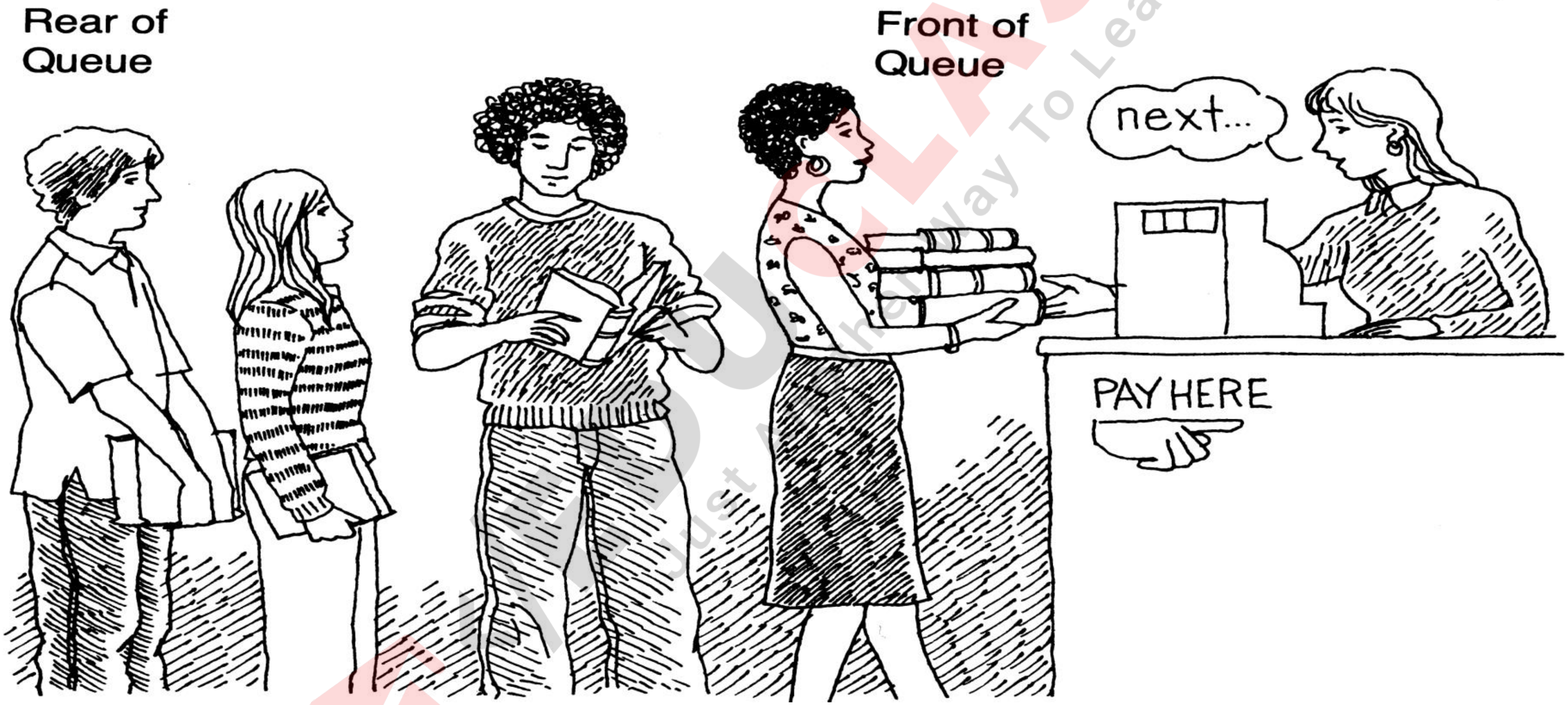
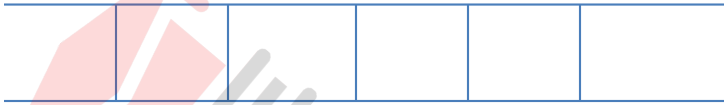
Queues



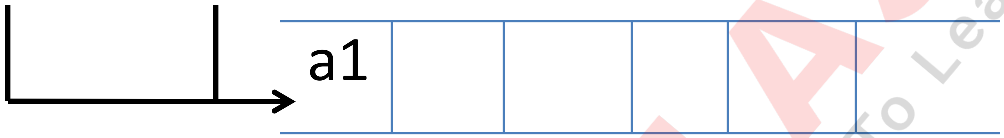
Queues

- A linear list of elements in which
  - – Inserting can take place at the other end called **rear.**
  - – deletion can take place only at one end called **front.**
- Queue principle : FIFO
- Conditions
  - – If there is no element in queue/ empty queue
    - **FRONT =-1 ; REAR =-1**



Queues

- If there is exactly one element in queue
  - FRONT = REAR



- Whenever an element is added to the queue,

  REAR =REAR +1 or REAR++

- If the item being added is the first element, FRONT=0, indicating the queue is no longer empty.

Operations on queues

- Four basic operations on queue
  - **Enqueue :**
    - Queue insert operation is called as **enqueue.**
    - The data is inserted in the rear.
    - If there is not room to insert another data in the queue , the queue is said to be in an **overflow state**.
    - Diagram
    - algorithm

Algorithm Enqueue(struct queue *q, int item)

To add the data to the queue using array implementation

pre: struct stack *s : pointer to the queue structure

item : data to be pushed in the stack

post : to add data to the queue

```
1.[check if array is full]
    if (q->rear=ARR-1)
        print "\n QUEUE IS  FULL"
2. [increment rear and add item
    to the array]
    1. q->rear++
    2. q->a[q->rear]=item;
    3. count++;
3. [if the array is empty
    (front =-1) ,set front=0]
    if(q->front =-1)
        q->front=0
```

Operations on queues

– **Dequeue:**

- Queue delete operation is called as **dequeue.**
- Data is removed from the front of the queue.
- If there are no data in the queue, when a dequeue is attempted , then the queue is in an **underflow state.**
- Diagram
- algorithm

Algorithm int dequeue(struct queue *q)

To remove data from the queue using array implementation

pre: struct stack *s : pointer to the queue structure

post:data is removed from the queue and returned to the calling program

```
1. [if the queue is empty return null]
   if(q->front =-1)
       return NULL
2. [fetch data from the front of the queue]
   1. data=q->a[q->front]
   2. q->a[q->front]=NULL;
3. [ if last element was removed from the queue,
   if(q->front=q->rear)
       q->front=q->rear=-1;
   else
       q->front++;
   return data;
```

Operations on queues

- **QueueFront:**
  - Returns data which is at the front of the queue without changing the contents of the queue.
  - If there are no data in the queue, when a queue front is attempted , then the queue is in an **underflow state.**
  - Diagram
  - Algorithm

Assignment

Algorithm int queuefront(struct queue *q)

To display the data which is in the front of the queue

pre: struct stack *s : pointer to the queue structure

Post : data which is in the front of the queue is returned to the calling program

Operations on queues

- **QueueRear:**
  - Returns data which is at the rear of the queue without changing the contents of the queue.
  - If there are no data in the queue, when a queue rear is attempted , then the queue is in an **underflow state.**
  - Diagram
  - Algorithm

Algorithm int queuerear(struct queue *q)

To display the data which is in the rear end of the queue

pre: struct stack *s : pointer to the queue structure

Post : data which is in the rear end of the queue is returned to the calling program

Assignment

Display the contents of the queue

```
Algorithm displayqueue(struct queue *q)
To display the contents of the queue
Pre : struct stack *s : pointer to the queue structure
1.[initialize]
        x=0
2.[if rear=-1 then the queue is empty]
    1.    if(q->rear==-1)
                print “QUEUE IS EMPTY”
                return;
    2. repeat  while(x<=q->rear)
          1. printf q->a[x]
          2. x=x+1
```

Circular Queues

- Disadvantage of linear queues:
  - If an item is removed from the linear array, the space remains unutilized. To overcome this, the circular array is used.
  - Algorithm:

Algorithm to insert data in a Circular Queue

```
Algorithm ins_circular_q(Struct
    queue *cq, int item)
To insert data into circular queue
Pre :
Post:
1.  [Check if the queue is full]
      If q->REAR =MAX -1  and
         q-> Front =0
            print " circular queue is
           full"
            return
2.  If q-> REAR = MAX -1
          q->REAR=0
      else
          increment q->REAR
      if(q->a[q->rear]==0 ||
            q->a[q->rear] ==-999)
          q->a[q->REAR]= item
      else
          printf("QUEUE IS FULL");
3. [If an item is inserted , then
     queue is not empty]
     if q-> front =-1
          q->front =0
```

Algorithm to delete data from a Circular Queue

```
Algorithm int
    delete_from_circular_queue(
   {struct queue *q)
1. [Check if the queue is empty]
    if(q->front ==-1)
         return
2. Assign the value of array to the
    variable data and set the array
    element to 0
          data=a[q->front]
          a[q->front]=0
3.[check if the queue is empty]
    if(q->front==q->rear)
         q->front=q->rear=-1
    else
         if(q->front=MAX-1)
              q->front=0;
         else
              q->front++
    return data
```

Priority Queues

- Collection of elements where each element is assigned a priority and the order in which elements are deleted and processed.
- The following are the rules:
  - All elements of higher priority is before any element of lower priority.
  - 2 elements with the same priority are processed according to the order in which they are added to the queue.

Priority Queues

- Eg: Time Sharing System
  - Programs of high priority are processed first and programs with the same priority form a standard queue.

- Eg:
- Algorithm

Algorithm for inserting data in the priority queue

```
Algorithm add_priority_jobs(struct priorityqueue *pq,struct data
    dt)
Post: to add jobs in a priority queue
1.[declare variables]
    struct data temp
    int i,j
2.  if(pq->rear==MAX -1)
      display " QUEUE IS FULL"
          return
```

```
3.  rear++;
     d[pq->rear]=dt;
      if pq->front==-1
            pq->front =0;
4. Repeat for(i=0;i<=pq->rear;i++)
     1.  Repeat for(j=i+1; j<= pq-> rear;j++)
            A.  If(priority1 >priority2)
                  swap the jobs
             B. If(priority1=priority2)
                 1. if(orderno1>orderno2)
                          swap the jobs
```

Algorithm for displaying data in the priority queue

```
void displaycircularqueue(struct circularqueue *q)
1[initialize]
    x=0
2.  Repeat while(x<=MAX -1)
        1. display  q->a[x]
        2. x++
```