# Unit 4: Queues

a. Concept and Definition
b. Queue as an ADT
c. Implementation of Insert and Delete operation of
    - Linear Queue
    - Circular Queue
d. Concept of Priority Queue

## Concept and Definition

- A ***queue*** is an ordered collection of items from which items may be deleted at one end (called the ***front*** of the queue) and into which items may be inserted at the other end (called the ***rear*** of the queue).
- Since the first element inserted into a queue is the first element to be removed, a queue is also called a **FIFO** (first-in, first-out) list or **LILO** list.
- A line at a bank or at a bus stop or a waiting list of students are real life examples of a queue.

## QUEUE

## TU Exam Question (2065)

- How can you use Queue as an ADT.

## Queue as an ADT

A **queue** is a collection of items, where an item to be added to the queue must be placed at the end of the queue and items that are removed from the queue must be removed from the front. The end of the queue is known as the **rear** and the front of the queue is known as the **front**. The term **enqueue** means to add an item to the queue, and the term **dequeue** means to remove an item from the queue.

- **<u>Operations</u>**

  The operations on a queue are:

  - **enqueue(q, x) -** adds an item **x** to the **rear** of the queue **q**.
  - **dequeue(q) -** remove an item from the **front** of the queue **q**.
  - **front(q) -** access the first item at the **front** of the queue.

## Implementation of Queue using Array

- Use an array to hold the elements of the queue and use two variables, *front* and *rear*, to hold the positions within the array of the first and last elements of the queue.
- Thus a queue ***q*** of integers may be declared as:

```
#define MAXQUEUE 100
struct queue
      {
      int items[MAXQUEUE];
      int front, rear;
      }q;
```

- Now the operation ***enqueue(q, x)*** may be implemented as:

  ***q.items[++q.rear] = x;***
- Similarly the operation ***x=dequeue(q)*** may be implemented as:

  ***x = q.items[q.front++];***
- Initially

  ***q.rear=-1;***

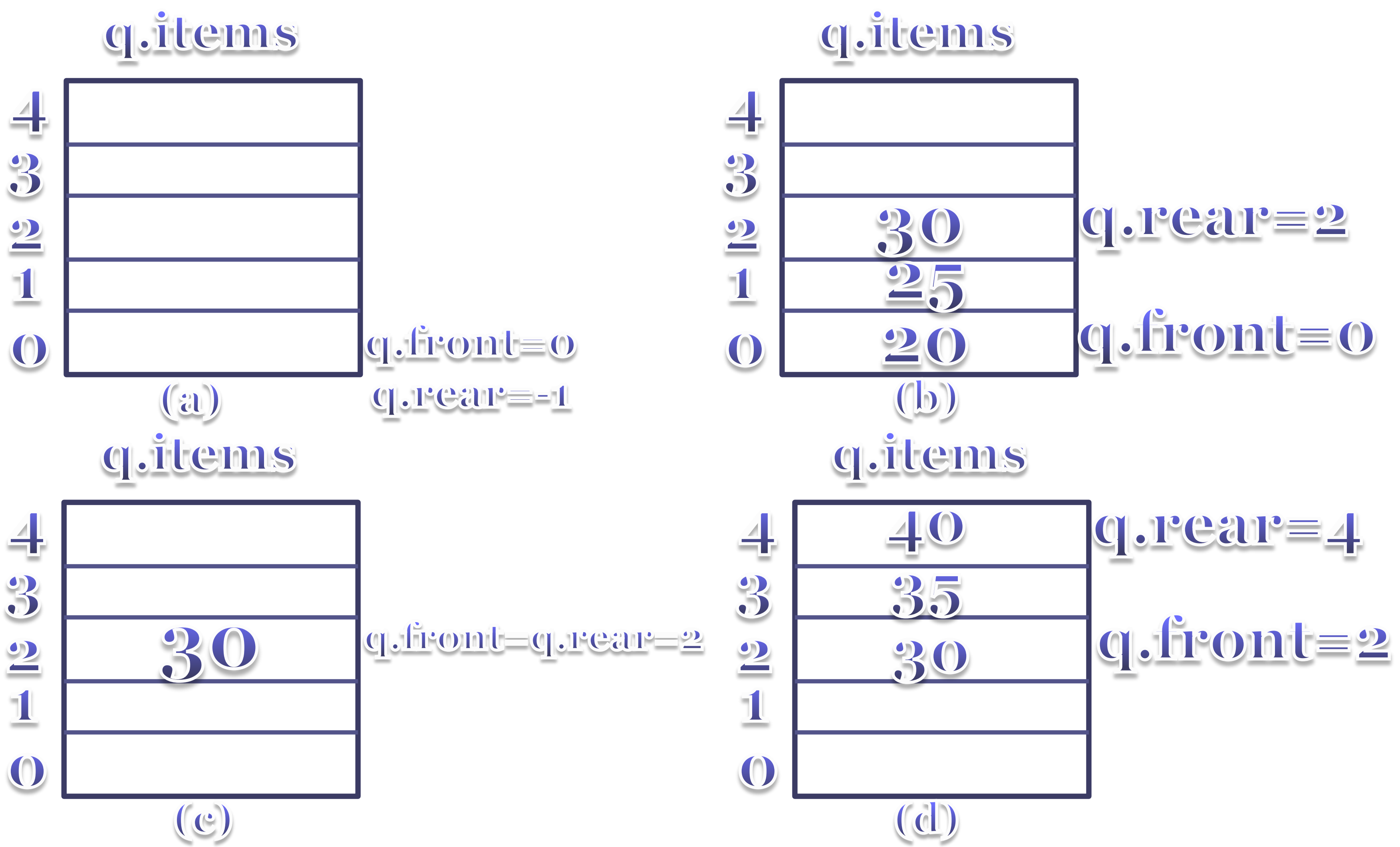
  ***q.front=0;***
- Queue is empty whenever ***q.rear<q.front***.
- When ***q.rear==MAXQUEUE-1***, queue is full.
- No. of elements in the queue at any time is equal to the value of ***q.rear-q.front+1***.

```
#define MAXQUEUE 5
void enqueue(struct queue *,int);
int dequeue(struct queue *);
void display(struct queue *);
struct queue
    {
    int items[MAXQUEUE];
    int front;
    int rear;
    }q;

void main()
{
int option;
char ch='y';
int x;
q.rear=-1;
q.front=0;
clrscr();
printf("What do you want to do???");
printf("\n1.Add item to queue");
printf("\n2.Remove item from queue");
printf("\n3.Display queue");
while(ch=='y')
    {
    printf("\n Enter your option (1, 2 or 3):");
    scanf("%d", &option);
    switch(option)
            {
            case 1:
                        printf("\n Enter item to insert:");
                        scanf("%d", &x);
                        enqueue(&q, x);
                        break;
            case 2:
                        x=dequeue(&q);
                        printf("\n The removed item is:%d", x);
                        break;
            case 3:
                        display(&q);
                        break;
            default:
                        printf("WRONG OPTION");

            }
    printf("\n Do you want to continue (y/n)?");
    scanf(" %c", &ch);
    }
getch();
}
```

```
void enqueue(struct queue *q, int x)
{
if(q->rear==MAXQUEUE-1)
    {
    printf("\n Queue if full.");
    return;
    }
else
    q->items[++q->rear]=x;
}

int dequeue(struct queue *q)
{
if(q->rear<q->front)
    {
    printf("\n Queue is empty.");
    getch();
    exit();
    }
else
    return q->items[q->front++];
}
```

```
void display(struct queue *q)
{
int i;
printf("\n No. of elements in the queue:%d", q->rear-q->front+1);
for(i=q->front;i<=q->rear;i++)
   {
   printf("\nq[%d]=%d", i, q->items[i]);
   }
}
```

## Problem:


q.items
4
3
2
1
0
q.front=0
q.rear=-1
(a)
q.items
4
3
2 30
1 25
0 20
q.rear=2
q.front=0
(b)
q.items
4
3
2 30
1
0
q.front=q.rear=2
(c)
q.items
4 40
3 35
2 30
1
0
q.rear=4
q.front=2
(d)

- In fig.(a), the queue is initially empty.
- In fig.(b), three items 20, 25 and 30 are inserted so that ***q.rear*** is incremented to 2.
- In fig.(c), two items are removed from the queue so that ***q.front=q.rear=2***.
- In fig.(d), two items 35 and 40 are inserted again, so that ***q.rear=4***. Thus, no. of elements in fig.(d) is ***q.rear-q.front+1=4-2+1=3***.
- Since the array (i.e. queue) contains 5 elements, we must have room for the queue to expand without worrying for overflow.
- But to insert another item (say 45) into the queue, ***q.rear*** must be increased by 1 to 5, so that ***q.items[5]=45***. As ***q.items*** is an array of 5 elements, insertion cannot be made.

- **Solution: (Always deleting 0th element strategy)** Modify the ***dequeue*** operation, so that whenever an item is deleted, the entire queue is shifted to the beginning of the array.
- In this representation the queue does not need a ***front*** field, since the element at position 0 of the array is always at the ***front*** of the queue.
- The empty queue in this representation is represented by the queue in which ***rear=-1***.
- The full queue situation is represented by ***rear=MAXQUEUE-1***.
- **For insertion-** increment rear and insert.
- **For deletion-**
  a) delete 0th element and shift all elements forward
  b) set ***rear=rear-1***
- **For display-** from 0th element to rear.

```
#define MAXQUEUE 5
void enqueue(struct queue *,int);
int dequeue(struct queue *);
void display(struct queue *);
struct queue
    {
    int items[MAXQUEUE];
    int rear;
    }q;

void main()
{
int option;
char ch='y';
int x;
q.rear=-1;
clrscr();
printf("What do you want to do???");
printf("\n1.Add item to queue");
printf("\n2.Remove item from queue");
printf("\n3.Display queue");
while(ch=='y')
    {
    printf("\n Enter your option (1, 2 or 3):");
    scanf("%d", &option);
    switch(option)
            {
            case 1:
                        printf("\n Enter item to insert:");
                        scanf("%d", &x);
                        enqueue(&q, x);
                        break;
            case 2:
                        x=dequeue(&q);
                        printf("\n The removed item is:%d", x);
                        break;
            case 3:
                        display(&q);
                        break;
            default:
                        printf("WRONG OPTION");

            }
    printf("\n Do you want to continue (y/n)?");
    scanf(" %c", &ch);
    }
getch();
}
```

```
void enqueue(struct queue *q, int x)
{
if(q->rear==MAXQUEUE-1)
    {
    printf("\n Queue is full.");
    return;
    }
else
    q->items[++q->rear]=x;
}

int dequeue(struct queue *q)
{
int x, i;
if(q->rear==-1)
    {
    printf("\n Queue is empty.");
    getch();
    exit();
    }
else
    {
    x=q->items[0];
    for(i=0;i<q->rear;i++)
            q->items[i]=q->items[i+1];
    q->rear--;
    return x;
    }
}
```

```
void display(struct queue *q)
{
int i;
printf("\n No. of elements in the queue:%d", q->rear+1);
for(i=0;i<=q->rear;i++)
   {
   printf("\nq[%d]=%d", i, q->items[i]);
   }
}
```

## Problem:

- Each deletion involves moving every remaining element of the queue.
- If queue contains 500 to 1000 elements, this is clearly too high price to pay.
- Further, the operation of removing an element from a queue logically involves manipulation of only one element: the one currently at the front of the queue. So the implementation of that operation should reflect this and should not involve a host of extraneous operations.

## Solution: Perform shifting operations only when *rear=MAXQUEUE-1* strategy

- Requires ***front*** index to know when shifting operation on the queue is to be performed.
- Initialization: ***q.front=0*** and ***q.rear=-1***
- Queue is empty: whenever ***q.rear<q.front***
- Queue is full: whenever ***q.rear==MAXQUEUE-1*** and ***q.front==0***
- Insertion: do shifting **if** ***q.rear=MAXQUEUE-1*** and ***q.front>0*** and insert, **else** increment ***q.rear*** and insert
- Deletion: increment ***q.front***
- Display: from ***q.front*** to ***q.rear***

```
#define MAXQUEUE 5
void enqueue(struct queue *,int);
int dequeue(struct queue *);
void display(struct queue *);
struct queue
     {
     int items[MAXQUEUE];
     int front;
     int rear;
     }q;

void main()
{
int option;
char ch='y';
int x;
q.front=0;
q.rear=-1;
clrscr();
printf("What do you want to do???");
printf("\n1.Add item to queue");
printf("\n2.Remove item from queue");
printf("\n3.Display queue");
while(ch=='y')
    {
    printf("\n Enter your option (1, 2 or 3):");
    scanf("%d", &option);
    switch(option)
            {
            case 1:
                        printf("\n Enter item to insert:");
                        scanf("%d", &x);
                        enqueue(&q, x);
                        break;
            case 2:
                        x=dequeue(&q);
                        printf("\n The removed item is:%d", x);
                        break;
            case 3:
                        display(&q);
                        break;
            default:
                        printf("WRONG OPTION");

            }
    printf("\n Do you want to continue (y/n)?");
    scanf(" %c", &ch);
    }
getch();
}
```

```
void enqueue(struct queue *q, int x)
{
int i, j;
if(q->rear==MAXQUEUE-1 && q->front==0)
    {
    printf("\n Queue if full.");
    return;
    }

else if(q->rear==MAXQUEUE-1 && q->front>0)
    {
    for(j=0,i=q->front;i<=q->rear;i++,j++)
            q->items[i-q->front]=q->items[q->front+j];
    q->front=0;
    q->rear=j;
    q->items[q->rear]=x;
    }
else
    q->items[++q->rear]=x;
}
```

```
int dequeue(struct queue *q)
{
int x, i;
if(q->rear<q->front)
     {
     printf("\n Queue is empty.");
     getch();
     exit();
     }

else
     return q->items[q->front++];
}

void display(struct queue *q)
{
int i;
if(q->rear<q->front)
     printf("\n Queue is empty.");
else
     {
     printf("\n No. of elements in the queue:%d", q->rear+1);
     for(i=q->front;i<=q->rear;i++)
               printf("\nq[%d]=%d", i, q->items[i]);
     }
}
```

## TU Exam Question (2065)

- Explain the implementation of stack and queue with example.

## Circular Queue

- A circular queue is a queue in which the insertion of a new element is done at the very first location of the queue, if the last location of the queue is full and the first location is empty.
- In other words, if we have a queue **q** of size say of **N** elements, then after inserting an element to the (**N-1**)th location of the array, the next element will be inserted at the very first location (i.e. location with subscript 0) of the array, only if the first location is empty.
- ***Advantage:*** A circular queue overcomes the problem of unutilized space in linear queue using array.

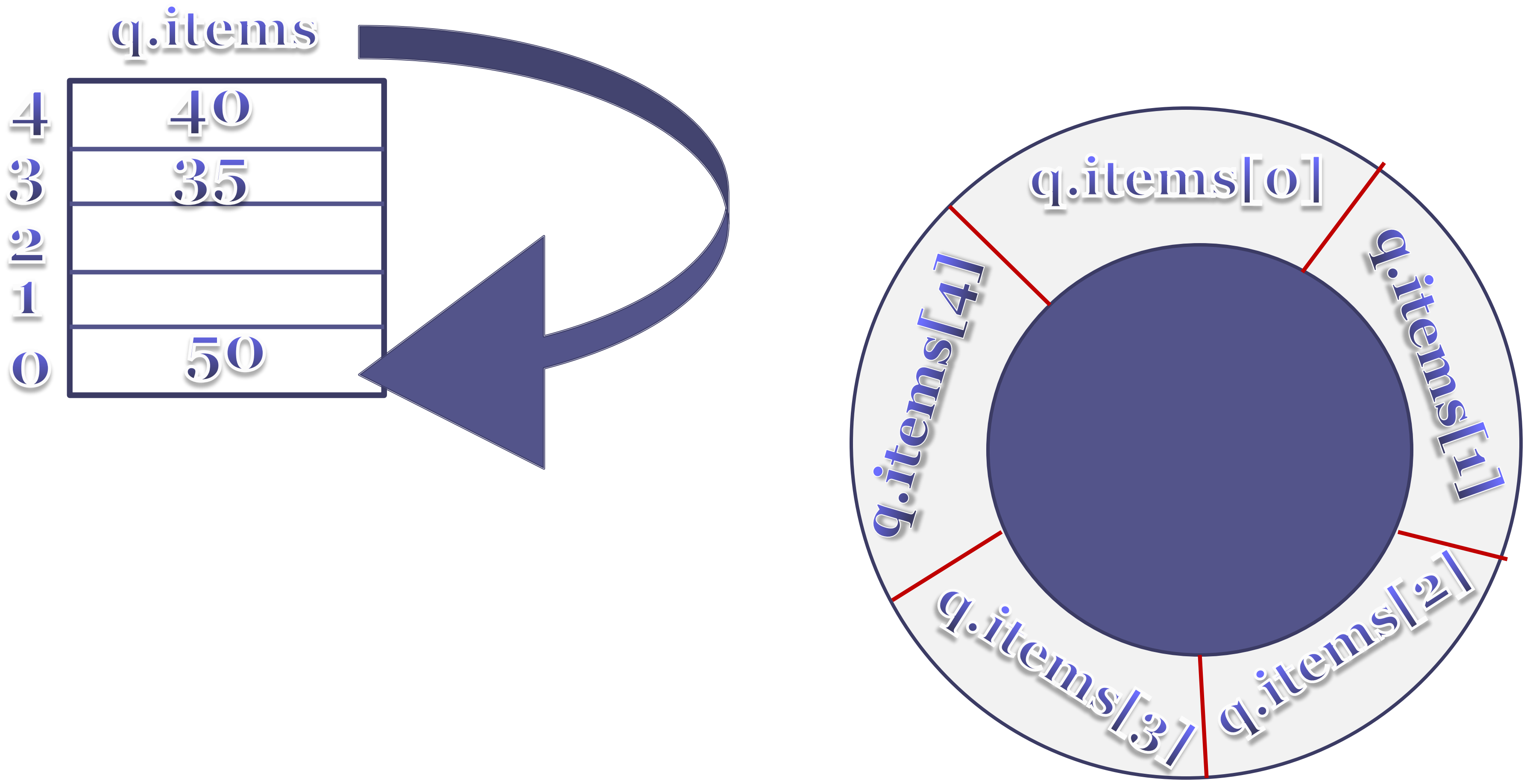


Fig: Circular Queue

## Implementation of Circular Queue using Array

***Assumptions***

- An array of ***items*** is used to hold the elements of the circular queue and two variables, ***front*** and ***rear***, is used to keep the track of the elements to be deleted and inserted and therefore to maintain the unique characteristic of the queue. Thus a circular queue ***q*** of integers may be declared as:

```
#define MAXQUEUE 100
struct queue
        {
        int items[MAXQUEUE];
        int front, rear;
        }q;
```

***Assumptions…***

- The front and rear of the queue are initialized as:

  ***q.front=MAXQUEUE-1;***

  ***q.rear=MAXQUEUE-1;***
- Whenever ***q.front==q.rear***, the queue is empty.
- If ***(q.rear+1)%MAXQUEUE==q.front***, the queue is full. **WHY???**
- **Insertion:** Check queue full otherwise increment rear ***(q.rear=(q.rear+1)%MAXQUEUE)*** and insert.
- **Deletion:** Check queue empty otherwise return the (***q.front+1)%MAXQUEUE*** element and increment front ***(q.front=(q.front+1)%MAXQUEUE)***.
- **Display:** From ***(q.front+1)%MAXQUEUE*** until ***(q.rear+1)%MAXQUEUE***.

## Note:

- In the circular queue representation, we have to sacrifice one element of the array and allow the queue to grow only as large as one less than the size of the array. Thus if an array of 100 elements is declared as a queue, the queue may contain up to 99 elements only.
- The sacrifice is done to distinguish between empty queue and full queue condition.
  - Queue Empty: ***q.rear==q.front***
  - Queue Full: Consider an array of 5 elements declared as a queue. Initially, ***q.rear=q.front=4***. Let's insert items 10, 20, 30 and 40. At this moment, ***q.front=4*** and ***q.rear=3***. If we allow ***q.rear*** to increment again (say by adding another element 50) to ***q.rear=4***, then the condition ***q.rear=q.front*** is satisfied which is actually the condition of empty queue… … … … … … … **PROBLEM**

***"Allow the array of the queue to grow one less than the size of the array"***

```
#define MAXQUEUE 5

void enqueue(struct queue *, int);
int dequeue(struct queue *);
void display(struct queue *);

struct queue
    {
    int items[MAXQUEUE];
    int front, rear;
    }q;

void main()
{
int option;
int x;
char ch='y';
q.front=MAXQUEUE-1;
q.rear=MAXQUEUE-1;
clrscr();
printf("\n What do you want to do?");
printf("\n1.Insert item into queue");
printf("\n2.Remove item from queue");
printf("\n3.Display queue");
while(ch=='y')
    {
    printf("\n Enter your option:");
    scanf("%d", &option);
    switch(option)
            {
            case 1:
                printf("\n Enter item to insert:");
                scanf("%d", &x);
                enqueue(&q, x);
                break;
            case 2:
                x=dequeue(&q);
                printf("\n The removed item is:%d", x);
                break;
            case 3:
                display(&q);
                break;
            default:
                        printf("\n Invalid Option");

            }
    printf("\n Do you want to continue (y/n)?:\t");
    scanf(" %c", &ch);
    }
getch();
}
```

```
void enqueue(struct queue *q, int l)
{
if((q->rear+1)%MAXQUEUE==q->front)
    {
    printf("\n Queue is full");
    return;
    }
q->rear=(q->rear+1)%MAXQUEUE;
q->items[q->rear]=l;
}

int dequeue(struct queue *q)
{
int x;
if(q->rear==q->front)
    {
    printf("\n Queue is empty");
    exit();
    }
x=q->items[(q->front+1)%MAXQUEUE];
q->front=(q->front+1)%MAXQUEUE;
return x;
}
```

```
void display(struct queue *q)
{
int i;
if(q->rear==q->front)
    printf("\n Queue is empty");
else
    {
    printf("\n Items of queue are:");
    for(i=(q->front+1)%MAXQUEUE;i!=(q->rear+1)%MAXQUEUE;i=(i+1)%MAXQUEUE)
            printf("\n q[%d]=%d", i, q->items[i]);
    }

}
```

## TU Exam Question (2066)

- Write C function to display all items in a circular queue in array implementation. Write assumptions you need.

## Priority Queue

- The ***priority queue*** is a queue data structure in which each item in the queue has a "***priority value***" associated with it. This priority determines the intrinsic ordering of the elements in the queue and the result of its basic operations.
- There are two types of priority queues: an ***ascending priority queue*** and a ***descending priority queue***.
- An ***ascending priority queue*** is a collection of items into which items can be inserted arbitrarily and from which only the smallest item can be removed.
- If ***apq*** is an ascending priority queue, the operation ***pqinsert(apq, x)*** inserts element ***x*** into ***apq*** and ***pqmindelete(apq)*** removes the minimum element from ***apq*** and returns its value.

## Priority Queue…

- A ***descending priority queue*** is a collection of items into which items can be inserted arbitrarily and from which only the largest item can be removed.
- If ***dpq*** is a descending priority queue, the operation ***pqinsert(dpq, x)*** inserts element ***x*** into ***dpq*** and ***pqmaxdelete(dpq)*** removes the maximum element from ***dpq*** and returns its value.

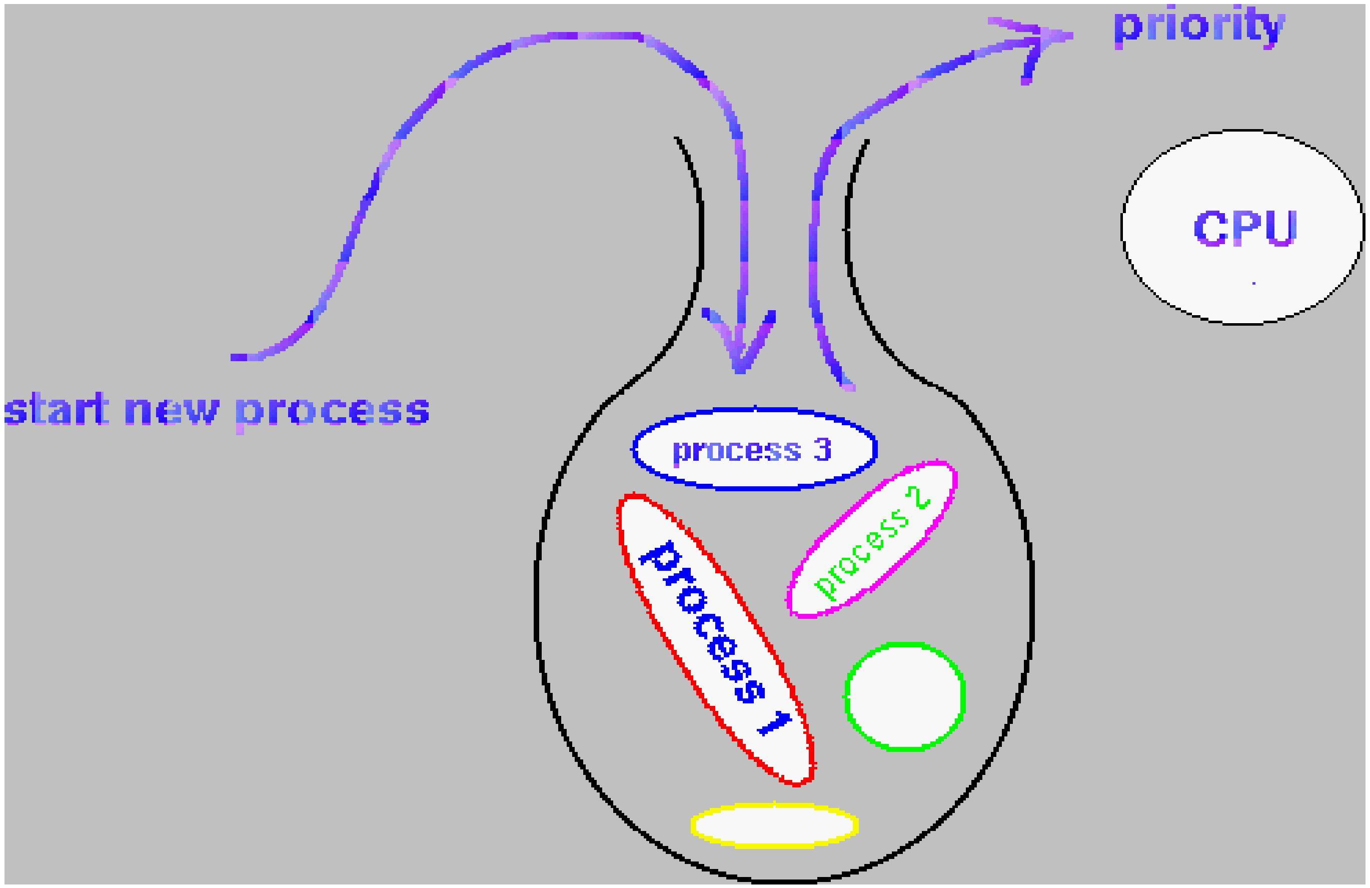


**Fig: Priority Queue**

**(The memory contains multiple processes and each process has a priority associated with it. The process with the highest priority is allowed to execute first.)**

## Model Question (2008)

***Explain the concept of priority queue with an example.***

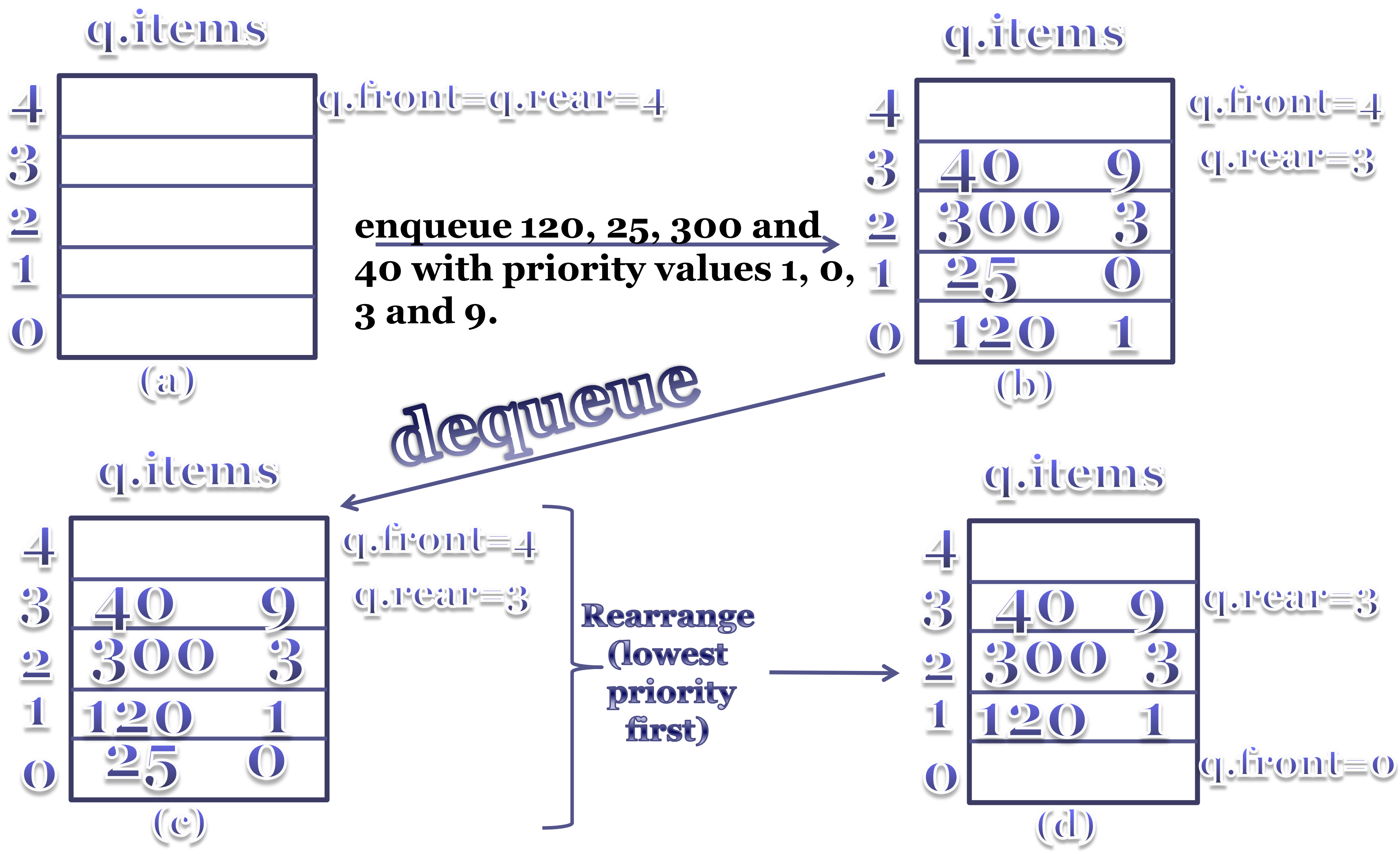
- ***Answer:*** A priority queue is a queue data structure in which a "***priority***" is maintained for each data item and the result of the queue operations like ***enqueue*** and ***dequeue*** is based on the "***priority***" which is set for the queue data items.

  So a priority queue ***q*** may be declared as-

```
#define MAXQUEUE 5
struct queue
        {
        int items[MAXQUEUE];
        int front, rear;
        int priority[MAXQUEUE];
        }q;
```

In fact, there are two types of priority queues: ***ascending priority queue*** and ***descending priority queue***. In both types of priority queues, the data items are ***enqueued*** arbitrarily, however we can only ***dequeue*** the data item with the minimum priority (at an instance of the queue) from the ascending priority queue and the data item with the maximum priority (at an instance of the queue) from the descending priority queue.

**Example:** Let ***q*** be an ascending priority queue (implemented as a circular queue using array) of size 5. Initially, ***q*** has no data items and ***q.front=MAXQUEUE-1*** and ***q.rear=MAXQUEUE-1***.



**enqueue 120, 25, 300 and 40 with priority values 1, 0, 3 and 9.**

## Assignment

- ***Implement Priority Queue.***

## Array Implementation of Ascending Priority Queue using Circular Queue Concept

```
#define MAXQUEUE 5

void enqueue(struct queue *, int);
int dequeue(struct queue *);
void display(struct queue *);

struct queue
   {
   int items[MAXQUEUE];
   int front, rear;
   int priority[MAXQUEUE];
   }q;
```

```
void main()
{
int option;
int x;
char ch='y';
q.front=MAXQUEUE-1;
q.rear=MAXQUEUE-1;
clrscr();
printf("\n What do you want to do?");
printf("\n1.Insert item into queue");
printf("\n2.Remove item from queue");
printf("\n3.Display queue");
    while(ch=='y')
    {
    printf("\n Enter your option:");
    scanf("%d", &option);
    switch(option)
            {
            case 1:
                 printf("\n Enter item to insert:");
                 scanf("%d", &x);
                 enqueue(&q, x);
                 break;
```

```
   case 2:
              x=dequeue(&q);
              printf("\n The removed item is:%d",x);
              break;
   case 3:
              display(&q);
              break;
   default:
                  printf("\n Invalid Option");

         }
   printf("\n Do you want to continue (y/n)?:\t");
   scanf(" %c", &ch);
   }
getch();
}
```

```
void enqueue(struct queue *q, int l)
{
if((q->rear+1)%MAXQUEUE==q->front)
  {
  printf("\n Queue is full");
  return;
  }
q->rear=(q->rear+1)%MAXQUEUE;
q->items[q->rear]=l;
printf("\n Enter priority:");
scanf("%d",&q->priority[q->rear]);
}
```

```
int dequeue(struct queue *q)
{
int x;
int temp;
int p1,p2;
int i,j;
if(q->rear==q->front)
    {
    printf("\nQueue is empty");
    exit();
    }
else
    {
    for(i=(q->front+1)%MAXQUEUE;i!=q->rear%MAXQUEUE;i=(i+1)%MAXQUEUE)
            {
            for(j=(i+1)%MAXQUEUE;j!=(q->rear+1)%MAXQUEUE;j=(j+1)%MAXQUEUE)
                    {
                    if(q->priority[i]>q->priority[j])
                            {
                            p1=q->priority[i];
                            p2=q->priority[j];
                            temp=q->items[i];
                            q->items[i]=q->items[j];
                            q->priority[i]=p2;
                            q->items[j]=temp;
                            q->priority[j]=p1;
                            }
                    }
            }
    }
x=q->items[(q->front+1)%MAXQUEUE];
q->front=(q->front+1)%MAXQUEUE;
return x;
}
```

```
void display(struct queue *q)
{
int i;
if(q->rear==q->front)
    printf("\n Queue is empty");
else
    {
    printf("\n Items of queue are:");
    for(i=(q->front+1)%MAXQUEUE;i!=(q->rear+1)%MAXQUEUE;i=(i+1)%MAXQUEUE)
            printf("\nq[%d]=%d, priority=%d", i, q->items[i], q->priority[i]);
    }

}
```

## Applications of Queues

- All types of customer service (like railway/airplane ticket reservation) center softwares are designed using queues to store customer's information.
- Round robin technique for processor scheduling is implemented using queues.
- Printer server (a dedicated computer to which a printer is attached) routines are designed using queues. A number of users (computers) share a printer using printer server, the printer server then spools all the jobs from all the users, to the server's hard disk in a queue. From here, jobs are printed one-by-one according to their number in the queue.