Queues

B.Sc. CSIT

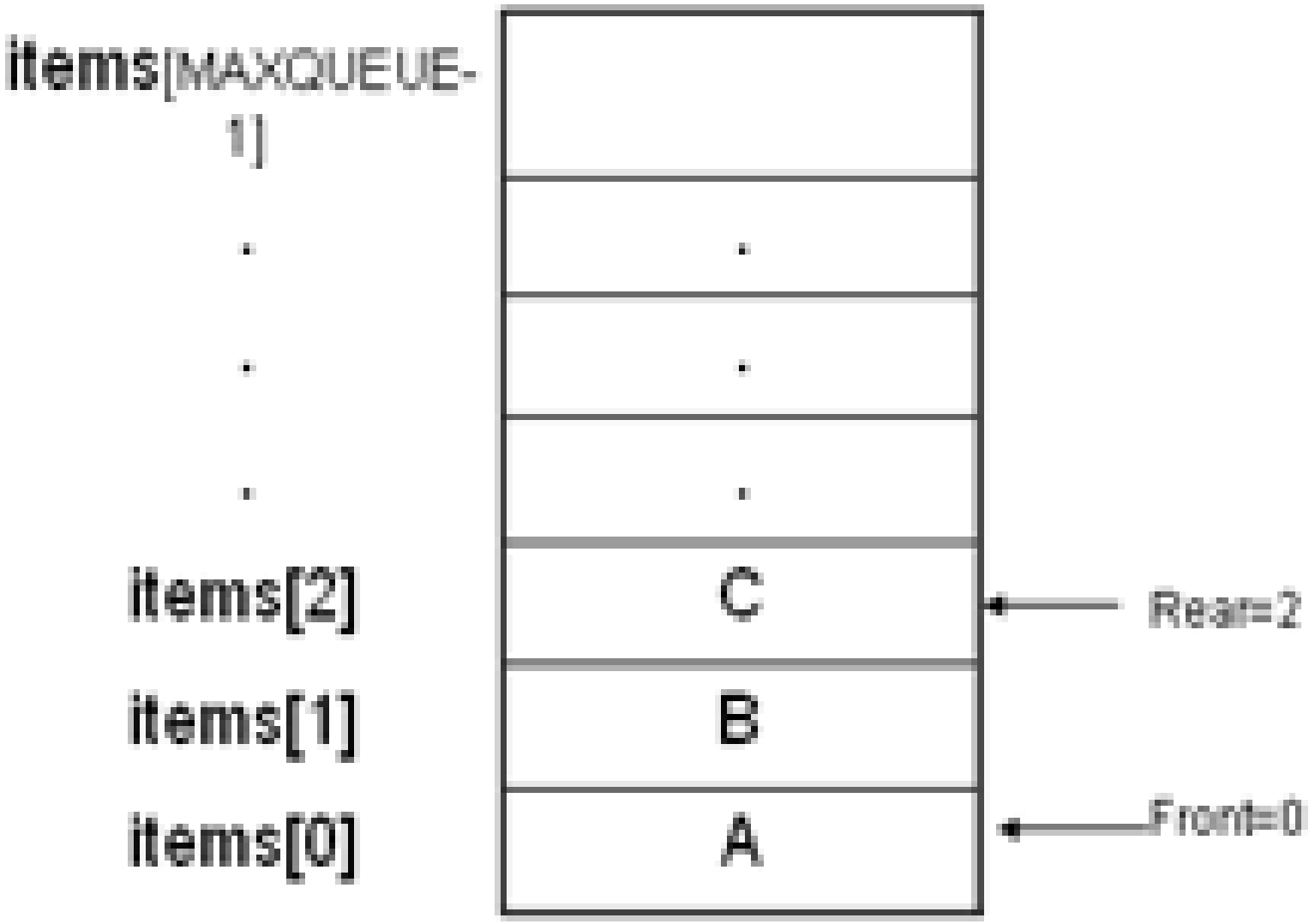
3rd Semester

- ✓ Concept and definition
- ✓ Queue as ADT
- ✓ Implementation of insert and delete operation of
  - · Linear queue
  - · Circular queue
- ✓ Concept of priority queue

What is a queue?

- 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 (the ***rear*** of the queue).
- The first element inserted into the queue is the first element to be removed.
- For this reason a queue is sometimes called a ***fifo*** (first-in first-out) list as opposed to the stack, which is a ***lifo*** (last-in first-out).

Example:

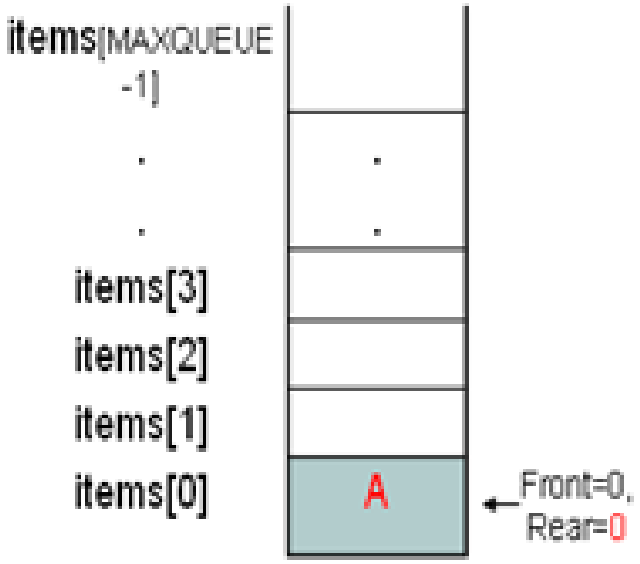


Operations on queue

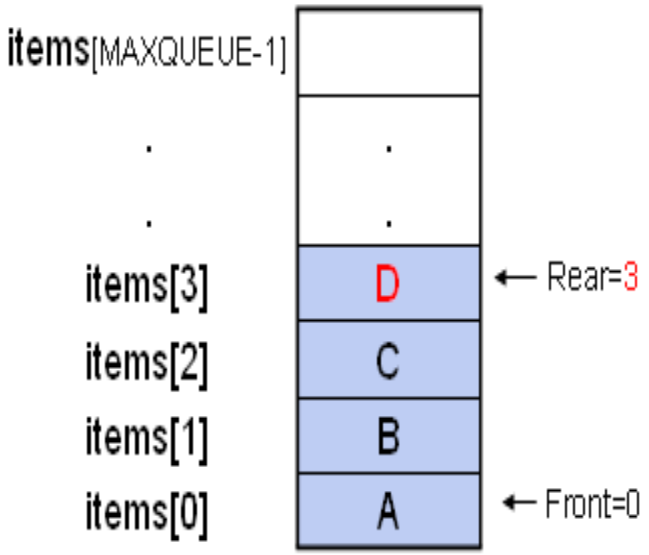
- ***MakeEmpty(q):*** To make q as an empty queue
- ***Enqueue(q, x):*** To insert an item x at the rear of the queue, this is also called by names add, insert.
- ***Dequeue(q):*** To delete an item from the front of the queue q. this is also known as Delete, Remove.
- ***IsFull(q):*** To check whether the queue q is full.
- ***IsEmpty(q):*** To check whether the queue q is empty
- ***Traverse (q):*** To read entire queue that is display the content of the queue.

Cont....

Enqueue(A)

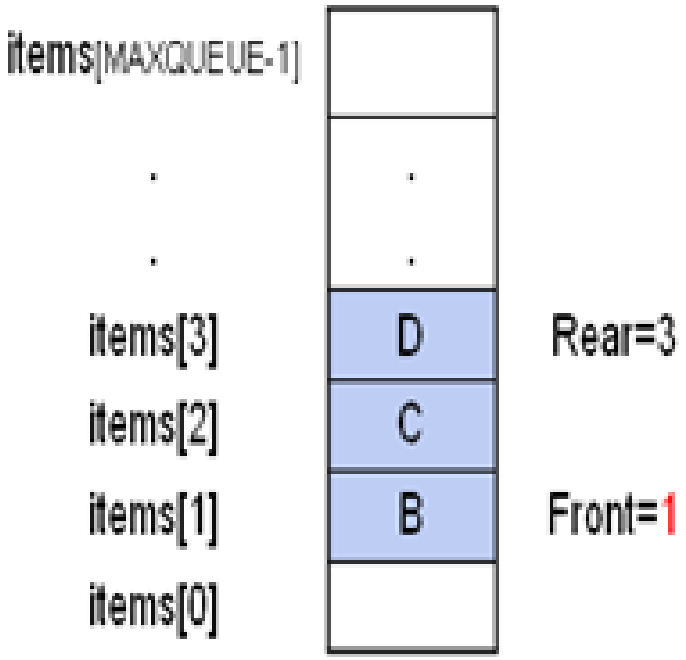


Enqueue(B,C, D)

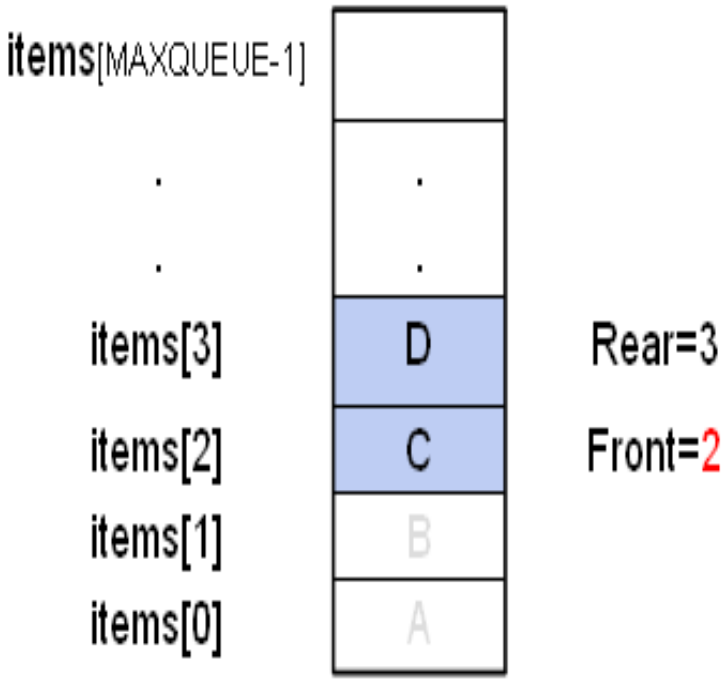


Cont....

Dequeue(A):



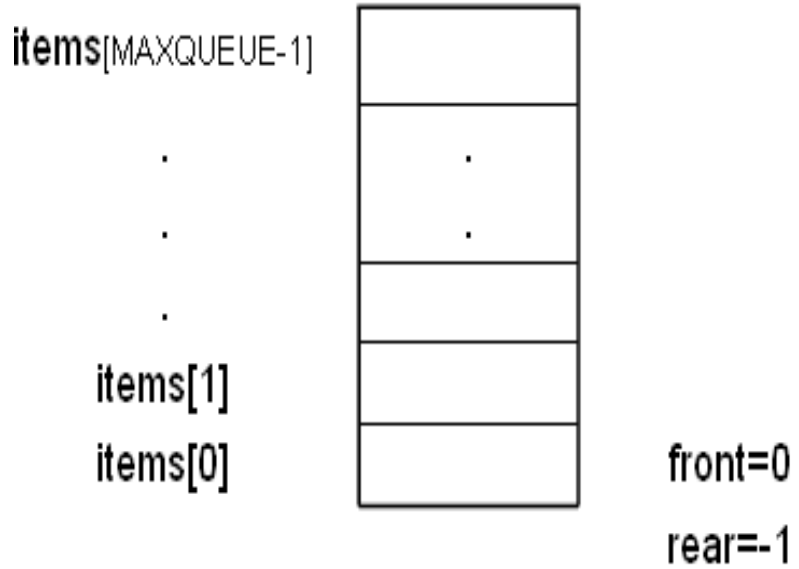
Dequeue(B):



Initialization of queue

- The queue is initialized by having the *rear* set to *-1*, and *front* set to *0*.
- Let us assume that maximum number of the element we have in a queue is MAXQUEUE elements as shown below

Rare= -1
Front=0



Applications of queue

- Task waiting for the printing
- Time sharing system for use of CPU
- For access to disk storage
- Task scheduling in operating system

The Queue as a ADT

- A queue q of type T is a finite sequence of elements with the operations
- ***MakeEmpty(q):*** To make q as an empty queue
- ***IsEmpty(q):*** To check whether the queue q is empty. Return true if q is empty, return false otherwise.
- ***IsFull(q):*** To check whether the queue q is full. Return true in q is full, return false otherwise.
- ***Enqueue(q, x):*** To insert an item x at the rear of the queue, if and only if q is not full.
- ***Dequeue(q):*** To delete an item from the front of the queue q. if and only if q is not empty.
- ***Traverse (q):*** To read entire queue that is display the content of the queue.

Implementation of queue

There are two techniques for implementing the queue:

- Array implementation of queue (static memory allocation)
- Linked list implementation of queue (dynamic memory allocation)

Array implementation of queue:

- In array implementation of queue, an array is used to store the data elements.
- Array implementation is also further classified into two types

Linear array implementation:

- A linear array with two indices always increasing that is rear and front. Linear array implementation is also called linear queue

Circular array implementation:

This is also called circular queue.

Linear queue

Algorithm for insertion (or Enqueue) and deletion (Dequeue) in queue

Algorithm for insertion an item in queue:

1. Initialize front=0 and rear=-1

if rear>=MAXSIZE-1

print "queue overflow" and return

else

set rear=rear+1

queue[rear]=item

2. end

Algorithm to delete an element from the queue

1. if rear<front

print "queue is empty" and return

else

item=queue[front++]

2. end

Declaration of a Queue

```
# define MAXQUEUE 50 /* size of the queue items*/
struct queue
{
        int front; int rear;
        int items[MAXQUEUE];
};
typedef struct queue qt;
```

Defining the operations of linear queue

The MakeEmpty function

```
void makeEmpty(qt *q)
{
        q->rear=-1;
        q->front=0;
}
```

The IsEmpty function:

```
int IsEmpty(qt *q)
{
        if(q->rear<q->front)
        return 1;
        else
        return 0;
}
```

The Isfull function:

```
int IsFull(qt *q)
{
        if(q->rear==MAXQUEUEZIZE-1)
        return 1;
        else
return 0;
}
```

The Enqueue function

```
void Enqueue(qt *q, int newitem)
{
    if(IsFull(q))
    {
        printf("queue is full");
    exit(1);
    }
    else
    {
        q->rear++;
        q->items[q->rear]=newitem;
    }
}
```

The Dequeue function

```
int Dequeue(qt *q)
{
    if(IsEmpty(q))
    {
        printf("queue is Empty");
        exit(1);
    }
    else
    {
        return(q->items[q->front]);
        q->front++;
    }
}
```

Problems with Linear queue implementation

- Both rear and front indices are increased but never decreased.
- As items are removed from the queue, the storage space at the beginning of the array is discarded and never used again.
- Wastage of the space is the main problem with linear queue which is illustrated by the following example.

Write a menu driven program to illustrate basic operations of Linear queue.

- **Enqueue**
- **Dequeue**
- **Display all values**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define N 10      //Size of queue is 10

struct queue      //Definition of structure
{
          int n[N];             //Data holder
          int  front, rear;  //Front and Rear
};
//Function Declaration
void enqueue(struct queue *);
void dequeue(struct queue *);
void show(struct queue *);
void front(struct queue *);
void makeempty(struct queue *);
```

```
void main()
{
        int c;
        struct queue q;          //variable of type queue is declared (q)
        makeempty(&q);     //Initialization of the queue

        do
        {
                clrscr();
                printf("\n\t\t LINEAR QUEUE IMPLEMENTATION USING ARRAY");
                printf("\n\n 1:Enqueue\ n2: Dequeue\n 3:Show all\n 4:Front\n 5:Make empty\n
                6:Exit\n");
                printf("\n\nEnter your choice: ");
                scanf("%d",&c);
                switch(c)
                {
                        case 1:
                        enqueue(&q);             //for 1 call function enqueue
                        break;

                        case 2:
                        dequeue(&q);    //for 2 call function dequeue
                        break;

                        case 3:
                        show(&q);      //for 3 call function show
                        break;
```

```
            case 4:
            front(&q);      //for 4 call function front
            break;

            case 5:
            makeempty(&q);  //for 5 call function makeempty
            break;

            case 6:
            exit(1);        //for 6 call function exit to exit the program

            default:
            printf("\n Wrong option");  //default "wrong option"

            }  //switch close
            }while(c!=6);        //do close
getch();
}  //void main close
```

```
//insert data
void enqueue(struct queue *q)
{
        int val;
        if(q->rear==N-1)            //check for overflow
        printf("\n Queue is full");
        else
        {
                printf("\nEnter the number to enqueue\n");
                scanf("%d",&val);
                q->rear++;                  //rear is incremented
                q->n[q->rear]=val;                  //value is enqueued
                printf("\n %d is enqueued", val);
        }
        getch();
}
```

```
//delete data
void dequeue(struct queue *q)
{
        if(q->rear<q->front)          //check for underflow
                printf("\n Queue is empty");
        else
        {
        printf("\n Dequeued value is %d",
                q->n[q->front]);
                q->front++;                 //front is incremented
        }
        getch();
}
```

```
//Display data
void show(struct queue *q)
{
        int i;
        if(q->rear<q->front)        //check for empty
        printf("\n Queue is empty");
  else
        {
                printf("\nThe numbers in the queue are:(Front to Rear)\n");
                for(i=q->front;i<=q->rear;i++) // if not empty, print all the values in
the queue
                                        {
        printf("\t%d",q->n[i]);
      }
        }

        getch();
}
```

```
//display front data item in queue
void front(struct queue *q)
{
        if(q->rear<q->front)        //check for empty
        {
        printf("\n Queue is empty");
  getch();
        }
        else                        //if not empty, print the value at front
        {
                printf("\n The value at front is %d", q->n[q->front]);
                getch();
        }
}
```

// to make queue empty

```
void makeempty(struct queue *q)  //makes the queue empty or reinitialize
{
        q->front=0;             //reinitialize front to 0 and
        q->rear=-1;             //rear to -1
}
```

Circular queue