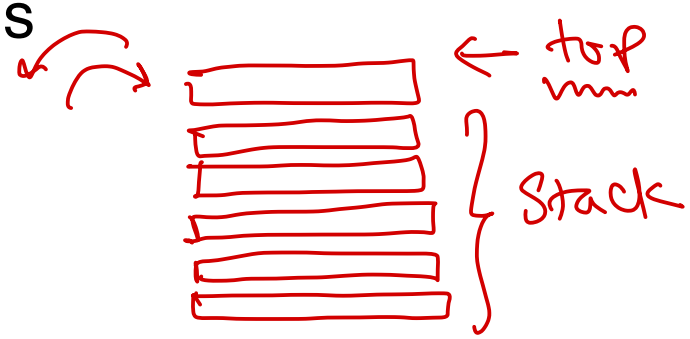
Data Structure & Algorithms

Sunbeam Infotech

Stack and Queue

- Stack & Queue are utility data structures.
- Can be implemented using array or linked lists.
- Usually time complexity of stack & queue operations is O(1).
- Stack is Last-In-First-Out structure.
- Stack operations
  - push()
  - pop()
  - peek()
  - isEmpty()
  - isFull()* → for array



add/del done from same end.

- Simple queue is First-In-First-Out structure.
- Queue operations
  - push()
  - pop()
  - peek()
  - isEmpty()
  - isFull()* → for array

- Queue types
  - Linear queue
  - Circular queue
  - Deque
  - Priority queue

Linear Queue

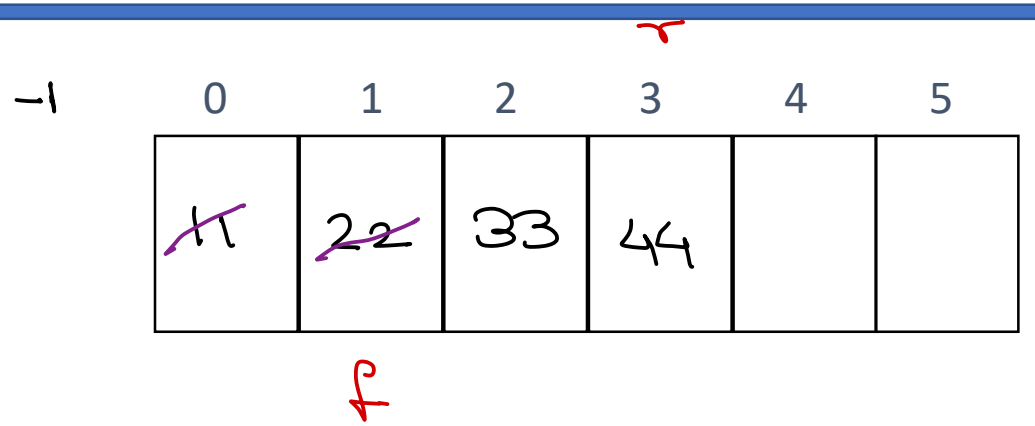
f+1 → rear

init:

```
f = -1
r = -1
```

isfull:

```
r == size-1
```



push: (enque)

```
r++
arr[r] = val.
```

isempty:

```
f == r
```

f & r increments from (0) to (n-1).

pop: (deque)

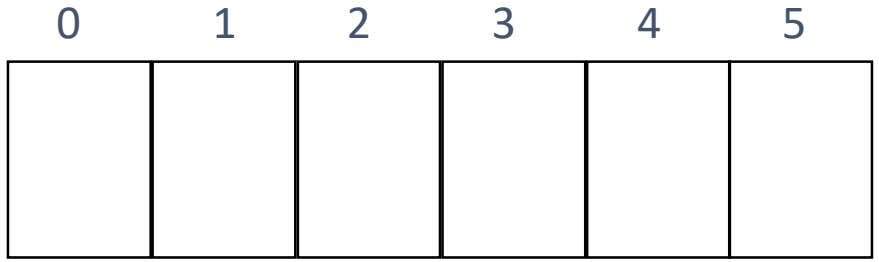
```
f++
```

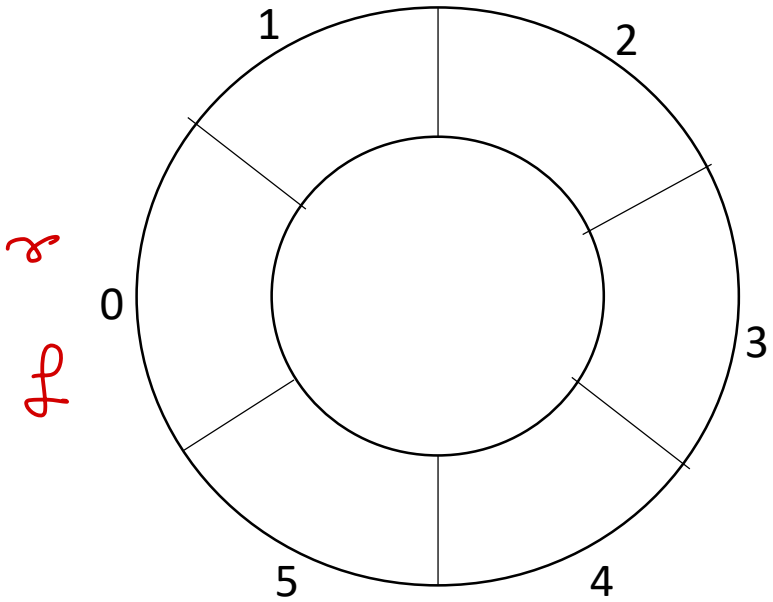
peek:

```
return arr[f+1]
```

Circular Queue

- In linear queue (using array) when *rear* reaches last index, further elements cannot be added, even If space is available due to deletion of elements from *front*. Thus space utilization is poor.
- Circular queue allows adding elements at the start of array if *rear* reaches last index and space is free at the start of the array.
- Thus *rear* and *front* can be incremented in circular fashion i.e. 0, 1, 2, 3, …, n-1. So they are said to be circular queue.
- However queue full and empty conditions become tricky.





Circular Queue

init :

```
r = -1
f = -1
c = 0
```

pop :

```
f = (f+1) % size
c--;
```

push :

```
r = (r+1) % size;
arr[r] = ele;
c++;
```

is_empty :

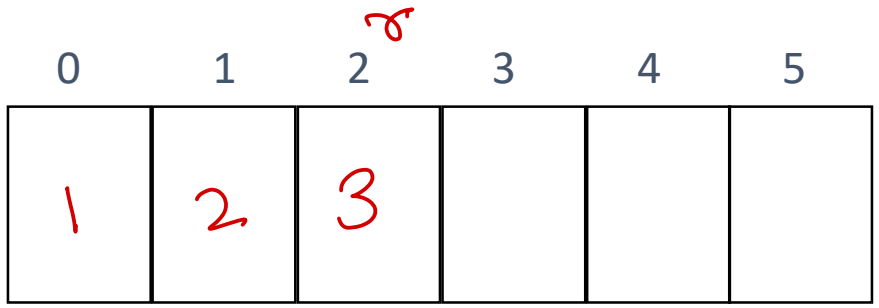
```
c == 0
```

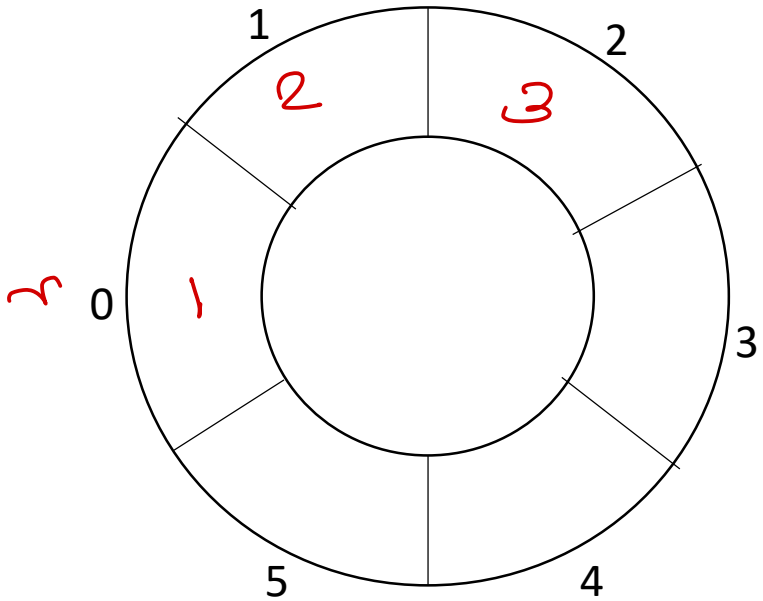
is_full

```
c == size
```

peek :

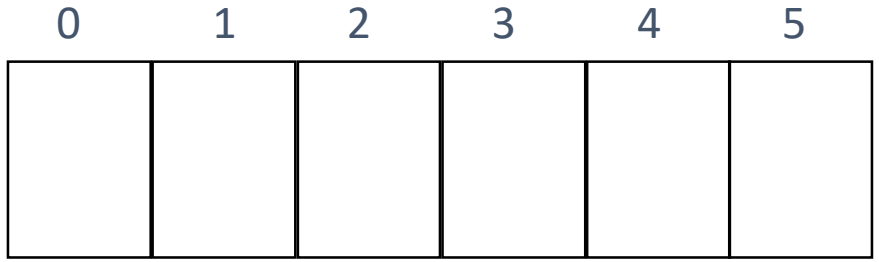
```
i = (f+1) % size
return arr[i]
```

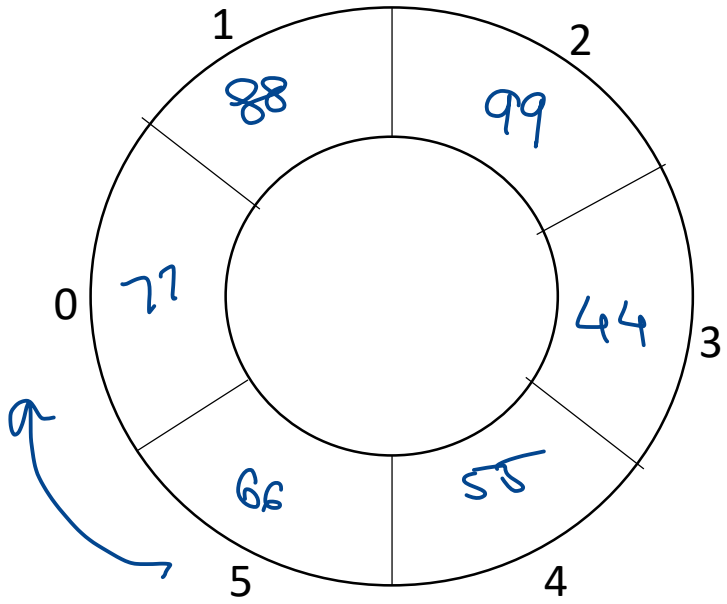




Circular Queue

(-1 + 1) % 6 = 0
(0 + 1) % 6 = 1
(1 + 1) % 6 = 2
(2 + 1) % 6 = 3
(3 + 1) % 6 = 4
(4 + 1) % 6 = 5
(5 + 1) % 6 = 0

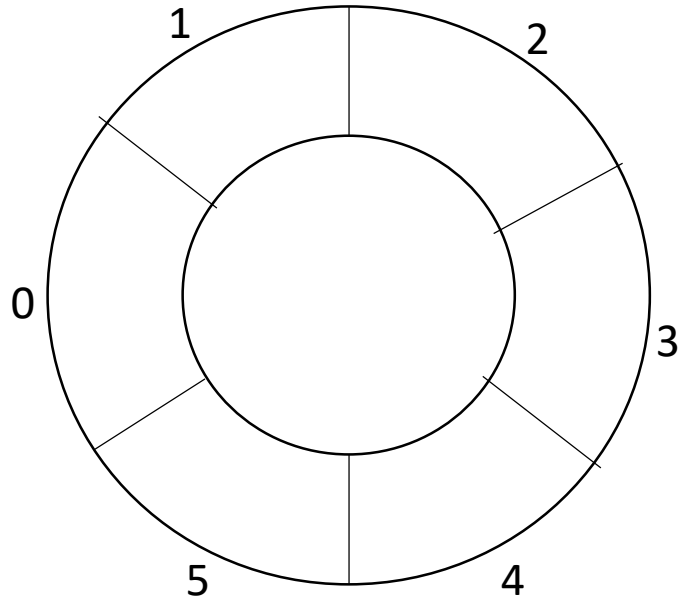




Circular Queue – Full

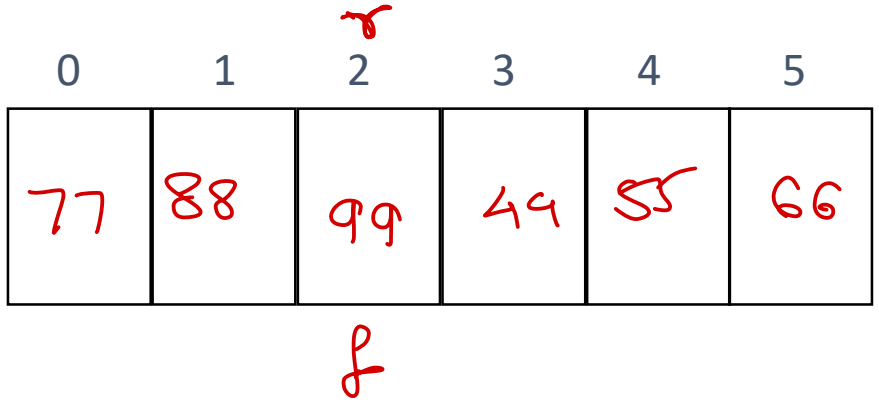
(f == -1 && r == size-1)

-1

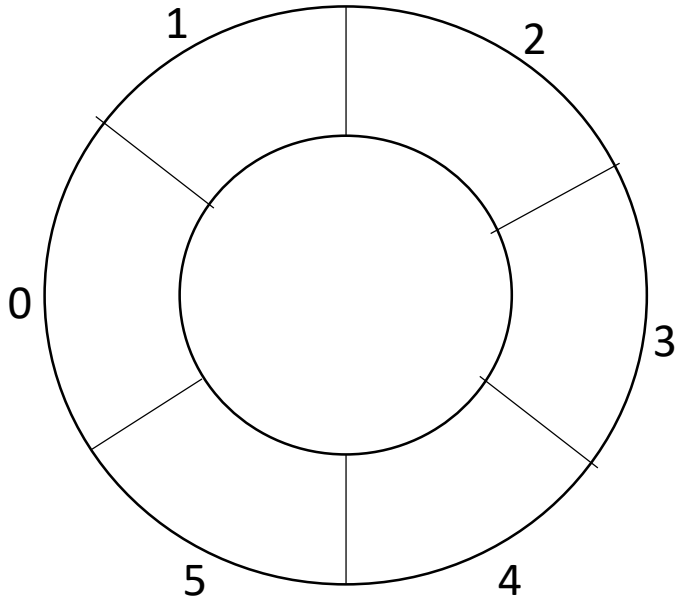


Circular Queue – Full

(f == -1 && r == size-1) -1



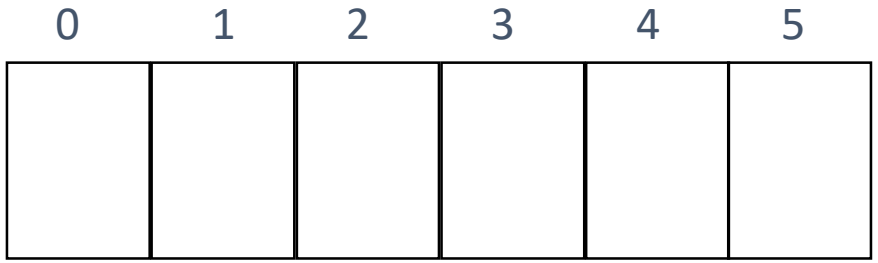
(r == f && r != -1)



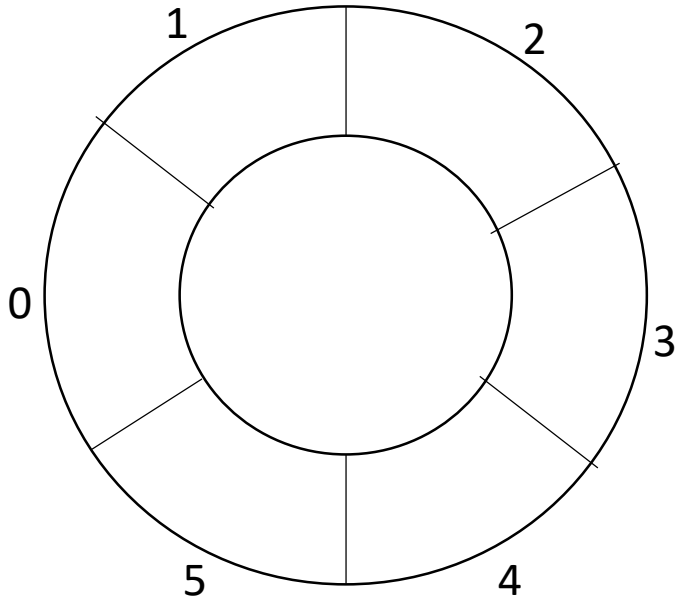
Circular Queue – empty

(r == f && r == -1)

r
-1



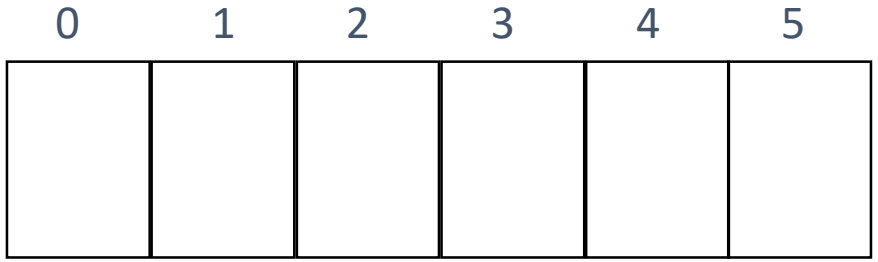
f



Circular Queue – Full & Empty

full

```
(f == -1 && r == size-1) || r
(r == f && r != -1)
```

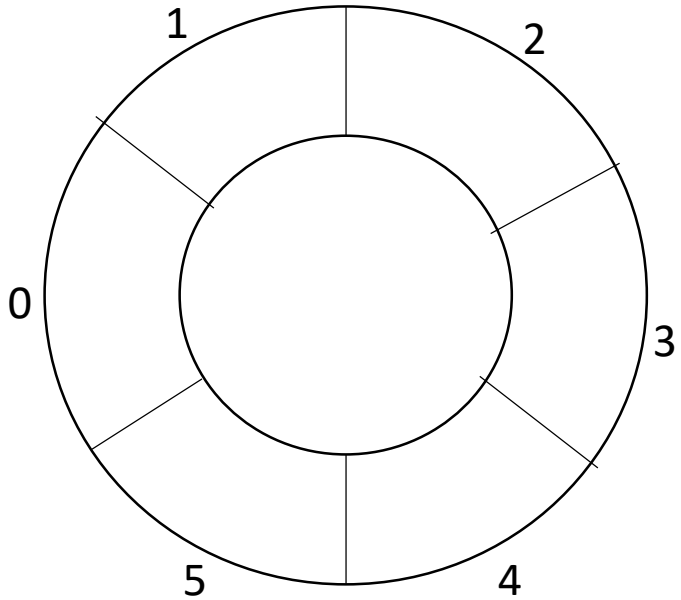


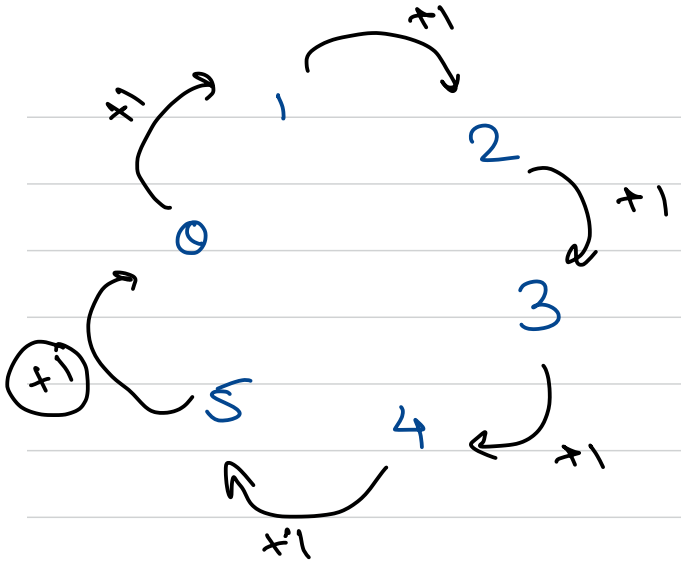
empty

```
(r == f && r == -1)
```

pop

```
f = (f+1) % size;
if (r == f) {
    r = -1;
    f = -1;
}
```





size = 6.

$(0+1) \% 6 = 1$

$(1+1) \% 6 = 2$

$(2+1) \% 6 = 3$

$(3+1) \% 6 = 4$

$(4+1) \% 6 = 5$

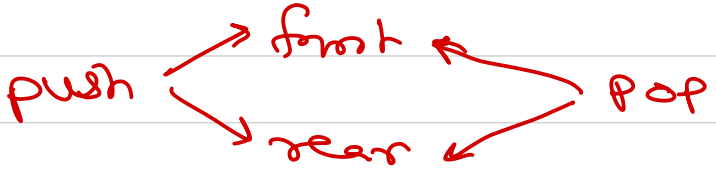
$(5+1) \% 6 = 0$

$r = (r+1) \% \text{size}$

$f = (f+1) \% \text{size}$

Deque: Double Ended Queue.

push & pop ops are possible from both ends



push_front()
push_rear()

pop_front()
pop_rear()

is_empty()

priority queue

* each ele is associated with priority.
* this queue is NOT FIFO.
* ele with highest priority comes first.
* most efficient impl of priority queue is using max heap,

  Time complexity of push & pop = $O(\log n)$

Stack

init:

```
top = -1
```

push:

```
top++;
arr[top] = ele;
```

pop:

```
top--;
```

peek:

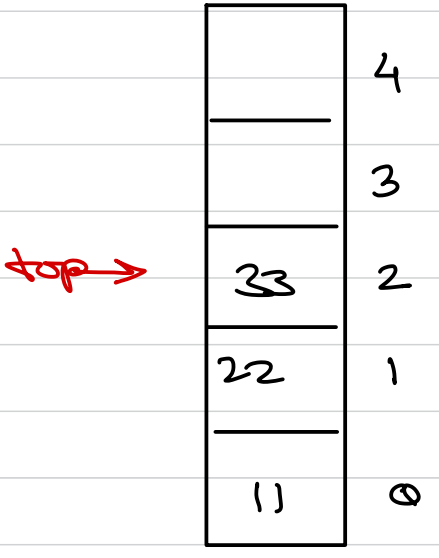
```
return arr[top];
```

is_empty:

```
top == -1
```

is_full

```
top == size - 1
```



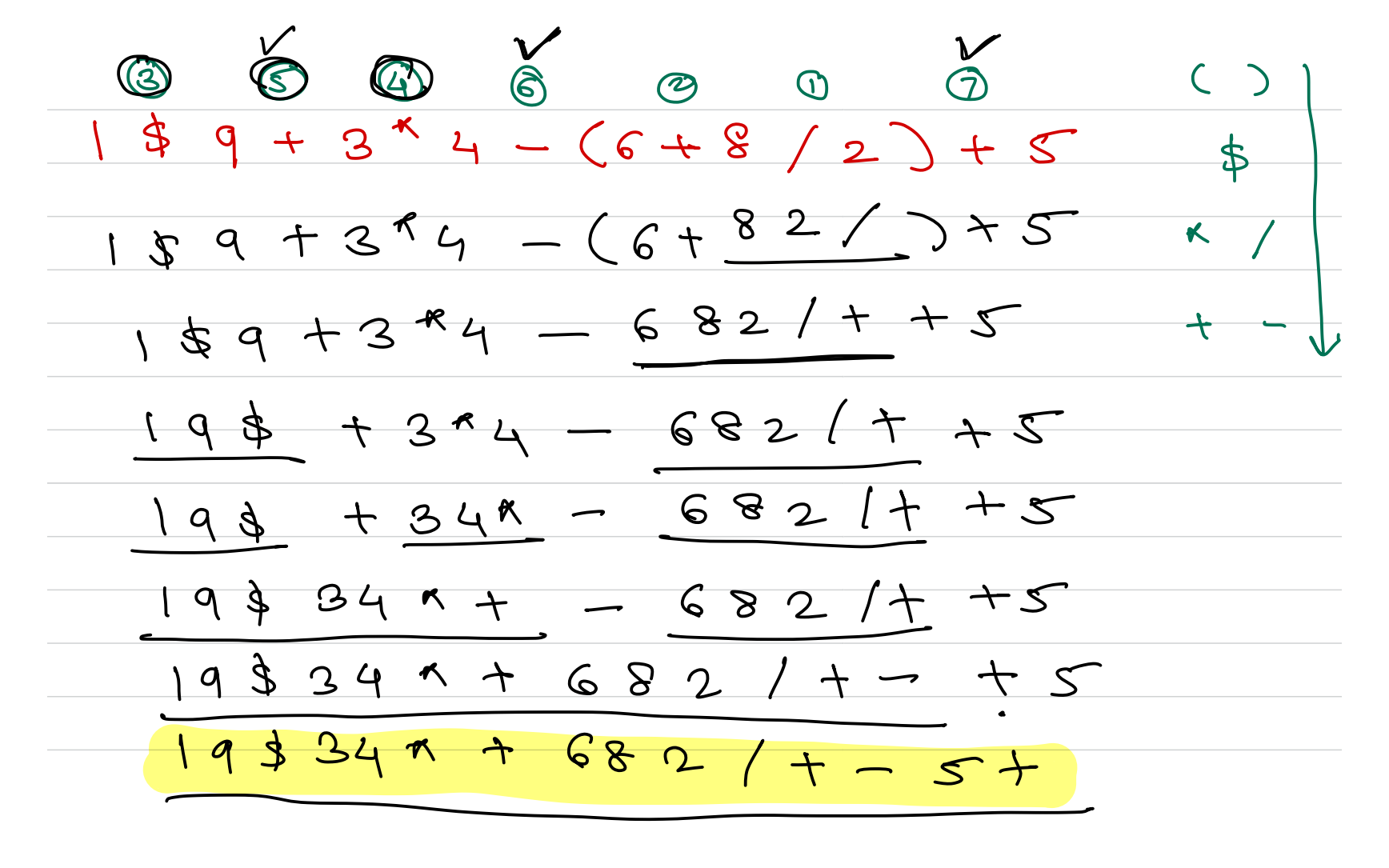
Expression Notations

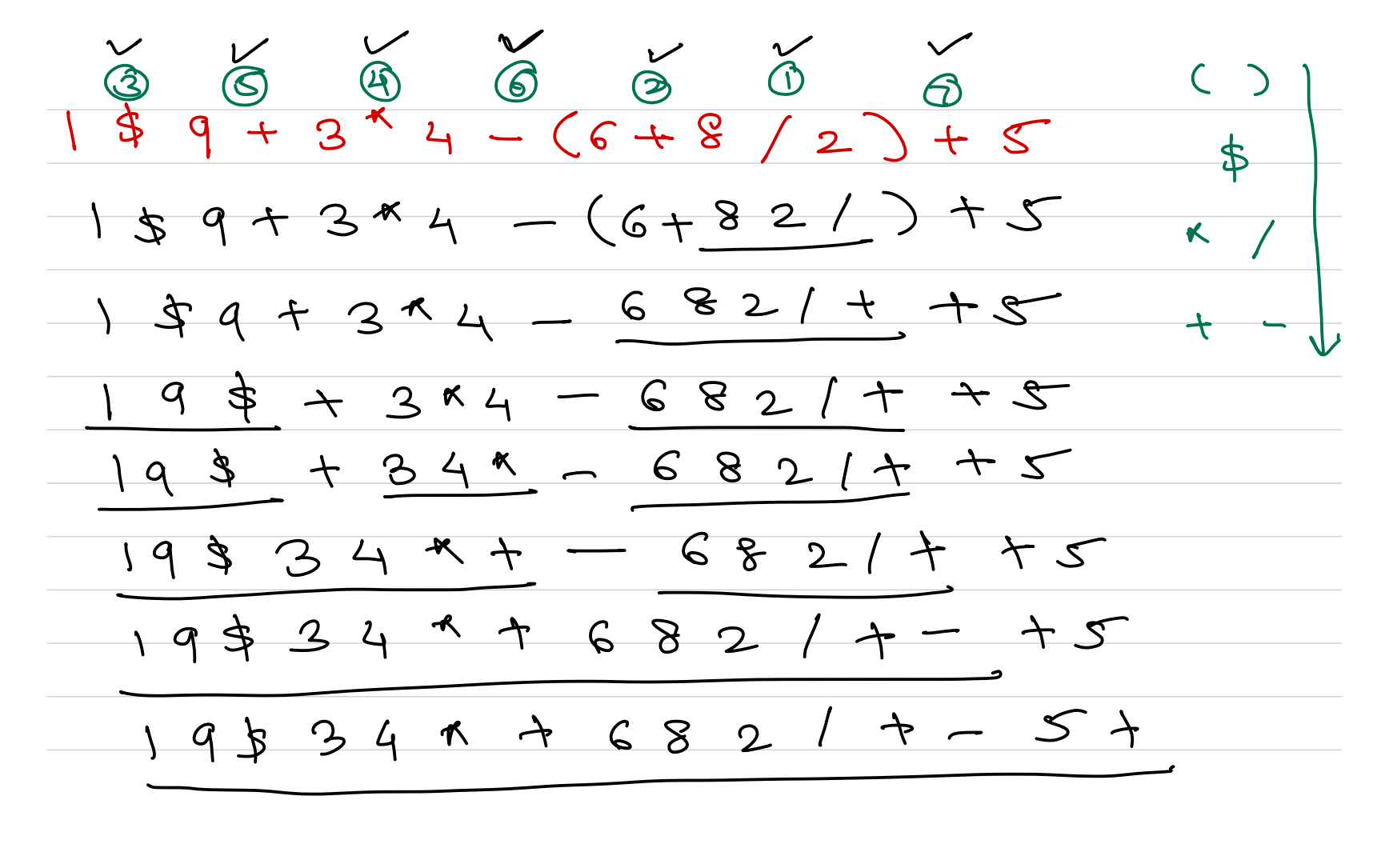
infix : a + b ← human

postfix : a b +

prefix : + a b

↑ ↓ computer algo.


3 5 4 6 2 1 7
1 $ 9 + 3 * 4 - (6 + 8 / 2) + 5
1 $ 9 + 3 * 4 - (6 + 8 2 /) + 5
1 $ 9 + 3 * 4 - 6 8 2 / + + 5
1 9 $ + 3 * 4 - 6 8 2 / + + 5
1 9 $ + 3 4 * - 6 8 2 / + + 5
1 9 $ 3 4 * + - 6 8 2 / + + 5
1 9 $ 3 4 * + 6 8 2 / + - + 5
1 9 $ 3 4 * + 6 8 2 / + - 5 +
( )
$
* /
+ -


3 5 4 6 2 1 7
1 $ 9 + 3 * 4 - (6 + 8 / 2) + 5
1 $ 9 + 3 * 4 - (6 + 8 2 /) + 5
1 $ 9 + 3 * 4 - 6 8 2 / + + 5
1 9 $ + 3 * 4 - 6 8 2 / + + 5
1 9 $ + 3 4 * - 6 8 2 / + + 5
1 9 $ 3 4 * + - 6 8 2 / + + 5
1 9 $ 3 4 * + 6 8 2 / + - + 5
1 9 $ 3 4 * + 6 8 2 / + - 5 +
( )
$
* /
+ -

③ ⑤ ④ ⑥ ② ① ⑦

1 $ 9 + 3 * 4 − (6 + 8 / 2) + 5

+ − + $ 19 * 34 + 6 / 8 2 5 ← pre

1 9 $ 34 * + 6 8 2 / + − 5 + ← post

( )

$

* /

+ −

Assignments

1. Implement stack. Pass size of stack to the constructor & dynamically allocate array.

2. Implement stack. Instead of initializing top = -1, start with top = 0. Do necessary changes in push, pop & other functions.

3. Input a string (char[]) from user. Reverse string using stack.

4. How we can simulate stack using queue?

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>