FIFO Queues

CSE 2320 – Algorithms and Data Structures

Alexandra Stefan
University of Texas at Arlington

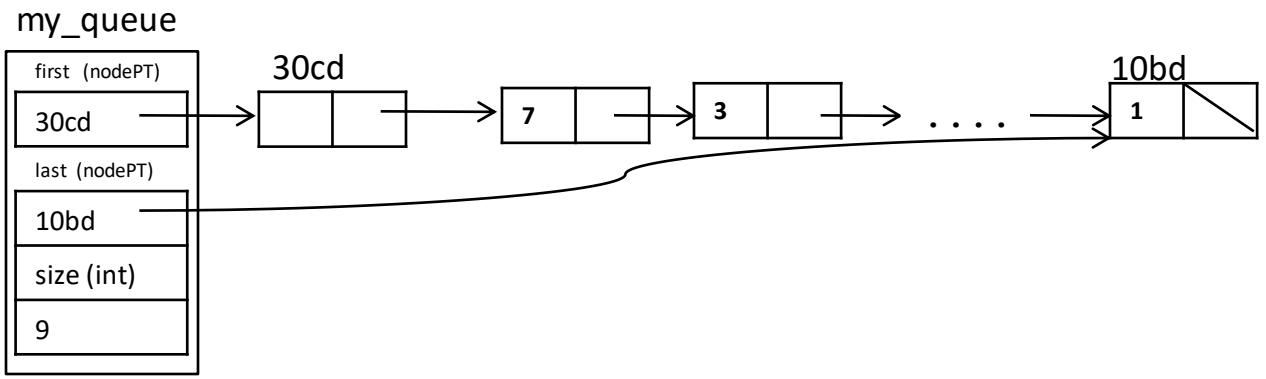
FIFO Queues

- *First-in first-out (FIFO)* queues.
- Examples of usage of FIFO queues:
  - Program execution:
    - Requests for access to memory, disk, network...
  - Resource allocation:
    - Forwarding network traffic in network switches and routers.
  - Search algorithms.
    - E.g. part of BFS in Graphs, level-order traversal for trees. (See later in the course)
- Main operations:
  - **put** - inserts an item at the end of the queue.
  - **get** - removes the item from the head of the queue.
- 2 implementations for FIFO queues: lists & arrays

List Implementation for FIFO Queues

- A FIFO queue is essentially a list.
- **put(queue, item)** inserts that item at the **end** of the list. - O(1)
  - Assumption: the list data type contains a pointer to the last element.
- **get(queue)** removes (and returns) the item at the **beginning** of the list. - O(1)

```
typedef struct node * nodePT;
struct queue_list {
   nodePT firstD; // dummy
   nodePT last;
   int size;
};
```



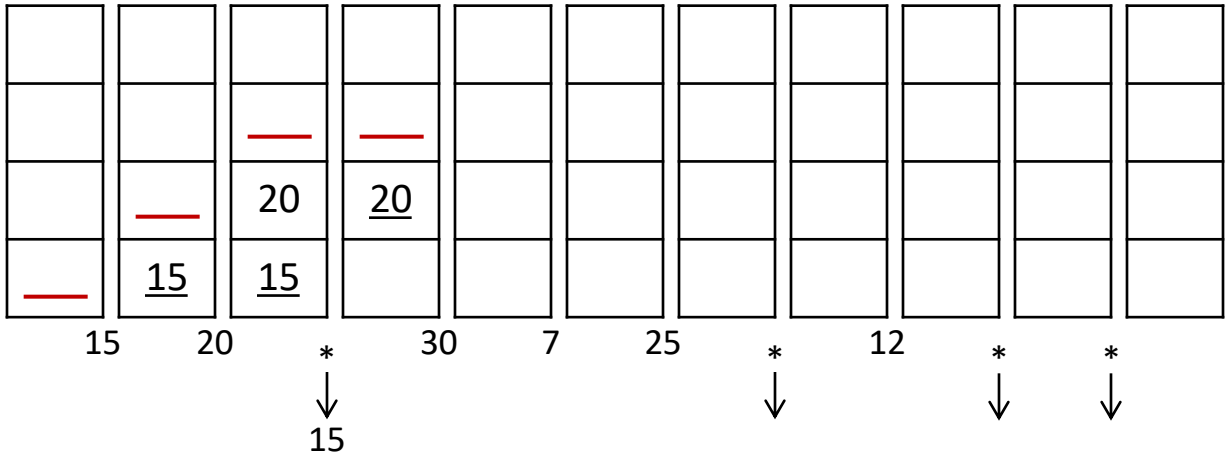
Array Implementation for FIFO Queues

```
typedef int Item;
struct queue_array {
   int capacity;
   int size;
   int first_index;   // index OF first item
   int last_index;    // index AFTER last item
   Item * items;
};
```

Conventions:

- ___ place where the new item will be added (last_index).
- underline: first item in the queue (first_index).
- x – put(x)
- * – get()

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



Array-Based Queue: Example

```
typedef int Item;
struct queue_array {
   int capacity;
   int size;
   int first_index;   // index OF first item
   int last_index;    // index AFTER last item
   Item * items;
};
```

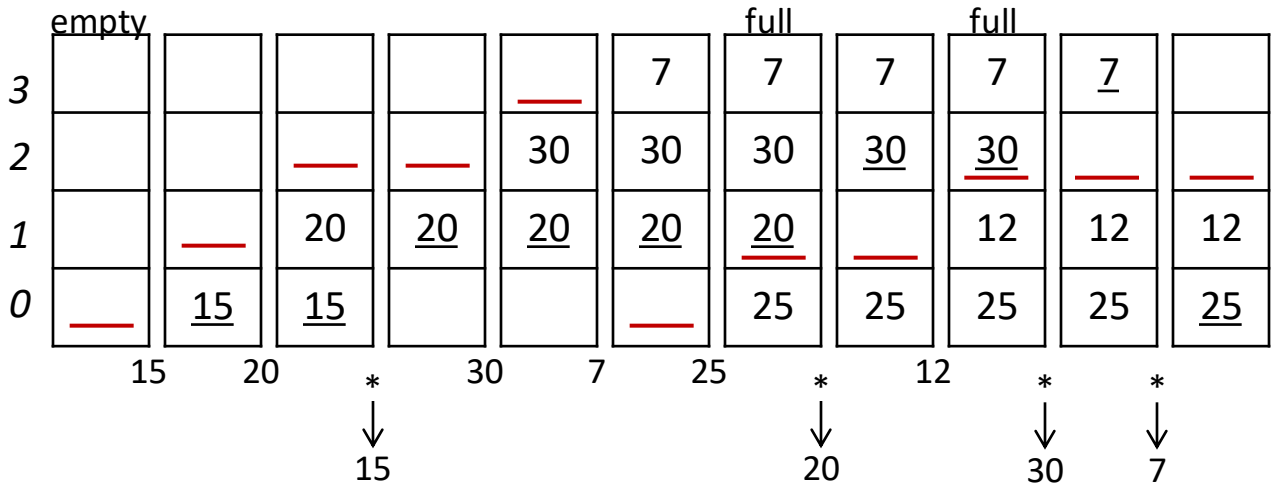
Text notations:

:_,_,_,_
15:15^,_,_,_
20:15^,20,_,_
*:_,20^,_,_ (15)
30:_,20^,30,_
7:_,20^,30,7
25:25,20^,30,7
*:25,_,30^,7 (20)
12:25,12,30^,7
*:25,12,_,7^ (30)
*:25^,12,_,_ (7)

Conventions:

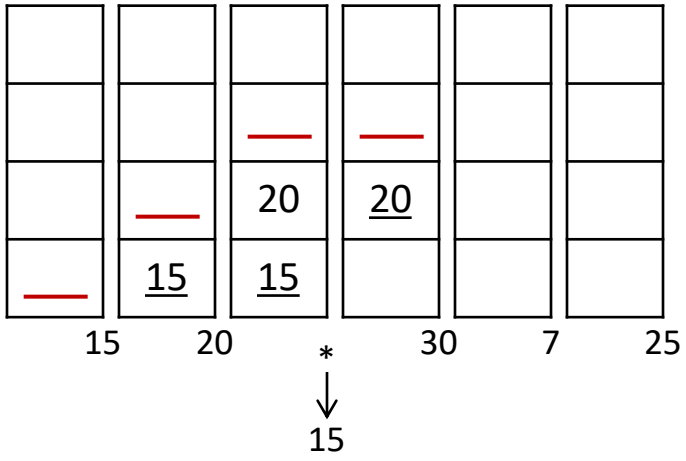
- ___ place where the new item will be added (last_index).
- underline: first item in the queue (first_index).
- x – put(x)
- * – get()

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



Array Implementation for FIFO Queues

```
typedef int Item;
struct queue_array {
   int capacity;
   int size;
   // index of first item
   int first_index;
    // index AFTER last item
   int last_index;
   Item * items;
};
typedef struct queue_array * Queue;
```




```
bool put(Queue * Q, Item val){
  if (Q->size == Q->capacity) {  // full
     return false;
  }
  if (Q->size==0) {
    Q->first_index = 0;
    Q->last_index  = 1;
    Q->items[Q->first_index] = val;
  }
  else {
     Q->items[Q->last_index] = val;
     Q->last_index = (Q->last_index+1)%Q->capacity;
  }
  Q->size++;
  return true;
}

bool get(Queue * Q, Item* ret){
  if (Q->size == 0){
    return false;
  }
  *ret = Q->items[Q->first_index];
  Q->first_index = (Q->first_index+1)%Q->capacity;
  Q->size--;
  return true;
}
```

Issues with reallocation

Assume that if the queue is full and a put operation is called you will NOT refuse the insertion, instead you will reallocate the array to make a bigger queue. (Assume the initial max_size is 10)

Q1. How do reallocate?

a) +10 (an extra 10 spaces)

b) *2 (double the space)

Assume you allocate a queue of max size 10 at first. The user keeps inserting items until they put M items in the queue. (M can be a 1000000)

For Q1 a) and b) above answer:

- how many reallocations are needed
- Time complexity of all the reallocations and data copying to put all M items in

Q2. How do you reallocate memory and how do you "copy"? Consider these options:

- (Realloc) or (malloc with memcopy)
- Malloc with copy one by one (e.g. newArr[i] = oldArr[i])
- Reinsert one by one

002

- Old = [25,20^, 30, 7] , full double the space
- 0 1 2 3
- New = [25 ,20^,30, 7, , , ,]
- Any sequence of put and get will work as expected
- Put(100), *,*,*,*,*
- Output: 20,30,7,25,100
- O1 : [25 ,20^,30, 7, *100* , , ,] - bad
- [20,30,7,25,100, _, _, _]

001

- 25,20^, 30, 7 // old
- *0, 1, 2, 3*
- *Insert 100 in old, reallocate*
- *After that, if I do any ops, they should be correct: get will remove the data in this order: 20,30,7,25,100*
- 25, 20^, 30, 7, ?100?, _, _, _ // BAD
- *0, 1, 2, 3, 4, 5,6, 7*
- *20^,30,7,25,100,_,_,_*
- *Correct: use [work of get from old array and [work of put operation*