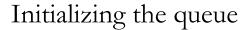
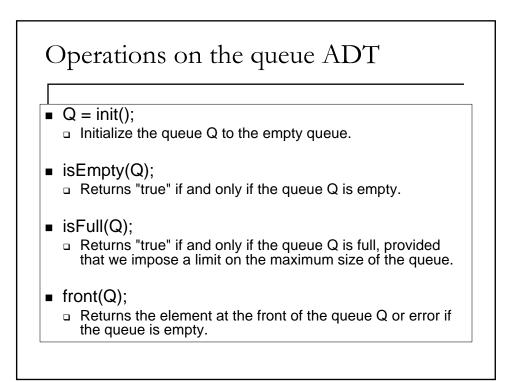
CS11001/CS11002 Programming and Data Structures (PDS) (Theory: 3-1-0)

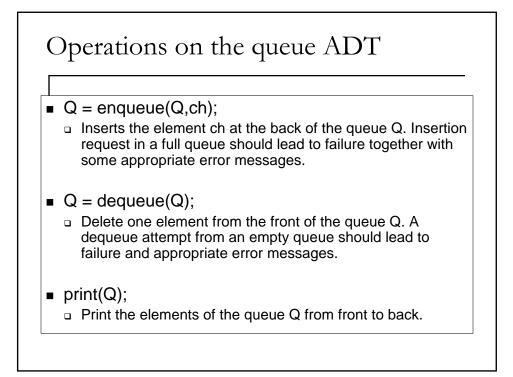
The queue ADT

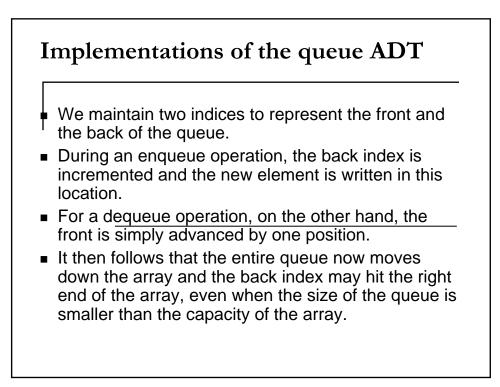
- A queue is like a "natural" queue of elements.
- It is an ordered list in which all insertions occur at one end called the **back** or **rear** of the queue.
- All deletions occur at the other end called the front or head of the queue.
- In the popular terminology, insertion and deletion in a queue are respectively called the enqueue and the dequeue operations.
- The element dequeued from a queue is always the first to have been enqueued among the elements currently present in the queue.
 - In view of this, a queue is often called a First-In-First-Out or a FIFO list.



```
queue init ()
{
queue Q;
Q.front = 0;
Q.back = MAXLEN - 1;
return Q;
}
```

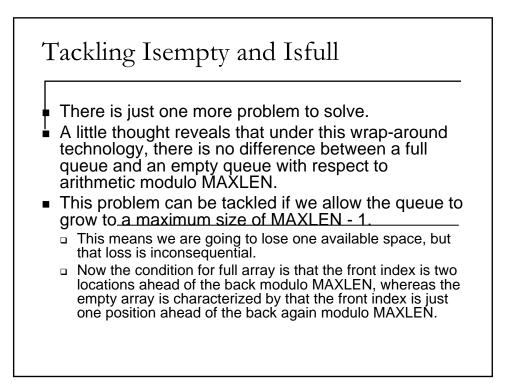






Reducing wastage in the queue

- In order to avoid waste of space, we allow our queue to wrap at the end.
- This means that after the back pointer reaches the end of the array and needs to proceed further down the line, it comes back to the zeroth index, provided that there is space at the beginning of the array to accommodate new elements.
- Thus, the array is now treated as a circular one with index MAXLEN treated as 0, MAXLEN + 1 as 1, and so on.
- That is, index calculation is done modulo MAXLEN.
- We still don't have to maintain the total queue size.
- As soon as the back index attempts to collide with the front index modulo MAXLEN, the array is considered to be full.



```
Isempty and Isfull
```

```
int isEmpty ( queue Q )
{
return (Q.front == (Q.back + 1) % MAXLEN);
}
int isFull ( queue Q )
{
return (Q.front == (Q.back + 2) % MAXLEN);
}
```

```
Enqueue

queue enqueue ( queue Q , char ch )

{

if (isFull(Q))

{

fprintf(stderr,"enqueue: Queue is full\n");

return Q;

}

++Q.back;

if (Q.back == MAXLEN)

Q.back = 0;

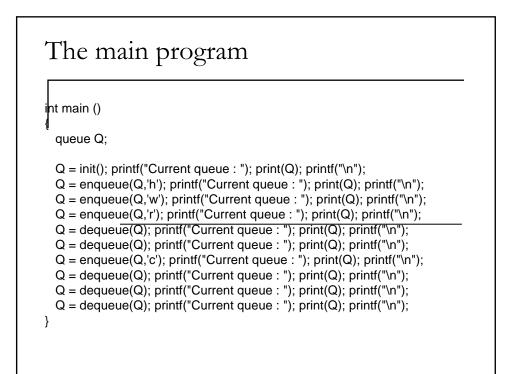
Q.element[Q.back] = ch;

return Q;

}
```

Dequeue

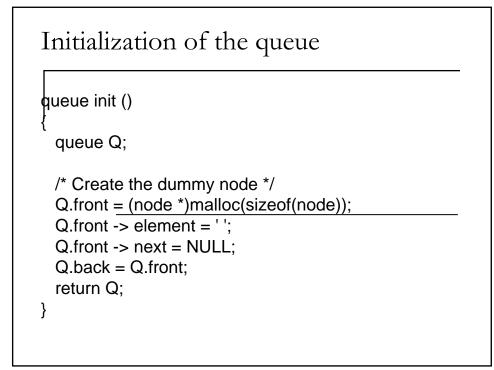
```
queue dequeue ( queue Q )
{
    if (isEmpty(Q))
      {
      fprintf(stderr,"dequeue: Queue is empty\n"); return Q;
      }
      ++Q.front;
    if (Q.front == MAXLEN)
      Q.front = 0;
    return Q;
    }
```



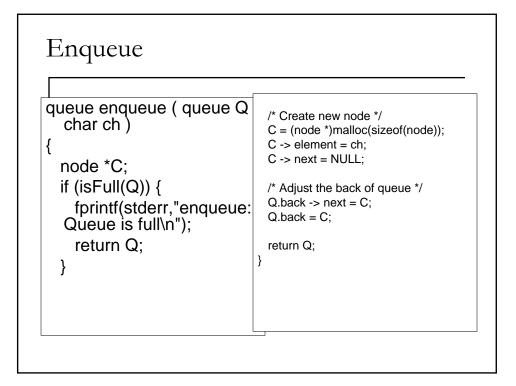
Output

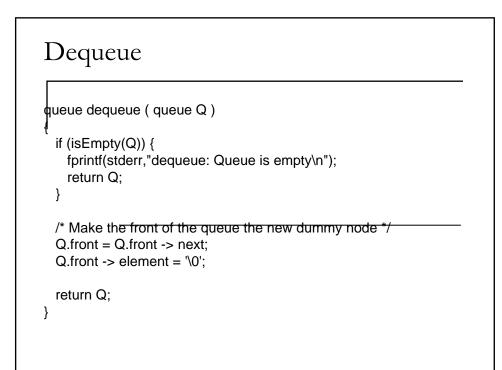
Current queue : Current queue : h Current queue : hw Current queue : hwr Current queue : wr Current queue : r Current queue : r Current queue : c Current queue : dequeue: Queue is empty Current queue :

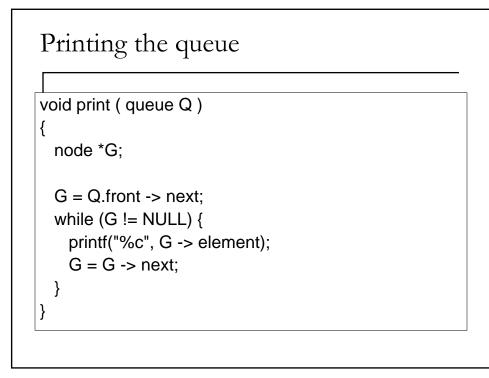
Defining a node in the queue
#include <stdio.h></stdio.h>
#include <malloc.h></malloc.h>
typedef struct _node { char element;
struct _node *next; } node;
typedef struct { node *front; node *back;
} queue;

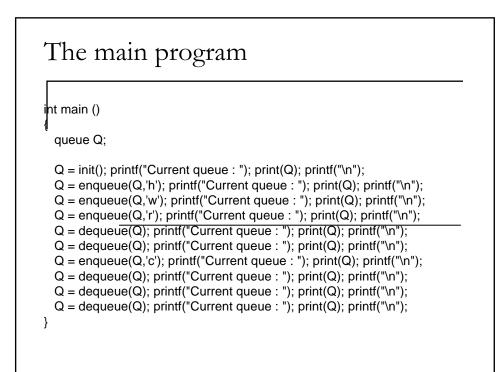


int isEmpty (queue Q)	
{	char front (queue Q)
return (Q.front == Q.back); }	{ if (isEmpty(Q)) { fprintf(stderr,"front: Queue is empty\n"); return '\0';
int isFull(queue Q)	}
{	return Q.front -> element;
return 0;	}
return 0;	}









Output

- Current queue :
- Current queue : h
- Current queue : hw
- Current queue : hwr
- Current queue : wr
- Current queue : r
- Current queue : rc
- Current queue : c
- Current queue :
- dequeue: Queue is empty
- Current queue :

Two Examples

(incomplete progr a	a m)
finclude <stdio.h> ypedef struct { int x; int y; points; ypedef points line [3]; main() line I1;</stdio.h>	$if((!!1[0].x-l1[0].x)\&\&(!!1[2].x-l1[1].x)) \\ \{ m1=(float)(l1[1].y-l1[0].y)/(l1[1].x-l1[0].x); \\ m2=(float)(l1[2].y-l1[1].y)/(l1[2].x-l1[1].x); \\ \}$
int i; float m1, m2; for(i=0;i<3;i++){ printf("Enter Point %d\n",i+1); scanf("%d",&I1[i].x); scanf("%d",&I1[i].y); }	if(m1==m2) printf("Line is straight…\n"); }

