DATA STRUCTURES

QUEUE

Prof. Khalil Al-Shqeerat

CS Dept. – Qassim University

QUEUE

1. Queue definition
2. Queue as an Array
3. Queue as a Linked list
4. Circular Queue
5. Deque
6. Priority Queue

QUEUE DEFINITION

- Queue is a linear data structure that permits insertion of new element at one end and deletion of an element at the other end.
- The end at which the deletion of an element takes place is called **front** , and the end at which insertion of a new element takes place is called **rear**.
- The first element that gets added into the queue is the first one to get removed from the list.
  - Queue is also referred to as **first-in-first- out (FIFO)** list

الـ Queue هو هيكل بيانات (Data Structure) خطّي، يعني العناصر تكون مرتبة وراء بعض.

فكرته تشبه طابور الناس عند الكاشير أو البنك:

أول شخص يدخل الطابور → أول شخص يخرج.

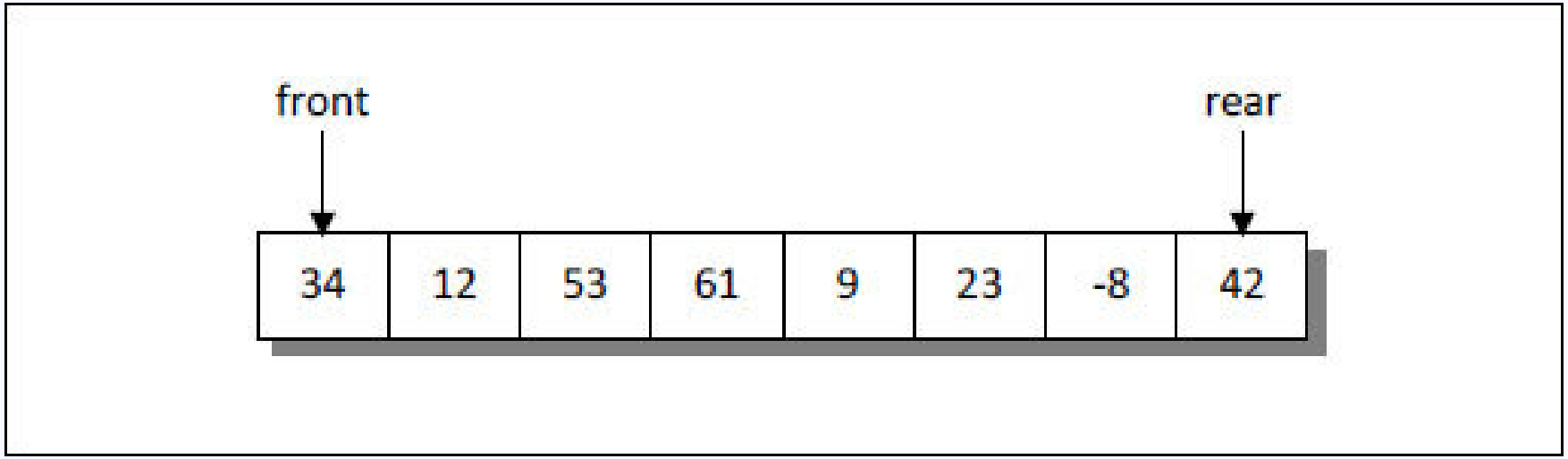
آخر شخص يدخل → ينتظر دوره.

لهذا نسميه:

FIFO = First In First Out

أول داخل = أول خارج

REPRESENTATION OF A QUEUE



الإضافة (insertion) تكون من جهة اسمها Rear الخلف.

الحذف (deletion) يكون من جهة اسمها Front الأمام.

QUEUE AS AN ARRAY

- An array can store a fixed number of elements.
- Queue keeps on changing as element is removed from the front end or a new element is added at the rear end.
- Declaring an array with a maximum size would solve this problem.
- The maximum size should be large enough for a queue to expand or shrink.

بدل ما نخزن عناصر الـ Queue بطريقة عشوائية،
نستخدم مصفوفة Array لتخزين العناصر.
لماذا نستخدم Array؟
لأن الـ Array: سهلة، سريعة
تخزن البيانات بترتيب
لكن يوجد مشكلة:
الـ Queue حجمه يتغير دائمًا:
أحيانًا نضيف عناصر
أحيانًا نحذف عناصر
بينما الـ Array:
حجمه ثابت Fixed Size
الحل: نحدد حجمًا أقصى للـ Queue.
مثال:
int queue[5];
يعني: أقصى عدد عناصر = 5

IMPLEMENTATION OF QUEUE AS AN ARRAY

```
#include
using namespace std;
const int MAX = 10;  أقصى عدد عناصر هو 10
class queue
{
      private:
            int arr[MAX];  مصفوفة من نوع int تخزن ١٠ عناصر
            int front, rear;  الأمام والخلف
      public:
            queue();
            void addq (int item);  إضافة عنصر
            int delq();  حذف عنصر
};
```

```
// initializing data members
queue:: queue()
{
front = -1;
rear = -1;
}
```

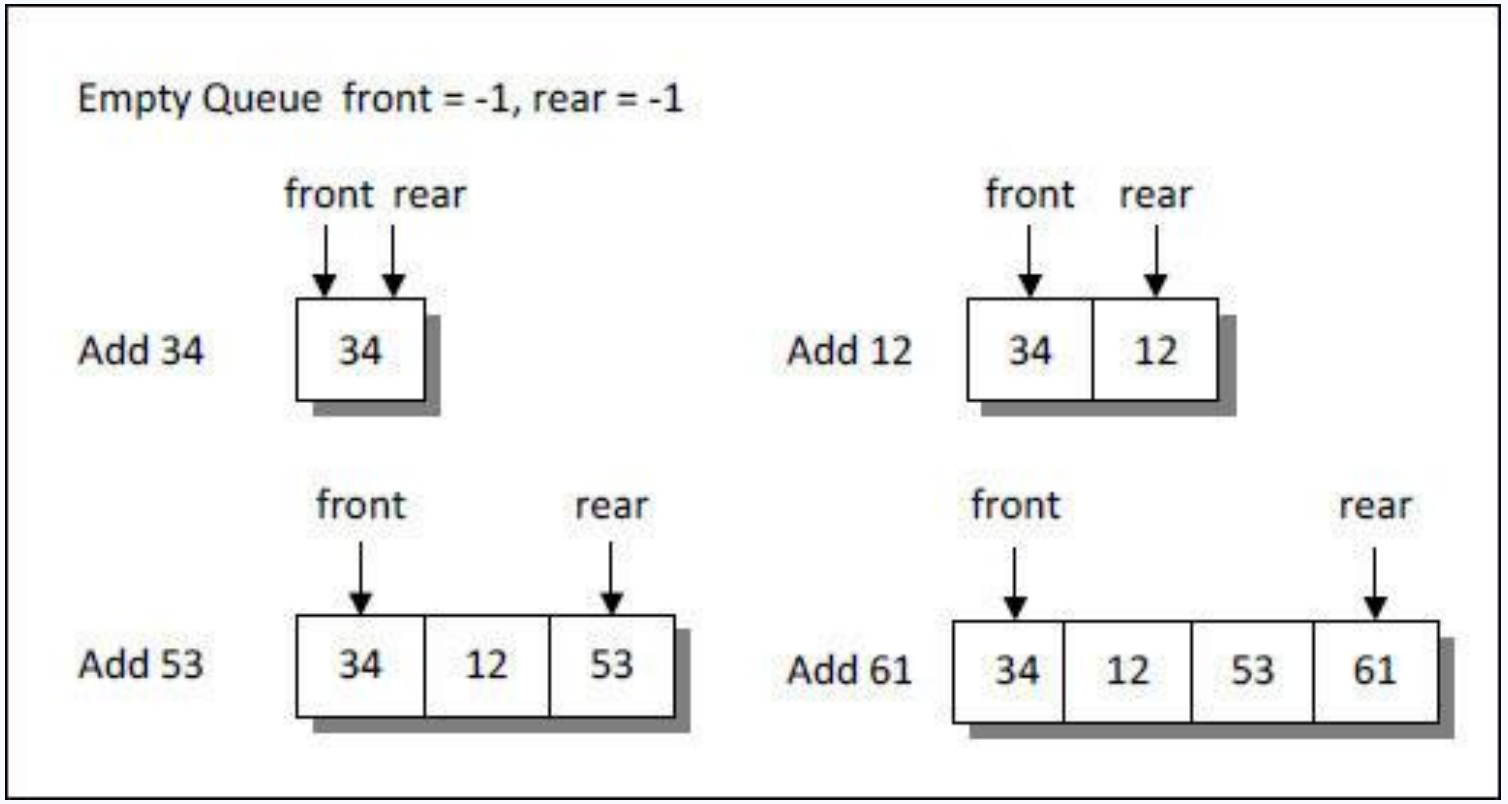
- Two variables **front** and **rear** are declared to monitor the two ends of the queue.
- The initial values of **front** and **rear** are set to -1 to mark the queue as empty.

ADDING AN ELEMENT TO THE QUEUE

```
void queue:: addq (int item)
{
      if (rear == MAX - 1)
      {
            cout << "Queue is full" << endl;
            return;
      }
      rear++;
      arr[rear] = item;
      if (front == -1)
            front = 0;
}
```

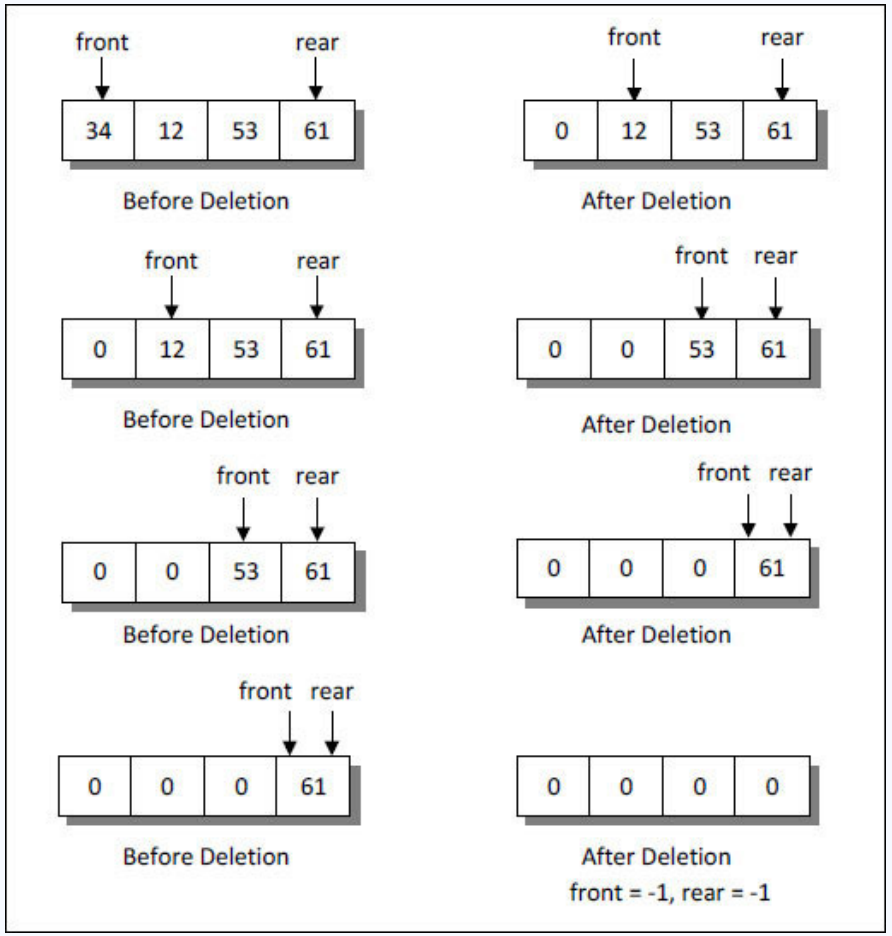
- Since the array indexing begins with 0 the maximum number of elements that can be stored in the queue are MAX - 1.
- If an element can be added to the queue then value of **rear** is incremented and the new item is stored in the array.
- If the item being added to the queue is the first element then the queue **front** is set to 0 indicating that the queue is no longer empty.

ADDITION OF AN ELEMENT TO A QUEUE



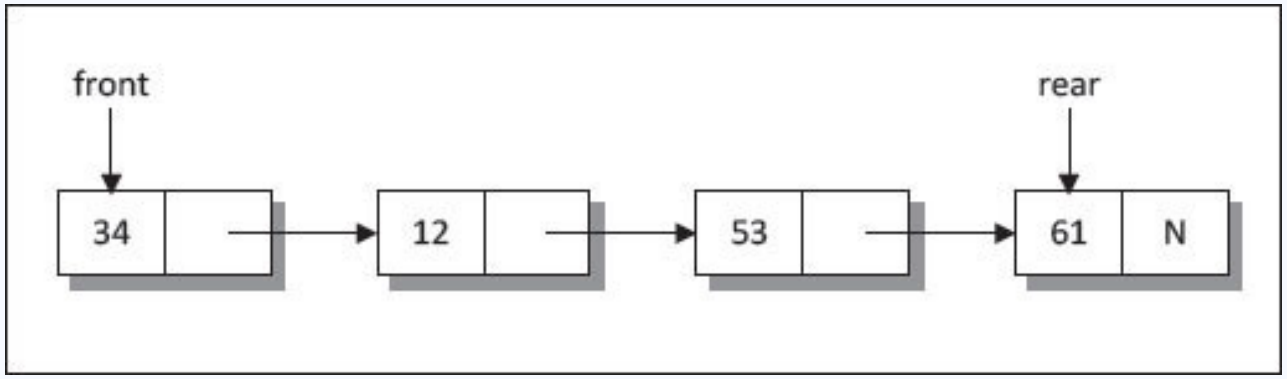
REMOVING AN ELEMENT FROM THE QUEUE

```
int queue:: delq()
{
      int data;
      if (front == -1)
      {
            cout << "Queue is Empty" << endl;
            return NULL;
      }
      data = arr[front];
      arr[front] = 0;
      if (front == rear)
            front = rear = -1;
      else
            front++;
      return data;
}
```



QUEUE AS A LINKED LIST

- Linked lists do not have any restrictions on the number of elements it can hold.
- Space for the elements in a linked list is allocated dynamically.
  - It can grow as long as there is enough memory available for dynamic allocation.



بدل استخدام Array ذات الحجم الثابت،
نستخدم Linked List.
وهذا يحل أغلب مشاكل الـ Array Queue 👌
لماذا نستخدم Linked List؟ لأن الـ Linked List:
✅ لا تحتاج حجم ثابت
✅ تكبر وتصغر ديناميكيًا
✅ لا يوجد هدر للمساحات
✅ لا يوجد Overflow إلا إذا انتهت الذاكرة
الفكرة الأساسية: كل عنصر في الـ Queue يكون
عبارة عن Node تحتوي على:
القيمة → Data
مؤشر للعنصر التالي → Next
Node يشير لأول Front
Node يشير لآخر Rear

IMPLEMENTATION QUEUE AS A LINKED LIST

```
#include
using namespace std;
class queue
{
      private:
            struct node
            {
                  int data;
                  node *link;
            } *front, *rear;
      public:
            queue();
            void addq (int item);
            int delq();
            ~queue();
};

// initializing a data member
queue:: queue()
{
front = rear = NULL;
}
```

- The queue is empty hence both **front** and **rear** are set to NULL.

ADDING A NEW NODE TO THE QUEUE

```
void queue:: addq (int item)
{
      node *temp;
      temp = new node;
      if (temp == NULL){
            cout << "Queue is full" << endl;
            return;}
      temp -> data = item;
      temp -> link = NULL;
      if (front == NULL)
      {
            rear = front = temp;
            return;
      }
      rear -> link = temp;
      rear = rear -> link;
}
```

- If the element added is the first element, then both **front** and **rear** are made to point to the new node.
  - If the element added is not the first element then only **rear** is made to point to the new node,
  - whereas **front** continues to point to the first node in the list

REMOVING AN ELEMENT FROM THE QUEUE

```
int queue:: delq()
{
      if (front == NULL)
      {
      cout << "Queue is empty" << endl;
      return NULL;
      }
      node *temp;
      int item;
      item = front -> data;
      temp = front;
      front = front -> link;
      delete temp;
      return item;
}
```

- Removal of an element from the list actually deletes the node to which **front** is pointing.
- After deletion, **front** is made to point to the next node that comes in the list, whereas **rear** continues to point to the last node in the list.

DEALLOCATING MEMORY

```
queue:: ~queue()
{
      if (front == NULL)
            return;
      node *temp;
      while (front != NULL)
      {
            temp = front;
            front = front -> link;
            delete temp;
      }
}
```

CIRCULAR QUEUE

- The queue using an array suffers from one limitation.
  - The queue is reported as full (since rear has reached the end of the array), even though in actuality there might be empty slots at the beginning of the queue
- To overcome this limitation the queue can be implemented as a circular queue.
  - When elements are added to the queue and the end of the array is reached, the next element is stored in the first slot of the array.
- In a circular queue, the queue would be reported as full only when all the slots in the array stand occupied.

لماذا نحتاج Circular Queue؟
تذكري مشكلة الـ Queue العادي باستخدام Array:
بعد عدة عمليات حذف، تظهر فراغات بالبداية لكن لا نستفيد منها.
مثال:
الحجم = 5
[50] [40] [30] [20] [10]
إذا حذفنا عنصرين:
[50] [40] [30] [ ] [ ]
أصبح:
Front = 2
Rear = 4
الآن إذا أردنا إضافة عنصر جديد:
❌ الـ Rear وصل لنهاية الـ Array
فيعتقد النظام أن الـ Queue ممتلئ!
مع أن هناك أماكن فارغة بالبداية.
الحل: Circular Queue
نجعل الـ Queue دائريًا.
يعني:
بعد آخر خانة
نعود لأول خانة مرة أخرى

IMPLEMENTATION OF CIRCULAR QUEUE

```
#include
using namespace std;
const int MAX = 8;
class queue
{
      private:
            int arr[MAX];
            int front, rear;
      public:
            queue();
            void addq (int item);
            int delq();
            void display();
            ~queue();
};
// initializing a data member
queue:: queue()
{
      front = rear = -1;
      for (int i = 0; i < MAX; i++)
            arr[i] = 0;
}
```

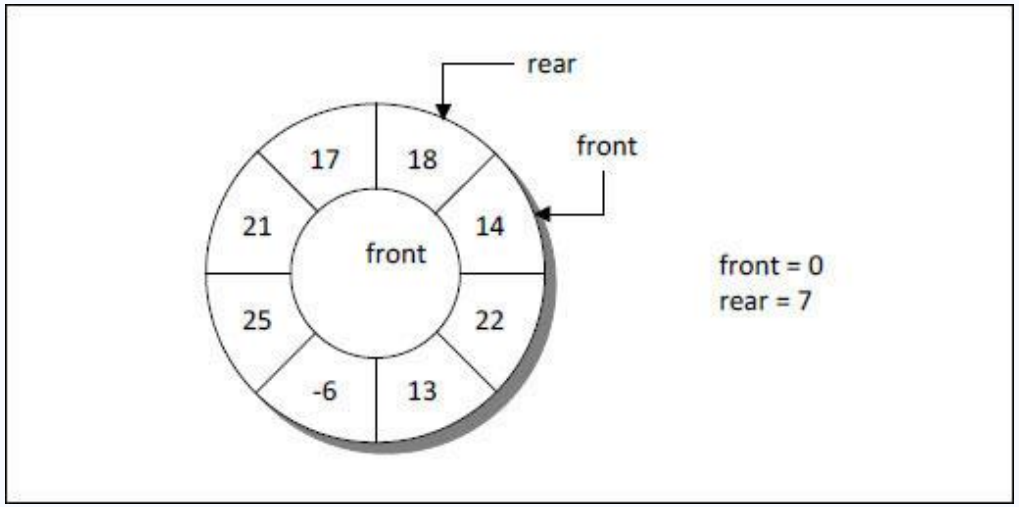
ADDING AN ELEMENT TO THE CIRCULAR QUEUE

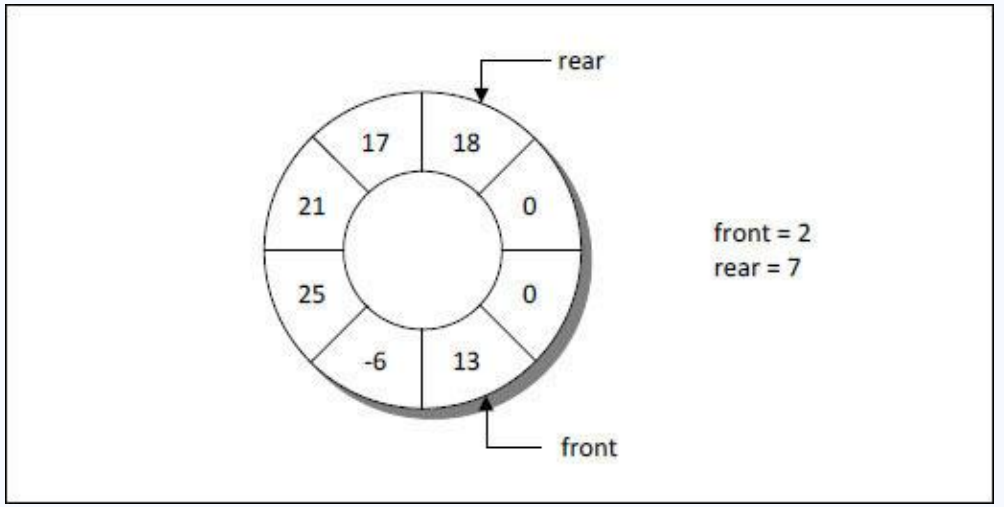
```
void queue:: addq (int item)
{
      if ((rear == MAX - 1 && front == 0)  ||  (rear + 1 == front))
      {
            cout << "Queue is full" << endl;
            return;
      }
      if (rear == MAX - 1)
            rear = 0;
      else
            rear++;
      arr[rear] = item;
      if (front == -1)
            front = 0;
}
```

REMOVING AN ELEMENT FROM THE QUEUE

```
int queue:: delq()
{
      int data;
      if (front == -1)
      {
      cout << "Queue is empty" << endl;
      return NULL;
      }
      data = arr[front];
      arr[front] = 0;
      if (front == rear)
      {
            front = -1;
            rear = -1;
      }
      else
      {
            if (front == MAX - 1)
                  front = 0;
            else
                  front++;
      }
      return data;
}
```

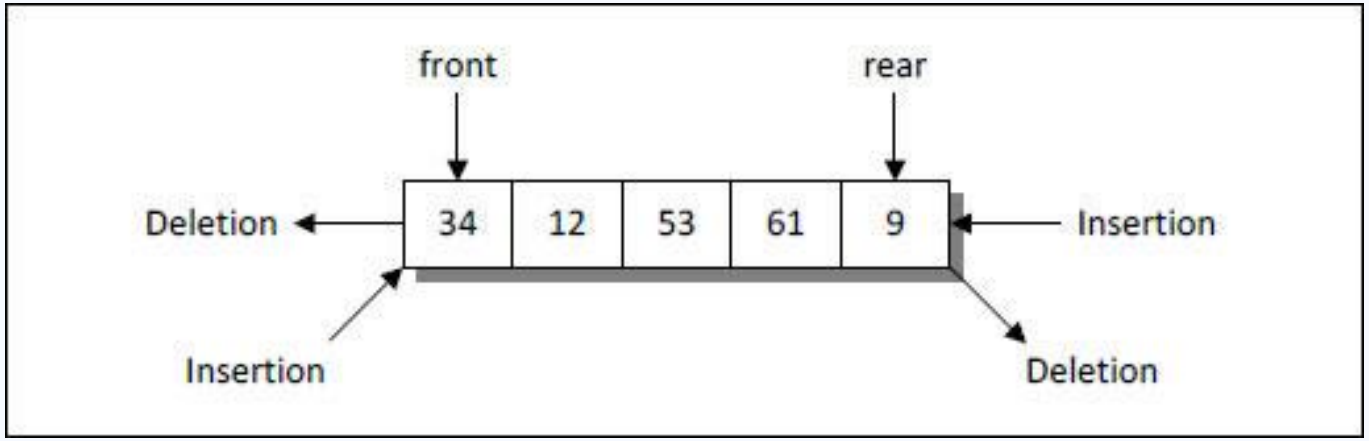
CIRCULAR QUEUE AFTER ADDITION OR DELETION





DEQUE

- The word deque is a short form of double-ended queue
  - Items can be added or deleted at either the front or rear
  - No changes can be made elsewhere in the list.
- It is a generalization of both a stack and a queue.



DEQUEUE – CONT.

- Two variations of a deque
  - Input-restricted deque
    - It restricts the insertion of elements at one end only, but the deletion of elements can be done at both the ends of a queue
  - Output-restricted deque.
    - It restricts the deletion of elements at one end only, and allows insertion to be done at both the ends of a deque.

PRIORITY QUEUE

- A priority queue is a collection of elements where the elements are stored according to their priority levels.
  - The order in which the elements should get added or removed is decided by the priority of the element.
- Rules are applied to maintain a priority queue.

(a) The element with a higher priority is processed before any element of lower priority.

(b) If there are elements with the same priority, then the element added first in the queue would get processed