Queues

rear insertion (enqueue)

front deletion (dequeue)

First in first out (FIFO)

Applications

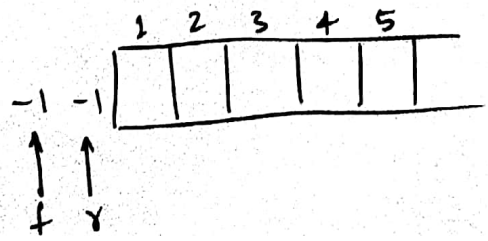
1. process and job scheduling.
2. resource allocation.

Implementation of queue

Let SIZE denote the size of the queue and the following structure definition is used to represent the queue.

```
Struct queue
{
    int f, r;
    int data[SIZE];
};  typedef struct queue QUEUE;
```

where f denotes front end and r denotes rear end. and array data is used to hold the elements. Initial value of f and r are -1. An empty queue can be represented as



A C function to insert an element onto a queue

```
void insertq (QUEUE *q, int ele)
{
    if (q->r == SIZE-1)
        pf("Queue is full");
    else
    {
        q->data[++(q->r)] = ele;
        if (q->f == -1)
            (q->f)++;
    }
```

A C func to delete an element from the queue.

```
int deleteq (QUEUE *q)
{
    int e

    if (q->f == -1)
    {   printf("Queue empty");
        return -1;
    }
    else
    {
        e = q->data[q->f];
        if (q->f == q->r)
        {
            q->f = -1;
            q->r = -1;
        }
        else
            (q->f)++;
        return e;
    }
}
```

A function to display the content of queue.

```
void display q (QUEUE *q)
{
    int i;
    if (q.f == -1)
        pf("Queue empty");
    else
    {
        pf("\n Queue content is");
        for (i= q.f ; i<= q.r ; i++)
            pf("%d \t", q.data[i]);
    }
}
```

There are 4 types of queues.

1. Ordinary queue (Linear queue)
2. Circular queue
3. Priority queue
4. Double ended queue (D-queue)

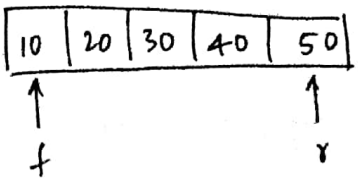
   Two types of D-queue

   i) input restricted queue. (ins only r , del using both)

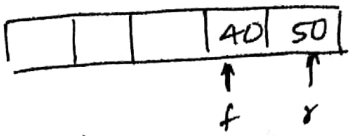
   ii) output restricted queue (del " f , ins " ")

of queue.

Consider a linear or ordinary queue with 5 elements as.



After deleting 3 elements, the queue will look like.



even though there are some empty slots in the begining of the queue, it cant perform an insertion operation as the rear end has reached a value of (size-1). To overcome this disadvantage, we make use of a queue type named circular queue.

Implementation of a circular queue

Let SIZE denotes the size of the queue and the following struct definition is used to represent a circular queue.

```
struct queue
{
    int f,r;
    int data[SIZE];
};
typedef struct queue CQUEUE;
```

insertion operation

SIZE 5

A c function to perform insertion op.

```
void    cqueue insert (CQUEUE *q, int item)
{
    if (q->f == (q->r + 1) % SIZE)
        pf ("\n queue full");
    else
    {
        q->r = (q->r+1) % SIZE;
        q->data [q->r] = item;
        if (q->f == -1)
            q->f = 0;
    }
}
```

Deletion operation

```
int    cqueue delete (CQUEUE *q)
{
    if (q->f == -1)
    {   pf ("\n q empty");
        return -1;
    }
    e = q->data [q->f];
    if (q->f == q->r)
```

→ `q->f = (q->f+1) % SIZE`
`if (q->f == q->r)`

```
        q->f=-1;
        q->r=-1;
    }
    else
        q->f=(q->f+1)%SIZE;
    return e;
}
```

Display operation

```
void cqdisplay(CQUEUE q)
{
    if (q.f==-1)
        pf("\n empty");
    else if (q.f <= q.r)
    {
        pf("Que data");
        for( i=q.f ; i<=q.r ,i++).
            pf("%d\t", q.data[i]);
    }
    else
    {
        pf("Queue data");
        for (i=q.f; i<SIZE ; i++)
            pf("%d \t", q.data [i]);
        for (i=0 ;  i<=q.r ; i++)
            pf("%d\t", q.data[i]);
    }
```

```
6  i=q.f
   i!=q.r
   i=(i+1)%SIZE
```

Lab Prog 3

Implement a message queuing system using circular queue.

```
#include <stdio.h>
        <stdlib.h>
        <string.h>
#define   SIZE   5

struct   queue
{
     int  f,r;
     char  data[SIZE][20];
};
typedef   struct   queue   CQUEUE;

void   cqinsert (CQUEUE *q, char item[20])
{
     if (q->f==(q->r+1) % SIZE)
          pf("\n queue full");
     else
     {
          q->r = (q->r +1) % SIZE;
          strcpy (q->data[q->r], item);
          if (q->f==-1)
               q->f=0;
     }
}
```

```
char *  cqdelete (CQUEUE *q)
{
    char *e;
    if (q->f == -1)
    {   pf ("\n empty");
        return NULL;
    }
    e=q->data[q->f];
    if (q->f == q->r)
    {
        q->f=q->r=-1;
    }
    else
        q->f = (q->f+1) % SIZE;
    return e;
}
void  cqdisplay (CQUEUE q)
{
    if (q.f==-1)
        pf("\n empty");
    else if (f<=r)
    {
        pf("\n data is");
        for (i=q.f ; i<=q.r ; i++)
            pf ("%s\n", q.data[i]);
    }
```

```
        else
        {
            pf ("\n Queue data\n")
            for (i=q.f ; i<SIZE ; i++)
                pf ("%s\n", q.data [i]);
            for (i=0 ; i<=q.r; i++)
                pf ("%s\n", q.data[i]);
        }
}

int main()
{
    char item[20], *e

        int ch
    CQUEUE q
    q.f=-1; q.r=-1;

    for(;;)
    {
        pf(1. ins 2. del 3. disp 4. exit);
        scanf("%d", &ch); getchar();
        switch (ch)
        {
            case 1: pf("\n read message to be ins");
                    gets(item);
                    cqinsert (&q, item);
                    break
```

```
case 2: e= cq_delete(&q);
        if (*e != NULL)
            pf("The message deleted is %s", *e);
        break.
case 3: cq_display(q);
default: exit(0);
    }
}
```

1st test portions

Priority Queue

is a queue type where in the deletion (dequeue) op is based on the priority.

There are 2 types of priority queues

1. Ascending priority queues
2. Descending priority queues

There are 2 ways of implemendation

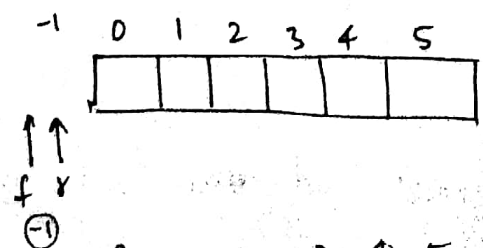
1. The insertion op is the normal one as compared to queue and the deletion operation needs to be modified as compared to the normal queue
2. The insertion op is a modified one where in the elements are inserted in an order onto the queue and the deletion is the normal deletion op

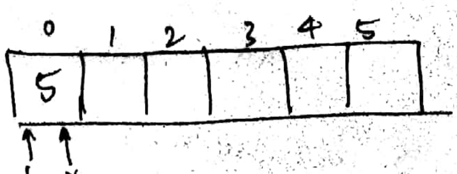
A C func to insert an element onto a priority queue

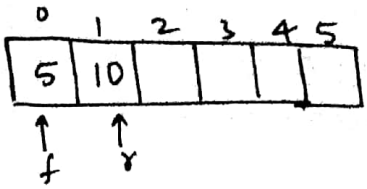
```
void    pqinsert (QUEUE *q, int item)
{
    if (q->r == SIZE-1)
        pf(" full");
    else
    {
        int pos = q->r;
        q->r = q->r + 1;
        while (p>=0 && q->data[pos] >= item)
        {
            q->data[pos+1] = q->data[pos];
            pos--;
        }
        q->data[pos+1] = item;
        if (q->f == -1)
            q->f = 0;
    }
}
```

Trace the insertion function on a priority queue for the following elements

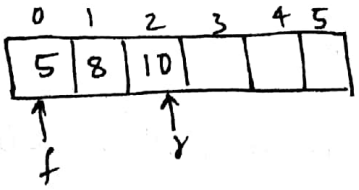
5 10 8 4 6.







pos = 0
r = 1



pos = 1
r = 2