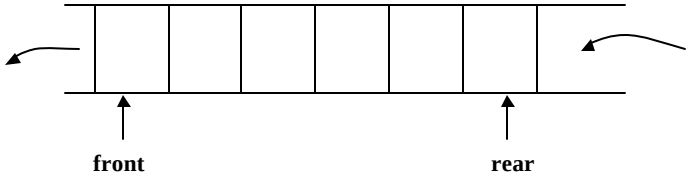
QUEUES

- A queue is a list from which items may be deleted at one end (front) and into which items may be inserted at the other end (rear)

- Similar to checkout line in a grocery store - first come first served.



- It is referred to as a first-in-first-out (FIFO) data structure.

- Queues have many applications in computer systems:
  - jobs in a single processor computer
  - print spooling
  - information packets in computer networks.

- **Primitive operations**

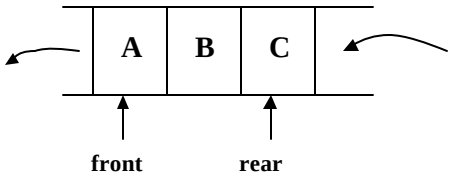
  `enqueue (q, x):` inserts item x at the rear of the queue q

  `x = dequeue (q):` removes the front element from q and returns its value.

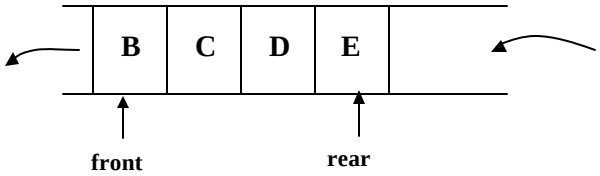
  `isEmpty(q)` : true if the queue is empty, otherwise false.

Example

```
enqueue(q, 'A');
enqueue(q, 'B');
enqueue(q, 'C');
x = dequeue(q);
enqueue(q, 'D');
enqueue(q, 'E');
```

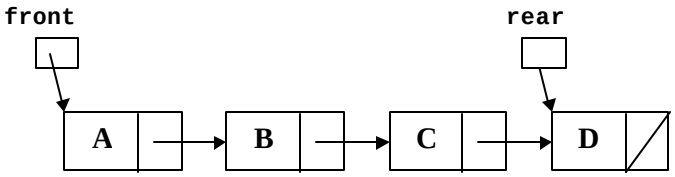


x= dequeue (q) -> x= 'A'



Linked List Implementation

We need to keep two pointers: front and rear




```
struct queueNode{
   char data;
   struct queueNode * next;
};

struct queue{
   struct queueNode *front;
   struct queueNode *rear;
};
```

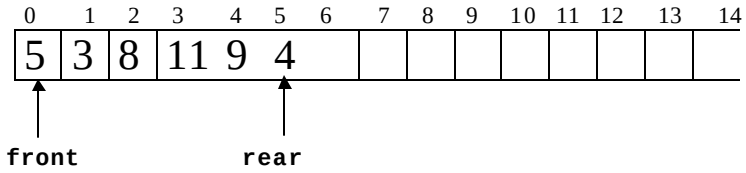

Inserting a node:

```
void enqueue(struct queue *q, char value)
{
    struct queueNode * newPtr;

    newPtr = malloc(sizeof(struct queueNode));
    if (newPtr != NULL) {
        newPtr->data = value;
        newPtr->next = NULL;
        if (isEmpty(*q))
            q->front = newPtr;
        else
            q->rear->next = newPtr;
        q->rear = newPtr;
    }
    else
        printf("%c is not inserted. No memory "
               "available.\n", value);
}
```

```
char dequeue(struct queue *q)
{
    char value;
    struct queueNode * tempPtr;

    value = q->front->data;
    tempPtr = q->front;

    q->front = q->front->next;
    if (q->front == NULL)
        q->rear = NULL;
    free (tempPtr);
    return value;
}

int isEmpty(struct queue q)
{
    return q.front == NULL;
}
```


Array Implementation

A huge array and two variables (indices) front and rear to point the first and the last elements of the queue.




```
struct queue{
    int items[MAX];
    int front;
    int rear;
};

struct queue q;
```

Initially:

```
    q.rear = -1;
    q.front = 0;
/* queue is empty when rear < front */
```

- Addition and deletion are simple.
- Good if the queue is often emptied.
- Disadvantage: needs a huge array.

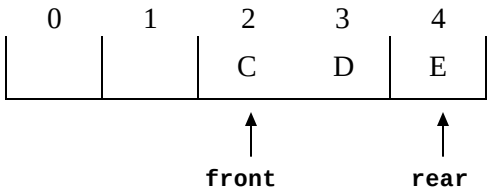
Ignoring overflow and underflow, insert and remove can be implemented as:

/* number of elements in the queue = rear – front + 1 */

```
enqueue(q, x):
    q.rear = q.rear +1;
    q.items[q.rear] = x;

x = dequeue(q):
    x = q.items[q.front];
    q.front = q.front + 1;
```

Problems with this representation:

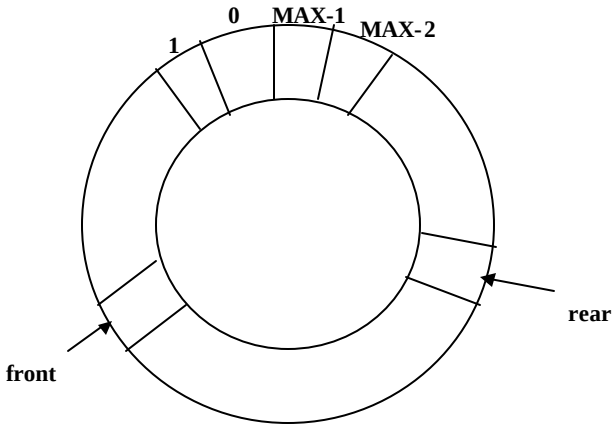
Although there is space we may not be able to add a new item. An attempt will cause an overflow.



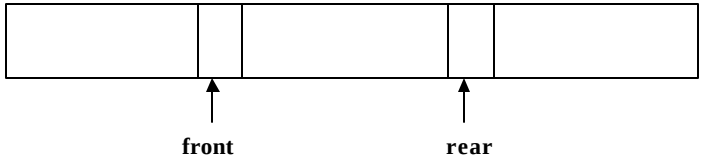
It is possible to have an empty queue yet no new item can be inserted.

A Solution: Circular Array

– A good method to implement queues (efficient use of space) is to view the array as if it is a circular array.



equivalently:



– when we pass the MAX-1, we return to 0.

– to increment index in a circular array:

```
if (i == MAX-1)
  i = 0;
else i = i+ 1;
```

(i.e. use % operator)

- The condition rear < front is no longer valid as a test for empty queue.

- One solution: Keep a counter that holds the number of elements in the queue.

```
struct queue{
    int count;
    int front;
    int rear;
    int items[max];
};
```

```
void function initialize (struct queue *q)
{
    q->count = 0;
    q->front = 0;
    q->rear = -1;
}
```

```
int isEmpty(struct queue q)
{
    return (q.count == 0);
}
```

```
int isFull(struct queue q)
{
    return (q.count == max);
}
```



```
void enqueue(struct queue *q, int x)
{
    if (q-> count == max)
        printf("%d is not inserted. Queue is "
                    "full.\n", x);
    else{
        q->count  = q->count + 1;
        q->rear = (q->rear + 1) % max;
        q->items[rear] = x;
    }
}
```

```
int dequeue(struct queue *q)
{
    int x;

    q->count = q->count –1;
    x = q->items[front];
    q->front = (q->front + 1)% max;
    return x;
}
```

Exercises

- Empty one stack onto the top of another stack.
- Move all items from a queue to a stack.
- Start with a queue and an empty stack and use the stack to reverse the order of all items in the queue.
- How can you implement a queue of stacks?
- How can you implement a stack of queues?