Data Structures

Fall 2023

---

13. Queues

Queues

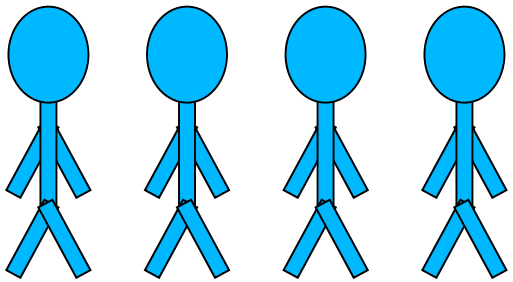
- Queue is First-In-First-Out (FIFO) data structure
  - First element added to the queue will be first one to be removed

- Queue implements a special kind of list
  - Items are inserted at one end (the rear)
  - Items are deleted at the other end (the front)

Queue – Analogy (1)

- A queue is like a line of people waiting for a bank teller
- The queue has a front and a rear

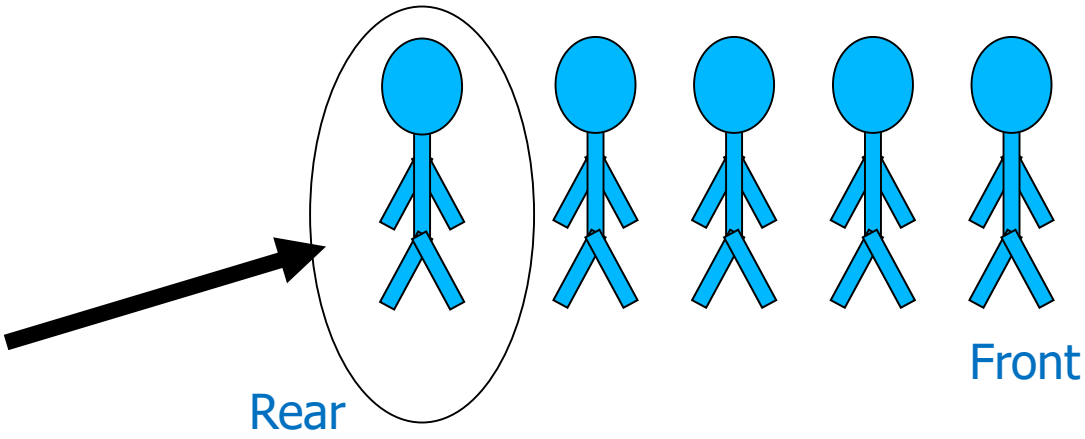
Rear



Front

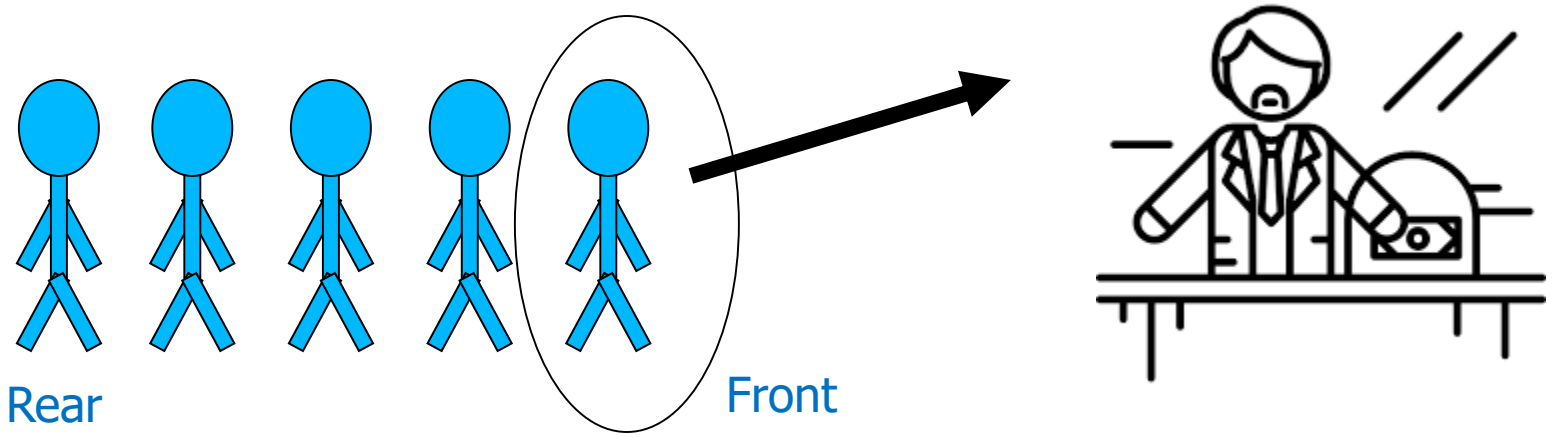
Queue – Analogy (2)

- New people must enter the queue at the rear



Queue – Analogy (3)

- An item is always taken from the front of the queue



Queues – Examples

- Billing counter
  - Booking movie tickets
  - Queue for paying bills
- A print queue
- Vehicles on toll-tax bridge
- Luggage checking machine
- And others?

Queues – Applications

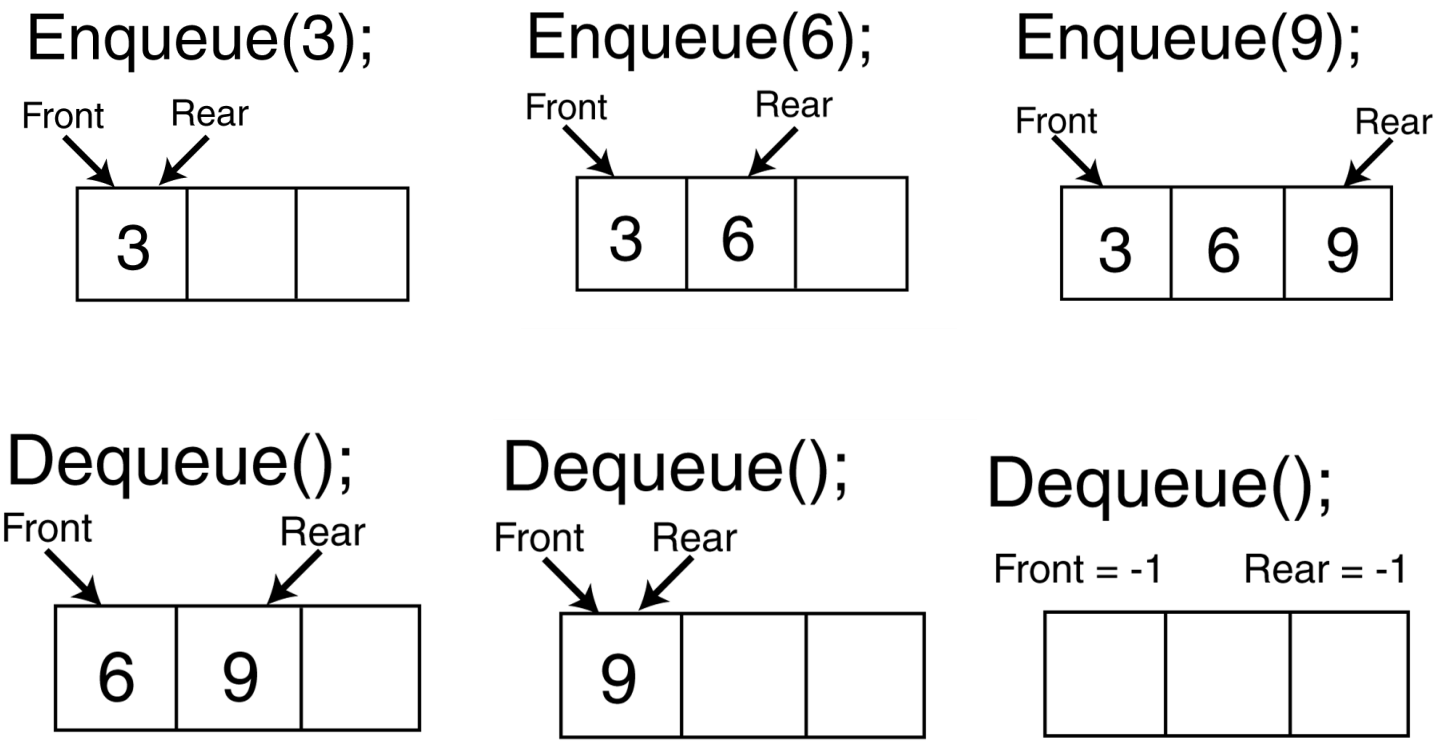
- Operating systems
  - Process scheduling in multiprogramming environment
  - Controlling provisioning of resources to multiple users (or processing)

- Middleware/Communication software
  - Hold messages/packets in order of their arrival
    - Messages are usually transmitted faster than the time to process them
  - The most common application is in client-server models
    - Multiple clients may be requesting services from one or more servers
    - Some clients may have to wait while the servers are busy
    - Those clients are placed in a queue and serviced in the order of arrival

Basic Operations (Queue ADT)

- `MAKENULL(Q)`
  - Makes Queue Q be an empty list
- `FRONT(Q)`
  - Returns the first element on Queue Q
- **`ENQUEUE(x,Q)`**
  - Inserts element x at the end of Queue Q
- **`DEQUEUE(Q)`**
  - Deletes the first element of Q
- `EMPTY(Q)`
  - Returns true if and only if Q is an empty queue

Enqueue And Dequeue Operations



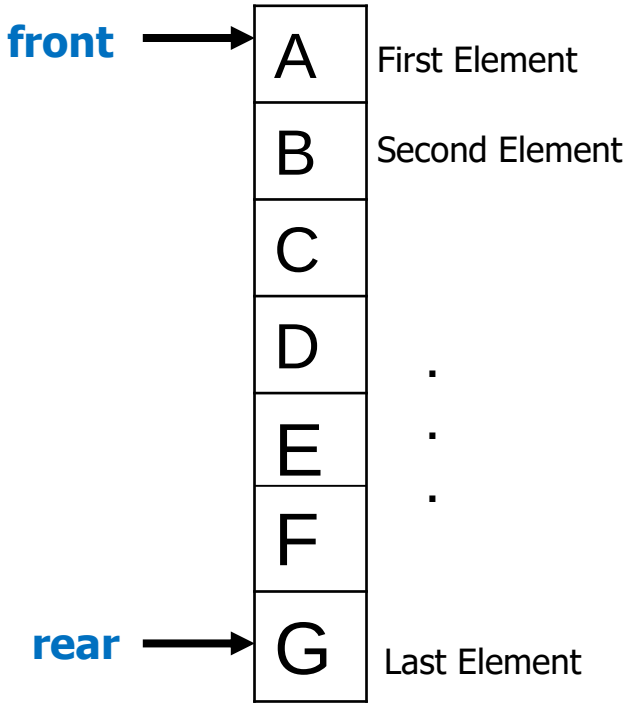
Implementation

- Static
  - Queue is implemented by an array
  - Size of queue remains fixed

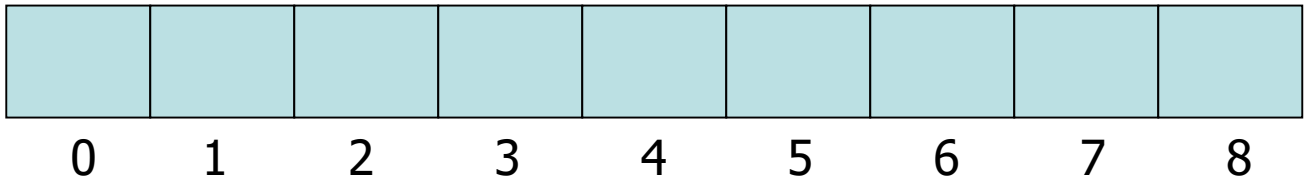
- Dynamic
  - A queue can be implemented as a linked list
  - Expand or shrink with each enqueue or dequeue operation

Array Implementation

- Use two counters that signify rear and front
- When queue is empty
  - Both front and rear are set to -1
- When there is only one value in the Queue,
  - Both rear and front have same index
- While enqueueing increment rear by 1
- While dequeueing, increment front by 1

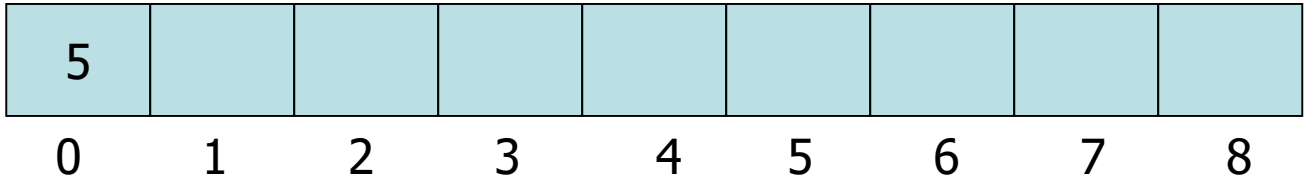


Array Implementation Example (1)



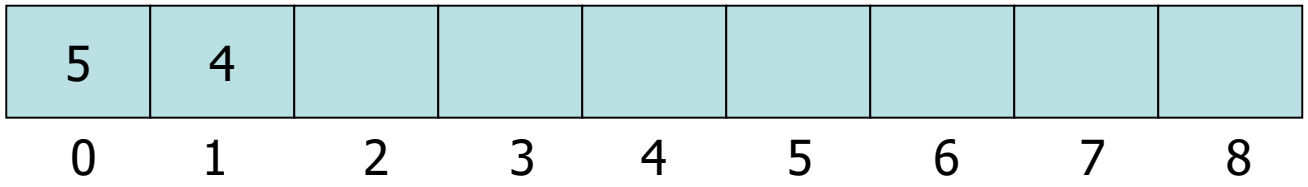
front= -1
rear = -1

Enqueue 5



front= 0
rear = 0

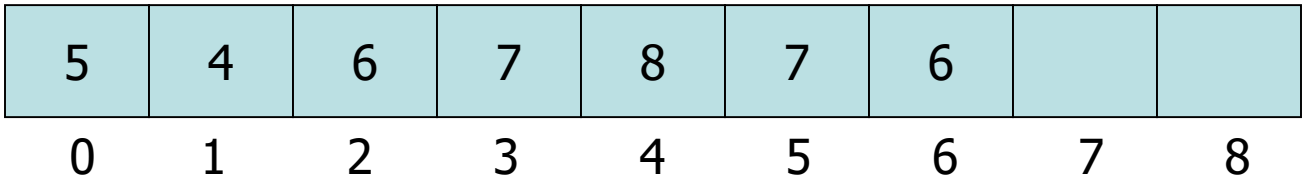
Enqueue 4



front= 0
rear = 1

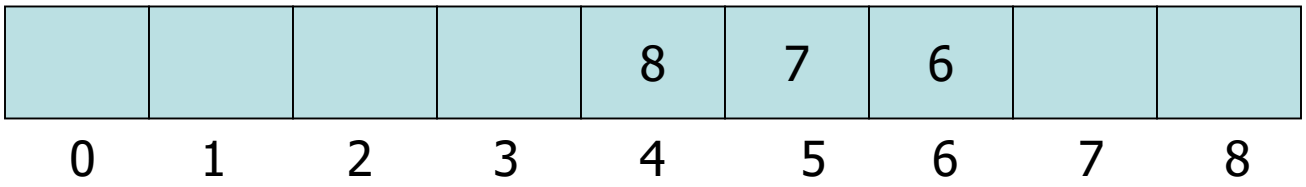
Array Implementation Example (2)

Enqueue 6, 7, 8, 7, 6



front=0
rear=6

Dequeue 5, 4, 6, 7



front=4
rear=6

Array Implementation – Code (1)

```
class queue
{
    private:
        int a[size];
        int front, rear;
    public:
        queue()
        {
            front = -1;
            rear = -1;
        }
}
```

Array Implementation – Code (2)

```
void enqueue(int x)
{
    rear++;
    if (rear >= size)
    {
        cout << "Queue full\n";
        rear = size-1;
    }
    else
    {
        a[rear] = x;
        if (front == -1)
            front=0;
    }
}
```

Array Implementation – Code (3)

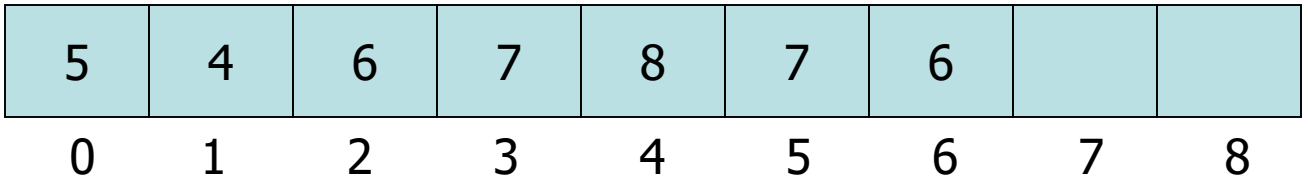
```
void dequeue()
{
    if (front > rear || front == -1)
    {
        cout << "Queue empty\n";
        front = -1;
        rear = -1;
        return -1;
    }
    else {
        int data;
        data = a[front];
        front++;
        return data;
    }
}
```

Array Implementation – Code (4)

```
void display()
{
    if (front > rear || front == -1 && rear == -1)
    {
        cout << "Queue empty\n";
    }
    else {
        cout << "Queue is\n";
        for (int i=front;i<=rear;i++)
            cout << a[i] << " ";
        cout << endl;
    }
}
```

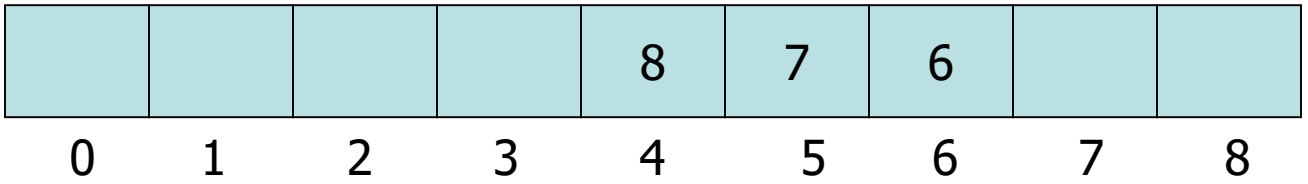
Array Implementation Example (3)

Enqueue 6, 7, 8, 7, 6



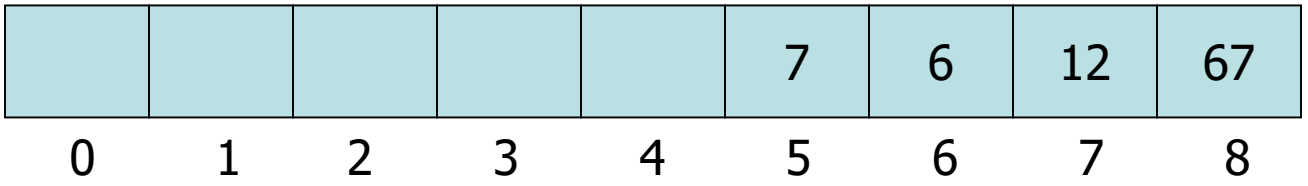
front=0
rear=6

Dequeue 5, 4, 6, 7



front=4
rear=6

Enqueue 12, 67



front=5
rear=8

Problem: How can we insert more elements?
Rear index can not move beyond the last element….

Using Circular Queue

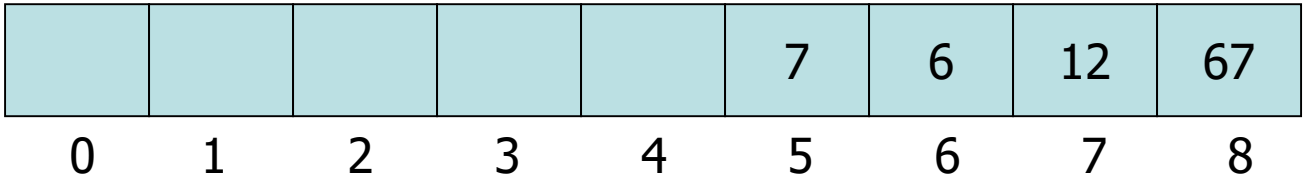
- Allow rear to wrap around the array

```
if(rear == queueSize-1)
        rear = 0;
else
        rear++;
```

- Alternatively, use modular arithmetic

```
rear = (rear + 1) % queueSize;
```

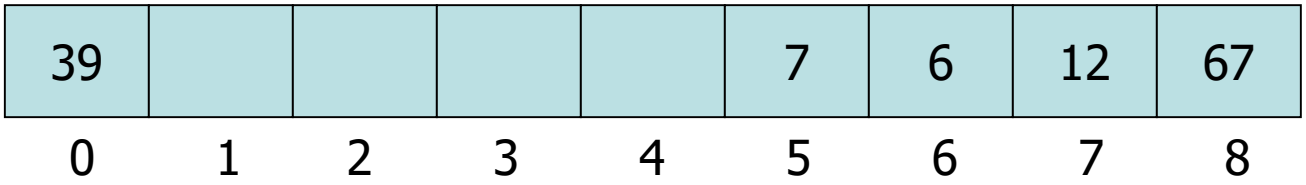
Array Implementation Example (4)



front=5
rear=8

Enqueue 39

- `Rear = (Rear+1) mod queueSize = (8+1) mod 9 = 0`



front=5
rear=0

Problem: How to avoid overwriting an existing element?

How to Determine Empty and Full Queues?

- A counter indicating number of values/items in the queue
  - Covered in first array-based implementation

```
class cqueue
{
    private:
        int a[size];
        int front, rear, count;
    public:
        cqueue()
        {
            front = -1;
            rear = -1;
            count = 0;
        }
}
```

Array Implementation – Code (5)

```
void enqueue(int x)
{
    if (count == size)
    {
        cout << "Queue full\n";
        return;
    }
    else
    {
        rear=(rear+1)%size;
        a[rear] = x;
        count++;
    }
}
```

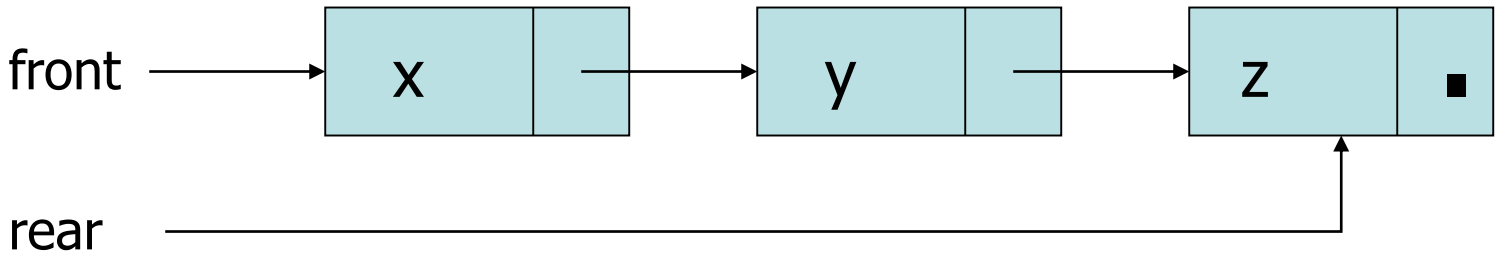
```
void dequeue()
{
    if (count == 0)
    {
        cout << "Queue empty\n";
        return -1;
    }
    else {
        int data;
        front = (front+1)%size;
        data = a[front];
        count--;
        return data;
    }
}
```

Array Implementation – Code (6)

```
void display()
{
    if (count == 0)
        cout << "Queue empty\n";
    else if (front <= rear)
    {
        for (int i=front; i<=rear; i++)
            cout << array[i] << " ";
        cout << endl;
    }
    else        // front > rear
    {
        for (int i=front; i<size; i++)
            cout << array[i] << " ";
        for (int i=0; i<=rear; i++)
            cout << array[i] << " ";
        cout << endl;
    }
}
```

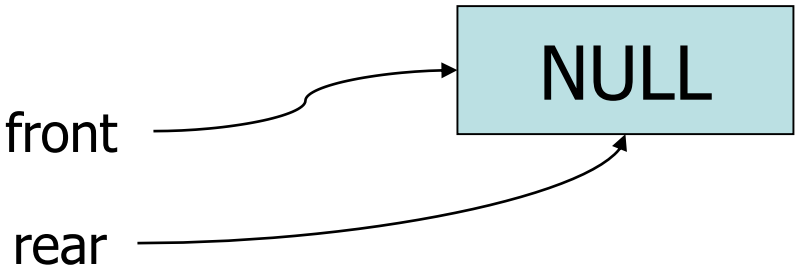
Linked List Implementation of Queues

- Queue Class maintains two pointers
  - `front:` A pointer to the first element of the queue
  - `rear:` A pointer to the last element of the queue

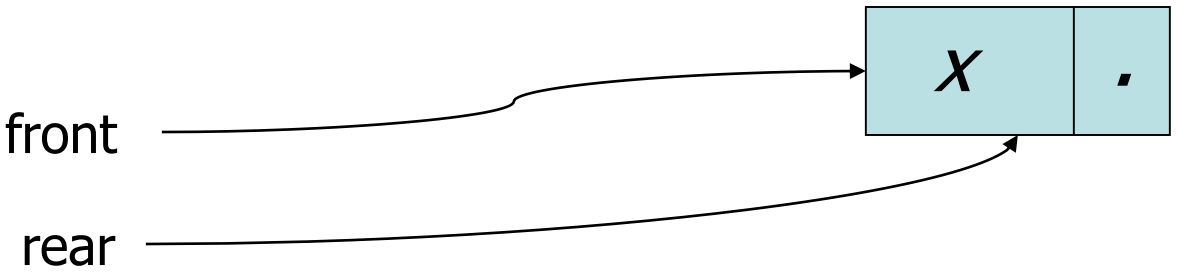


- Enqueue: Insert at End
- Dequeue: Delete at Start

Queue Operations (1)

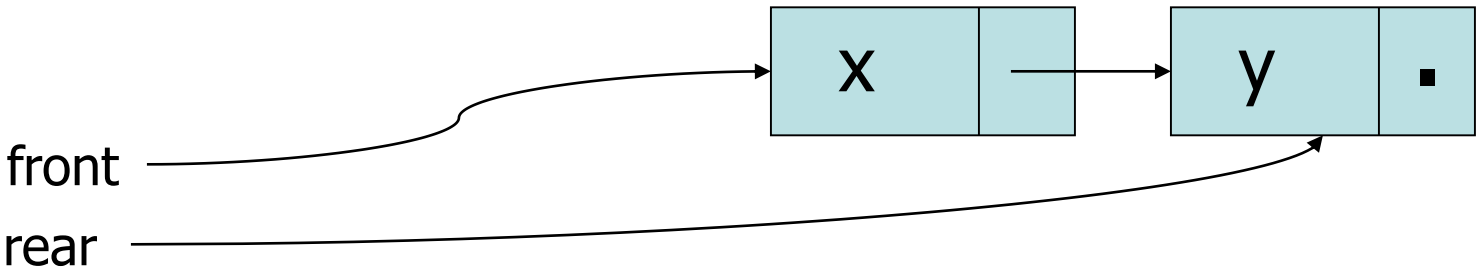


- `ENQUEUE (x)`

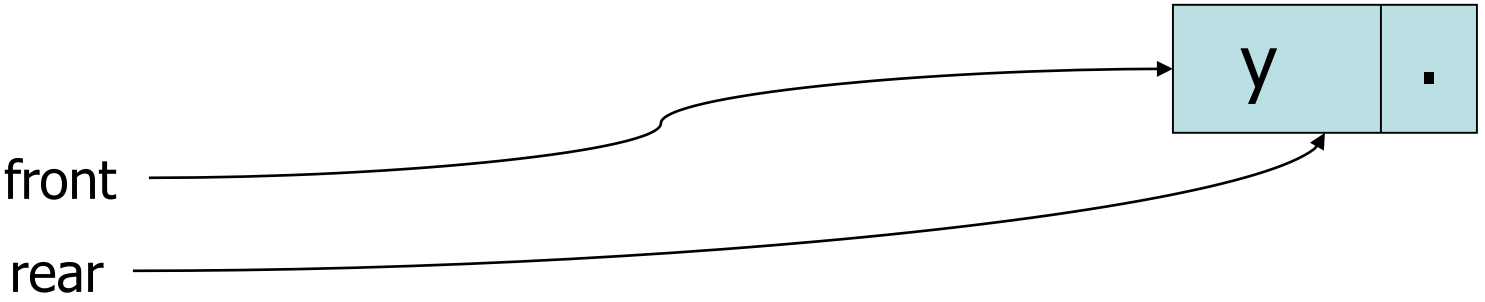


Queue Operations

- `ENQUEUE (y)`



- `DEQUEUE`



Any Question So Far?