

Advanced Computer Algorithms (CSC813)

Review of Linear Data Structure cont'd

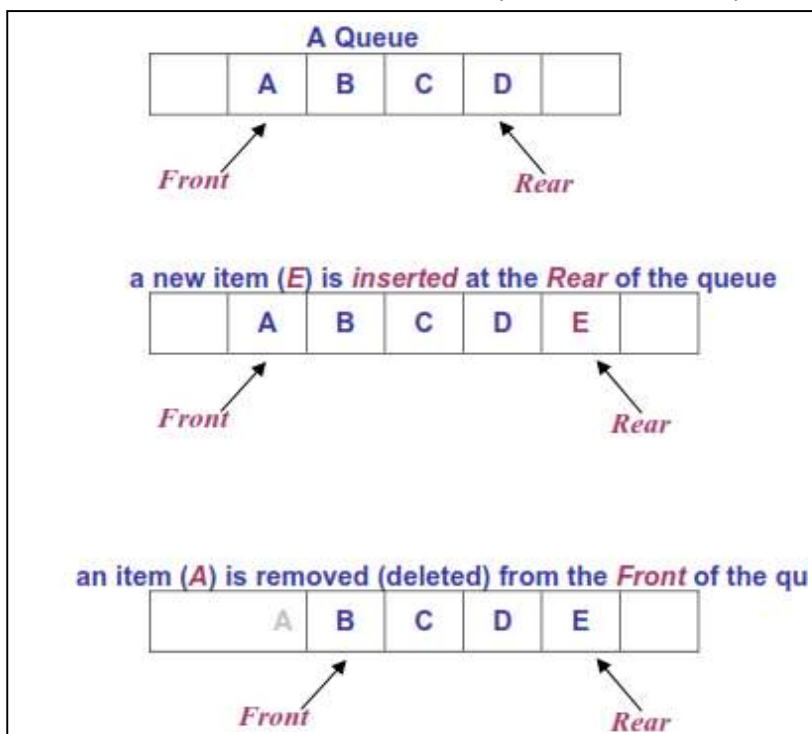
QUEUES

- 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).

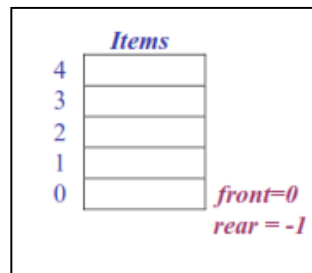


QUEUE OPERATIONS

- ✓ 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

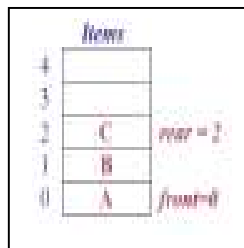
INITIALIZE THE QUEUE (USING LINEAR ARRAY)

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.

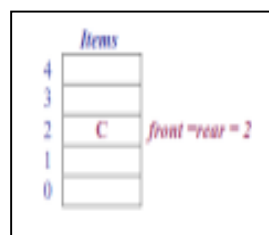


INSERT / REMOVE ITEMS

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



Remove two items from the front of the queue.



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

Items	
4	E
3	D
2	C
1	
0	

rear = 4
front = 2

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)																									
<p>◆ Initialize the queue.</p> <table> <tr><th colspan="2">Items</th></tr> <tr><td>4</td><td></td></tr> <tr><td>3</td><td></td></tr> <tr><td>2</td><td></td></tr> <tr><td>1</td><td></td></tr> <tr><td>0</td><td></td></tr> </table> <p><i>count=0</i> <i>front=rear=4</i></p>	Items		4		3		2		1		0		<p>◆ Insert A,B,C to the rear of the queue.</p> <table> <tr><th colspan="2">Items</th></tr> <tr><td>4</td><td></td></tr> <tr><td>3</td><td></td></tr> <tr><td>2</td><td>C</td></tr> <tr><td>1</td><td>B</td></tr> <tr><td>0</td><td>A</td></tr> </table> <p><i>count=3</i> <i>front=4</i> <i>rear=2</i></p>	Items		4		3		2	C	1	B	0	A
Items																									
4																									
3																									
2																									
1																									
0																									
Items																									
4																									
3																									
2	C																								
1	B																								
0	A																								
<p>◆ Remove two items from the queue.</p> <table> <tr><th colspan="2">Items</th></tr> <tr><td>4</td><td></td></tr> <tr><td>3</td><td></td></tr> <tr><td>2</td><td>C</td></tr> <tr><td>1</td><td></td></tr> <tr><td>0</td><td></td></tr> </table> <p><i>count=1</i> <i>rear=2</i> <i>front=1</i></p>	Items		4		3		2	C	1		0		<p>◆ Insert D,E to the queue.</p> <table> <tr><th colspan="2">Items</th></tr> <tr><td>4</td><td>E</td></tr> <tr><td>3</td><td>D</td></tr> <tr><td>2</td><td>C</td></tr> <tr><td>1</td><td></td></tr> <tr><td>0</td><td></td></tr> </table> <p><i>count=3</i> <i>rear=4</i> <i>front=1</i></p>	Items		4	E	3	D	2	C	1		0	
Items																									
4																									
3																									
2	C																								
1																									
0																									
Items																									
4	E																								
3	D																								
2	C																								
1																									
0																									
<p>◆ Insert F to the queue.</p> <table> <tr><th colspan="2">Items</th></tr> <tr><td>4</td><td>E</td></tr> <tr><td>3</td><td>D</td></tr> <tr><td>2</td><td>C</td></tr> <tr><td>1</td><td></td></tr> <tr><td>0</td><td>F</td></tr> </table> <p><i>count=4</i> <i>front=1</i> <i>rear=0</i></p>	Items		4	E	3	D	2	C	1		0	F	<p>◆ Insert G to the queue. (Queue is full)</p> <table> <tr><th colspan="2">Items</th></tr> <tr><td>4</td><td>E</td></tr> <tr><td>3</td><td>D</td></tr> <tr><td>2</td><td>??</td></tr> <tr><td>1</td><td>G</td></tr> <tr><td>0</td><td>F</td></tr> </table> <p><i>count=5</i> <i>front=rear=1</i></p> <p>Queue Overflow!!</p>	Items		4	E	3	D	2	??	1	G	0	F
Items																									
4	E																								
3	D																								
2	C																								
1																									
0	F																								
Items																									
4	E																								
3	D																								
2	??																								
1	G																								
0	F																								

Declaration and Initialization of a Queue.

```
#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

```
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

```
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);
    }
}
```

```

    qpтр->items[qpтр->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.

```

#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("\nEnter 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("\nEnter 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.

```

#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("\nEnter 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 *qp)
{

    qp->front=MAXQUEUE-1;
    qp->rear =MAXQUEUE-1;
    printf("Now the Queue is Empty\n");
}
```

Queues (Continued)

Examples on Queue Applications

Priority Queues

- Ascending Priority Queues
- Descending Priority Queues

- **Ex:** Assume that we have a queue of integer numbers. Write a function, *QueueSearch* to search a given key element in the queue until the search key is found. Once the search key is found, the function returns its position in the queue, otherwise returns -1 to indicate that the searched key is not in the queue.

Assume that the Queue contains integer elements and has the following structure:

```
typedef struct{
    int front;
    int rear;
    int items[MAXQUEUE]; /* Assume that MAXQUEUE is defined*/
}QUEUE;
```

Then the following function can be written:

```
int QueueSearch(QUEUE *qp, int searchkey)
{
    int pos = -1, f;
    f=qp->front;
    while (qp->front != qp->rear) {
        if (qp->front == MAXQUEUE-1)
            qp->front = 0;
        else
            qp->front++;
        if (qp->items[qp->front] == searchkey) {
            pos = qp->front;
            qp->front = f;
            return pos;
        }
    }
    qp->front=f;
    return pos;
}
```

Ex: Write a function, *QueueCopyReverse*, to copy the integer elements of Queue 1 to Queue 2 in reverse order.

Assume that there is enough space in Queue 2 for copying and the size of both of the Queues is MAXQUEUE.

```
void QueueCopyReverse(QUEUE *qp1, QUEUE *qp2)
{
    int r,x;
    r=qp1->rear;
    do{
        x = qp1->items[qp1->rear];
        insert(qp2,x);
        if(qp1->rear ==0)
            qp1->rear = MAXQUEUE-1;
        else
            qp1->rear --;
    }while(qp1->rear != qp1->front);
    qp1->rear = r;
}
```

```

}

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

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

```

PRIORITY QUEUES

The priority queue is a data structure in which intrinsic ordering of the elements determines the results of its basic operations.

An **ascending priority queue** is a collection of items into which items can be inserted arbitrarily and from which only the smallest item can be removed. On the other hand a **descending priority queue** allows only the largest item to be removed.

Insertion

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

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

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

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

***PriQremove Operation using removing the element with highest priority and shifting the elements up in the array and decrementing rear.
Consider Ascending Priority Queue.***

```
int PRIQremove(QUEUE *qp)
{
    int smallest, loc, f, i;
    f=qp->front;
    if(qp->front == qp->rear){
        printf("Queue underflow");
        exit(1);
    }
    smallest = qp->items[(qp->front+1)%MAXQUEUE];
    loc = (qp->front+1)%MAXQUEUE;
    (qp->front++)%MAXQUEUE; /* Circular increment*/
    while(qp->front != qp->rear){
        if(qp->items[(qp->front+1)%MAXQUEUE] < smallest){
            smallest = qp->items[(qp->front+1)%MAXQUEUE];
            loc = (qp->front+1)%MAXQUEUE;
        }
        qp->front = (qp->front++)%MAXQUEUE; /* Circular
inc.*/
    }
    while(loc != qp->rear){
        qp->items[loc] = qp->items[(loc+1)%MAXQUEUE];
        (loc++)%MAXQUEUE;
    }
    qp->front=f;
    if(qp->rear == 0) /*Decrement rear after removing
one item*/
```

```
    qptr->rear = MAXQUEUE -1;
else
    qptr->rear--;
return smallest;
}
```

Insert Operation of Priority Queue is the same as the insert of the non-priority queues.

```
void insert(struct queue *qptr, int x)
{
qptr->rear = (qptr->rear++)%MAXQUEUE; /*Circular
increment*/
if(qptr->rear == qptr->front){
    printf("Queue overflow");
    exit(1);
}
qptr->items[qptr->rear]=x;
}
```