Outline
Queues
- Definition of Queue
- Queue Operations
  - Insertion (Enqueue)
  - Removing (Dequeue)
- Applications of the Queues

**Definition of Queue**

A *queue* is an ordered collection of items from which items may be deleted at one end (*front* of the queue) and into which items may be inserted at the other end (*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).

![A Queue Diagram](image)

A new item (*E*) is *inserted* at the *Rear* of the queue

![Queue Operations Diagram](image)

An item (*A*) is removed (deleted) from the *Front* of the queue

**Queue Operations**

- Initialize the queue
- *Insert* to the rear of the queue (also called as Enqueue)
- *Remove* (Delete) from the front of the queue (also called as Dequeue)
- Is the Queue Empty
- Is the Queue Full
- What is the size of the Queue
**INITIALIZE THE QUEUE (USING LINEAR ARRAY)**
- 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 5 elements as shown below.

<table>
<thead>
<tr>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>front=0</td>
</tr>
<tr>
<td>rear = -1</td>
</tr>
</tbody>
</table>

**INSERT / REMOVE ITEMS**
- *Insert A, B, C to the rear of the queue.*

<table>
<thead>
<tr>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>front=0</td>
</tr>
<tr>
<td>rear = 2</td>
</tr>
</tbody>
</table>

- *Remove two items from the front of the queue.*

<table>
<thead>
<tr>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>front = rear = 2</td>
</tr>
</tbody>
</table>

- *Insert D, E, to the rear of the queue.*

<table>
<thead>
<tr>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td>D</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>front = 2</td>
</tr>
<tr>
<td>rear = 4</td>
</tr>
</tbody>
</table>

- What happens if we want to insert a new item F into the queue?
  Although there is some empty space, the queue is full. One of the methods to overcome this problem is to consider a circular array, where the next item position after position 4 in the above queue is position 0.
**Initialize the Queue Insert / Remove Items (Circular array)**

- **Initialize** the queue.

<table>
<thead>
<tr>
<th>Items</th>
<th>count=0</th>
<th>front= rear =4</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Insert** A,B,C to the rear of the queue.

<table>
<thead>
<tr>
<th>Items</th>
<th>count=3</th>
<th>front =4</th>
<th>rear = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>

- **Remove** two items from the queue.

<table>
<thead>
<tr>
<th>Items</th>
<th>count=1</th>
<th>front =1</th>
<th>rear =2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Insert** D,E to the queue.

<table>
<thead>
<tr>
<th>Items</th>
<th>count=3</th>
<th>front =1</th>
<th>rear =4</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Insert** F to the queue.

<table>
<thead>
<tr>
<th>Items</th>
<th>count=4</th>
<th>front =1</th>
<th>rear =0</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Insert** G to the queue. (Queue is full)

<table>
<thead>
<tr>
<th>Items</th>
<th>count=5</th>
<th>front = rear=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>G</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

Queue Overflow!!

- **Declaration and Initialization of a Queue.**

```c
#define MAXQUEUE 10 /* size of the queue items*/
struct queue{
    int front;
    int rear;
    int items[MAXQUEUE];
};
struct queue q;
q.front = MAXQUEUE-1;
q.rear= MAXQUEUE-1;
```

- **Remove Operation**

```c
int remove(struct queue *qptr)
{
    if(qptr->front == qptr->rear){
        printf("Queue underflow");
        exit(1);
    }
    if(qptr->front == MAXQUEUE-1)
        qptr->front=0;
    else
        qptr->front++;
    return qptr->items[qptr->front];
}
```
**Insert Operation**

```c
void insert(struct queue *qptr, int x)
{
    if(qptr->rear == MAXQUEUE-1)
        qptr->rear=0;
    else
        qptr->rear++;

    if(qptr->rear == qptr->front){
        printf("Queue overflow");
        exit(1);
    }
    qptr->items[qptr->rear]=x;
}
```

**Ex:** Write a program to simulate a queue of 5 integer items, where the program asks the user to enter number of items to be inserted/removed from the queue. In the case of insertion the programmer will have to enter the items one by one.

```c
#include<stdio.h>
#include<stdlib.h>
#define  MAXQUEUE 5 /* size of the queue items*/
struct queue{
    int front;
    int rear;
    int items[MAXQUEUE];
};
void insert(struct queue *, int);
int remove(struct queue *);
int main()
{
    struct queue q;
    q.front = MAXQUEUE-1;
    q.rear= MAXQUEUE-1;
    int n,i;
    int x;
    int choice;
    do{
        printf("Enter the type of operation\n");
        printf("Press 1 to insert into the queue\n");
        printf("Press 2 to remove from the queue\n");
        printf("Press 3 to quit\n");
        scanf("%d",&choice);
        switch(choice){
        case 1:
            printf("Enter number of insertions:");
            scanf("%d",&n);
            for(i=0;i<=n-1;i++){
                printf("Enter the item:");
                scanf("%d",&x);
                insert(&q,x);
            }
            break;
```
case 2:
    printf("Enter number of removals:");
    scanf("%d",&n);
    for(i=0;i<=n-1;i++){
        x=remove(&q);
        printf("\nRemoved item:%d",x);
    }
    break;
default:
    printf("\nWrong entry, Try again!!");
    break;
}
}while(choice != 3);
return 0;
}

void insert(struct queue *qptr, int x)
{
    if(qptr->rear == MAXQUEUE-1)
        qptr->rear=0;
    else
        qptr->rear++;

    if(qptr->rear == qptr->front){
        printf("Queue overflow");
        exit(1);
    }
    qptr->items[qptr->rear]=x;
}

int remove(struct queue *qptr)
{
    if(qptr->front == qptr->rear){
        printf("Queue underflow");
        exit(1);
    }
    if(qptr->front == MAXQUEUE-1)
        qptr->front=0;
    else
        qptr->front++;
    return qptr->items[qptr->front];
}
Ex: Write a program to simulate a queue structure with following specifications:
- User will be able to insert an item, remove an item, display all of the queue items and clear the queue.
- The items will be floating point numbers.

```c
#include<stdio.h>
#include<stdlib.h>
#define MAXQUEUE 5 /* size of the queue items*/
struct queue{
    int front;
    int rear;
    float items[MAXQUEUE];
};
void insert(struct queue *, float);
float remove(struct queue *);
void display(struct queue *);
void clear(struct queue *);

int main()
{
    int choice;
    float x;
    struct queue q;
    q.front = MAXQUEUE-1;
    q.rear= MAXQUEUE-1;

    do{
        printf("Enter your choice:\n");
        printf("Press 1 to insert \n");
        printf("Press 2 to remove \n");
        printf("Press 3 to display \n");
        printf("Press 4 to clear \n");
        printf("Press 5 to quit\n");
        scanf("%d",&choice);
        switch(choice){
        case 1:
            printf("Enter the item to be inserted:");
            scanf("%f",&x);
            insert(&q,x);
            break;
        case 2:
            x=remove(&q);
            printf("\nRemoved item:%f",x);
        break;
        case 3:
            display(&q);
        break;
        case 4:
            clear(&q);
        break;
        default:
            printf("\nWrong entry, Try again!!");
        break;
        }
    }while(choice != 5);
    return 0;
}
```
void insert(struct queue *qptr, float x)
{
    if(qptr->rear == MAXQUEUE-1)
        qptr->rear=0;
    else
        qptr->rear++;

    if(qptr->rear == qptr->front){
        printf("Queue overflow");
        exit(1);
    }
    qptr->items[qptr->rear]=x;
}

float remove(struct queue *qptr)
{
    if(qptr->front == qptr->rear){
        printf("Queue underflow");
        exit(1);
    }
    if(qptr->front == MAXQUEUE-1)
        qptr->front=0;
    else
        qptr->front++;
    return qptr->items[qptr->front];
}

void display(struct queue *qptr)
{
    int f,r;
    f=qptr->front;
    r=qptr->rear;
    while(qptr->front !=qptr->rear){
        if(qptr->front == MAXQUEUE-1)
            qptr->front =0;
        else
            qptr->front++;
        printf("%5.2f",qptr->items[qptr->front]);
    }
    printf("\n");
    qptr->front=f;
    qptr->rear=r;
}

void clear(struct queue *qptr)
{
    qptr->front=MAXQUEUE-1;
    qptr->rear =MAXQUEUE-1;
    printf("Now the Queue is Empty\n");
}