```
Queue is Underflow

Queue is underflow                    Menu

1. Insert

2. Delete

3. Display

4. Exit

Enter Your Choice :4

...Program finished with exit code 0

Press ENTER to exit console.
```

Deque [double-ended queue] :

In the preceding section we saw that a queue in which we insert items at one end and from which we remove items at the other end. In this section we examine an extension of the queue, which provides a means to insert and remove items at both ends of the queue. This data structure is a **deque**. The word **deque** is an acronym derived from **double-ended queue**. Below figure shows the representation of a deque.

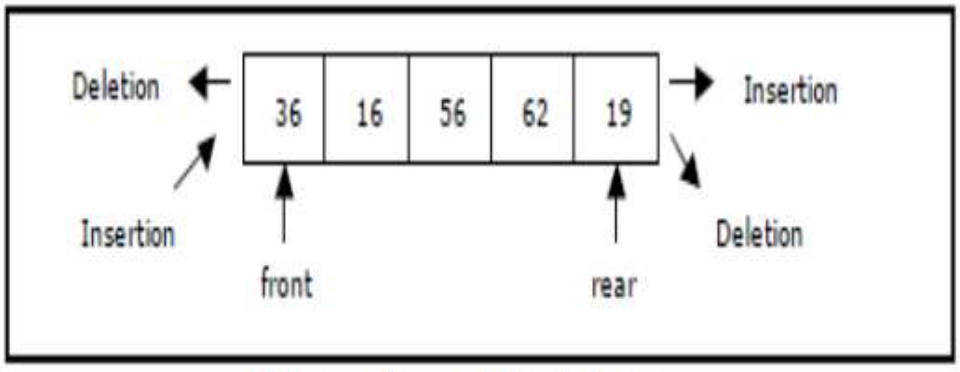


Figure Representation of a deque.

deque provides four operations. Below Figure shows the basic operations on a deque.

- enqueue_front: insert an element at front.
- dequeue_front: delete an element at front.
- enqueue_rear: insert element at rear.
- dequeue_rear: delete element at rear.

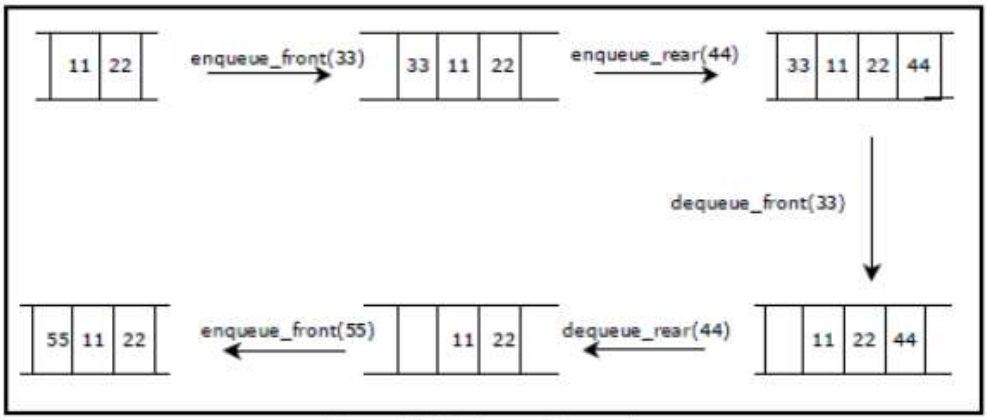


Figure ` `. Basic operations on deque

There are two variations of deque. They are:

- Input restricted deque (IRD)
- Output restricted deque (ORD)

An Input restricted deque is a deque, which allows insertions at one end but allows deletions at both ends of the list.

An output restricted deque is a deque, which allows deletions at one end but allows insertions at both ends of the list.

Double – Ended Queue

Algorithm – Insertion at rear end

```
Step -1 : [Check for overflow]
          if(rear==MAX)
                Output "Queue is Overflow"

                return;

Step-2  : [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-3 : return
```

Algorithm : Insertion at front end

```
Step-1 : [Check for the front position]
            If front<=1
                  Ourput "Cannot add value at front end"
                  return;
Step-2  : [Insert at front]
            else
                  front=front-1;
                  q[front]=no;
Step-3  : Return
```

Algorithm :Deletion from front end

```
Step-1 [ Check for front pointer]
        if front=0
            Output " Queue is Underflow"
            return;
Step-2 [Perform deletion]
        else
            no=q[front];
            Output "Deleted element is",no

            [Set front and rear pointer]

                  if front=rear
                        front=0;
                        rear=0;
                  else
                        front=front+1;

Step-3 : Return
```

Algorithm : deletion from rear end

```
Step-1 : [Check for the rear pointer]
            if rear=0
                  Output "Cannot delete value at rear end"
                  return;

Step-2: [ perform deletion]
            else
                  no=q[rear];
                  [Check for the front and rear pointer]
                  if front= rear
                        front=0;
                        rear=0;
                  else
                        rear=rear-1;
                        Output "Deleted element is",no
Step-3 : Return
```

Program for Double ended queue:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 10

int q[MAX], front=0, rear=0;

void add_rear();
void add_front();
void delete_rear();
void delete_front();
void display();

int main() {
   int ch;
   do {
      printf("\n DQueue Menu");
      printf("\n----------");
      printf("\n 1. AddRear");
      printf("\n 2. AddFront");
      printf("\n 3. DeleteRear");
      printf("\n 4. DeleteFront");
      printf("\n 5. Display");
      printf("\n 6. Exit");
```

```
printf("\n--------------");
printf("\n Enter your choice: ");
scanf("%d", &ch);
switch(ch) {
   case 1:
      add_rear();
      printf("\n Queue after insert at rear");
      display();
      break;
   case 2:
      add_front();
      printf("\n Queue after insert at front");
      display();
      break;
   case 3:
      delete_rear();
      printf("\n Queue after delete at rear");
      display();
      break;
   case 4:
      delete_front();
      printf("\n Queue after delete at front");
      display();
      break;
```

```
            case 5:
               display();
               break;
            case 6:
               exit(0);
               break;
            default:
               printf("\n Wrong Choice\n");
         }
   } while(ch!=6);
   return 0;
}

void add_rear() {
   int no;
   printf("\n Enter value to insert : ");
   scanf("%d",&no);
   if(rear==MAX) {
      printf("\n Queue is Overflow");
      return;
   }
   else {
      rear++;
      q[rear]=no;
```

```
        if(rear==0)
            rear=1;
        if(front==0)
            front=1;
    }
}

void add_front() {
    int no;
    printf("\n Enter value to insert: ");
    scanf("%d", &no);
    if(front<=1) {
        printf("\n Cannot add value at front end");
        return;
    }
    else {
        front--;
        q[front]=no;
    }
}

void delete_front() {
    int no;
    if(front==0) {
```

```
        printf("\n Queue is Underflow\n");
        return;
    }
    else {
        no=q[front];
        printf("\n Deleted element is %d\n", no);
        if(front==rear) {
            front=0;
            rear=0;
        }
        else {
            front++;
        }
    }
}

void delete_rear() {
    int no;
    if(rear==0) {
        printf("\n Cannot delete value at rear end\n");
        return;
    }
    else {
        no=q[rear];
```

```
        if(front==rear) {
            front=0;
            rear=0;
        }
        else {
            rear--;
            printf("\n Deleted element is %d\n", no);
        }
    }
}

void display() {
    int i;
    if(front==0) {
        printf("\n Queue is Underflow\n");
        return;
    }
    else {
        printf("\n Output");
        for(i=front; i<=rear; i++) {
            printf("\n %d", q[i]);
        }
    }
}
```

Output:

DQueue Menu

---

1. AddRear

2. AddFront

3. DeleteRear

4. DeleteFront

5. Display

6. Exit

Enter your choice: 1

Enter value to insert : 23

Queue after insert at rear

Output

23

DQueue Menu

---

1. AddRear

2. AddFront

3. DeleteRear

4. DeleteFront

5. Display

6. Exit

Enter your choice: 2

Enter value to insert: 56

Cannot add value at front end

Queue after insert at front

Output

23

DQueue Menu

---

1. AddRear

2. AddFront

3. DeleteRear

4. DeleteFront

5. Display

6. Exit

Enter your choice: 5

Output

23

DQueue Menu

---

1. AddRear

2. AddFront

3. DeleteRear

4. DeleteFront

5. Display

6. Exit

Enter your choice: 3 4

Deleted element is 23

Queue after delete at front

Queue is Underflow

DQueue Menu

---

1. AddRear

2. AddFront

3. DeleteRear

4. DeleteFront

5. Display

6. Exit

```
Enter your choice: 5


Queue is Underflow


DQueue Menu
--------------
1. AddRear
2. AddFront
3. DeleteRear
4. DeleteFront
5. Display
6. Exit
--------------
Enter your choice: 6
...Program finished with exit code 0
Press ENTER to exit console….
```

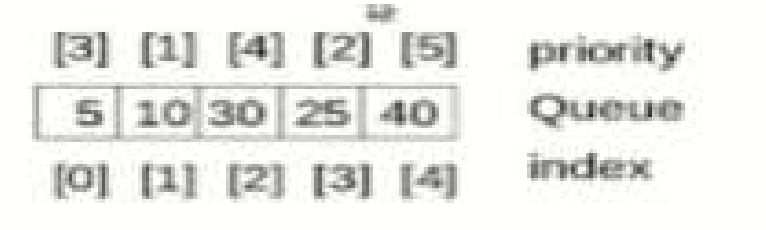
Priority Queue:

A **priority queue** is a collection of elements such that each element has been assigned a priority. We can insert an element in priority queue at the rare position. We can delete an element from the priority queue based on the elements priority and such that the order in which elements are deleted and processed comes from the following rules:

1. An element of higher priority is processed before any element of lower priority.

2. Two elements with same priority are processed according to the order in which they were added to the queue. It follows **FIFO** or **FCFS**(**First Comes First serve**) rules.

We always remove an element with the highest priority, which is given by the minimal integer priority assigned.



A prototype of a priority queue is time sharing system: programs of high priority are processed first, and programs with the same priority form a standard queue.

Priority queues are two types:

1. Ascending order priority queue

2. Descending order priority queue

1. Ascending order priority queue: It is Lower priority number to high priority number. Examples: order is 1,2,3,4,5,6,7,8,9,10

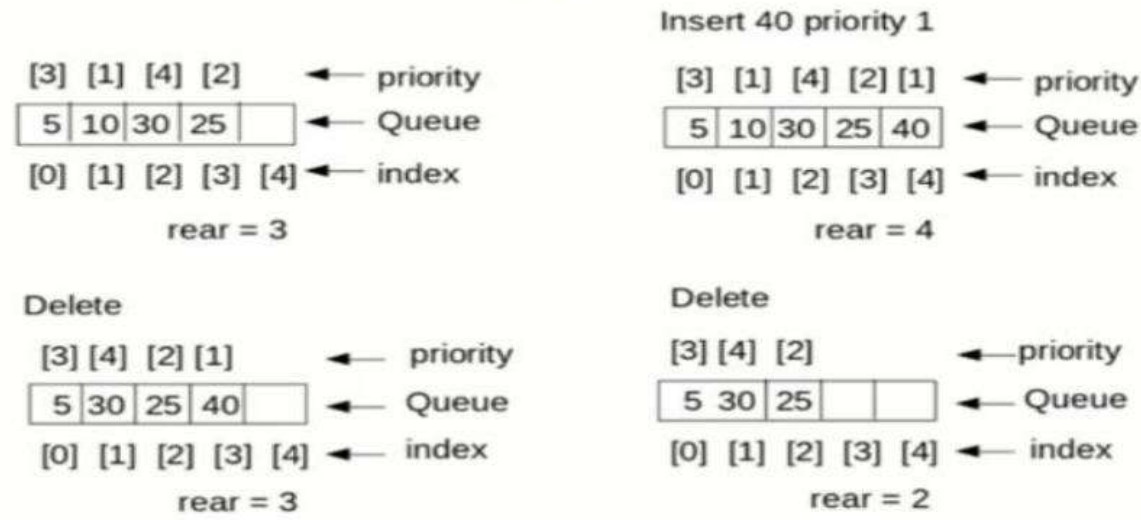
2. Descending order priority queue: It is high priority number to lowest priority number. Examples: Order is 10,9,8,7,6,5,4,3,2,1

Implementation of Priority Queue:

Implementation of priority queues are two types:

1. Through Queue(Using Array)
2. Through Sorted List(Using Linked List)

1. Through Queue (Using Array): In this case element is simply added at the rear end as usual. For deletion, the element with highest priority is searched and then deleted.



Priority Queue

Algorithm : Insert in queue

```
Step-1 : [check for queue is overflow]
                if (rear = MAX – 1)
                        Output " Priority queue is full"

Step-2 : [Insert element]
                else
                        rear = rear + 1
                        d [rear] = d1
            step-3 : [Arrange priority]
                repeat  i=front  to  i<=rear  i=i+1
                repeat  j=i+1  to  j<=rear  j=j+1
                        if ( d[i].p > d[j].p) then
                                temp = d[i]
                                d[i] = d[j]
                                d[j] = temp
                        else
                                if(d[i].p==d[j].p) then
                                        if(d[i].o > d[j].o) then
                                                temp = d[i]
                                                d[i] = d[j]
                                                d[j] = temp
```

step-4 : exit

Algorithm : Deletion in a queue

```
Step-1 : [check for underflow]
            if(front=-1)
                  output " priority queue is empty"

step-2 : [Display deleted element]
            else
                  print  " value" = d[front].val
                  print  " priority" = d[front].p
                  print " order" = d[front].o

step-3 : [Increment value]
            if (front == rear) then
                  front = rear = -1
            else
                  front =front +1
```

Program for priority queue:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 5

struct data
{
   int val,p,o;
} d[MAX];

int front = -1, rear = -1;

void insert(struct data d1) {
   struct data temp;
   int i,j;

   if(rear == MAX-1) {
      printf("\n Priority Queue is Full...");
      return;
   } else {
      rear++;
      d[rear] = d1;
      if(front == -1) front = 0;
      for(i = front; i <= rear; i++) {
```

```
            for(j = i+1; j <= rear; j++) {
               if(d[i].p > d[j].p) {
                  temp = d[i];
                  d[i] = d[j];
                  d[j] = temp;
               } else {
                  if(d[i].p == d[j].p) {
                     if(d[i].o > d[j].o) {
                        temp = d[i];
                        d[i] = d[j];
                        d[j] = temp;
                     }
                  }
               }
            }
         }
      }

void deletion() {
   if(front == -1) {
      printf("\n Priority Queue is Empty...");
      return;
   } else {
```

```
        printf("-------------------------------\n");
        printf("\n Value: %d", d[front].val);
        printf("\n Priority: %d", d[front].p);
        printf("\n Order: %d", d[front].o);
        printf("\n -------------------------------");
        if(front == rear)
           front = rear = -1;
        else
           front++;
    }
}

void display() {
    int i;
    if(front == -1) {
        printf("\n Priority Queue is Empty...");
        return;
    } else {
        for(i = front; i <= rear; i++) {
            printf("\n -------------------------------");
            printf("\n Object : %d", i+1);;
            printf("\n Value : %d", d[i].val);
            printf("\n Priority: %d", d[i].p);
            printf("\n Order : %d", d[i].o);
```

```
            printf("\n -------------------------------");
        }
    }
}

int main() {
    struct data d1;
    int ch;

    do {
        printf("\n-------------------");
        printf("\n M E N U ");
        printf("\n-------------------");
        printf("\n1 Insertion");
        printf("\n2 Deletion");
        printf("\n3 Display");
        printf("\n4 Exit");
        printf("\n-------------------");
        printf("\nEnter Choice... ");
        scanf("%d", &ch);

        switch(ch) {
            case 1:
                printf("\n\t\tEnter Value:- ");
```

```
            scanf("%d", &d1.val);
            printf("\n\t\tEnter Priority:- ");
            scanf("%d", &d1.p);
            printf("\n\t\tEnter Order:- ");
            scanf("%d", &d1.o);
            insert(d1);
            break;
        case 2:
            deletion();
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
        default:
            printf("\nInvalid Choice...");
      }
   } while(1);

   return 0;
}
```

Output:

M E N U

---

1 Insertion

2 Deletion

3 Display

4 Exit

Enter Choice... 1

Enter Value:- 23

Enter Priority:- 0

Enter Order:- 2

M E N U

---

1 Insertion

2 Deletion

3 Display

4 Exit

Enter Choice... 1

Enter Value:- 45

Enter Priority:- 6

Enter Order:- 34

M E N U

---

1 Insertion

2 Deletion

3 Display

4 Exit

Enter Choice... 1

Enter Value:- 12

Enter Priority:- 1

Enter Order:- 1

M E N U

---

1 Insertion

2 Deletion

3 Display

4 Exit

Enter Choice... 2

Value: 23

Priority: 0

Order: 2

M E N U

---

1 Insertion

2 Deletion

3 Display

4 Exit

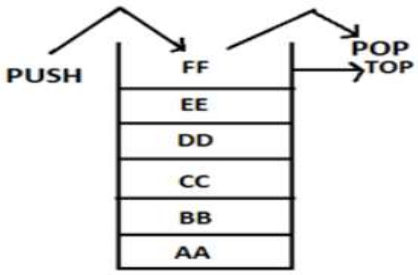
```
------------------
Enter Choice... 3


 ------------------------------
 Object : 2
 Value : 12
 Priority: 1
 Order : 1
 ------------------------------
 ------------------------------
 Object : 3
 Value : 45
 Priority: 6
 Order : 34
 ------------------------------
------------------
 M E N U
------------------
1 Insertion
2 Deletion
3 Display
4 Exit
------------------
Enter Choice... 4
```

1. Difference between stacks and Queues?

stacks	Queues
1.A stack is a linear list of elements in which the element may be inserted or deleted at one end. 2. In stacks, elements which are inserted last is the first element to be deleted. 3.Stacks are called LIFO (Last In First Out)list 4.In stack elements are removed in reverse order in which thy are inserted. 5.suppose the elements a,b,c,d,e are inserted in the stack, the deletion of elements will be e,d,c,b,a. 6.In stack there is only one pointer to insert and delete called "Top". 7.Initially top=-1 indicates a stack is empty. 8.Stack is full represented by the condition TOP=MAX-1(if array index starts from '0'). 9.To push an element into a stack, Top is incremented by one 10.To POP an element from stack,top is decremented by one.	1.A Queue is a linerar list of elements in which the elements are added at one end and deletes the elements at another end. 2. . In Queue the element which is inserted first is the element deleted first. 3. Queues are called FIFO (First In First Out)list. 4. In Queue elements are removed in the same order in which thy are inserted. 5. Suppose the elements a,b,c,d,e are inserted in the Queue, the deletion of elements will be in the same order in which thy are inserted. 6. In Queue there are two pointers one for insertion called "Rear" and another for deletion called "Front". 7. Initially Rear=Front=-1 indicates a Queue is empty. 8.Queue is full represented by the condition Rear=Max-1. 9.To insert an element into Queue, Rear is incremented by one. 10.To delete an element from Queue, Front is incremented by one.

11.The conceptual view of Stack is as follows:



11.The conceptual view of Queue is as follows: