CS 214 – DATA STRUCTURES FALL 2015

Lectures 4 & 5– Queues
Group **B**

9th & 16th October 2017

Dr. Mai Hamdalla

mai@fci.helwan.edu.eg

House Keeping

- Sections
  - Assignment
  - Scope
- Homework 1
- Homework 2
  - Uploaded to Piazza
- Mid-term exam
  - 7^{th} week

STACKS

What is a Stack?

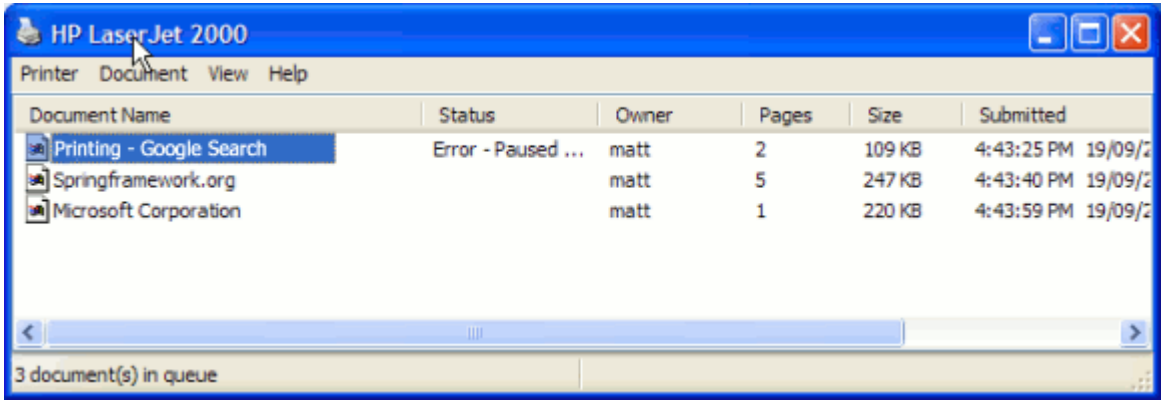
- Stack is LIFO Structure **[Last in First Out]**
- Stack is Ordered List of Elements of **Same Type**.
- Stack is a Linear List.
- In Stack, all Operations are permitted at only one end called **Top.**

Operations Performed on Stack

- Stack header file: `stack.h`
- Stack code file: `stack.c` (stack implementation in C)
- We can test the stack data structure in the `main.c` file (stack program in C)

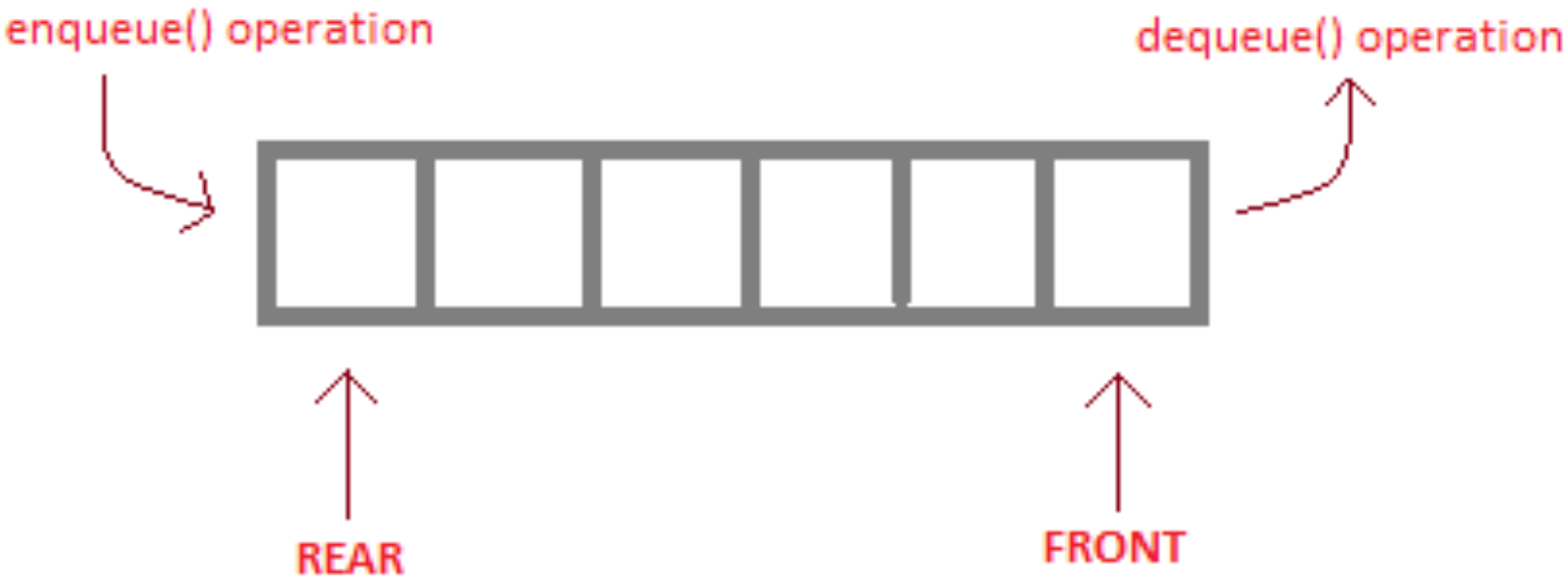
QUEUES

What is a Queue?



What is a Queue?

- Queue is FIFO Structure
- Queue is Ordered List of Elements of **Same Type**.
- New elements are added at one end called **rear** or **tail**.
- Existing elements are deleted from other end called **front** or **head**.



enqueue( ) is the operation for adding an element into Queue.

dequeue( ) is the operation for removing an element from Queue .

QUEUE DATA STRUCTURE

Queue Example -

- Trace the following example showing the how the **queue** would look and the value of the **front** and **rear** after each operation:
  - Add (5)
  - Add (-17)
  - Delete
  - Add (8)
  - Delete
  - Add (-2)

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 CONTIGUOUS IMPLEMENTATION

Queue Struct

```
#define MAX 10

typedef char entry_type;
typedef struct{
    int front;
    int rear;
    entry_type entry[MAX];
} queue_type;
```

Queue Initialization

- **Pre:** None.
- **Post:** The queue is initialized to be empty.

```
void create_queue(queue_type *q){
     q->front = 0;
     q->rear = -1;
}
```

Queue Enqueue Operation

- **Pre:** The queue is initialized and is not full.
- **Post:** Item is added to the rear of the queue.

```
void enqueue(entry_type item,
                    queue_type *q){
  q->rear = q->rear+1;
  q->entry[q->rear] = item;
}
```

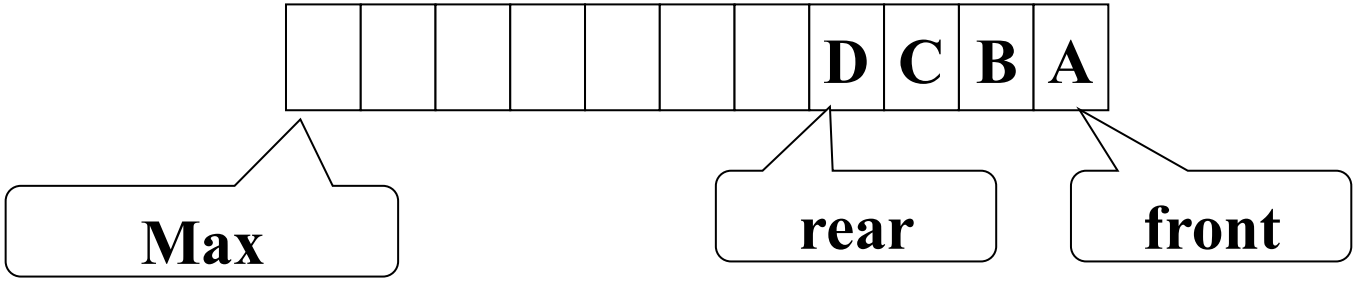
Queue Dequeue Operation

- **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.

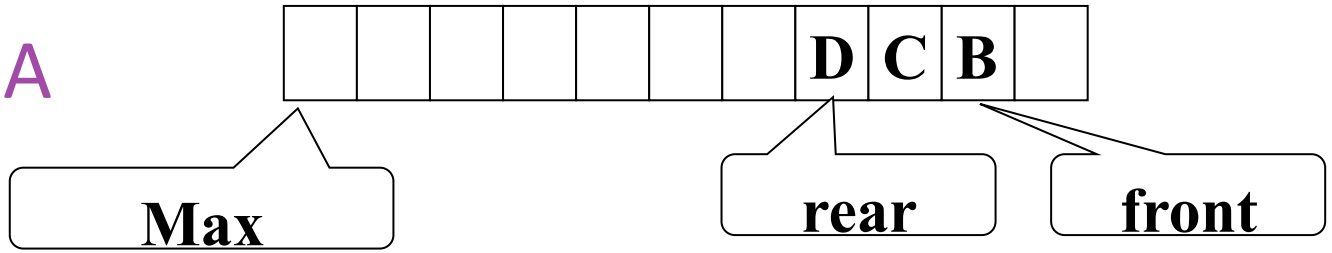
```
void dequeue(entry_type *item,
                    queue_type *q){
    *item = q->entry[q->front];
    q->front = q->frontr+1;
}
```

Queue Implementation

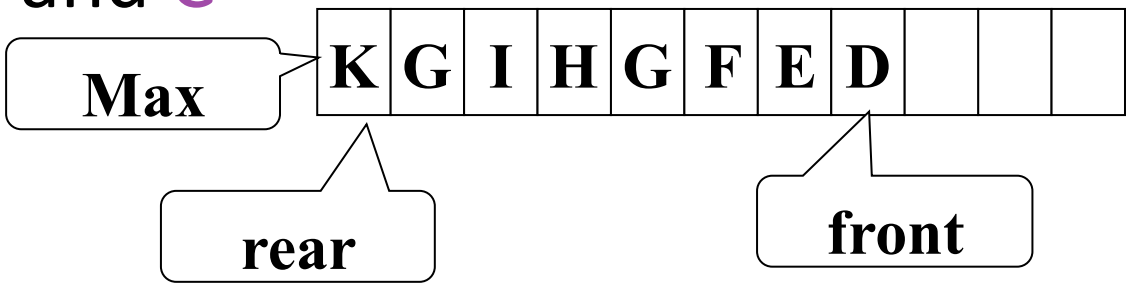
- A queue



- Dequeue A



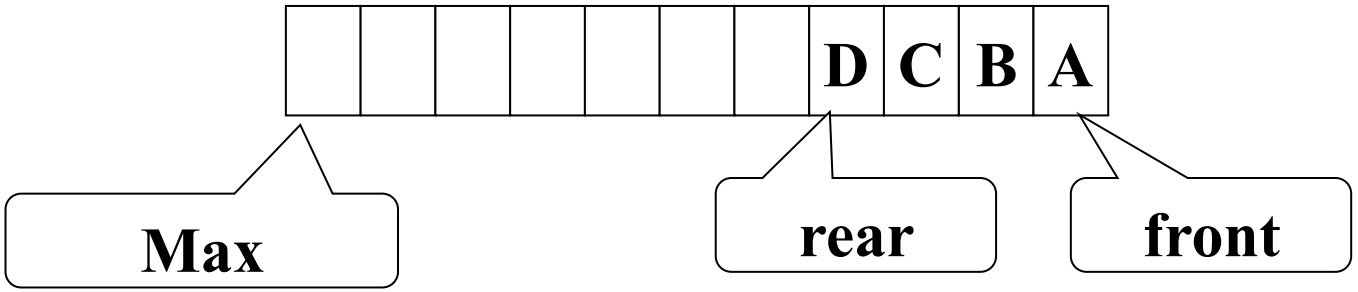
- Dequeue B and C



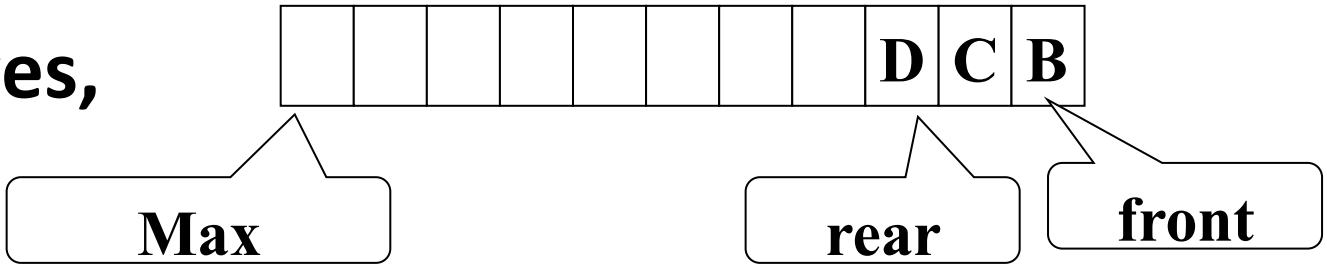
How can we insert a new element?!!

Queue Implementation

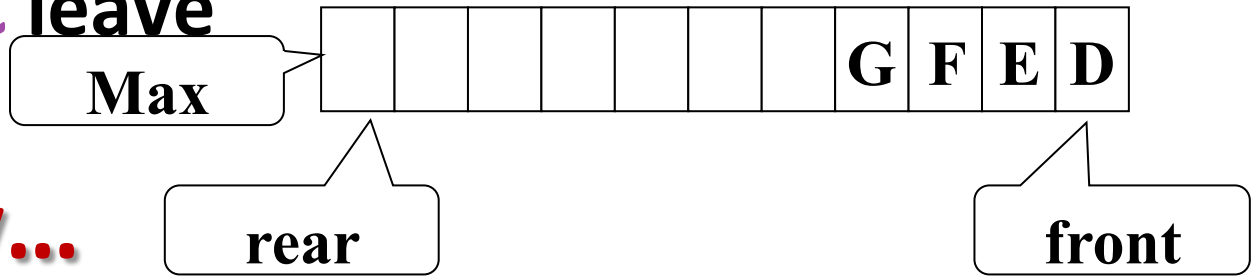
One solution is to shift all items to front when dequeue operation.



After A leaves,



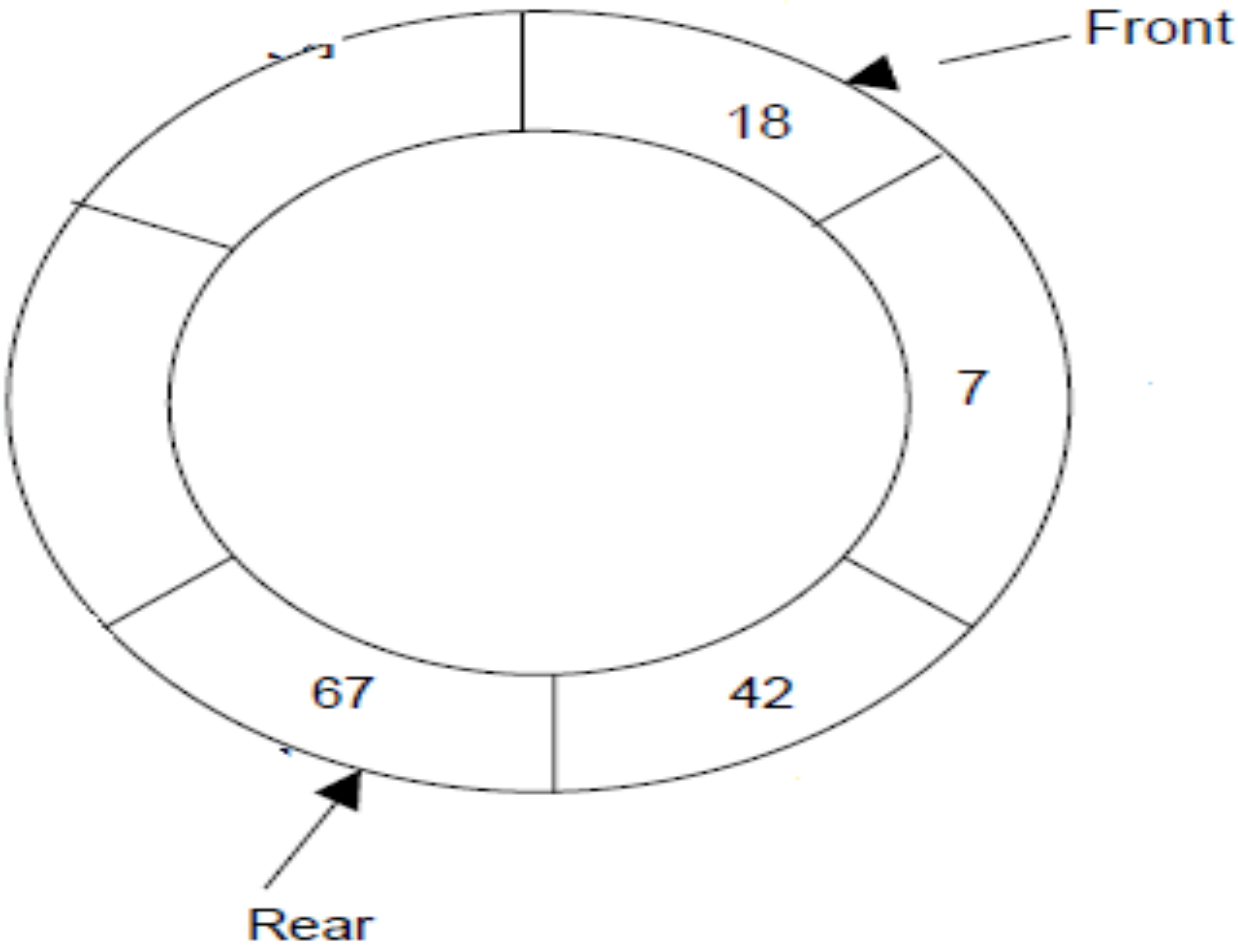
After B and C leave



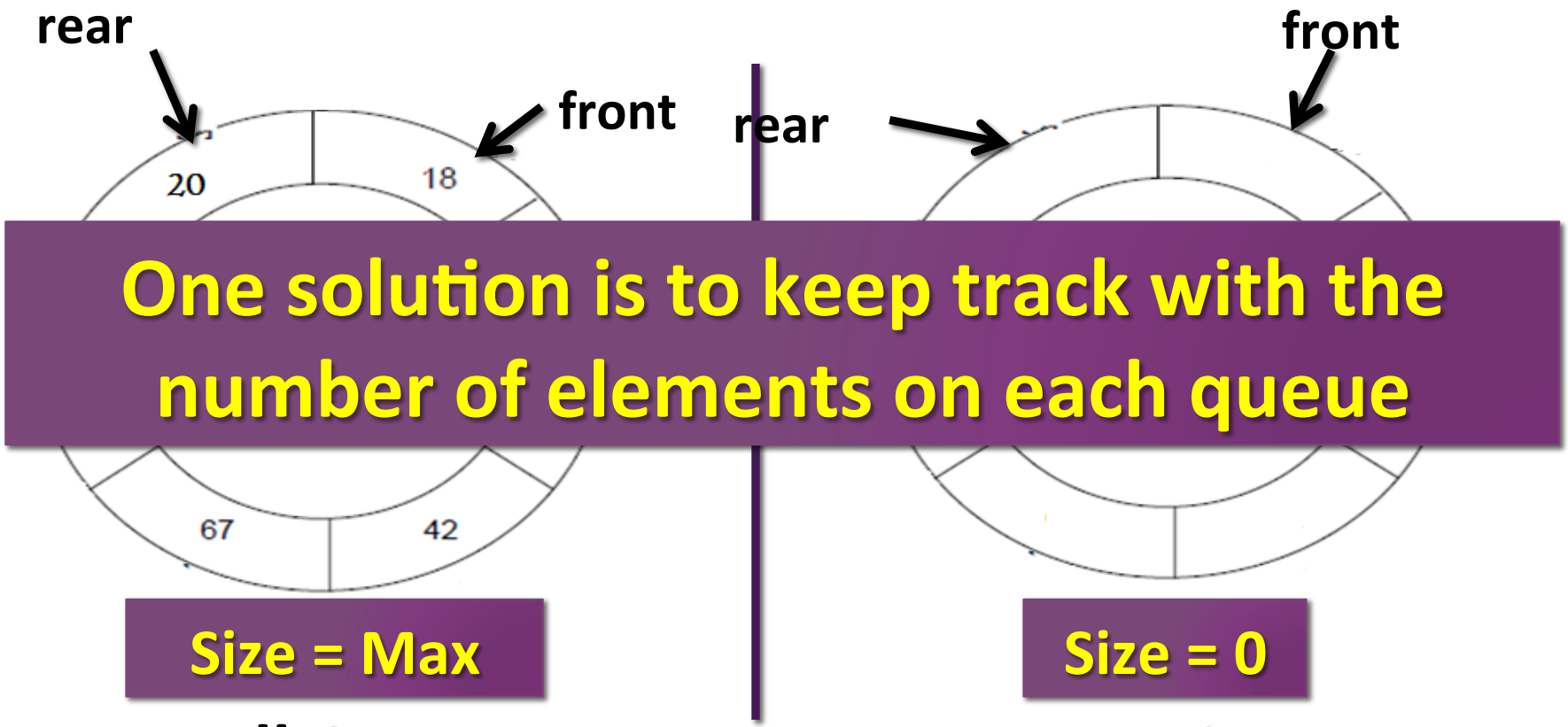
Too Costly…

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 distinguish between the two cases?!!

CIRCULAR QUEUE CONTIGUOUS IMPLEMENTATION

Queue Struct

```
#define MAX 10

typedef char entry_type;
typedef struct{
     int front;
     int rear;
     int size;
     entry_type entry[MAX];
} queue_type;
```

Queue Initialization

- **Pre:** None.
- **Post:** The queue is initialized to be empty.

```
void create_queue(queue_type *q){
     q->front = 0;
     q->rear = -1;
     q->size = 0;
}
```

Queue Enqueue Operation

- **Pre:** The queue is initialized and is not full.
- **Post:** Item is added to the rear of the queue.

```
void enqueue(entry_type item,
                    queue_type *q){
   // to circulate the queue
   q->rear = (q->rear+1)%MAX;
   q->entry[q->rear] = item;
   q->size++;
}
```

Queue 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_type *item,
                    queue_type *q){
    *item = q->entry[q->front];
    q->front = (q->frontr+1)%MAX;
    q->size--;
}
```

Queue Empty

- **Pre:** The queue is initialized.
- **Post:** If the queue is empty, (1) is returned.
  Otherwise (0) is returned.

```
int is_queue_empty (queue_type q)
 {
    return (q.size == 0);
}
```

Queue Full

- **Pre:** The queue is initialized.
- **Post:** If the queue is full, (1) is returned, otherwise (0) is returned.

```
int is_queue_full (queue_type q) {
    return (q.si
}
```

EXERCISE

Exercise

- In the implementation level of the queue ADT, write the **`queue_traverse`** which is defined as:

```
void queue_traverse(queue_type *pq,
                    void (*pf)(queue_entry))
```

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

Exercise Soln.

```
void queue_traverse(queue_type *q,
                    void(*f)(queue_entry)){
  int   i, siz;
  for(
                                        ){
        (*f)(q->entry[i]);
        i=(i+1)%MAX;
    }
}
```

```
print(q->entry(i))or
increment(q->entry(i))or …
```

Exercise

Write a user lever function to use the `queue_traverse` function to print all a queue elements on the screen.

Exercise Soln. Cont'd

```
void print(entry_type e){
      printf("e is: %d\n", e);
}
void main(){
     queue_type q;
     //Initialize the queue
     create_queue(&q);

     …
     queue_traverse(&q, &print);
}
```

CS 214 – DATA STRUCTURES FALL 2015

Lectures 4 & 5– Queues

Group **B**

9th & 16th October 2017

Dr. Mai Hamdalla

mai@fci.helwan.edu.eg