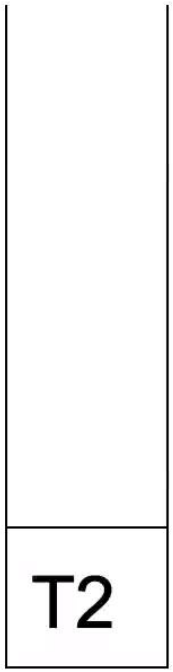
Example

+ * + a b + c d + e f.



$$T1 = (a+b) * (c+d)$$

$$T2 = (T1) + (e+f)$$

Example

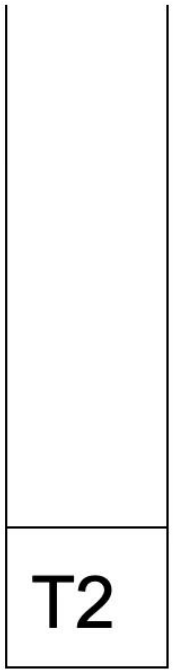
+ * + a b + c d + e f.

(+ab) x (+cd) + (+ef)

+ x +ab + cd + ef

T1 = (a+b) * (c+d)

T2 = (T1) + (e+f)



d) + (e + f)

Algorithm for Evaluating a Prefix Expression

```
Algorithm EvaluatePrefix(E)
begin
        Stack S;
        for i = n down to 1 do
        begin
        if E[i] is an operator, say o then
                operand2 = S.pop();
                operand1 = S.pop();
                value = operand1 o operand2;
                S.push(value);
        else
                S.push(E[i]);
        end-for
end-algorithm
```

- Here, n refers to the number of operators + the number of operands.
- The time taken for the above algorithm is linear in n.
  - There is only one for loop which looks at each element, either operand or operator, once.
- We will see an example next.

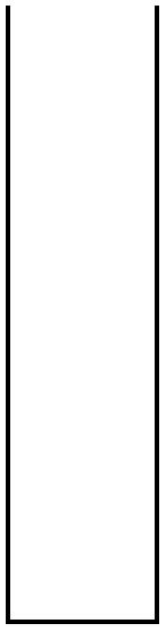
Reading Exercise

- We omitted a few details in our description.
- Some of them are:
  - How to handle unary operators?
  - How can this be extended to ternary operators?

- Another possibility is to use postfix expressions.
  - Also called as Reverse Polish Notation.
- They can be evaluated left to right with a stack.
- Try to arrive at the details.

infix-prefix

- Let us consider an expression of the form a + b + c * d + e * f.



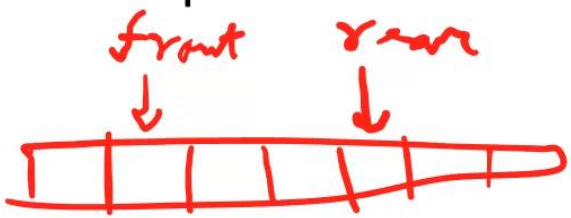
f e * d c * + b + a +

Invert as:

e f

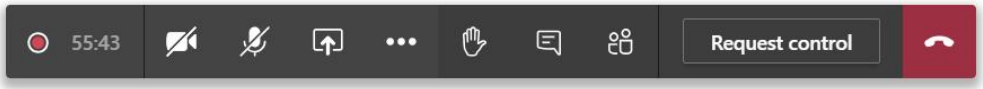
The Queue

- The fundamental operations for such a data structure are:
  - Create : create an empty queue
  - Enqueue : Insert an item into the queue
  - Dequeue : Delete an item from the queue.
  - size : return the number of elements in the queue.

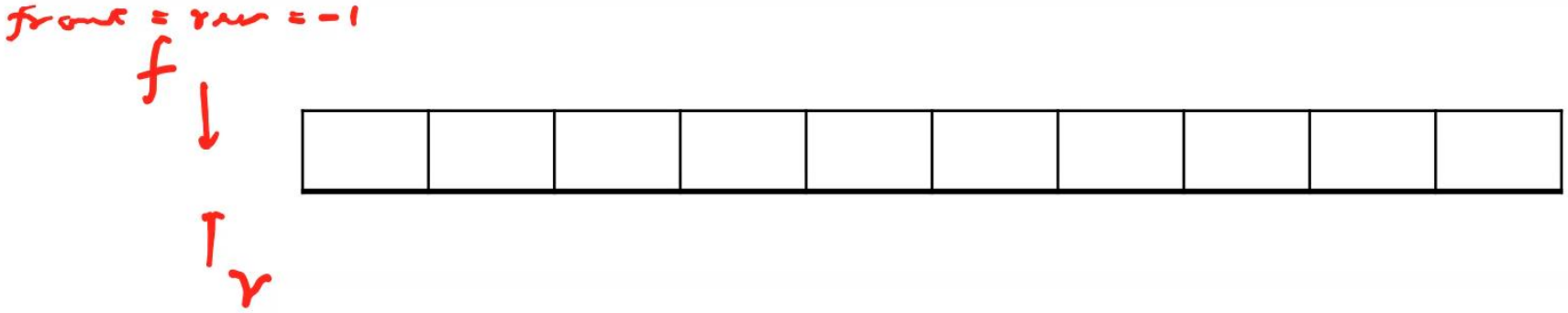


The Queue

- Can use an array also to implement a queue.
- We will show how to implement the operations.
  - We will skip create() and size().
- We will store two counters : front and rear
- Insertions happen at the rear
- Deletions happen from the front.



front = rear = -1



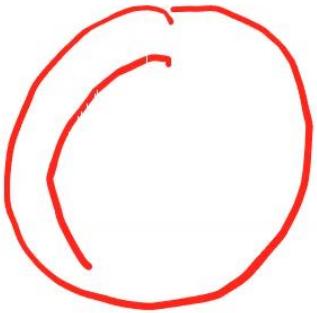
```
IsEmpty()
begin
if front==-1 && rear == -1
    return true;
else
    return false
end
```

```
Enqueue(x)
begin
if rear == MAXSIZE then
    return;
else if IsEmpty()
    front ← rear ← 0;
else
    rear = rear+1;
Queue[rear] = x;
end
```

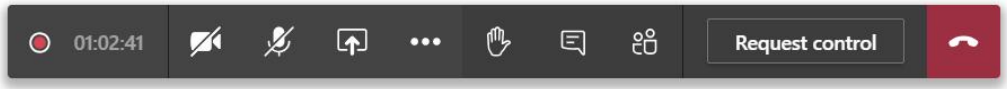
```
Dequeue(x)
begin
if IsEmpty()
    return;
else if front==rear
    front ← rear ← -1;
else
    front = front +1;
end
```

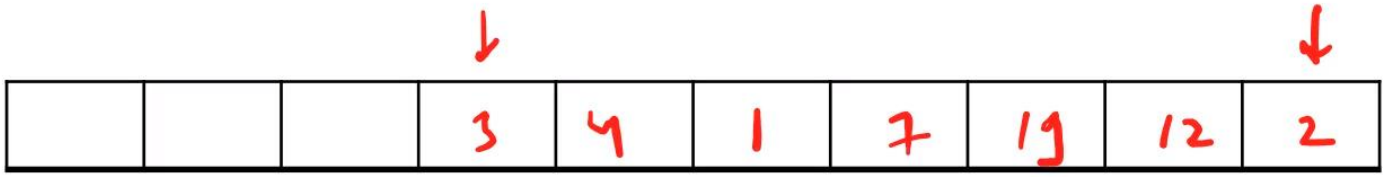
56:41 Request control

Other Variations of the Queue



- To save space, a circular queue is also proposed.
- Operations that update front and rear have to be based on modulo arithmetic.
- The circular queue is declared full when (rear+1)%N == front





Eg (7)

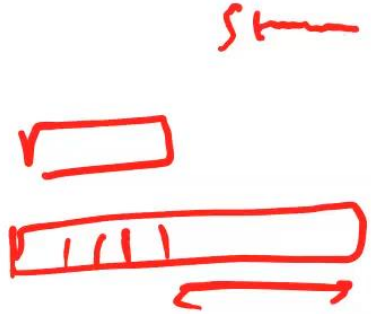
```
IsEmpty()
begin
if front==-1 && rear == -1
    return true;
else
    return false
end
```

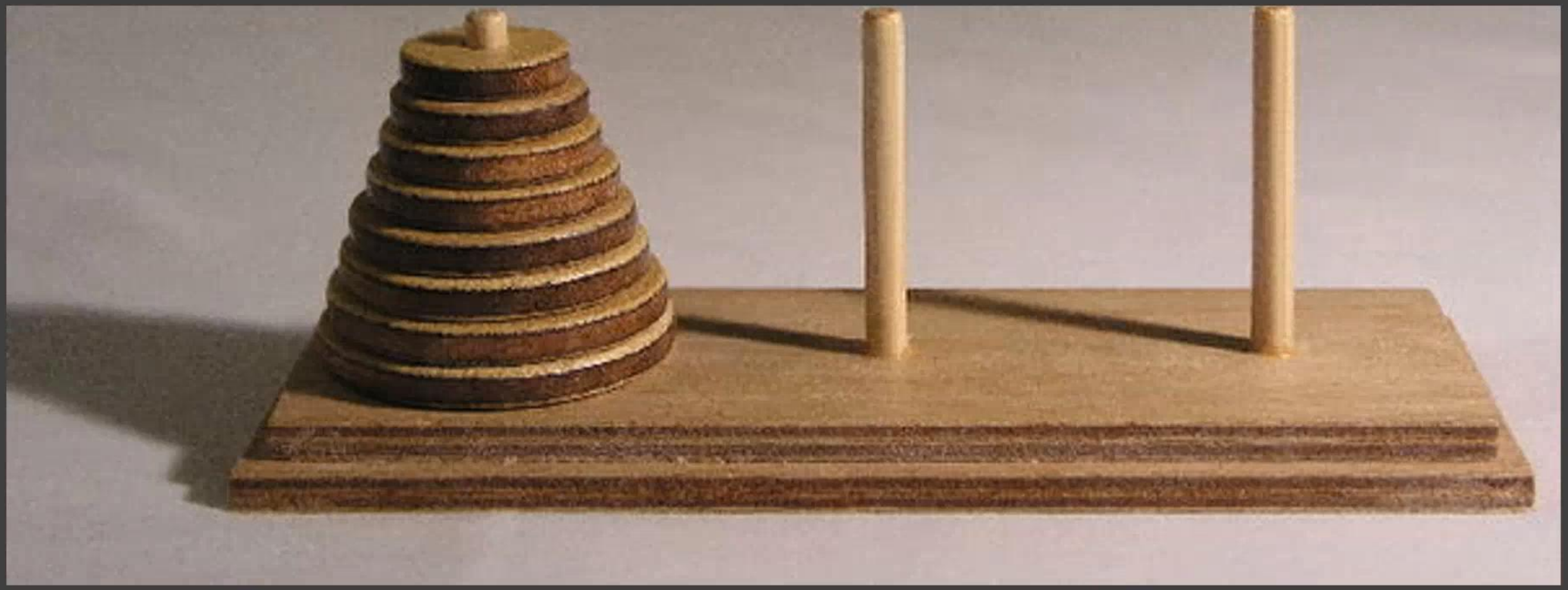
```
Enqueue(x)
begin
if (rear+1)%N+1 == front then
    return;
else if IsEmpty()
    front ← rear ← 0;
else
    rear = (rear+1)%N;
Queue[rear] = x;
end
```

```
Dequeue(x)
begin
if IsEmpty()
    return;
else if front==rear
    front ← rear ← -1;
else
    front = (front +1)%N;
end
```

A Sample Application with Stack and Queue

- A palindrome is a string that reads the same forwards and backwards, ignoring non-alphabetic characters.
- Examples:
  - Malayalam
  - Wonton? not now
  - Madam, i'm Adam
- Problem: Given a string, determine if it is a palindrome.
  - May not know the length of the string apriori.





Towers of Hanoi

- The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:
- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
- No larger disk may be placed on top of a smaller disk.