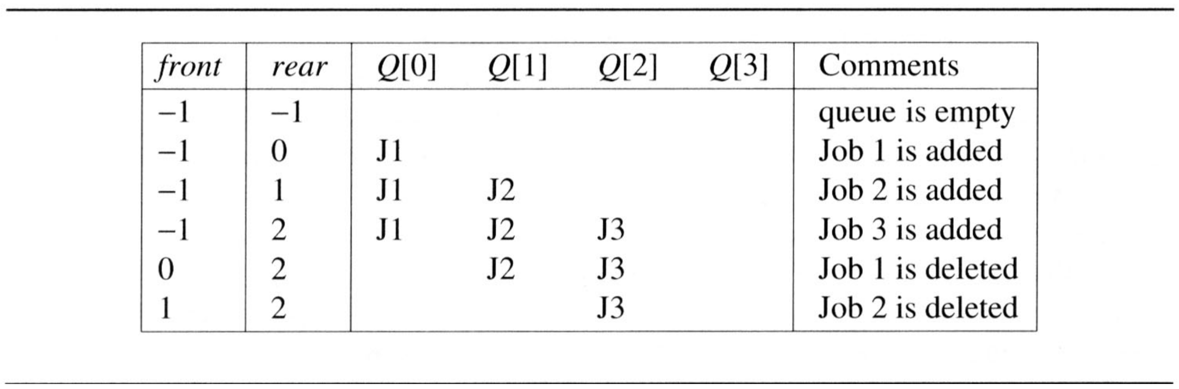
Stacks and Queues

Chapter - 8

The queue ADT

- **Example [*Job scheduling*]:**
  - Figure illustrates how an operating system might process jobs if it used a sequential representa



front	*rear*	*Q*[0]	*Q*[1]	*Q*[2]	*Q*[3]	Comments
−1	−1					queue is empty
−1	0	J1				Job 1 is added
−1	1	J1	J2			Job 2 is added
−1	2	J1	J2	J3		Job 3 is added
0	2		J2	J3		Job 1 is deleted
1	2			J3		Job 2 is deleted

    - As jobs enter and leave the system, the queue gradually shift to right.
    - In this case, *queue_full* should move the entire queue to the left so that the first element is again at *queue*[0], *front* is at -1, and *rear* is correctly positioned.
      - Shifting an array is very time-consuming, *queue_full* has a

Exercises

- Write a function that accepts a queue of integers as a parameter and replaces every element of the queue with two copies of that element.
  - `front [1, 2, 3] back`
    becomes
    `front [1, 1, 2, 2, 3, 3] back`
- Write a function that accepts a queue of strings as a parameter and appends the queue's contents to itself in reverse order.
  - `front [a, b, c] back`
    becomes
    `front [a, b, c, c, b, a] back`

Infix, Prefix and Postfix Expressions

- We usually write algebraic expressions like this:

  (a + b)

- This is called **infix notation**, because the operator ("+") is inside the expression
- With postfix and prefix notations, parentheses are no longer needed
- **Prefix** notation : + a b
- **Postfix** notation: a b +

Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

$$\begin{tikzcd}[column sep=tiny] (& (& A & + \arrow[l, bend left] & B) & * & (& C & + & D &) &) \end{tikzcd}$$

Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

$$\begin{tikzcd}[column sep=small] (& + & A & B & * \arrow[lll, bend left] & (& C & + & D &) &) \end{tikzcd}$$

Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

$$\begin{tikzcd}[column sep=small] * & + & A & B & (& C \arrow[ll, bend left] & + & D &) \end{tikzcd}$$

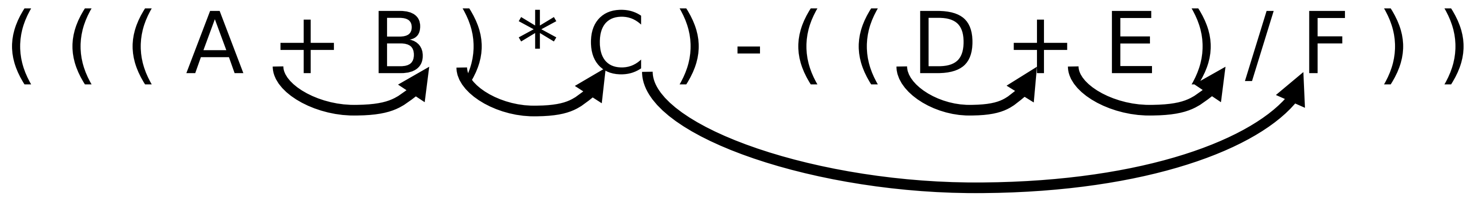
Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

* + A B + C   D

Order of operands does not change

Infix to Postfix



A B + C * D E + F / -

- Operand order does not change
- Operators are in order of evaluation

example: infix 1+2*3 to postfix

- (1+(2*3)) add parentheses
- (1+(23*)) convert multiplication
- (1(23*)+) convert addition
- 123*+ remove parentheses

example: infix (1+2)*(3+4)

- with parentheses: ((1+2)*(3+4))
- in postfix: 12+34+*
- in prefix: *+12+34

one more: infix 1^2*3-4+5/6/(7+8)

- paren.: ((((1^2)*3)-4)+((5/6)/(7+8)))
- in postfix: 12^3*4-56/78+/+
- in prefix: +-*^1234//56+78

Evaluate postfix expression

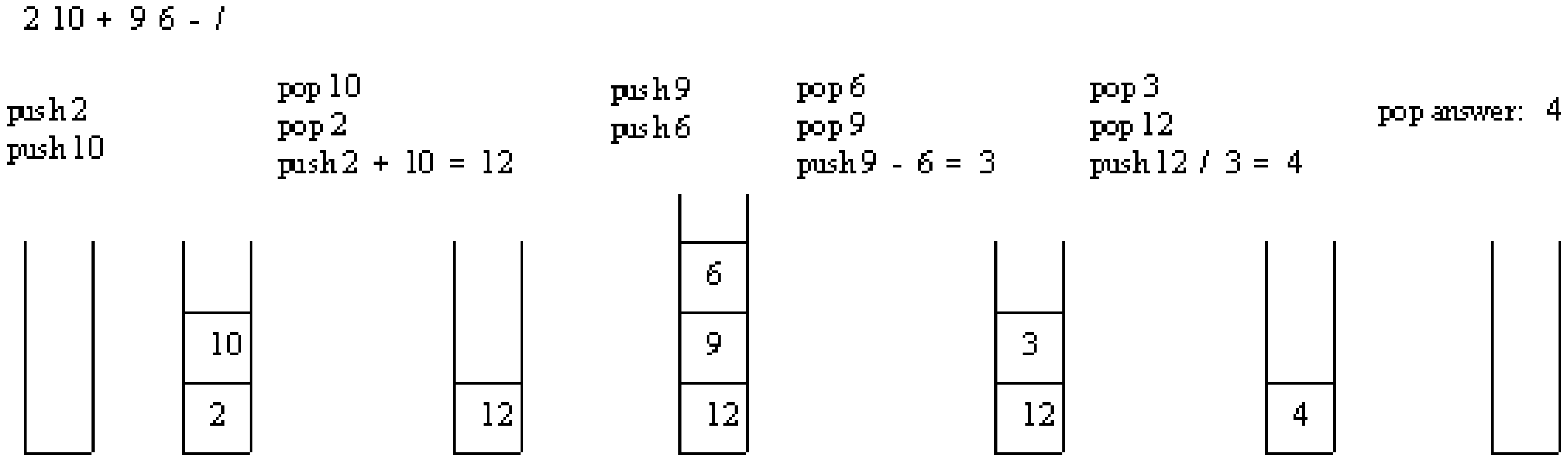
example: 623+-382/+*2^3+

- - scan from left to right: 6,2,3,+
- - apply + to 2 and 3: 6,5
- - scan further: 6,5,-
- - apply - to 6 and 5: 1
- - scan further: 1,3,8,2,/
- - apply / to 8 and 2: 1,3,4
- - scan further: 1,3,4,+
- - apply + to 3 and 4: 1,7
- - scan further: 1,7,*
- - apply * to 1 and 7: 7
- - scan further: 7,2,^
- - apply ^ to 7 and 2: 49
- - scan further: 49,3,+
- - apply + to 49 and 3: **52**

Application of Stacks - Evaluating Postfix Expression (Cont'd)

- Example: Consider the postfix expression, **2 10 + 9 6 - /,** which is **(2 + 10) / (9 - 6)** in infix, the result of which is 12 / 3 = 4.

- The following is a trace of the postfix evaluation algorithm for the postfix expression:


2 10 + 9 6 - /
push 2
push 10
pop 10
pop 2
push 2 + 10 = 12
push 9
push 6
pop 6
pop 9
push 9 - 6 = 3
pop 3
pop 12
push 12 / 3 = 4
pop answer: 4
10
2
12
6
9
12
3
12
4