CS214-Data Structure

Lecturer: Dr. Salwa Osama

Queue

Queue - What?

Let's Refresh our Mind

Refresh your mind

1. Queue is (**Linear** – **non linear)** data structure
2. Queue is (**non primitive - primitive**)
3. Queue is a (**Heterogenous – homogeneous)**collection of elements**.**
4. The new element is added in (**front - rear**) of the queue. And remove element from (**front – rear**) of the
5. The queue is called (First in first out, first in last out, last in first out, last in last out)
6. Examples of queue are ….(select one or more)
    1. Toll Station
    2. in Big super market
    3. Printer
    4. Function calls
    5. Backtracking
    6. Delimiter Checking
    7. All of mentioned

Queue- What

- Queue is **Linear** - **non-primitive** data structure
- Queue is a **homogeneous collection of elements.**
- In which new elements are added at one end called **rear or tail**, *and the existing elements are deleted from other end called* **front or head**.

So, the Queue is called First-in-First-out (FIFO)

- Examples of queues: Toll Station, Check-out in Big super market, Printer, …

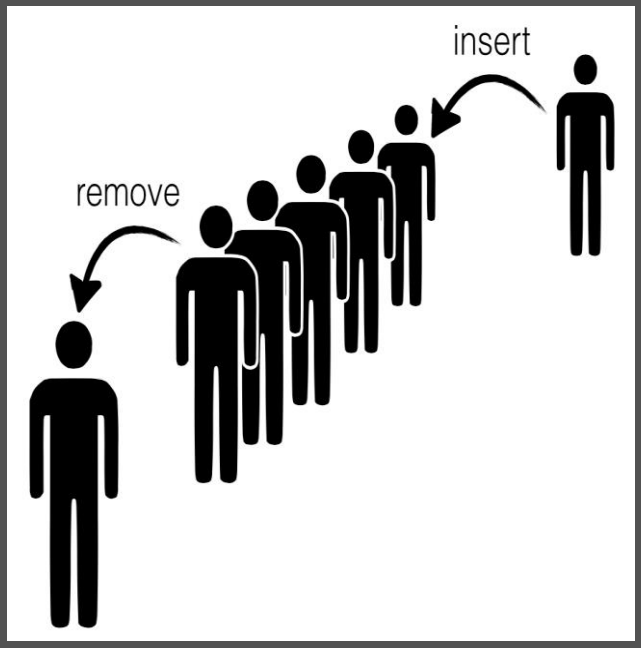
Queue- What

20
5
30

- Add (20)
- Add(5)
- Add(30)
- Delete
- Delete
- Add(0)
- Add(-3)

Stack Animation by Y. Daniel Liang (pearsoncmg.com)

Implementation



User View

Queue

OPERATIONS PERFORMED ON QUEUE

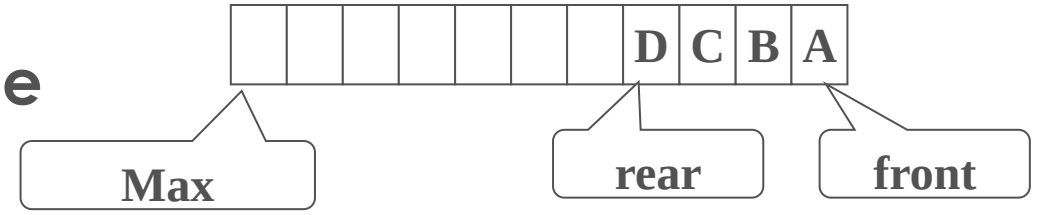
- **Create** the queue, leaving it empty.
- Determine whether the queue **is empty** or not.
- Determine whether the queue **is full** or not.
- **Enqueue** a new entry onto the end of the queue
- **Dequeue** the entry at the front of the queue.

Queue Implementation

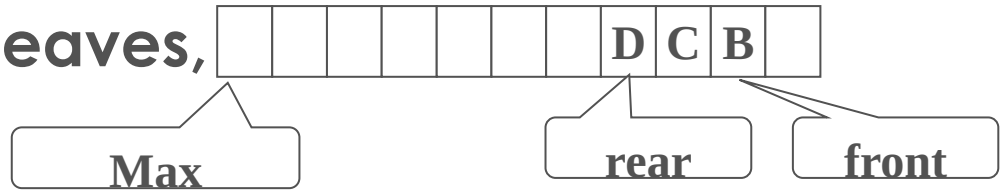
```
#define     MAX  10
typedef     char  EntryType;
typedef   struct {
    int       front;
    int       rear;
    EntryType    entry[MAX];
} QueueType;
```

Queue Implementation

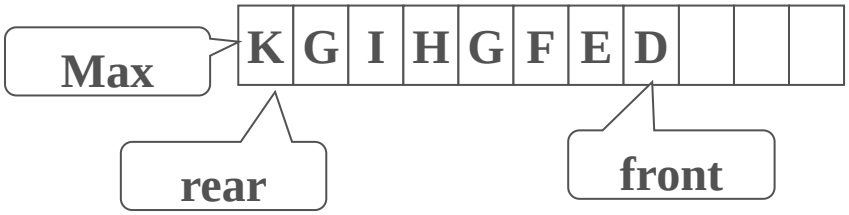
- A queue



- After A leaves,



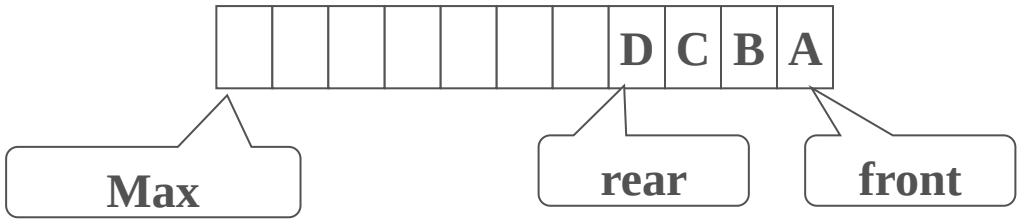
- After B and C leave



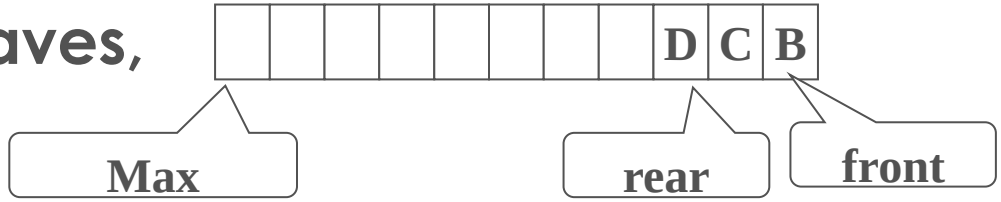
How can we insert a new element?!!

Queue Implementation

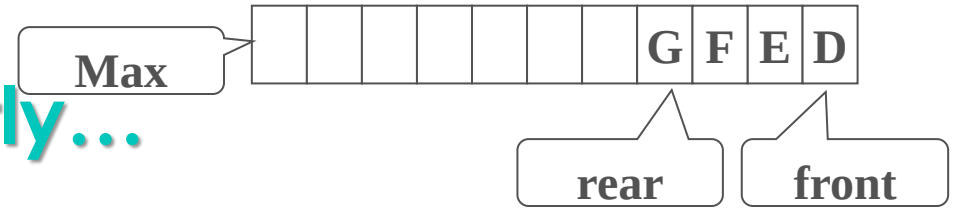
One solution is to shifting all items to front when dequeue operation.



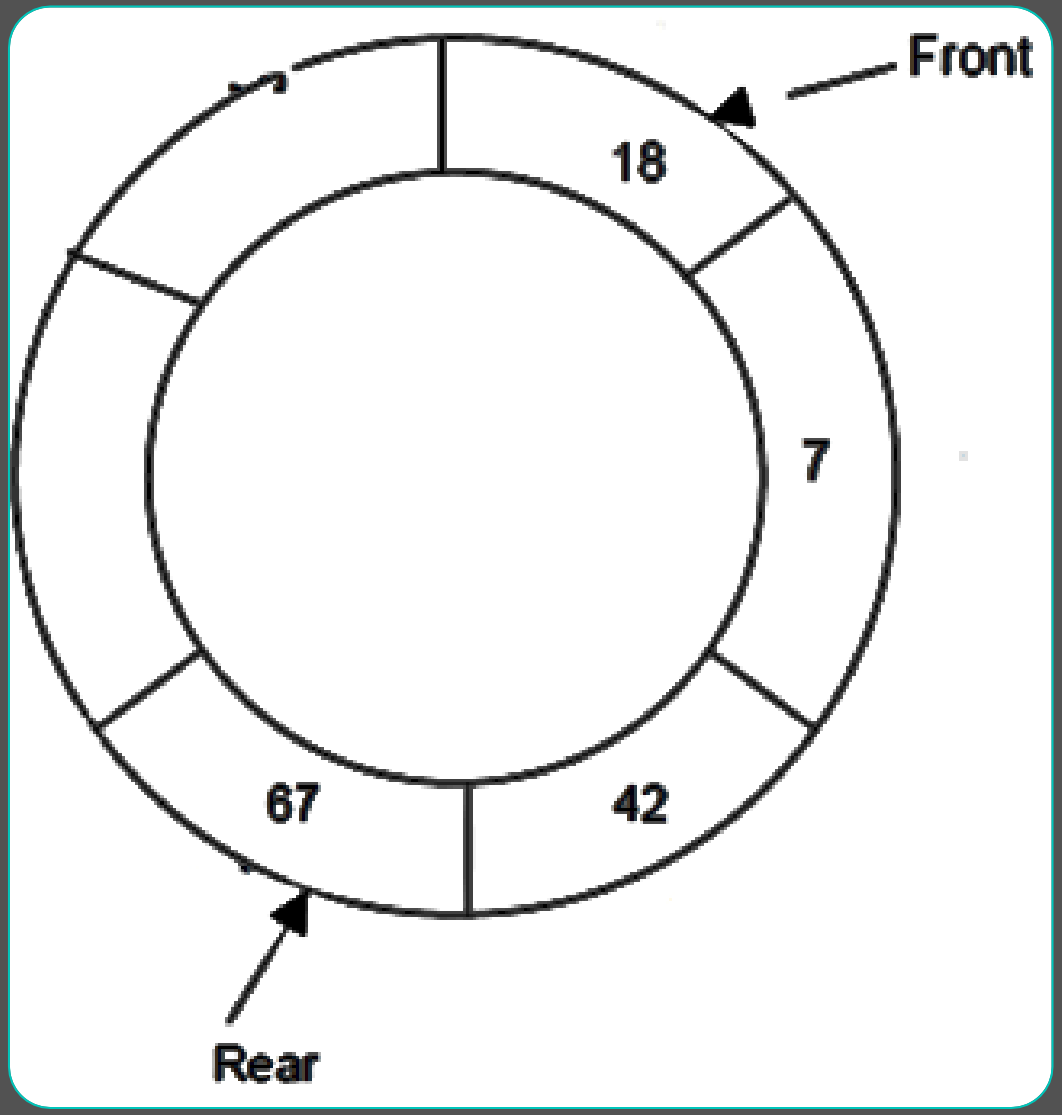
After A leaves,



After B and C leave



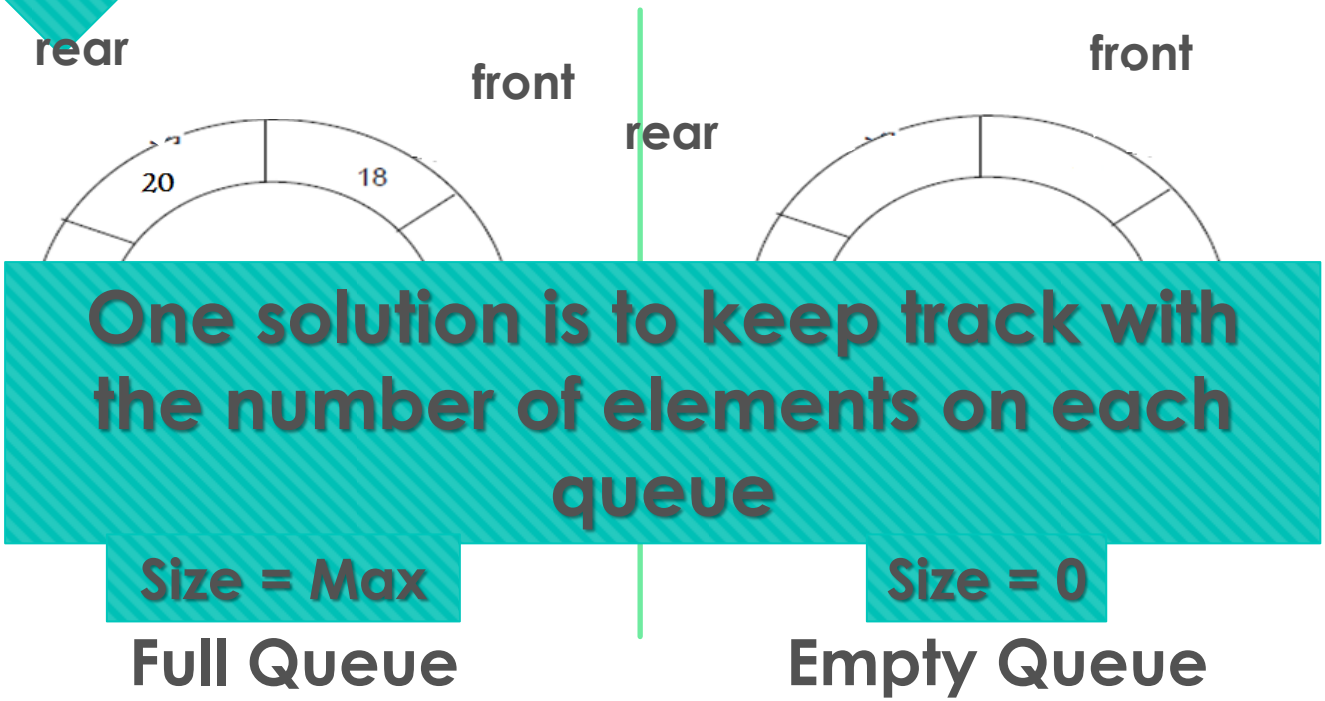
Too Costly…



Queue Implementation

- A better solution is a circular Queue

Circular Queue



In both cases *front* comes after *rear* with one step. Then, how can we distanguish between the two cases?!!

Queue Implementation

```
#define     MAX     10
typedef     char    EntryType;
typedef  struct {
    int         front;
    int         rear;
    int         size;
    EntryType     entry[MAX];
} QueueType;
```

Queue Implementation

- **Create operation:**

  Pre: None.

  Post: The queue is initialized to be empty.

```
void CreateQueue(QueueType *q){
  q->front= 0;
  q->rear =  max -1;
  q->size = 0;
}
```

Queue Implementation

Queue empty operation:

Pre: The queue is initialized.

Post: If the queue is empty (1) is returned. Otherwise (0) is returned.

```
int QueueEmpty(QueueType q){
    return (q.size==0);}
```

Queue full operation:

Pre: The queue is initialized.

Post: If the queue is full (1) is returned. Otherwise (0) is returned.

```
int QueueFull(QueueType q){
    return (q.size==MAX);}
```

Queue Implementation

- **Enqueue operation:**

  Pre: The queue is initialized and is not full.

  Post: Item is added to the end of the queue.

```
void Enqueue(EntryType item, QueueType *q){
        q->rear ++;
        q->rear = (q->rear+1)%Max; //to circulate the queue
        q->entry[q->rear] = item;
        q->Size ++;
}
```

It is for linear not circular

Queue Implementation

- **Dequeue operation:**

  Pre: The Queue is initialized and is not empty.

  Post: The front element of the Queue is removed from it and is assigned to item.

```
void dequeue (Entry *item, QueueType *q){
        *item = q->entry[ q->front];
        q->front = (q->front +1)% MAX
      q->Size--;
}
```

Exercise

- In the implentation level of the queue ADT, write the *QueueTraverse* which is defined as :

  void TraverseQueue(QueueType pq, void (*pf)(Entry *))*

  Where *pf* is a pointer to a function that is passed over all of the queue **pq* elements to perform a task

- Use the previous function to print on the screen all a queue elements in the user level.

Exercise

```
void TraverseQueue(Queue* q, void
   (*f)(EntryType*)){
   int  i, siz;
   for(i=q->front, siz=0; siz<q->size; siz++){

      (*f)(&q->entry[i]);

            i=(i+1)%MAX;
      }
}
```

could be

Print(&q->entry(i))
Or
Increment(&q->entry(i))
Or
…

Exercise

```
void Print(EntryType *e){
    printf("e is: %d\n", *e);
}
void main(){
    QueueType q;
    CreateQueue(&q);
     ...
    TraverseQueue(&q, &Print);
}
```

Excercise2

- Write a function that returns a copy from the first element in a queue. (Implementation level)

Excercise2

```
Type GetFirstElement(Queue queue) {
    if (isQueueEmpty(queue)) {
        printf("Queue is empty. Cannot retrieve the front element.\n");
        return (Type)-1;
    } else {
        return queue.data[queue.front];
    }
}
```

- Write a function that returns a copy from the first element in a queue. (Implementation level)

```
int mainFE() {
    Queue myQueue;
    createQueue(&myQueue);
    enqueue(&myQueue, 10);
    enqueue(&myQueue, 20);
    enqueue(&myQueue, 30);
    Type frontElement = GetFirstElement(myQueue);
    printf("Front element: %d\n", frontElement);
    return 0;
}
```

Exercises3

- We (as a user for QueueADT) have two filled queues; the first queue holds section code while the other holds group code (where number of groups inside the section is maximum 10). Merge those numbers (section code*10+group code) in a newly created queue.

Exercises3

```
Queue mergeCodes(Queue sectionCodes, Queue groupCodes) {
    Queue mergedQueue;
    createQueue(&mergedQueue);

    while (!isQueueEmpty(sectionCodes) && !isQueueEmpty(groupCodes)) {
        int sectionCode = dequeue(&sectionCodes);
        int groupCode = dequeue(&groupCodes);

        int mergedCode = sectionCode * 10 + groupCode;

        enqueue(&mergedQueue, mergedCode);
    }

    return mergedQueue;
}
```

- We (as a user for QueueADT) have two filled queues; the first queue holds section code while the other holds group code (where number of groups inside the section is maximum 10). Merge those numbers (section code*10+group code) in a newly created queue.

Exercises3

```
int main() {
    Queue sectionCodes;
    createQueue(&sectionCodes);
    Queue groupCodes;
    createQueue(&groupCodes);
    enqueue(&sectionCodes, 1);
    enqueue(&sectionCodes, 2);
    enqueue(&sectionCodes, 3);
    enqueue(&groupCodes, 1);
    enqueue(&groupCodes, 2);
    enqueue(&groupCodes, 3);
    Queue mergedQueue = mergeCodes(sectionCodes, groupCodes);
    while (!isQueueEmpty(mergedQueue)) {
        int mergedCode = dequeue(&mergedQueue);
        printf("Merged Code: %d\n", mergedCode);
    }
    return 0;
}
```


Any
Questions