Queue

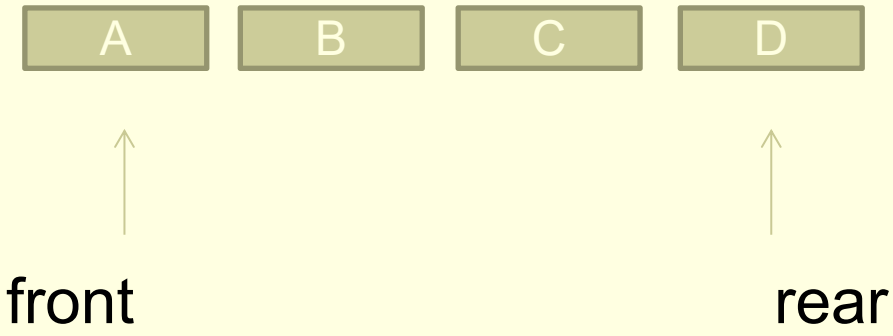
prepared by Harish Patnaik

Outline

- Intro to Queue
- Creation of Queue
- Queue representation using array
- Queue representation using Linked list

Queue

- Items are added only at one end (rear)
- Items are removed only from other end (front)
- First In First Out (FIFO)



Queue

- Every queue is associated with two pointers - front, rear
- Two basic operations of queue are -

  insert /enq - insert an element into queue
  delete/deq - remove an element from queue

Basic operations of Queue

Insertion

size - size of the queue

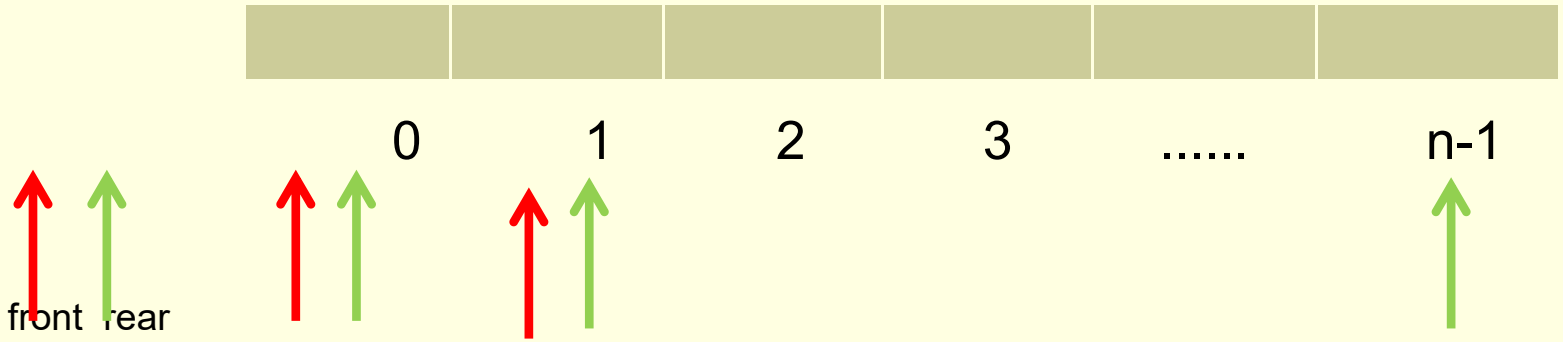
front, rear - pointers of queue

1. if rear >= size
   print “Queue overflow” and Exit
2. rear = rear+1
3. Q[rear] = value //inserting value
4. if front= -1 then front=0
5. Exit

Deletion

1. if front = -1 //queue is empty
   print “queue underflow” & Exit
2. value = Q[front]
3. if front= rear //single element
   front= -1
   rear = -1
   else front = front+1
4. Return value & Exit

Array implementation of Queue

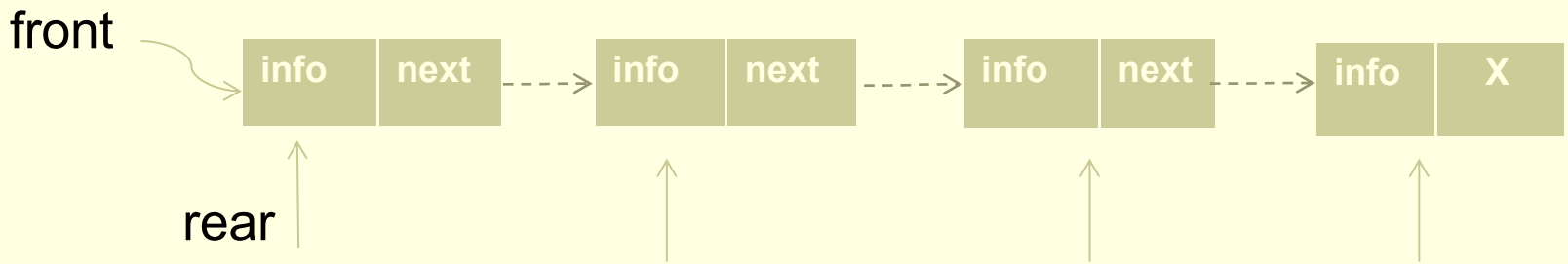


```
#define size 50
int rear = -1;
int q[size];

void insert(int s[], int d)
{
  if (rear== size-1)
            printf ("\n Queue overflow ");
  else if (front== -1 && rear== -1)
            front= rear= 0;
     else  rear++ ;
      q[rear]=d;
   }
```

```
int delete(int q[])
{
   int d;
   if (front== -1)
        { d = 0;
          printf ("\n Queue underflow "); }
   else
        { d = q[front] ;
          front++ ;
          if (front> rear)
              front= rear= -1;      }
    return (d);
}
 void display(int q[])
{    int i;
      if (front== -1)
          printf ("Queue is empty");
      else
          for (i=front; i<=rear; i++)
              printf (" \t %d", q[i]);
}
```

Linked List implementation of Queue

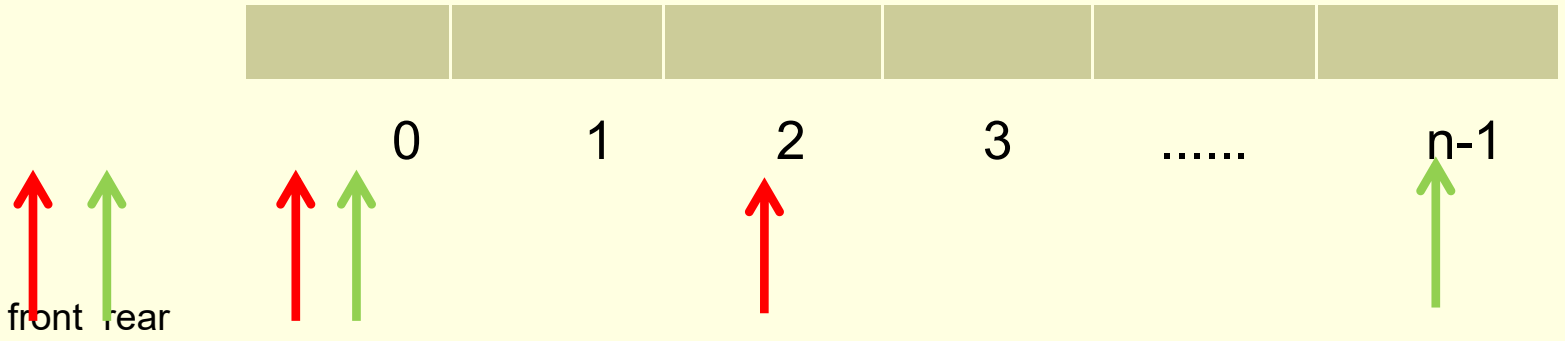


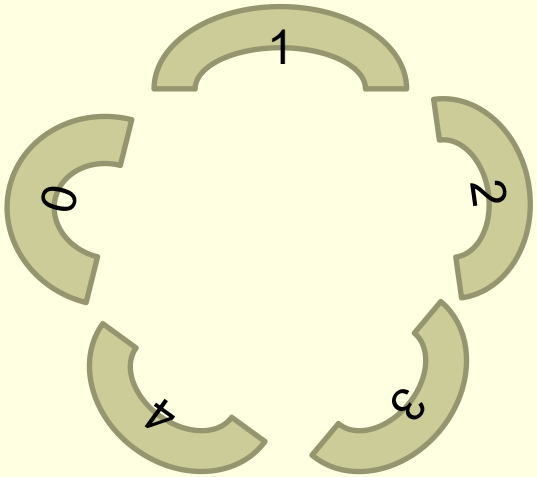

```
struct node
{
  int info;
  struct node * next;
};
struct node *front= NULL, *rear=NULL;
```

```
void insert (int x)
{
  struct node *qptr;
  qptr= (struct node *)
            malloc(sizeof(struct node));
 if (qptr== NULL)
    {printf ("\n Out of memory ");
     exit(0);
    }
qptr->info= x;
qptr->next= NULL;
if (rear==NULL)
     front= qptr;
else
     rear->next=qptr;
   rear=qptr;
 }
```

```
int delete()
{  struct node *temp;    int d;
   if (front==NULL)
        { printf ("\n Queue underflow ");
          exit(0); }
   temp =front;
   d = temp->info;
  front= front->next ;
  if (front== NULL)  //single element
     rear= NULL;
   free(temp) ;
    return (d);
}
void display()
{   struct node *curr;
      curr= front;
 if (curr== NULL)
        { printf ("Queue is empty"); exit(0); }
while (curr !=NULL)
              { printf ("\t %d", curr->info);
                curr= curr->next;     }
}
```

Circular Queue





Basic operations of Circular Queue

Insertion

n - size of the queue

front, rear - pointers of queue

1. if front=0 and rear=n OR front=rear +1
   print “Queue overflow” and Exit
2. else if front= -1
   front=0, rear=0,
3. else if rear=n
   rear= 0
4. else rear= rear+1
5. Q[rear] = value
6. Exit

Deletion

1. if front = -1 //queue is empty
   print “queue underflow” & Exit
2. value = Q[front]
3. if front= rear //single element
   front= -1
   rear = -1
4. else if front= n
   front=0
   else front = front+1
5. Return value & Exit

Circular Queue

```
#define size 10
int rear=-1, front=-1;
int q[size];
void insertq()
{ printf ("\n Enter the element ");
   scanf ("%d", &b);
 if (((front==0) && (rear= size-1))
            ||(front= rear+1))
     {printf("\n Queue overflow");
            return;}
else if (front<0)
            {front=0;  rear=0;}
else if (rear== size-1)
            rear=0;
else       rear=rear+1;
q[rear] = b;
}
```

```
int deleteq()
{ if front <0  //queue is empty
       {  printf( "queue underflow" );
            return;
        }
    b = q[front];
 if (front== rear)
      {  front= -1;
        rear = -1;
       }
   else if (front== size-1)
            front=0;
      else  front = front+1;
 return (b);
}
```

Priority Queue

- Each element is assigned a priority
- Elements are used in the order of priority

Rules for Priority queue-

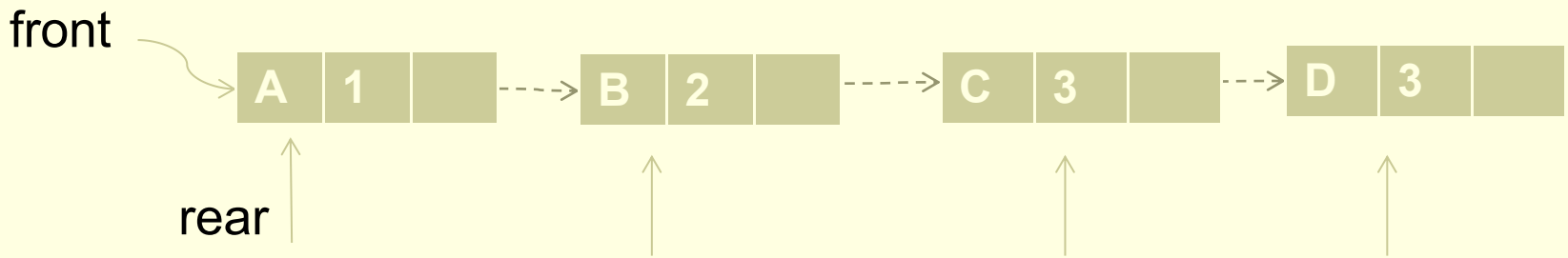
1. An element with higher priority is processed before an element with lower priority
2. Two elements with same priority are processed as First- Come- First- Served (FCFS) basis

Implementation of Priority Queue

a> Elements are stored in sorted order of priority and for deletion, the element with highest priority is deleted from the front end.

b> Elements are stored in the order of their arrival. So insertion is done only at the rear end of the list. But for deletion, the element with highest priority is searched and then deleted.

Linked List implementation of Priority Queue




```
struct node
{
  int info;
  int priority;
  struct node * next;
};
struct node *front= NULL, *rear=NULL;
```

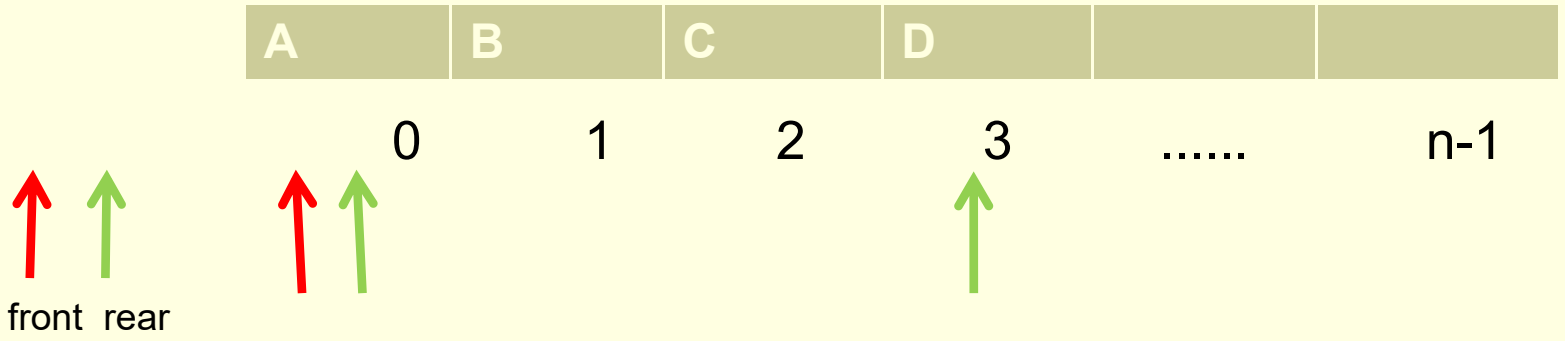
Priority Queue

```
void insert()
{ int val, pri;
 struct node *ptr, *p;
ptr=(struct node *) malloc(sizeof(struct node));
printf("\n Enter the value and priority :");
scanf ("%d %d", &val, &pri);
ptr->info= val;
ptr->priority= pri;
if (front==NULL || pri<front->priority)
{           ptr->next= front;
            front= ptr;
} else
{           p= front;
            while (p->next !=NULL && p->next->priority<= pri)
                        p= p->next;
            ptr->next= p->next;
            p->next=ptr;
} }
```

Priority Queue

```
int delete()
{
 struct node *ptr;
int d;
if (front== NULL)
        { printf("\n Queue underflow");
        return; }
else
        { ptr= front;
        d= ptr->info;
        front= front->next;
        free(ptr);
        return(d);
        }
 }
```

Queue using Stack



D
C
B
A

A
B
C
D

Queue using Stack

```
enque()
        { push into st1;
        }

deque()
        {
        check if st2 is empty;
        if empty,
                { transfer all element from st1 to st2;
                  pop st2;
                }
        else
                pop st2;
        }
```

Queue using stack

```
struct node
{
    int data;
    struct node *next;
};
void push(struct node** top, int data);
int pop(struct node** top);
struct queue
{
    struct node *stack1;
    struct node *stack2;
};
void enqueue(struct queue *q, int x)
{
    push(&q->stack1, x);
}
```

```
void dequeue(struct queue *q)
{
    int x;
    if (q->stack1 == NULL && q->stack2 == NULL) {
        printf("queue is empty");
        return;
    }
    if (q->stack2 == NULL) {
        while (q->stack1 != NULL) {
        x = pop(&q->stack1);
        push(&q->stack2, x);
        }
    }
    x = pop(&q->stack2);
    printf("%d\n", x);
}
```

Deque

Double ended queue

It is a list in which elements can be added or removed at either end but not at the middle. Operations are -

- Insertion at rear end
- Insertion at front end
- Deletion at front end
- Deletion at rear end

Two variations -

- Input- restricted deque
- Output- restricted deque

Input- restricted deque - insertion at only one end

Output- restricted deque - deletion at only one end

Thank you