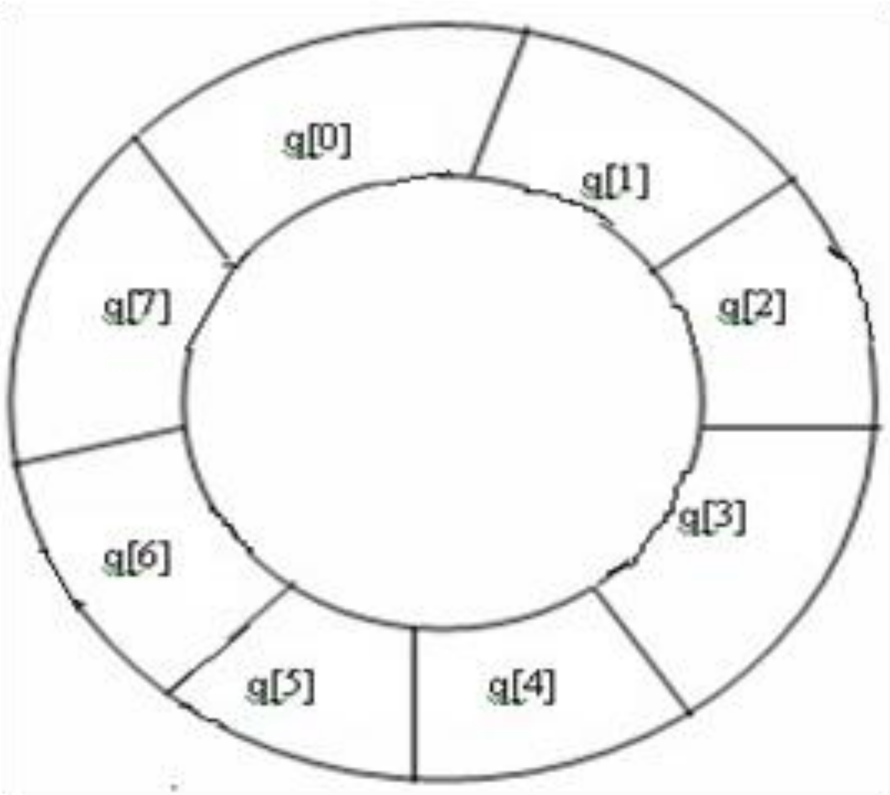
Circular queue

3nd Semester

B.Sc. CSIT, TU

Circular queue

A circular queue is one in which the **insertion of a new element is done at very first location** of the queue if the last location of the queue is full.



Cont…

- A circular queue overcomes the problem of unutilized space in linear queue implementation as array.
- In circular queue we sacrifice one element of the array thus to insert n elements in a circular queue
- we need an array of size n+1.

(or we can insert one less than the size of the array in circular queue).

Initialization of Circular queue

rear=front=MAXSIZE-1

Algorithms for inserting an element in a circular queue

This algorithm is assume that rear and front are initially set to MAZSIZE-1.

```
1. if (front==(rear+1)%MAXSIZE)
        print Queue is full and
        exit
   else
       rear=(rear+1)%MAXSIZE; [increment rear by 1]
2. cqueue[rear]=item;
3. end
```

Algorithms for deleting an element from a circular queue

This algorithm is assume that rear and front are initially set to MAZSIZE-1.

```
1. if (rear==front) [checking empty condition]
        print Queue is empty and
        exit
2. front=(front+1)%MAXSIZE; [increment front by 1]
3. Item = cqueue[front];
4. return item;
5. end.
```

Declaration of a Circular Queue

```
# define MAXSIZE 50
/* size of the circular queue items*/
struct cqueue
{
    int front;
   int rear;
    int items[MAXSIZE];
};
typedef struct cqueue cq;
```

Operations of a circular queue

The MakeEmpty function

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

The IsEmpty function

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

<u>The Isfull function</u>

```
int IsFull(cq *q)
{
            if(q->front==(q->rear+1)%MAXSIZE)
            return 1;
            else
            return 0;
}
```

<u>The Enqueue function</u>

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

The Dequeue function:

```
int Dequeue(cq *q)
{
        if(IsEmpty(q))
        {
                printf(“queue is Empty”); exit(1);
        }
        else
        {
                q->front=(q->front+1)%MAXSIZE;
                return(q->items[q->front]);
        }
}
```

Circular Queue Implementation

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAXQUEUE 5
struct queue
  {
    int items[MAXQUEUE];
    int front,rear;
  };
struct queue q;
void enqueue(struct queue *,int);
int  dequeue(struct queue *);
int empty(struct queue *);
```

```
void main()
 {
   int ch;
   int x,y;
   clrscr();
   q.front=q.rear=MAXQUEUE-1;
  do
  {
             clrscr();
             printf("1.Enqueue\n");
             printf("2.Dequeue\n");
             printf("3.Exit\n");
             scanf("%d", &ch);
             switch(ch)
             {
                case 1:
                 {
printf("Please enter the number:");
                      scanf("%d",&x);
                      clrscr();
                      enqueue(&q,x);
                      break;
```

```
        }
    case 2:
    {
     y=dequeue(&q);
    printf("\n\nDequeued
    item=%d",y);
    }
    getch();
    break;
    }
    case 3:
    {
       exit(1);
       break;
    }
    }
  }while(ch<=3);
}
```

```
int Empty(struct queue *pq)
{
        return (pq->rear == pq->front);
}
int Full(struct queue *pq)
{
        int newrear;
        newrear = (pq->rear+1)%MAXQUEUE;
        return newrear == pq->front;
}
```

```
void enqueue(struct queue *pq, int x)
{
if(Full(pq))
   {
                        printf("Queue Full\n");
                        getch();

            }
else
   {
            pq->rear = (pq->rear+1)%MAXQUEUE;
            pq->items[pq->rear] = x;
            printf("\n %d is enqueued",x);
            getch();
   }
}
```

```
int dequeue(struct queue *pq)
{
        if (Empty(pq))
        {
        printf("Queue is Empty\n");
        getch();
        }
        else
        {
                pq->front = (pq->front+1)%MAXQUEUE;
                return(pq->items[pq->front]);
        }
 return(0);
}
```

```
//Circular Queue Implementation with Counter
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAXQUEUE 5
struct queue
   {
      int items[MAXQUEUE];
      int front,rear;
   };
struct queue q;
void enqueue(struct queue *, int);
int  dequeue(struct queue *);
int empty(struct queue *);
int counter=0;
```

```
void main()
 {
  char ch;
  int x,y;
  clrscr();
  q.front=q.rear=MAXQUEUE-1;
  while(1)
   {
           label:
           printf("1.Enqueue\n");
           printf("2.Dequeue\n");
           printf("3.Exit\n");
           ch=getch();
           switch(ch)
           {
            case '1':
           {
           if(counter==MAXQUEUE)
            {
              printf("Queue overflow");
               getch();
               clrscr();
            }
```

Priority queue

A priority queue is a collection of elements such that each element has been assigned a priority and the order in which elements are deleted and processed comes

from the following rules:

- An element of higher priority is processed before any element of lower priority.
- If two elements has same priority then they are processed according to the order in which they were added to the queue.

The best application of priority queue is observed in CPU scheduling.

✓ The jobs which have higher priority are processed first.

✓ If the priority of two jobs is same this jobs are processed according to their position in queue.

✓ A short job is given higher priority over the longer one.

Types of priority queues

Ascending priority queue (min priority queue):

An ***ascending priority queue*** is a collection of items into which items can be inserted arbitrarily but from which only the smallest item can be removed.

Descending priority queue(max priority queue):

An **descending priority queue** is a collection of items into which items can be inserted arbitrarily but from which only the largest item can be removed.

Priority QUEUE Operations

Insertion

The insertion in Priority queues is **the same as** in non-priority queues.

Cont…

Deletion

Deletion requires a search for the element of highest priority and deletes the element with highest priority.

The following methods can be used for deletion/removal from a given Priority Queue

- ✓ An empty indicator replaces deleted elements.
- ✓ After each deletion elements can be moved up in the array decrementing the rear.
- ✓ The array in the queue can be maintained as an ordered circular array

Priority Queue Declaration

Queue data type of Priority Queue is the same as the Non-priority Queue.

```
#define MAXQUEUE 10 //size of the queue items struct
pqueue
{
        int front; int rear;
        int items[MAXQUEUE];
};
struct pqueue *pq;
```

The priority queue ADT

A ascending priority queue of elements of type T is a finite sequence of elements of T together with the operations:

MakeEmpty(p): Create an empty priority queue p

Empty(p): Determine if the priority queue p is empty or not

Insert(p, x): Add element x on the priority queue p

DeleteMin(p): If the priority queue p is not empty, remove the minimum element of the quque and return it.

FindMin(p): Retrieve the minimum element of the priority queue p.

Array implementation of priority queue

Unordered array implementation:

- ✓ To insert an item, insert it at the rear end of the queue.
- ✓ To delete an item, find the position of the minimum element and
- ✓ Either mark it as deleted (lazy deletion) or
- ✓ shift all elements past the deleted element by on position and then decrement rear. (Delete smallest element first)

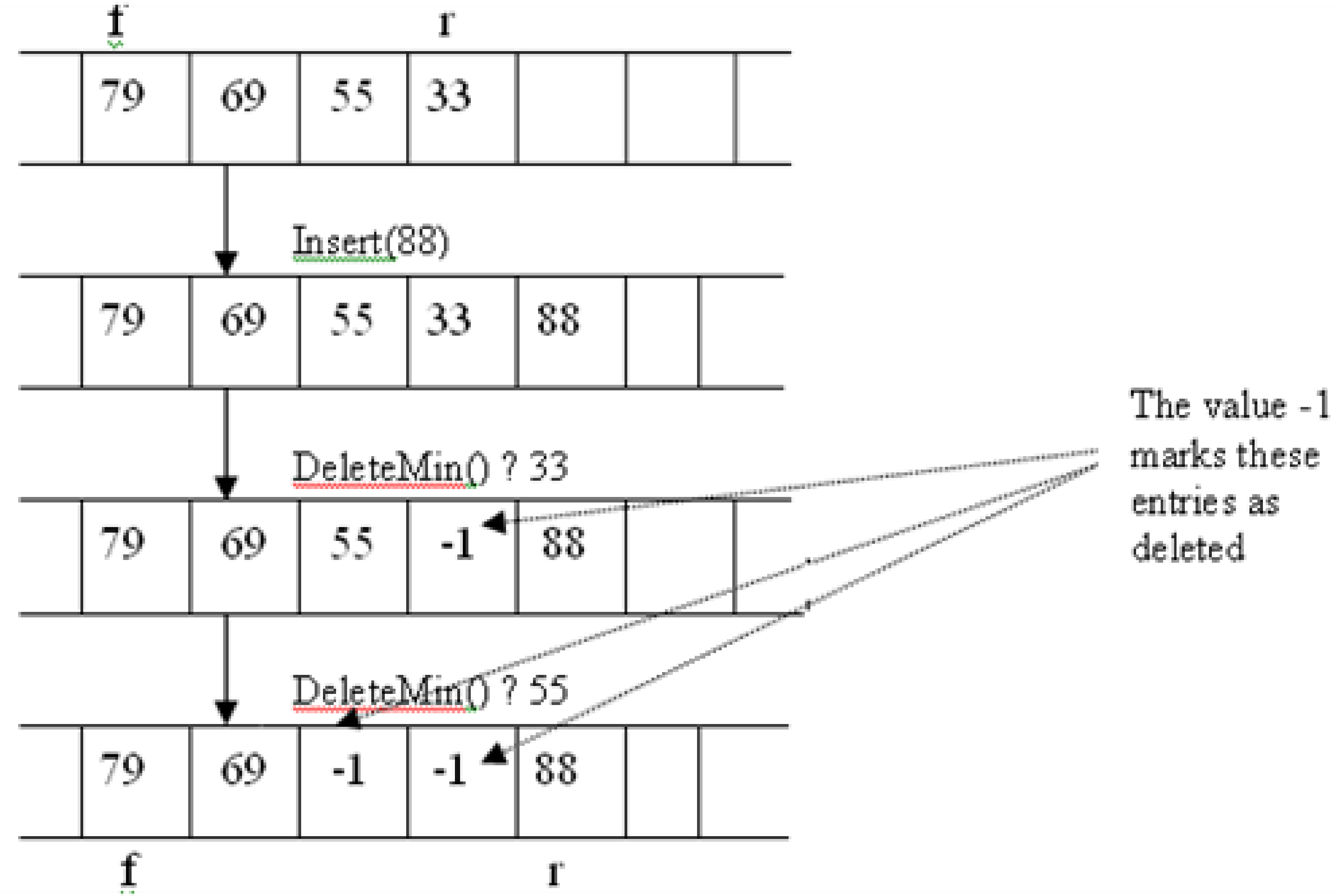


Fig: Illustration of unordered array implementation

Ordered array implementation:

✓ Set the front as the position of the smallest element and the rear as the position of the largest element.

✓ To insert an element, locate the proper position of the new element and shift preceding or succeeding elements by one position.

✓To delete the minimum element, increment the front position.

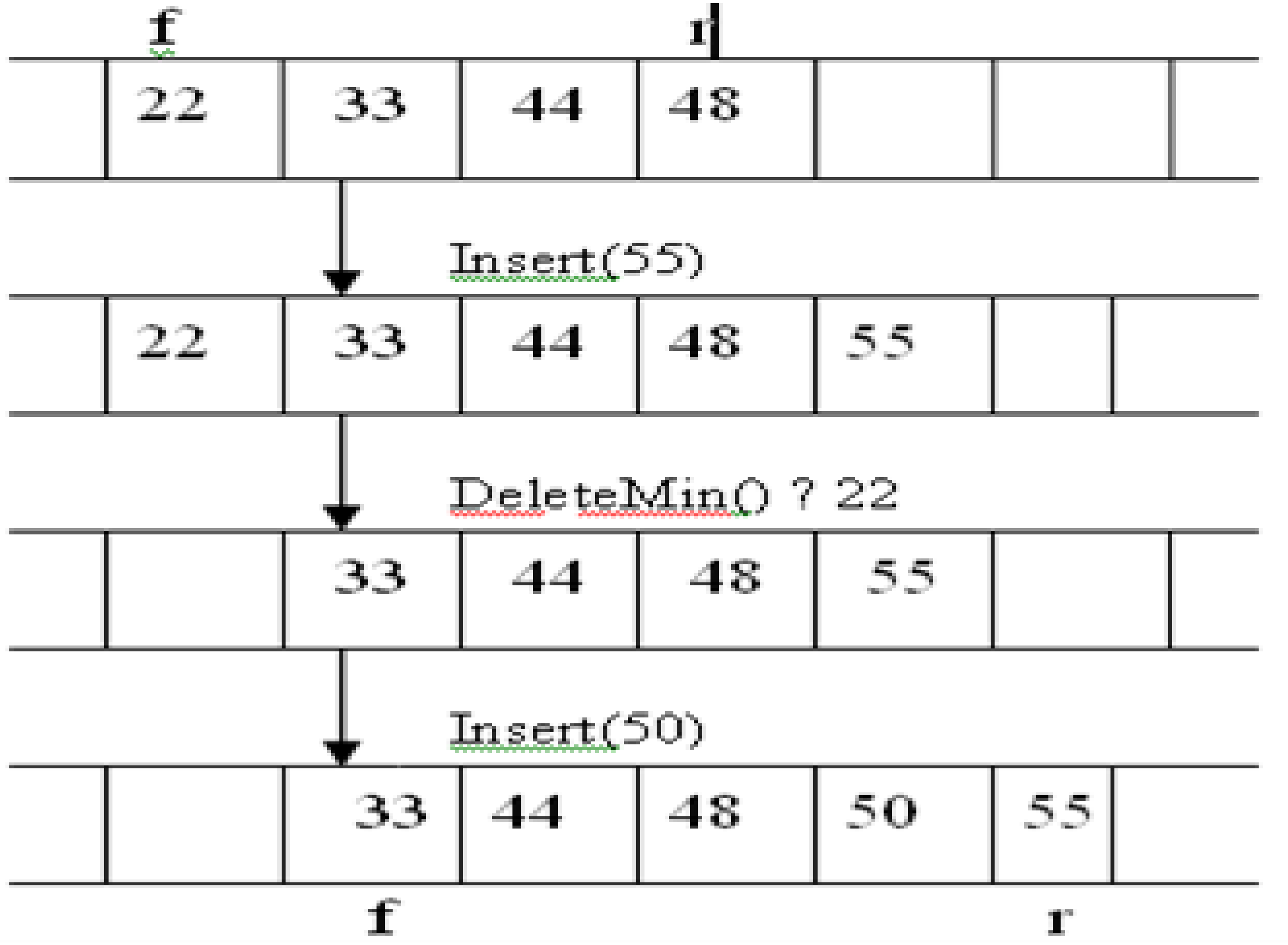


Fig. Illustration of ordered array implementation

Application of Priority queue

- In a time-sharing computer system, a large number of tasks may be waiting for the CPU, some of these tasks have higher priority than others.
- The set of tasks waiting for the CPU forms a priority queue.

```
/*implementation of ascending priority queue */
 #include<stdio.h>
#include<conio.h>
#define SIZE 20
struct cqueue
{
        int item[SIZE];
        int rear;
        int front;
};
```

```
struct queue q;
void insert(pq*);
void delet(pq*);
void display(pq*);
void main()
{
int ch;
q->rear=-1;
q->front=0;
clrscr();
printf("Menu for program:\n");
printf("1:insert\n2:delete\n3:display\n4:exit\n");
do
{
        printf("Enter youer choice\n");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1:
                 insert(&q);
                break;
                Case 2:
                delet(&q);
                break;
                case 3:
                display(&q);
                break;
                case 4:
                exit(1);
                break;
                default:
                printf("Your choice is wrong\n");
                break;
        }
}while(ch<5);
getch();
}
```

```
/***********insert function**************/
void insert(struct queue  *q)
{
int d;
if(q->rear==SIZE-1)
printf("Queue is full\n");
else
{
            printf ("Enter data to be inserted\n");
            scanf("%d",&d);
            q->rear++;
            q->item[q->rear]=d;
}
}
```

```
/**********delete function*****************/
void delet(struct queue *q){
        int i, temp=0, x;
        x=q->item[q->front];
        if(q->rear<q->front){
                printf("Queue is empty\n"); return 0;
        }
        else {
                for(i=q->front+1; i<q->rear; i++){
                        if(x>q->item[i]){
                                temp=i;
                                x=q->item[i];
                        }
                }
                for(i=temp; i< q->rear-1; i++) {
                        q->item[i]=q->item[i+1];
                 }
        q->rear--;
        return x;
} }
```

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

                }
}
```

Lab assignment

- WAP to implement ascending priority queue
- WAP to implement descending priority queue