Queues & Stacks in Practice

Part 4

1

Queues & Stacks in Practice

1001 Uses!
(I meant 1,001 – not 9)

2

HTML Tag Matching

- HTML is a hierarchical structure
- HTML consists of tags
  - each tag can also embed other tags
  - allows text to be aligned, made bold, etc…

3

HTML Tag Matching

- Web browsers read the text and apply a tag depending if it is active
- They maintain a stack…
  - push a start tag, pop and end tag
  - if the HTML is correct, they should match
  - *… with the exception of the unary tags*

4

HTML Tag Matching

```
<html>
<body>
<center>
<h1>Banks of Sacramento</h1>
</center>
<i>A bully ship and a bully crew.<br>
Hoo-da! Hoo-da!<br>
A bully mate and a captain too.<br>
Hoo-da! Hoo-da-day!<br>
And it's blow, ye winds, blow,<br>
for Californi-o.<br>
For there's plenty of gold,<br>
so I've been told,<br>
on the banks of the
Sacramento.</i><br>
</body>
</html>
```

Banks of Sacramento

A bully ship and a bully crew.
Hoo-da! Hoo-da!
A bully mate and a captain too.
Hoo-da! Hoo-da-day!

Then blow, ye winds, blow,
for Californi-o.
For there's plenty of gold,
so I've been told,
on the banks of the Sacramento.

5

Balanced Parentheses

- When analyzing arithmetic expressions…
  - it is important to determine whether it is balanced with respect to parentheses
  - otherwise, the expression is incorrect
- A great solution is a stack
  - push each (and pop each)
  - at the end, the stack should be empty
  - also, if you attempt to pop on an empty stack, the expression is invalid

6

Balanced Parenthesis Examples

`(a + b)`	Balanced
`(a + b))`	Pop empty stack
`) a + b (`	Pop empty stack
`(a + (b + 1) * c) / e`	Balanced
`(a * (b + ((d + e) * f))`	Stack has 1 left

7

Balanced Parentheses

- *But wait…*
  - can we just use a "parenthesis level" counter?
  - if it is `>= 1` at the end or if it ever is `< 0`, the expression is invalid
- Sorry, it won't work…
  - some expressions allow `{ }` and `[ ]`
  - a simple counter is insufficient
  - stack can check if the pop'd item matches

8

Balanced Parenthesis Examples

`[a + b]`	Balanced
`(a + b}`	Mismatch
`{[ a + b ]]`	Mismatch
`(a + (b + 1) * c / e`	Unbalanced
`(a * [b + {c + d} * e])`	Balanced

9

Evaluating Expressions

A Stack and Queue working together!

10

Evaluating Expressions

- It is a common task in programs to evaluate mathematical expressions and get a result
- Computers can perform this task using an algorithm *created by Dijkstra*, but we will get into that later

11

Evaluating Expressions

- First, we need to look at mathematical expressions
- We usually use infix notation
  - not stack or queue "friendly"
  - there are, however, two alternative notations
  - *one of which is stack friendly*

12

Infix Notation

- Using *infix notation*, we put the operator in between the two operands
- This is the standard format used today

To add the numbers *a* and *b*, we type:	`a + b`
To divide *a* by *b*, we type:	`a / b`

13

Prefix Notation

- *Prefix notation*, rather than putting the operator between the operands, puts it first
- It is also called *"Polish Notation"*
- Used by the LISP programming language

To add the numbers *a* and *b*, we type:	`+ a b`
To divide *a* by *b*, we type:	`/ a b`

14

Postfix Notation

- *Postfix notation* puts the operator at the end
- Also called *"Reverse Polish Notation" (RPN)*
- Since the operator is last, we can also use it as a "trigger" to perform math

To add the numbers *a* and *b*, we type:	`a b +`
To divide *a* by *b*, we type:	`a b /`

15

Where are My Parenthesis?

Infix	Prefix	Postfix
`a + b * c`	`+ a * b c`	`a b c * +`
`(a - b) * c`	`- a b * c`	`a b - c *`
`(a / (b - c) + d)`	`+ / a - b c d`	`a b c - / d +`
`(a + b / (c - d))`	`+ a / b - c d`	`a b c d - / +`

16

Where are My Parenthesis?

- Infix is the only notation that needs parentheses to change precedence
- The order of operators handles precedence in prefix and postfix

17

Compute Postfix Algorithm

- Computing a postfix expression is easy
- All you need is:
  - one queue that contains the values & operators
  - and one stack
- In fact, on classic Hewlett Packard calculators, all operations are stack based

18

Compute Postfix Pseudo-code

```
while there is data in the input queue
   dequeue a token (value or operator)
   if it's a value, push it on the stack
   if it's an operator
      pop two numbers from the stack
      compute the result (using the operator)
      push the result on the stack
   end if
end while

...Afterwards, the final result is on the stack
```


19

Compute Postfix Demo

Input Queue: 24 10 7 − / 34 +

24 / (10 − 7) + 34

Stack:

20

Compute Postfix Demo

Input Queue: 10 7 − / 34 +

Stack: 24

21

Compute Postfix Demo

Input Queue: 7 − / 34 +

Stack: 24 10

22

Compute Postfix Demo

Input Queue: − / 34 +

Stack: 24 10 7

23

Compute Postfix Demo

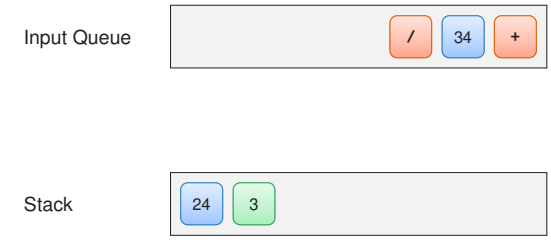
Input Queue: / 34 +

10 − 7

Stack: 24

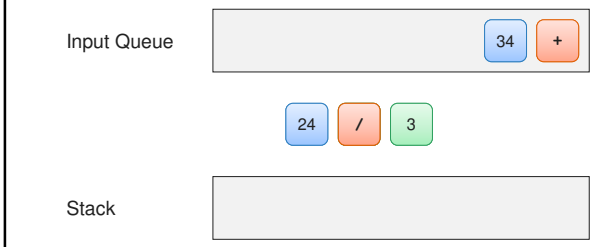
24

Compute Postfix Demo



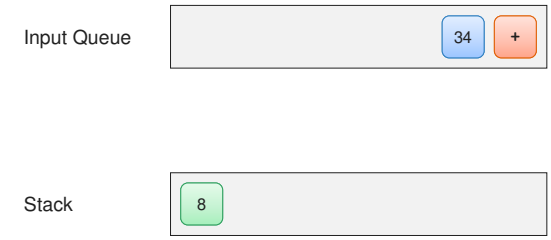
25

Compute Postfix Demo



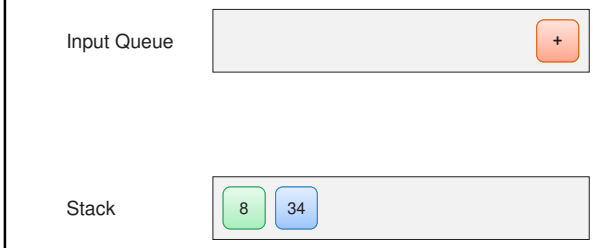
26

Compute Postfix Demo



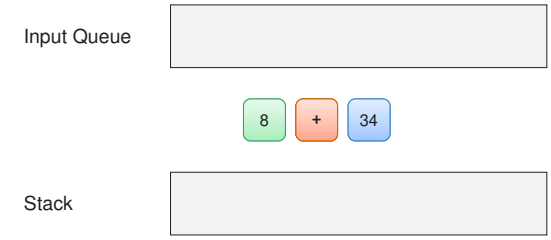
27

Compute Postfix Demo



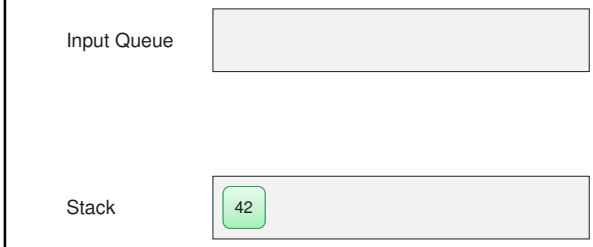
28

Compute Postfix Demo



29

Compute Postfix Demo



30

Converting to Prefix or Postfix

- Why are learning this... *be patient!*
- Converting infix to either postfix or prefix notation is easy to do by hand
- Did you notice that the operands did not change order? They were always *a, b, c…*
- We just need to rearrange the operators

31

Convert Infix to Prefix / Postfix

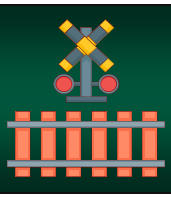
1. Make it a *Fully Parenthesized Expression (FPE)* - one pair of parentheses enclosing each operator and its operands
2. Move the operators to the start (prefix) or end (postfix) of each sub-expression
3. Finally, remove all the parenthesis

32

Infix to Postfix

```
a / (b – c) + d
1. ( ( a / (b – c) ) + d )
2. ( ( a ( b c – ) / ) d + )
3. a b c – / d +
```


33



Infix to Postfix Algorithm

Let the computer do the work…

34

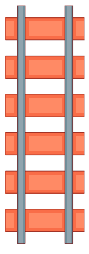
Edsger Dijkstra

- *Edsger Dijkstra* is a World-famous computer scientist
- He invented a wealth of algorithms
- For his contributions, he received the Turing Award

35

Infix to Postfix Algorithm

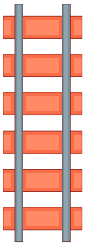
- Infix expressions need to be converted to postfix to be evaluated
- Dijkstra's *Shunting-yard algorithm* performs this task



36

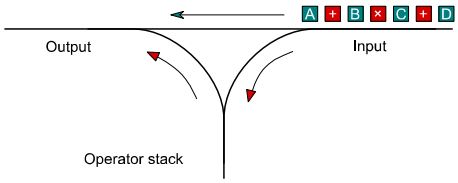
Shunting-yard algorithm

- Named after railroad shunting yards – which move trains onto different tracks
- Dijkstra's solution uses an input queue, operator stack, and output queue



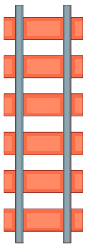
37

Shunting-yard Algorithm



38

Shunting-yard Algorithm



- The most basic version of this algorithm requires *Fully-Parenthesized Expression*
- This means, there is no precedence and parenthesis are put around every operator

39

FPE Shunting-yard Algorithm

```
while the input queue has tokens
   read a token from the input queue
   if the token is a…
      operand : add it to output queue
      operator : push it on the stack
      '(' : push it onto the stack
      ')' :
         while the top of stack isn't a '('
            pop an operator
            add it to the output queue
         end while
         pop and discard the extra '('
   end if
end while
```


40

FPE Shunting-yard Algorithm

Input Queue	((a * (b + c)) / d)
Operator Stack	
Output Queue	

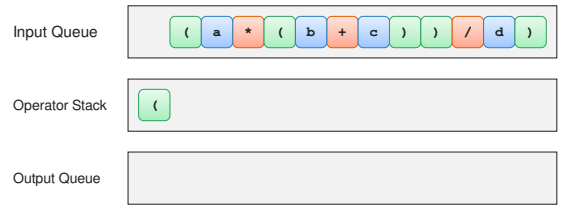
41

FPE Shunting-yard Algorithm

Input Queue	((a * (b + c)) / d)
Operator Stack	
Output Queue	

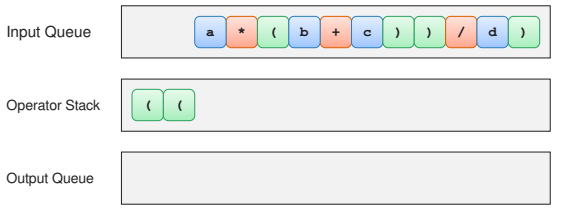
42

FPE Shunting-yard Algorithm



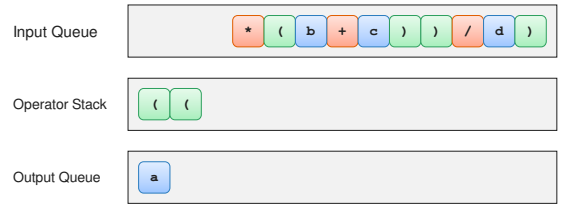
43

FPE Shunting-yard Algorithm



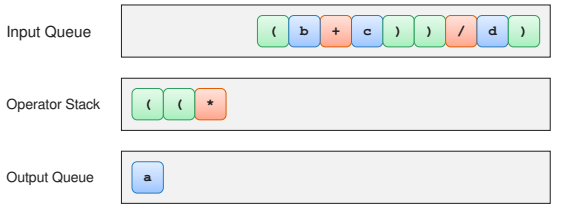
44

FPE Shunting-yard Algorithm



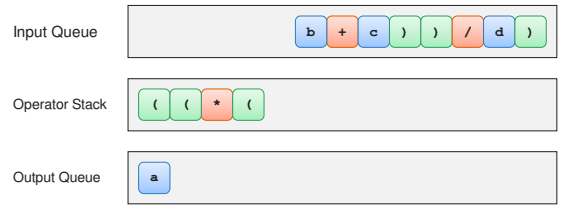
45

FPE Shunting-yard Algorithm



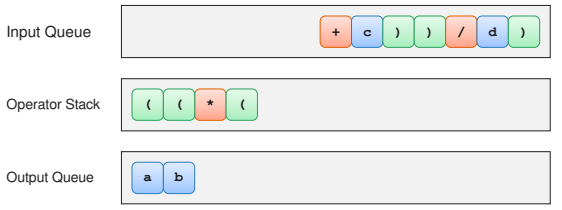
46

FPE Shunting-yard Algorithm



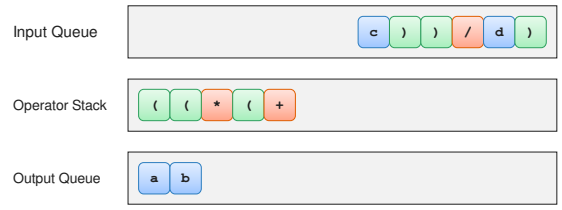
47

FPE Shunting-yard Algorithm



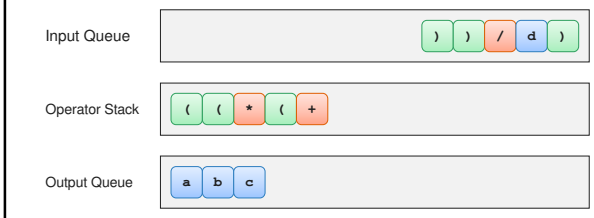
48

FPE Shunting-yard Algorithm



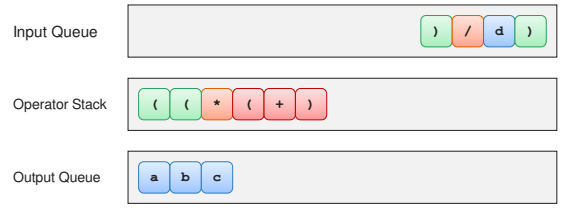
49

FPE Shunting-yard Algorithm



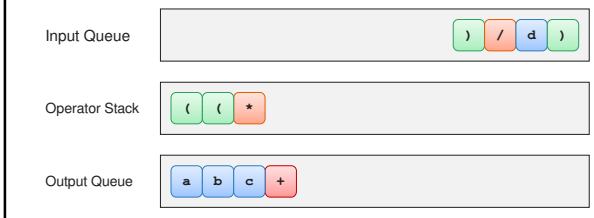
50

FPE Shunting-yard Algorithm



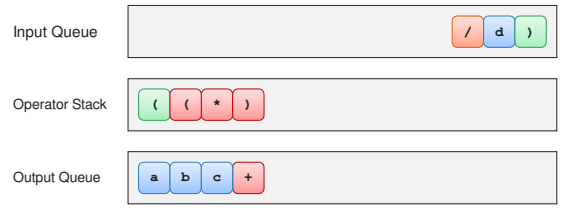
51

FPE Shunting-yard Algorithm



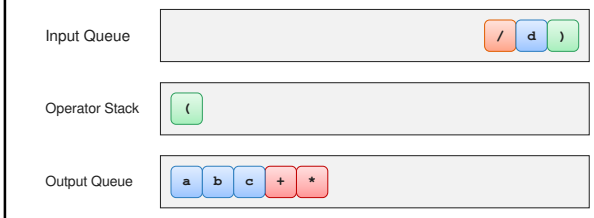
52

FPE Shunting-yard Algorithm



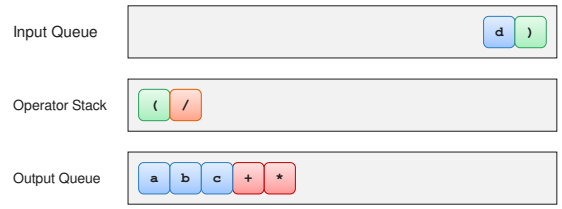
53

FPE Shunting-yard Algorithm



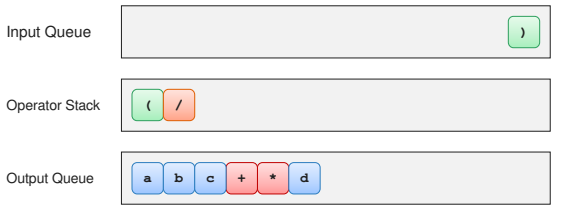
54

FPE Shunting-yard Algorithm



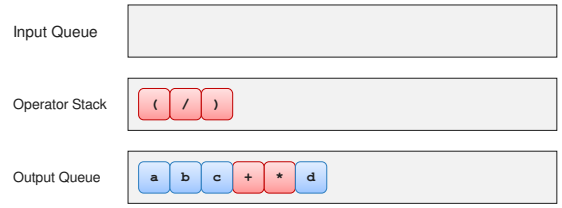
55

FPE Shunting-yard Algorithm



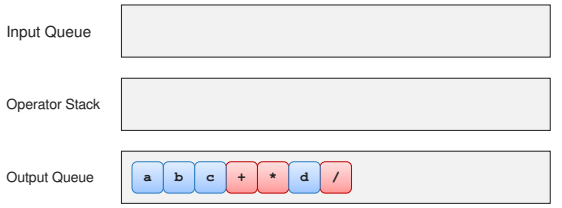
56

FPE Shunting-yard Algorithm



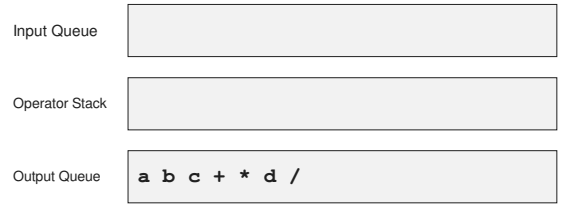
57

FPE Shunting-yard Algorithm



58

FPE Shunting-yard Algorithm



59

Too Many Paranthesis!

- FPE's are rarely used in real-World examples
- In fact, we use precedence rules to simplify expressions
- Fortunately, the algorithm can be modified, *very easily*, to handle precedence!

60

Non-FPE Shunting-yard Algorithm

```
while the input queue has tokens
   read a token from the input queue
   if the token is a…
      operand : add it to output queue
      operator : new rules - see next slide
      '(' : push it onto the stack
      ')' :
         while the top of stack isn't a '('
            pop an operator
            add it to the output queue
         end while
         pop and discard the '('
   end if
end while
```


61

Operator: New Rules

```
if operator is left-associative
    while top of stack is ≥ operator and not a '('
       pop the stack
       add it to the output queue
    end while
if operator is right-associative
    while top of stack is > operator and not a '('
       pop the stack
       add it to the output queue
    end while
push the operator onto the stack
```


62

Operator Associatively

Operator	Associatively
+ - * /	Left
^ (exponent)	Right

63

Shunting-yard Algorithm Example 1

Input Queue: a - b * c + d

Operator Stack:

Output Queue:

64

Shunting-yard Algorithm Example 1

Input Queue: a - b * c + d

Operator Stack:

Output Queue:

65

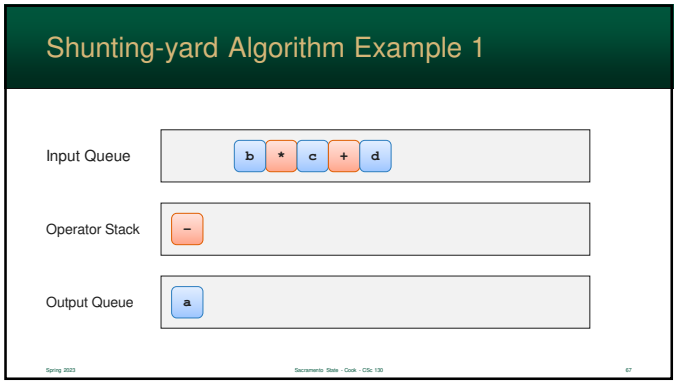
Shunting-yard Algorithm Example 1

Input Queue: - b * c + d

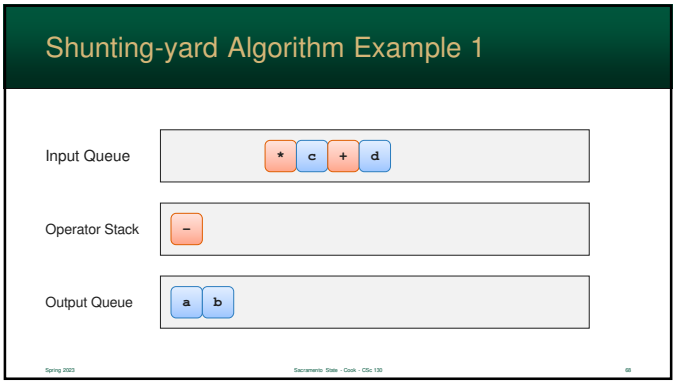
Operator Stack:

Output Queue: a

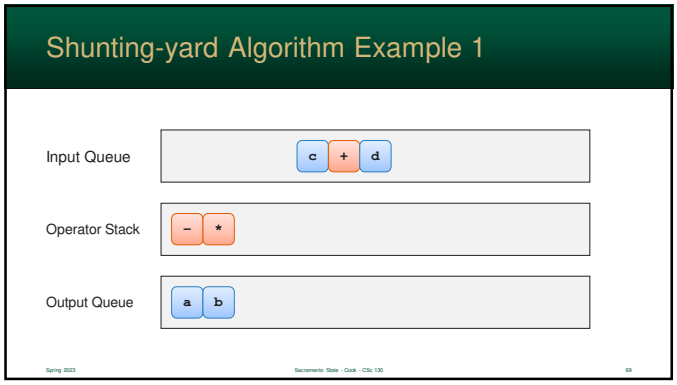
66



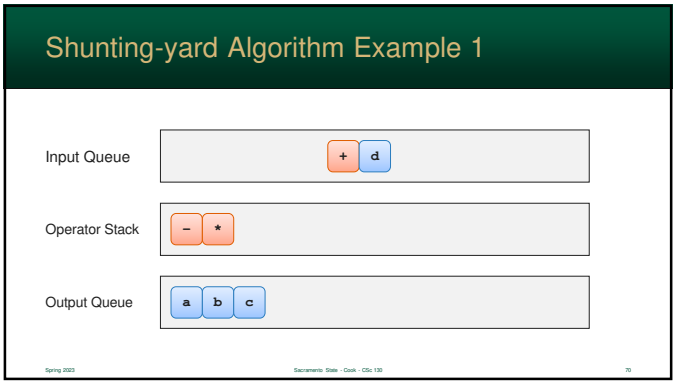
67



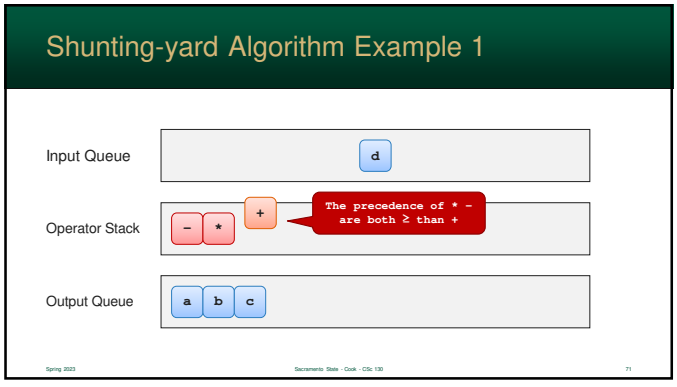
68



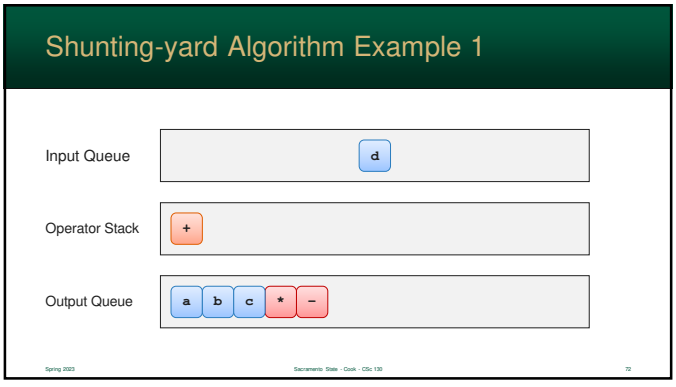
69



70

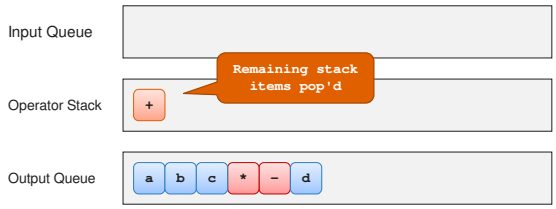


71



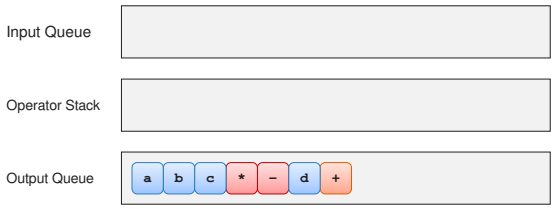
72

Shunting-yard Algorithm Example 1



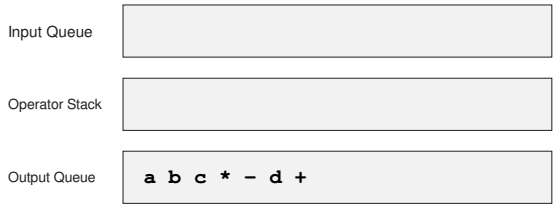
73

Shunting-yard Algorithm Example 1



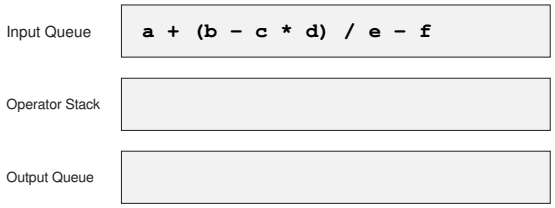
74

Shunting-yard Algorithm Example 1



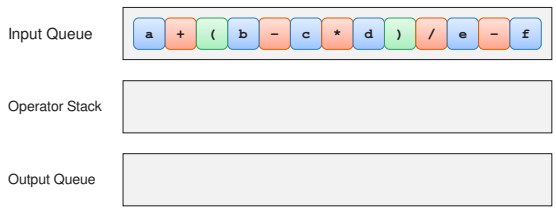
75

Shunting-yard Algorithm Example 2



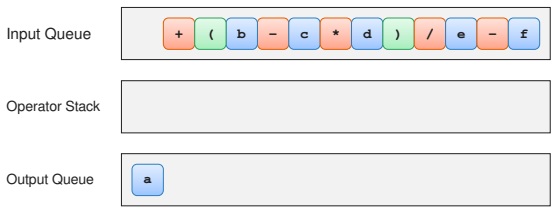
76

Shunting-yard Algorithm Example 2



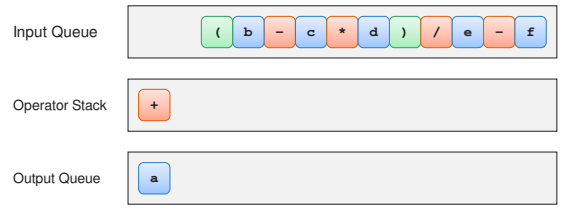
77

Shunting-yard Algorithm Example 2



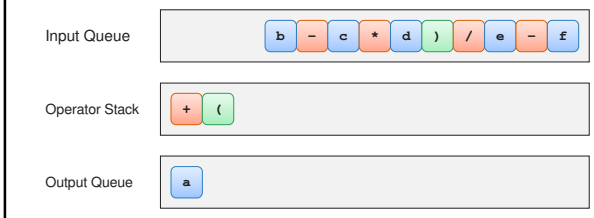
78

Shunting-yard Algorithm Example 2



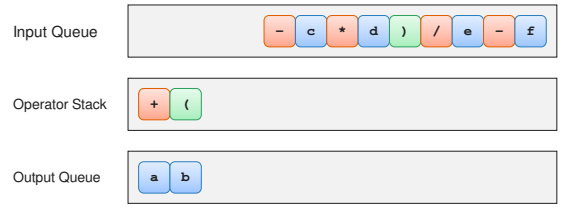
79

Shunting-yard Algorithm Example 2



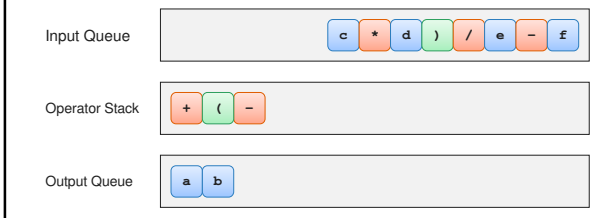
80

Shunting-yard Algorithm Example 2



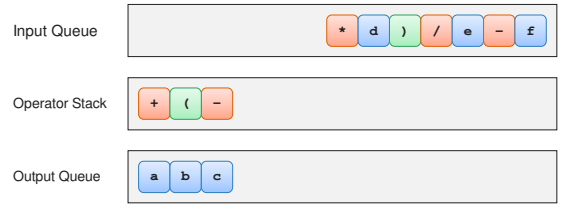
81

Shunting-yard Algorithm Example 2



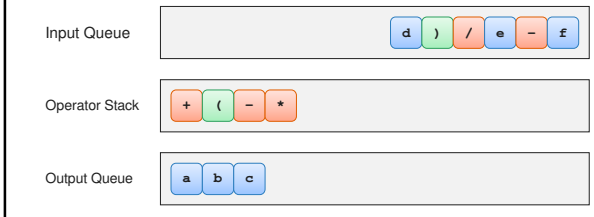
82

Shunting-yard Algorithm Example 2



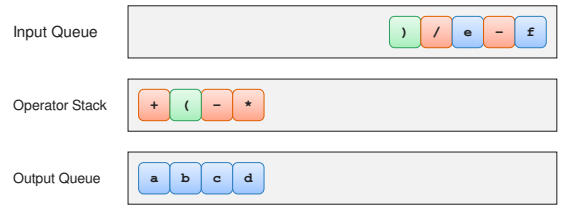
83

Shunting-yard Algorithm Example 2



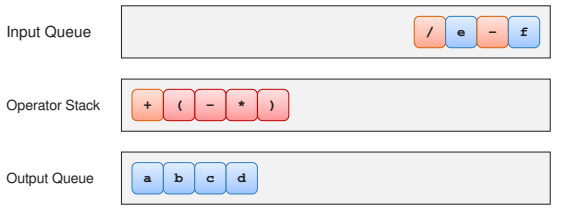
84

Shunting-yard Algorithm Example 2



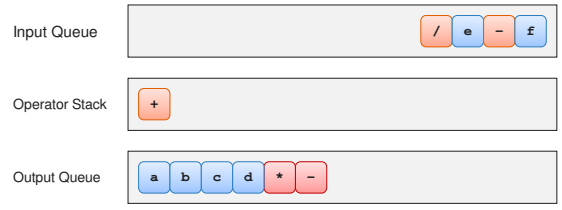
85

Shunting-yard Algorithm Example 2



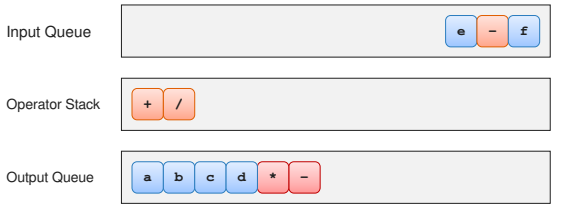
86

Shunting-yard Algorithm Example 2



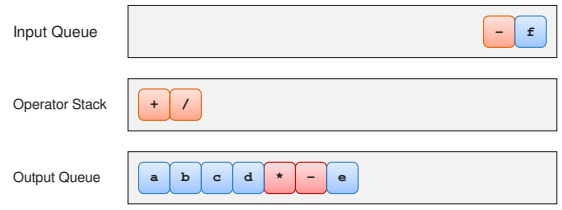
87

Shunting-yard Algorithm Example 2



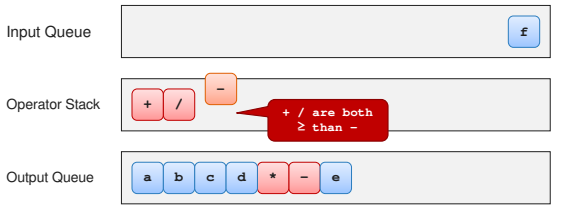
88

Shunting-yard Algorithm Example 2



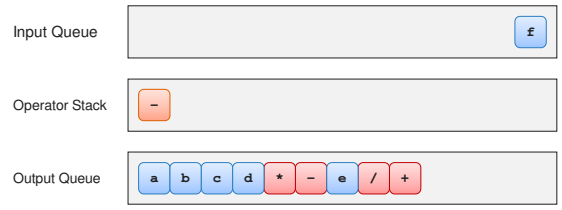
89

Shunting-yard Algorithm Example 2



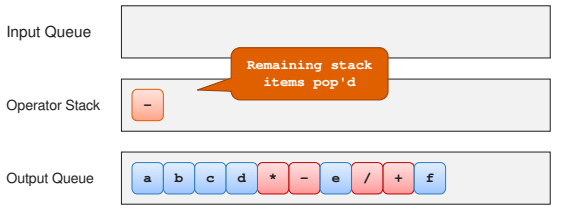
90

Shunting-yard Algorithm Example 2



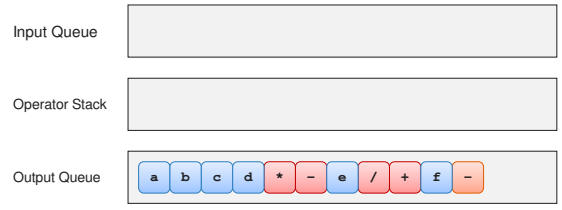
91

Shunting-yard Algorithm Example 2



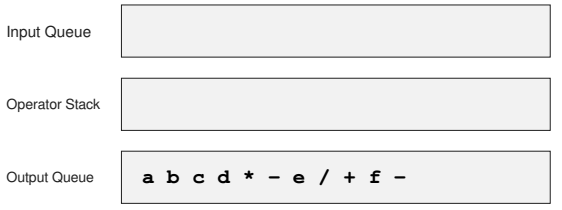
92

Shunting-yard Algorithm Example 2



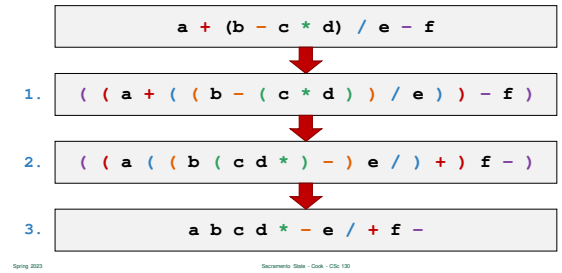
93

Shunting-yard Algorithm Example 2



94

Testing Our Result



95