FIFO Queues

CSE 2320 – Algorithms and Data Structures
Vassilis Athitsos
University of Texas at Arlington

FIFO Queues

- Stacks are *last-in first-out* queues.
- Another widely used model is *first-in first-out (FIFO)* queues.
- Examples of uses of FIFO queues:

FIFO Queues

- Stacks are *last-in first-out* queues.
- Another widely used model is *first-in first-out (FIFO)* queues.
- Examples of uses of FIFO queues:
- Program execution:
  - Requests for access to memory, disk, network...
- Resource allocation:
  - Forwarding network traffic in network switches and routers.
- Search algorithms.
  - More details later in the course

Put and Get

- The FIFO queue supports **insert** and **delete** as follows:
  - ~~**insert, push,**~~ **put:** This is what we call the insert operation when we talk about FIFO queues. It puts an item "at the end of the line".
  - ~~**delete, pop,**~~ **get**: This is what we call the delete operation when we talk about FIFO queues. It removes the item that is "at the head of the line".

- How can we implement FIFO queues?

FIFO Queues Using Lists

- A FIFO queue is essentially a list.
- **put(queue, item)** inserts that item at the **END** of the list.
- **get(queue)** removes (and returns) the item at the **beginning** of the list.
- What is the running time of these operations?

FIFO Queues Using Lists

- A FIFO queue is essentially a list.
- **put(queue, item)** inserts that item at the **END** of the list.
- **get(queue)** removes (and returns) the item at the **beginning** of the list.
- Both operations take O(1) time.
  - Assumption: the list data type contains a pointer to the last element.
  - Our new implementation at **lists.c** satisfies that assumption.

Queue Versus Stack Implementation

- Implementation-wise, compare:
  - list-based queue implementation.
  - list-based stack implementation.
- What are the key differences?

Queue Versus Stack Implementation

- Implementation-wise, compare:
  - list-based queue implementation.
  - list-based stack implementation.
- The only difference is in insertions.
  - Queues insert at the end of the list.
  - Stacks insert at the beginning of the list.
- It is very easy to implement queues starting from our list-based stack implementation.
  - Rename push to put.
  - Rename pop to get.
  - Change the get function to insert at the end.

FIFO Queues Using Arrays

- How do we implement queues using arrays?
- The stack definition looked like this:

```
typedef struct stack_struct * Stack;
struct stack_struct
{
    int max_size;
    int top_index;
    void ** items;
};
```

- What changes do we need to accommodate queues?

FIFO Queues Using Arrays

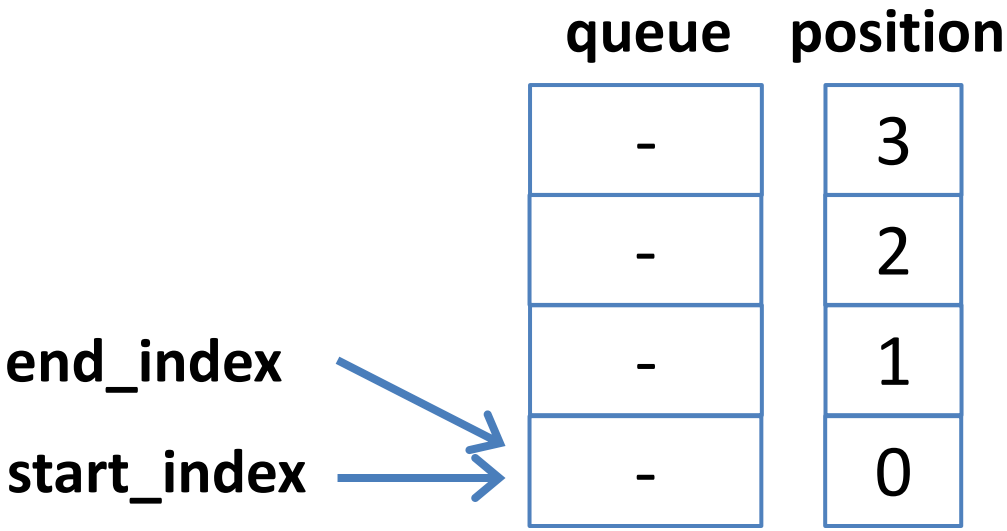
- How do we implement queues using arrays?
- A queue can be defined like this:

```
typedef struct queue_struct * queue;
struct queue_struct
{
   int max_size;
   int start_index;
   int end_index;
   void ** items;
};
```

- **end_index** tells us where to put a new item.
- **start_index** tells us where to remove an item from.

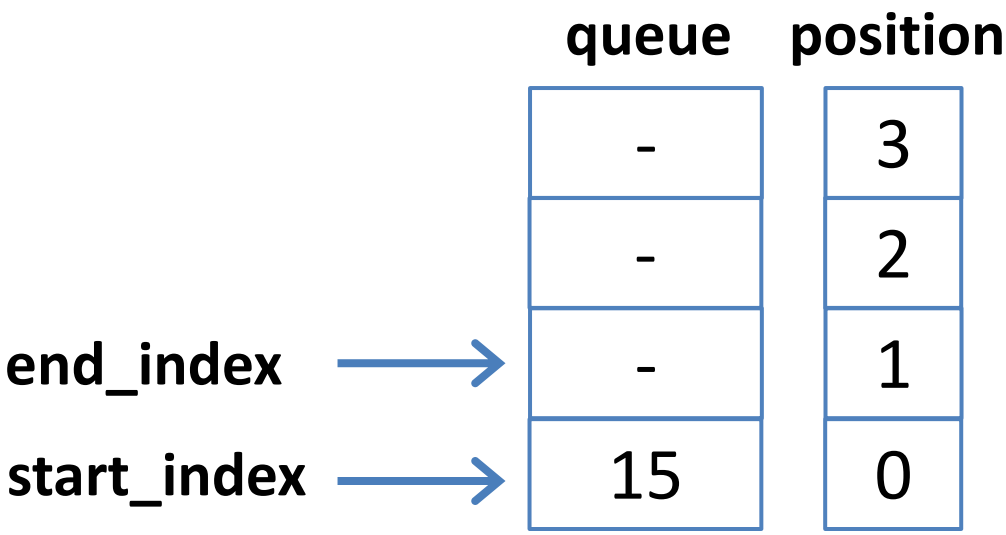
Examples of Put and Get

- put(15)
- put(20)
- get()
- put(30)
- put(7)
- put(25)
- get()
- put(12)
- get()
- get()



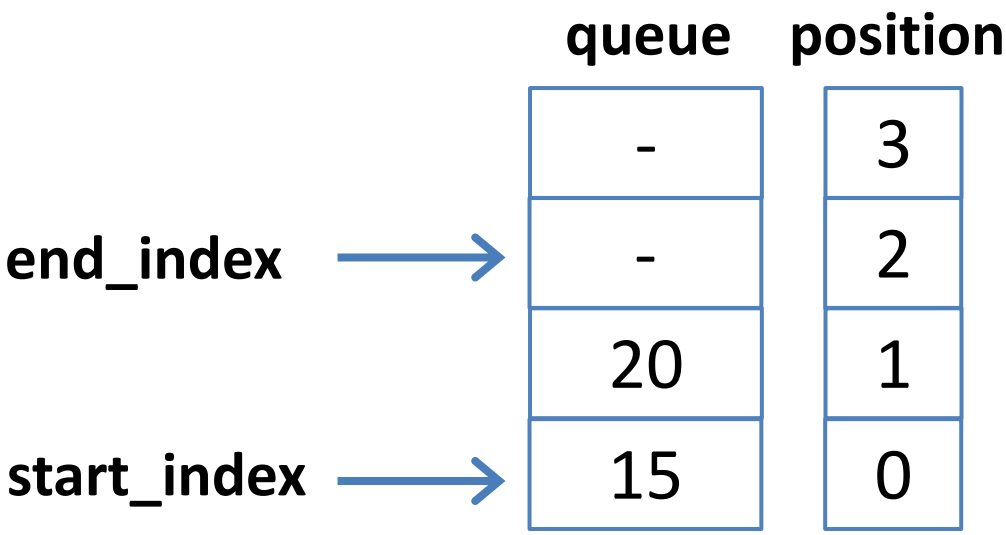
Examples of Put and Get

- **put(15)**
- put(20)
- get()
- put(30)
- put(7)
- put(25)
- get()
- put(12)
- get()
- get()



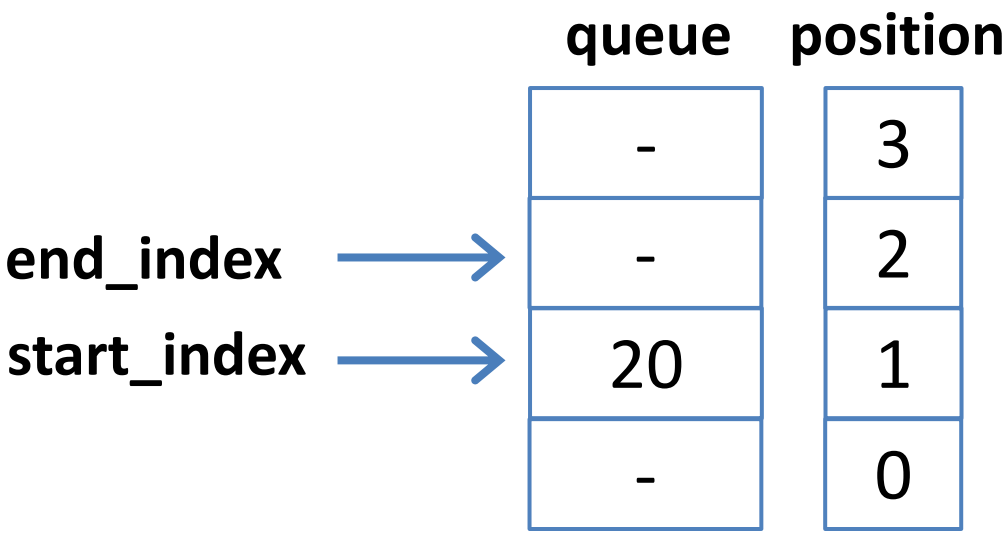
Examples of Put and Get

- put(15)
- **put(20)**
- get()
- put(30)
- put(7)
- put(25)
- get()
- put(12)
- get()
- get()



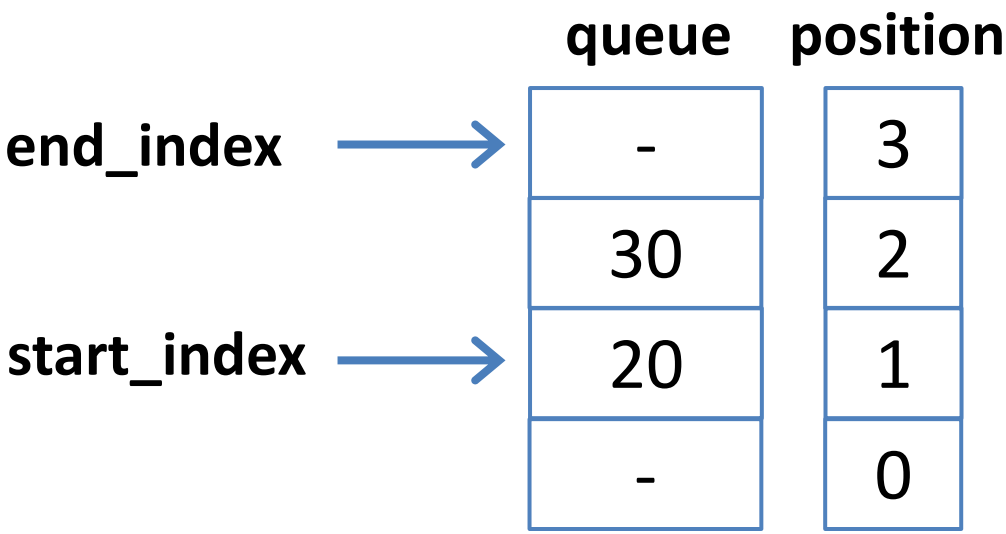
Examples of Put and Get

- put(15)
- put(20)
- **get(): returns 15**
- put(30)
- put(7)
- put(25)
- get()
- put(12)
- get()
- get()



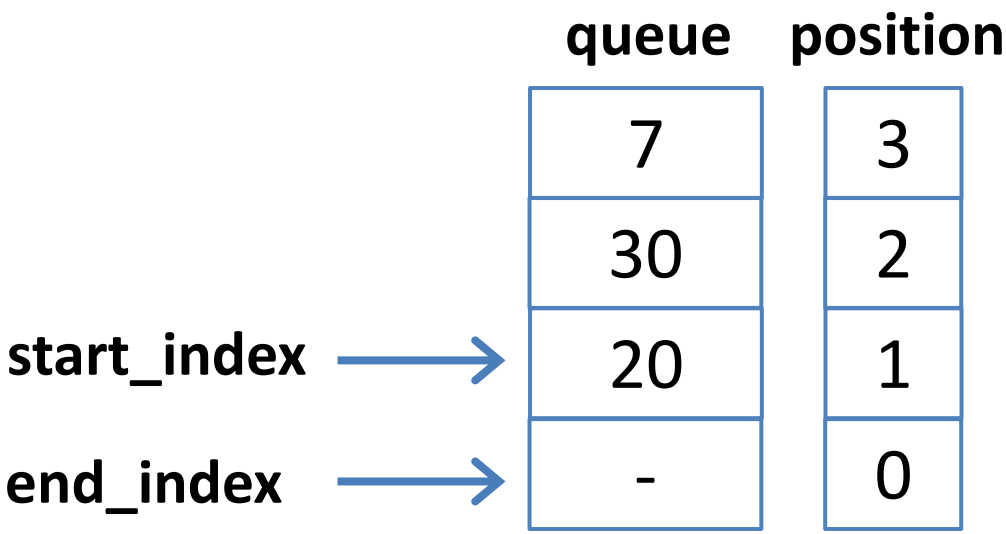
Examples of Put and Get

- put(15)
- put(20)
- get()
- **put(30)**
- put(7)
- put(25)
- get()
- put(12)
- get()
- get()



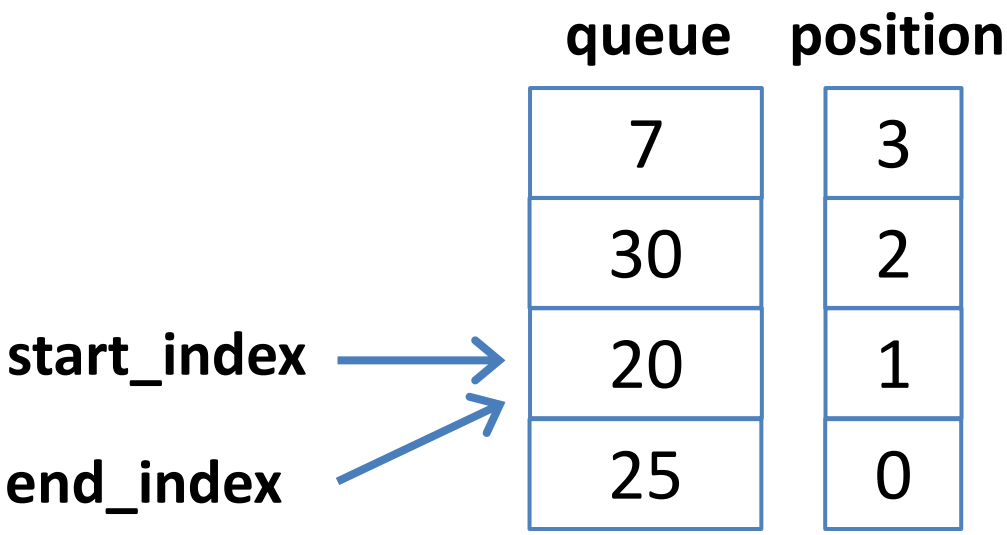
Examples of Put and Get

- put(15)
- put(20)
- get()
- put(30)
- **put(7)**
- put(25)
- get()
- put(12)
- get()
- get()



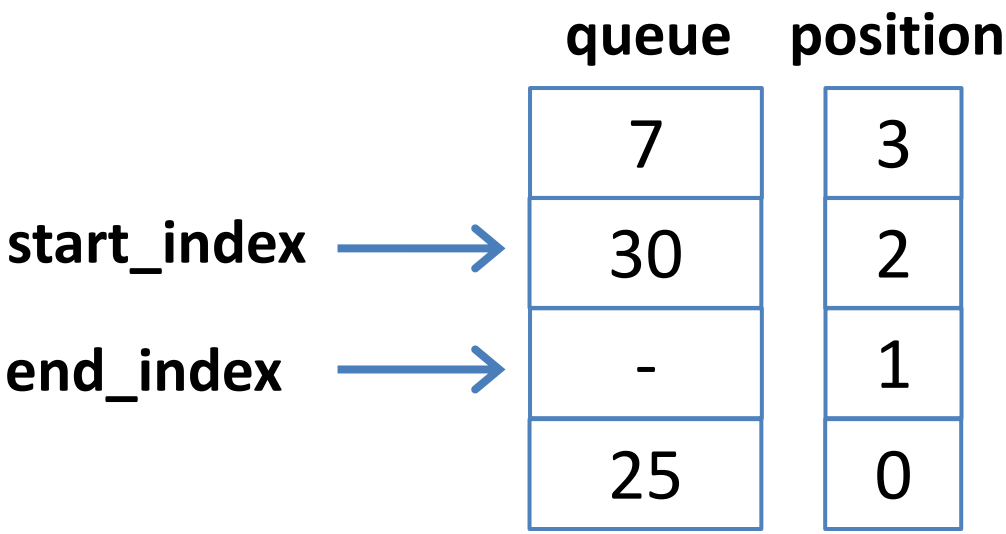
Examples of Put and Get

- put(15)
- put(20)
- get()
- put(30)
- put(7)
- **put(25)**
- get()
- put(12)
- get()
- get()



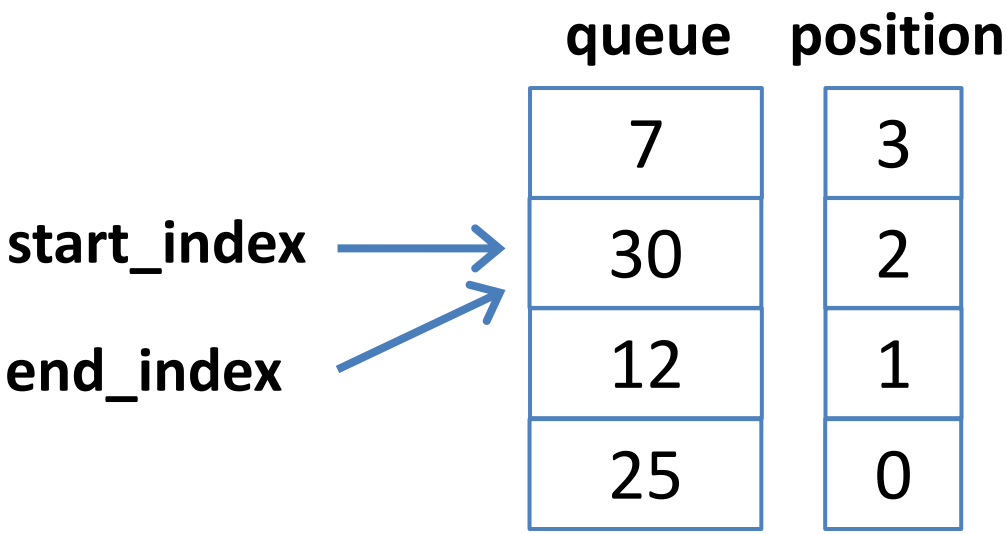
Examples of Put and Get

- put(15)
- put(20)
- get()
- put(30)
- put(7)
- put(25)
- **get(): returns 20**
- put(12)
- get()
- get()



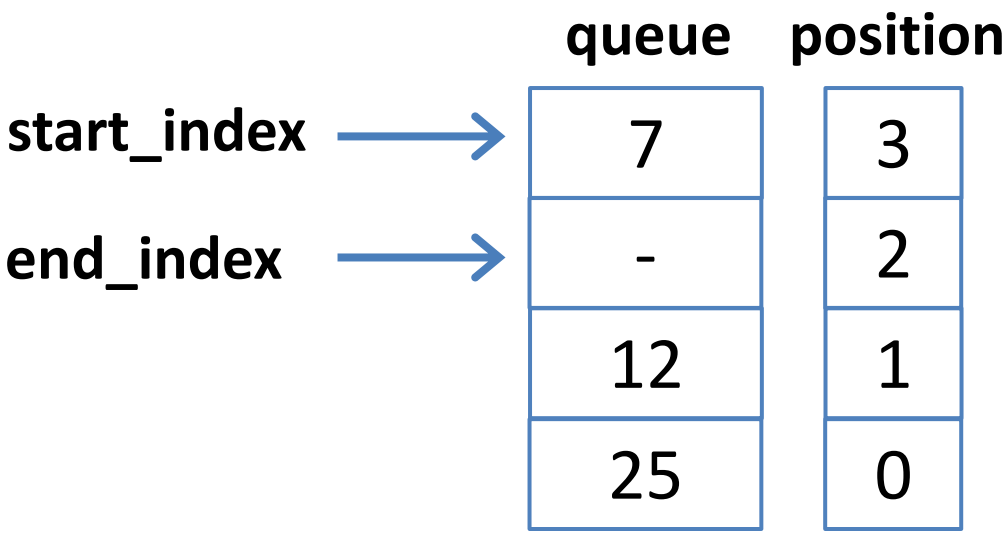
Examples of Put and Get

- put(15)
- put(20)
- get()
- put(30)
- put(7)
- put(25)
- get()
- **put(12)**
- get()
- get()



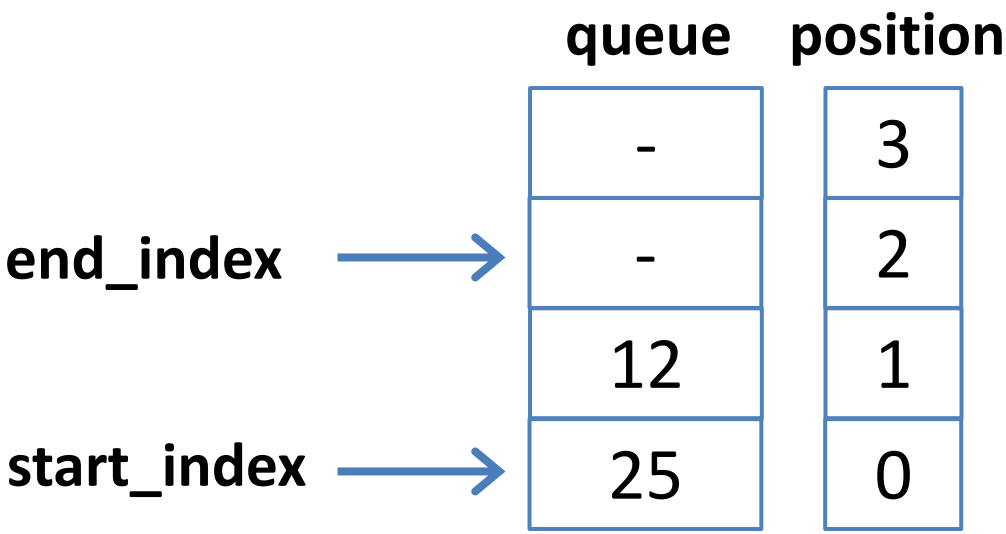
Examples of Put and Get

- put(15)
- put(20)
- get()
- put(30)
- put(7)
- put(25)
- get()
- put(12)
- **get(): returns 30**
- get()



Examples of Put and Get

- put(15)
- put(20)
- get()
- put(30)
- put(7)
- put(25)
- get()
- put(12)
- get()
- **get(): returns 7**



Book Implementation (Array-Based)

```
static Item *q;
static int N, head, tail;

void QUEUEinit(int maxN)
  { q = malloc((maxN+1)*sizeof(Item));
    N = maxN+1; head = N; tail = 0; }
int QUEUEempty()
  { return head % N == tail; }
void QUEUEput(Item item)
  { q[tail++] = item; tail = tail % N; }
Item QUEUEget()
  { head = head % N; return q[head++]; }
```

Limitations of This Implementation

- ???

Limitations of This Implementation

- Same as for stacks.
- Only one queue object can be used in the entire code.
- Queues can only store objects of a single data type:
  - Specified in Item.h, where type Item is defined using a typedef.
- It is easy to adapt the stacks_arrays.c code to obtain an array-based implementation of queues that does not have these two limitations.
- Possible homework...

More on Queues … Later

- Done with queues for now.
- We will see queues and queue-related topics quite a bit more in this course.