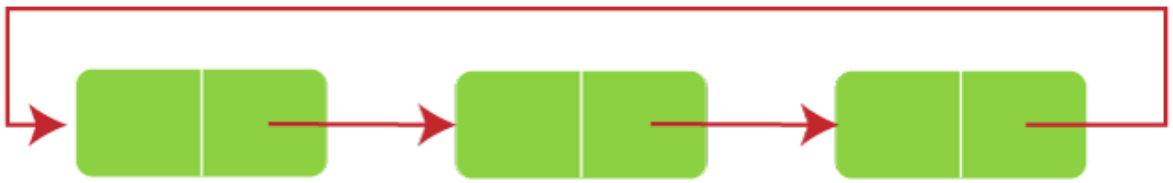
Simple Queue or Linear Queue

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 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.

Circular Queue

In Circular Queue, all the nodes are represented as circular. It is similar to the linear Queue except that the last element of the queue is connected to the first element. It is also known as Ring Buffer, as all the ends are connected to another end. The representation of circular queue is shown in the below image -

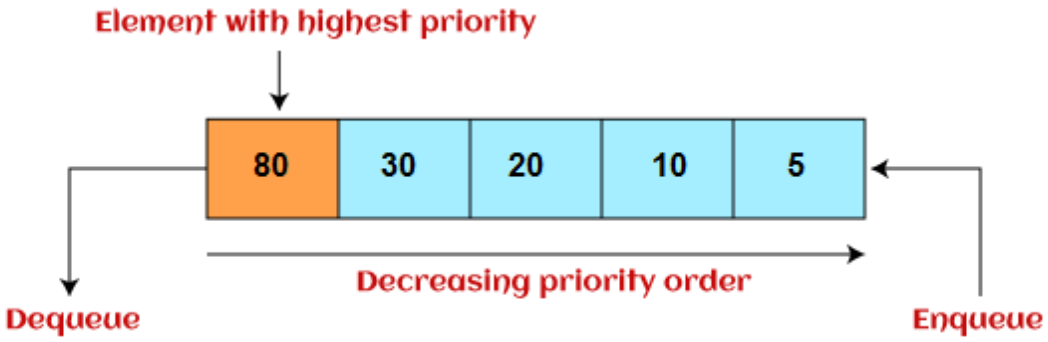


Circular Queue

The drawback that occurs in a linear queue is overcome by using the circular queue. If the empty space is available in a circular queue, the new element can be added in an empty space by simply incrementing the value of rear. The main advantage of using the circular queue is better memory utilization.

Priority Queue

It is a special type of queue in which the elements are arranged based on the priority. It is a special type of queue data structure in which every element has a priority associated with it. Suppose some elements occur with the same priority, they will be arranged according to the FIFO principle. The representation of priority queue is shown in the below image -



Insertion in priority queue takes place based on the arrival, while deletion in the priority queue occurs based on the priority. Priority queue is mainly used to implement the CPU scheduling algorithms.

There are two types of priority queue that are discussed as follows -

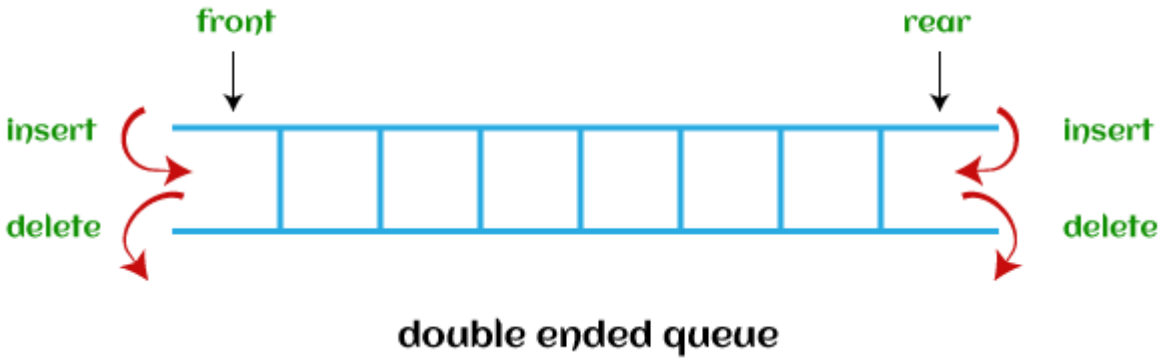
- **Ascending priority queue -** In ascending priority queue, elements can be inserted in arbitrary order, but only smallest can be deleted first. Suppose an array with elements 7, 5, and 3 in the same order, so, insertion can be done with the same sequence, but the order of deleting the elements is 3, 5, 7.
- **Descending priority queue -** In descending priority queue, elements can be inserted in arbitrary order, but only the largest element can be deleted first. Suppose an array with elements 7, 3, and 5 in the same order, so, insertion can be done with the same sequence, but the order of deleting the elements is 7, 5, 3.

Deque (or, Double Ended Queue)

In Deque or Double Ended Queue, insertion and deletion can be done from both ends of the queue either from the front or rear. It means that we can insert and delete elements from both front and rear ends of the queue. Deque can be used as a palindrome checker means that if we read the string from both ends, then the string would be the same.

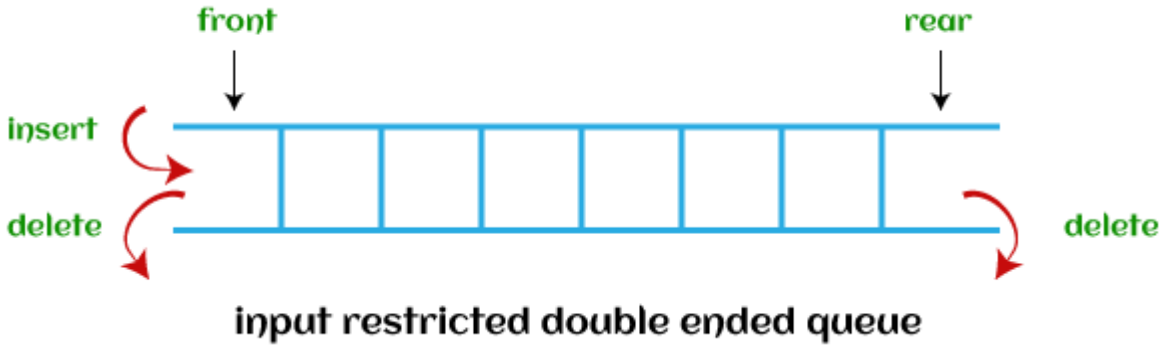
Deque can be used both as stack and queue as it allows the insertion and deletion operations on both ends. Deque can be considered as stack because stack follows the LIFO (Last In First Out) principle in which insertion and deletion both can be performed only from one end. And in deque, it is possible to perform both insertion and deletion from one end, and Deque does not follow the FIFO principle.

The representation of the deque is shown in the below image -

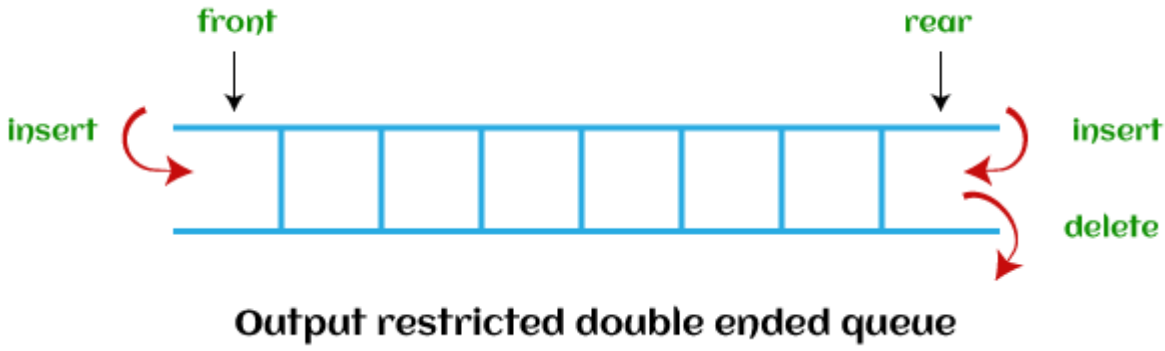


There are two types of deque that are discussed as follows -

- **Input restricted deque -** As the name implies, in input restricted queue, insertion operation can be performed at only one end, while deletion can be performed from both ends.



- **Output restricted deque -** As the name implies, in output restricted queue, deletion operation can be performed at only one end, while insertion can be performed from both ends.



Operations performed on queue

The fundamental operations that can be performed on queue are listed as follows -

- **Enqueue:** The Enqueue operation is used to insert the element at the rear end of the queue. It returns void.
- **Dequeue:** It performs the deletion from the front-end of the queue. It also returns the element which has been removed from the front-end. It returns an integer value.
- **Peek:** This is the third operation that returns the element, which is pointed by the front pointer in the queue but does not delete it.
- **Queue overflow (isfull):** It shows the overflow condition when the queue is completely full.
- **Queue underflow (isempty):** It shows the underflow condition when the Queue is empty, i.e., no elements are in the Queue.

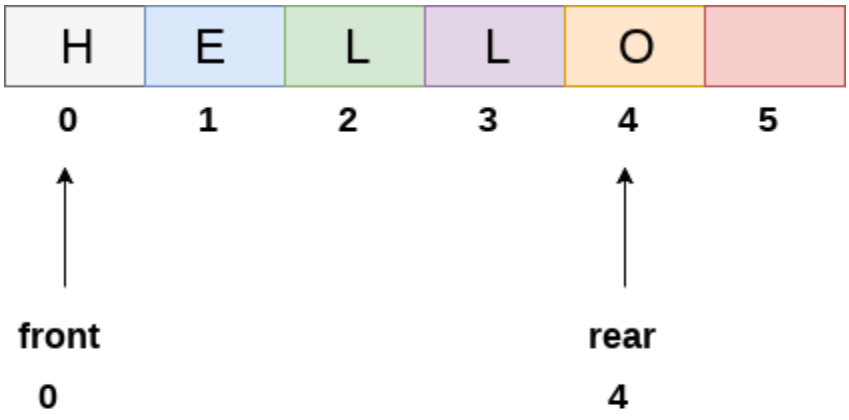
Ways to implement the queue

There are two ways of implementing the Queue:

- **Implementation using array:** The sequential allocation in a Queue can be implemented using an array.
- **Implementation using Linked list:** The linked list allocation in a Queue can be implemented using a linked list.

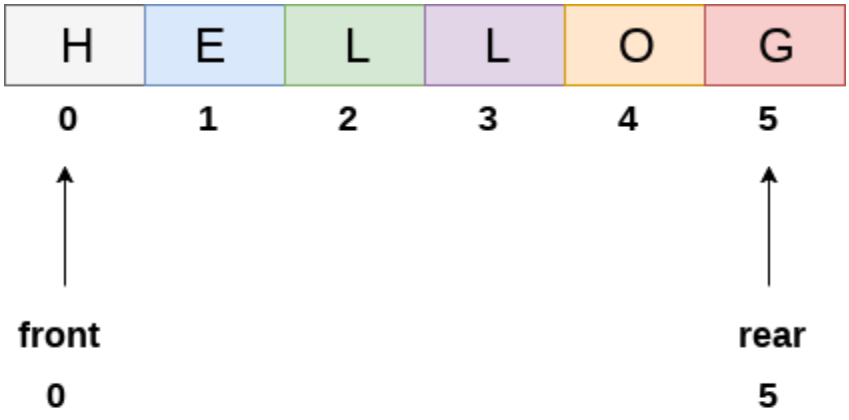
Array representation of Queue

We can easily represent queue by using linear arrays. There are two variables i.e. front and rear, that are implemented in the case of every queue. Front and rear variables point to the position from where insertions and deletions are performed in a queue. Initially, the value of front and queue is -1 which represents an empty queue. Array representation of a queue containing 5 elements along with the respective values of front and rear, is shown in the following figure.



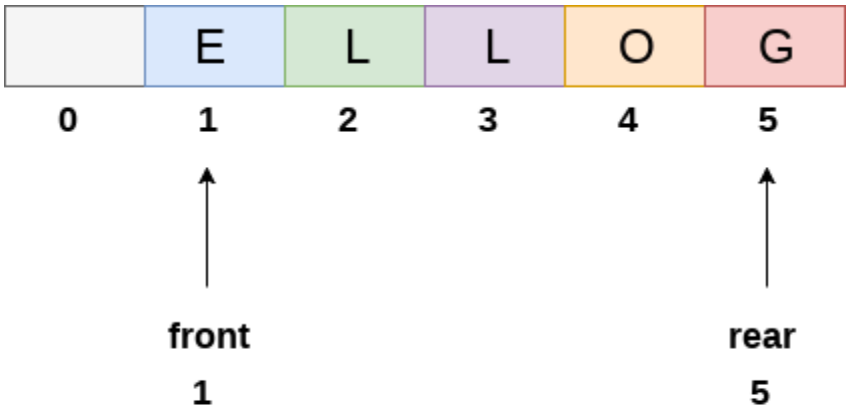
Queue

The above figure shows the queue of characters forming the English word **"HELLO"**. Since, No deletion is performed in the queue till now, therefore the value of front remains -1 . However, the value of rear increases by one every time an insertion is performed in the queue. After inserting an element into the queue shown in the above figure, the queue will look something like following. The value of rear will become 5 while the value of front remains same.



Queue after inserting an element

After deleting an element, the value of front will increase from -1 to 0. however, the queue will look something like following.



Queue after deleting an element

Algorithm to insert any element in a queue

Check if the queue is already full by comparing rear to max - 1. if so, then return an overflow error.

If the item is to be inserted as the first element in the list, in that case set the value of front and rear to 0 and insert the element at the rear end.

Otherwise keep increasing the value of rear and insert each element one by one having rear as the index.

Algorithm

- **Step 1:** IF REAR = MAX - 1
  Write OVERFLOW
  Go to step
  [END OF IF]
- **Step 2:** IF FRONT = -1 and REAR = -1
  SET FRONT = REAR = 0
  ELSE
  SET REAR = REAR + 1
  [END OF IF]
- **Step 3:** Set QUEUE[REAR] = NUM
- **Step 4:** EXIT

C Function

```
1.  void insert (int queue[], int max, int front, int rear, int item)
2.  {
3.      if (rear + 1 == max)
4.      {
5.          printf("overflow");
6.      }
7.      else
8.      {
9.          if(front == -1 && rear == -1)
10.         {
11.             front = 0;
12.             rear = 0;
13.         }
14.         else
15.         {
16.             rear = rear + 1;
17.         }
18.         queue[rear]=item;
19.     }
20. }
```

Algorithm to delete an element from the queue

If, the value of front is -1 or value of front is greater than rear , write an underflow message and exit.

Otherwise, keep increasing the value of front and return the item stored at the front end of the queue at each time.

Algorithm

- **Step 1:** IF FRONT = -1 or FRONT > REAR
  Write UNDERFLOW

ELSE
SET VAL = QUEUE[FRONT]
SET FRONT = FRONT + 1
[END OF IF]

- **Step 2:** EXIT

C Function

```
1.  int delete (int queue[], int max, int front, int rear)
2.  {
3.      int y;
4.      if (front == -1 || front > rear)
5.
6.      {
7.          printf("underflow");
8.      }
9.      else
10.     {
11.         y = queue[front];
12.         if(front == rear)
13.         {
14.             front = rear = -1;
15.             else
16.             front = front + 1;
17.
18.         }
19.         return y;
20.     }
21. }
```

Menu driven program to implement queue using array

```
1.  #include<stdio.h>
```

```
2.  #include<stdlib.h>
3.  #define maxsize 5
4.  void insert();
5.  void delete();
6.  void display();
7.  int front = -1, rear = -1;
8.  int queue[maxsize];
9.  void main ()
10. {
11.     int choice;
12.     while(choice != 4)
13.     {
14.         printf("\n*************************Main Menu*****************************\
    n");
15.         printf("\n=========================================
    ========================\n");
16.         printf("\n1.insert an element\n2.Delete an element\n3.Display the queue\
    n4.Exit\n");
17.         printf("\nEnter your choice ?");
18.         scanf("%d",&choice);
19.         switch(choice)
20.         {
21.             case 1:
22.             insert();
23.             break;
24.             case 2:
25.             delete();
26.             break;
27.             case 3:
28.             display();
29.             break;
30.             case 4:
31.             exit(0);
32.             break;
33.             default:
34.             printf("\nEnter valid choice??\n");
35.         }
```

```
    }
}
void insert()
{
    int item;
    printf("\nEnter the element\n");
    scanf("\n%d",&item);
    if(rear == maxsize-1)
    {
        printf("\nOVERFLOW\n");
        return;
    }
    if(front == -1 && rear == -1)
    {
        front = 0;
        rear = 0;
    }
    else
    {
        rear = rear+1;
    }
    queue[rear] = item;
    printf("\nValue inserted ");

}
void delete()
{
    int item;
    if (front == -1 || front > rear)
    {
        printf("\nUNDERFLOW\n");
        return;

    }
    else
    {
        item = queue[front];
```

```
73.         if(front == rear)
74.         {
75.             front = -1;
76.             rear = -1 ;
77.         }
78.         else
79.         {
80.             front = front + 1;
81.         }
82.         printf("\nvalue deleted ");
83.     }
84.
85.
86. }
87.
88. void display()
89. {
90.     int i;
91.     if(rear == -1)
92.     {
93.         printf("\nEmpty queue\n");
94.     }
95.     else
96.     {   printf("\nprinting values .....\n");
97.         for(i=front;i<=rear;i++)
98.         {
99.             printf("\n%d\n",queue[i]);
100.                }
101.            }
102.        }
```

Output:

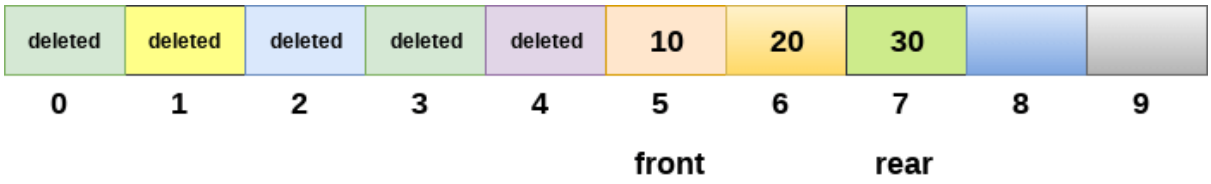
```
*************Main Menu**************

===============================================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit
```

```
Enter your choice ?1

Enter the element
123

Value inserted

*************Main Menu**************

==========================================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?1

Enter the element
90

Value inserted

*************Main Menu**************

==================================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?2

value deleted

*************Main Menu**************
==========================================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?3

printing values .....

90

*************Main Menu**************

==========================================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?4
```

Drawback of array implementation

Although, the technique of creating a queue is easy, but there are some drawbacks of using this technique to implement a queue.

- **Memory wastage :** The space of the array, which is used to store queue elements, can never be reused to store the elements of that queue because the elements can only be inserted at front end and the value of front might be so high so that, all the space before that, can never be filled.



limitation of array representation of queue

The above figure shows how the memory space is wasted in the array representation of queue. In the above figure, a queue of size 10 having 3 elements, is shown. The value of the front variable is 5, therefore, we can not reinsert the values in the place of already deleted element before the position of front. That much space of the array is wasted and can not be used in the future (for this queue).

- **Deciding the array size**

On of the most common problem with array implementation is the size of the array which requires to be declared in advance. Due to the fact that, the queue can be extended at runtime depending upon the problem, the extension in the array size is a time taking process and almost impossible to be performed at runtime since a lot of reallocations take place. Due to this reason, we can declare the array large enough so that we can store queue elements as enough as possible but the main problem with this declaration is that, most of the array slots (nearly half) can never be reused. It will again lead to memory wastage.