

## §3 The Stack ADT

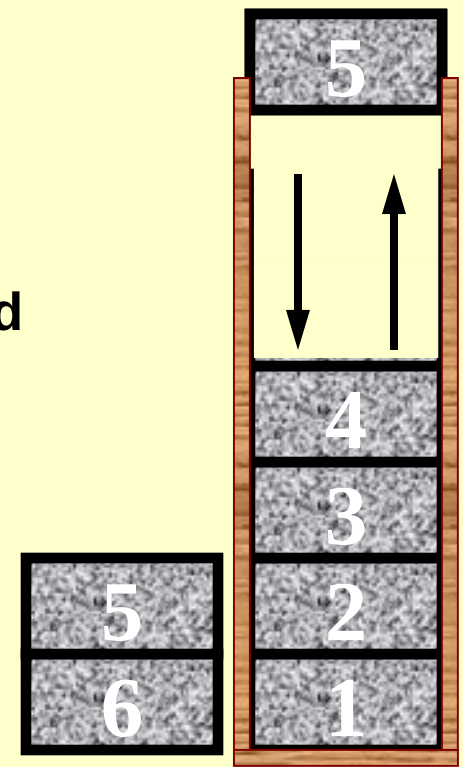
### 1. ADT

A **stack** is a Last-In-First-Out (LIFO) list, that is, an ordered list in which insertions and deletions are made at the **top** only.

**Objects:** A finite ordered list with zero or more elements.

**Operations:**

☞ **Int** **IsEmpty**( Stack S );  
☐ **Stack** **CreateStack**( );  
☐ **DisposeStack**( Stack S );  
☐ **MakeEmpty**( Stack S );  
☐ **Push**( ElementType X, Stack S );  
☐ ElementType **Top**( Stack S );  
☐ **Pop**( Stack S );



**Note:** A **Pop** (or **Top**) on an **empty** stack is an error in the stack ADT.

**Push** on a **full** stack is an implementation error but not an ADT error.

## 2. Implementations

### ➤ Linked List Implementation (with a header node)

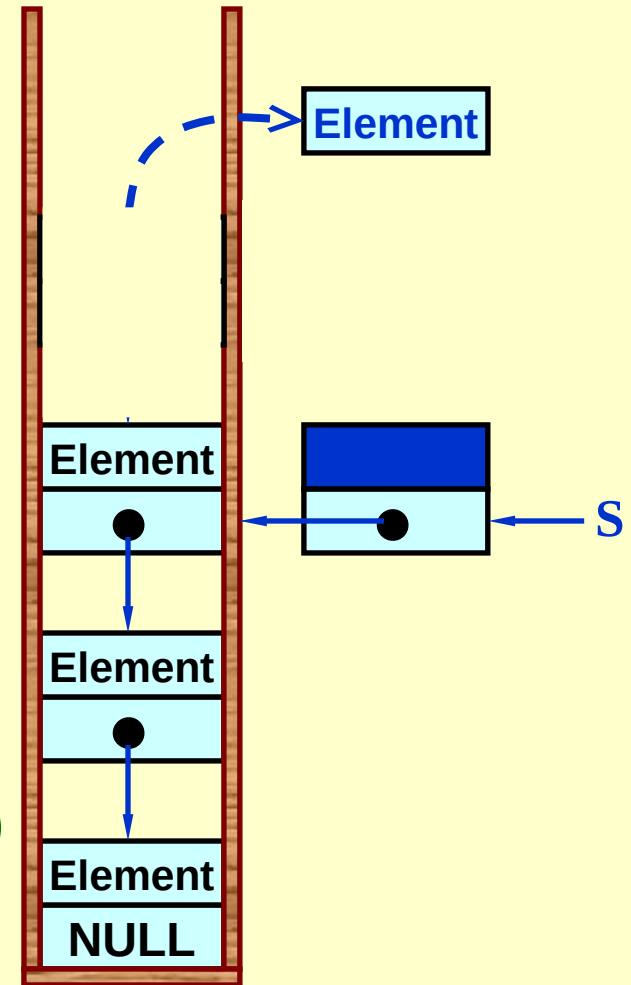
👉 **Push:** ①  $\text{TmpCell} \rightarrow \text{Next} = \text{S} \rightarrow \text{Next}$   
 ②  $\text{S} \rightarrow \text{Next} = \text{TmpCell}$

👉 **Top:**  $\text{return S} \rightarrow \text{Next} \rightarrow \text{Element}$

👉 **Pop:** ①  $\text{FirstCell} = \text{S} \rightarrow \text{Next}$   
 ②  $\text{S} \rightarrow \text{Next} = \text{S} \rightarrow \text{Next} \rightarrow \text{Next}$   
 ③  $\text{free}(\text{FirstCell})$



Easy! Simply keep  
 another stack as  
 a **recycle bin**.



## ➤ Array Implementation

```
struct StackRecord {  
    int    Capacity ;           /* size of stack */  
    int    TopOfStack;         /* the top pointer */  
    /* ++ for push, -- for pop, -1 for empty stack */  
    ElementType *Array; /* array for stack elements */  
};
```

**Note:** ① The stack model must be well **encapsulated**. That is, no part of your code, except for the stack routines, can attempt to access the **Array** or **TopOfStack** variable.

② Error check must be done before **Push** or **Pop** (**Top**).

Read Figures 3.38-3.52 for detailed implementations of stack operations.

### 3. Applications

#### ★ Balancing Symbols



Check if parenthesis ( ), brackets [ ], and braces { } are balanced.

```

Algorithm {
    Make an empty stack S;
    while (read in a character c) {
        if (c is an opening symbol)
            Push(c, S);
        else if (c is a closing symbol) {
            if (S is empty) { ERROR; exit; }
            else { /* stack is okay */
                if (Top(S) doesn't match c) { ERROR, exit; }
                else Pop(S);
            } /* end else-stack is okay */
        } /* end else-if-closing symbol */
    } /* end while-loop */
    if (S is not empty) ERROR;
}
  
```

$T(N) = O(N)$   
 where  $N$  is the length  
 of the expression.  
 This is an  
**on-line** algorithm.

# ★ Postfix Evaluation

〔 Example 〕 An **infix** expression:  $a + b * c - d / e$

A **prefix** expression:  $- + a * b c / d e$

A **postfix** expression:  $a b c * + d e / -$

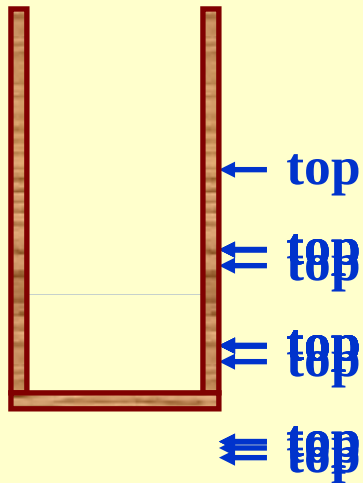
Reverse Polish notation

operand

operator with  
the highest  
precedence

operator

〔 Example 〕  $6\ 2\ /\ 3\ -\ 4\ 2\ *\ +\ 8\ ?$



Get token: 6 ( operand )	Get token: 2 ( operand )
Get token: / ( operator )	Get token: 3 ( operand )
Get token: - ( operator )	Get token: 4 ( operand )
Get token: 2 ( operand )	Get token: * ( operator )
Get token: + ( operator )	Pop: 8

$T(N) = O(N)$ . No need to know precedence rules.

# ★ Infix to Postfix Conversion

[ Example ]  $a + b * c - d \ a \ b \ c * + d -$

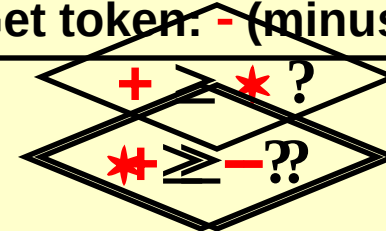
Note:

- The order of operands is the **same** in infix and postfix.
- **higher** precedence appear **before** those

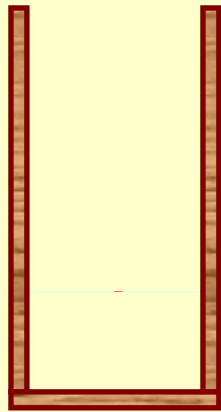
Isn't that  
simple?

Wait till  
you see the next  
example...

Get token: $a$ (operand)	Get token: $+$ (plus)
Get token: $b$ (operand)	Get token: $*$ (times)
Get token: $c$ (operand)	Get token: $-$ (minus)
Get token: $d$ (operand)	

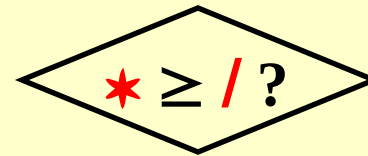


[ Example ]

 $a * (b + c) / c a b c + * d /$ Output:  $a b c + * d /$ 

← top

Get token: $a$ (operand)	Get token: $*$ (times)
Get token: $($ (lparen)	Get token: $b$ (operand)
Get token: $+$ (plus)	Get token: $c$ (operand)
Get token: $*$ (times)	Get token: $/$ (divide)
Get token: $d$ (operand)	



NO!!  $T(N) = O(N)$

## Solutions:

- ① Never pop a ( from the stack except when processing a ) .
- ② Observe that when ( is **not in** the stack, its precedence is the **highest**; but when it is **in** the stack, its precedence is the **lowest**. Define **in-stack** precedence and **incoming** precedence for symbols, and each time use the corresponding precedence for comparison.

Note:  $a - b - c$  will be converted to  $a b - c -$ . However,  $2^2 \wedge 3$  ( $2^{2^3}$ ) must be converted to  $2 2 3 \wedge \wedge$ , not  $2 2 \wedge 3 \wedge$  since exponentiation associates **right to left**.



# ★ Function Calls -- System Stack

Recursion can always be **completely removed**.  
 Non recursive programs are generally **faster** than  
 equivalent recursive programs.  
 However, recursive programs are in general  
 much **simpler and easier to understand**.

```
void PrintList ( List L )
{
    if ( L != NULL ) {
        PrintElement ( L->Element );
        PrintList( L->next );
    }
} /* a bad use of recursion */
```

```
void PrintList ( List L )
{
    top: if ( L != NULL ) {
        PrintElement ( L->Element );
        L = L->next;
        goto top; /* do NOT do this */
    }
} /* compiler removes recursion */
```

# §4 The Queue ADT

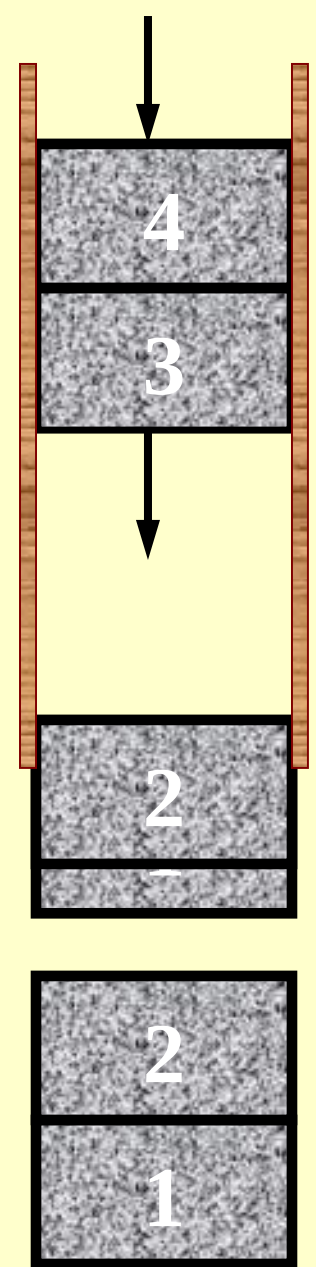
## 1. ADT

A **queue** is a First-In-First-Out (FIFO) list, that is, an ordered list in which insertions take place at one end and deletions take place at the opposite end.

**Objects:** A finite ordered list with zero or more elements.

**Operations:**

- int **IsEmpty**( Queue Q );
- Queue **CreateQueue**( );
- **DisposeQueue**( Queue Q );
- **MakeEmpty**( Queue Q );
- **Enqueue**( ElementType X, Queue Q );
- ElementType **Front**( Queue Q );
- **Dequeue**( Queue Q );

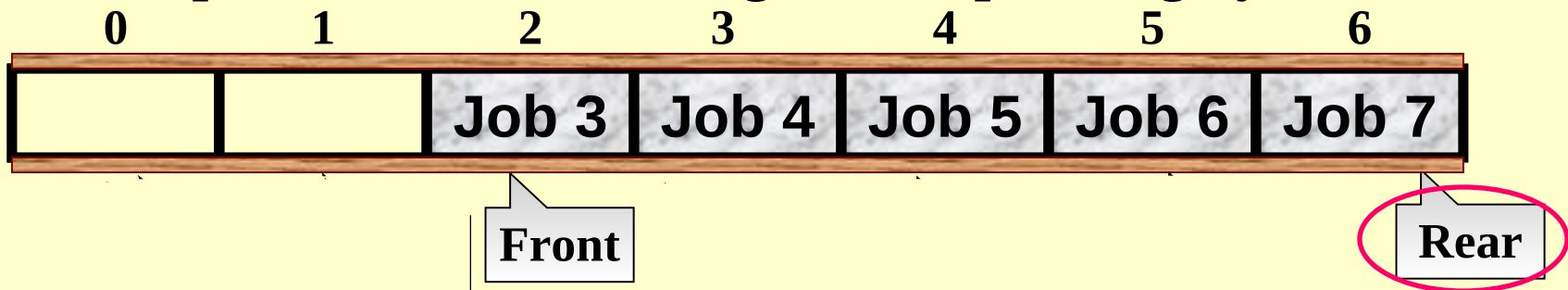


## 2. Array Implementation of Queues

(Linked list implementation is trivial)

```
struct QueueRecord {
    int    Capacity ; /* max size of queue */
    int    Front;     /* the front pointer */
    int    Rear;      /* the rear pointer */
    int    Size; /* Optional - the current size of queue */
    ElementType *Array; /* array for queue elements */
};
```

[ Example ] Job Scheduling in an Operating System

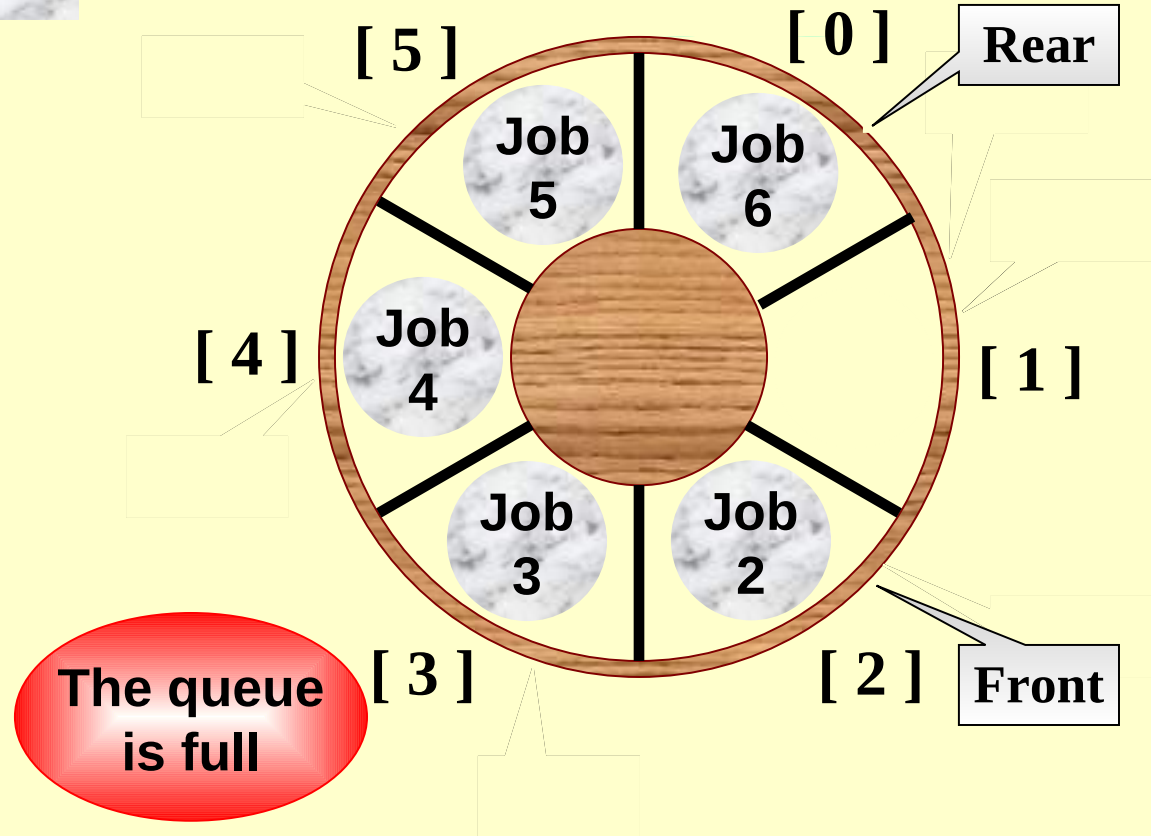


Enqueue Job 1	Enqueue Job 2	Enqueue Job 3	Dequeue Job 1
Enqueue Job 4	Enqueue Job 5	Enqueue Job 6	Dequeue Job 2
Enqueue Job 7	Enqueue Job 8		

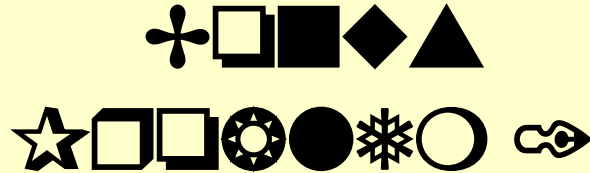
# Circular Queue:

Enqueue Job 1
Enqueue Job 2
Enqueue Job 3
Enqueue Job 4
Enqueue Job 5
Enqueue Job 6
Enqueue Job 7
Enqueue Job 8
Enqueue Job 9
Enqueue Job 10
Enqueue Job 11
Enqueue Job 12
Enqueue Job 13
Enqueue Job 14
Enqueue Job 15
Enqueue Job 16
Enqueue Job 17
Enqueue Job 18
Enqueue Job 19
Enqueue Job 20
Enqueue Job 21
Enqueue Job 22
Enqueue Job 23
Enqueue Job 24
Enqueue Job 25
Enqueue Job 26
Enqueue Job 27
Enqueue Job 28
Enqueue Job 29
Enqueue Job 30
Enqueue Job 31
Enqueue Job 32
Enqueue Job 33
Enqueue Job 34
Enqueue Job 35
Enqueue Job 36
Enqueue Job 37
Enqueue Job 38
Enqueue Job 39
Enqueue Job 40
Enqueue Job 41
Enqueue Job 42
Enqueue Job 43
Enqueue Job 44
Enqueue Job 45
Enqueue Job 46
Enqueue Job 47
Enqueue Job 48
Enqueue Job 49
Enqueue Job 50
Enqueue Job 51
Enqueue Job 52
Enqueue Job 53
Enqueue Job 54
Enqueue Job 55
Enqueue Job 56
Enqueue Job 57
Enqueue Job 58
Enqueue Job 59
Enqueue Job 60
Enqueue Job 61
Enqueue Job 62
Enqueue Job 63
Enqueue Job 64
Enqueue Job 65
Enqueue Job 66
Enqueue Job 67
Enqueue Job 68
Enqueue Job 69
Enqueue Job 70
Enqueue Job 71
Enqueue Job 72
Enqueue Job 73
Enqueue Job 74
Enqueue Job 75
Enqueue Job 76
Enqueue Job 77
Enqueue Job 78
Enqueue Job 79
Enqueue Job 80
Enqueue Job 81
Enqueue Job 82
Enqueue Job 83
Enqueue Job 84
Enqueue Job 85
Enqueue Job 86
Enqueue Job 87
Enqueue Job 88
Enqueue Job 89
Enqueue Job 90
Enqueue Job 91
Enqueue Job 92
Enqueue Job 93
Enqueue Job 94
Enqueue Job 95
Enqueue Job 96
Enqueue Job 97
Enqueue Job 98
Enqueue Job 99
Enqueue Job 100

**Question:**  
Why is the queue  
announced full  
while there is  
still a free  
space left?



**Note:** Adding a **Size** field can avoid wasting one empty space to distinguish “full” from “empty”. Do you have any other ideas?



## One Way In, Two Ways

Due: Tuesday, January 4<sup>th</sup>, 2022 at 10:00pm

(2 points)

The problem can be found and submitted at

<https://pintia.cn/>