



Queues & Stacks in Practice

Part 4



Queues & Stacks in Practice

1001 Uses!
(I meant 1,001 – not 9)

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...



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

HTML Tag Matching

```
<html>
<body>
<center>
<h1>Banks of Sacramento</h1>
</center>
<i>A bully ship and a bully crew.
Hoo-da! Hoo-da!
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.*

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

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

Fall 2021

Stacks and Queues - CS50.10B

7

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

Fall 2021

Stacks and Queues - CS50.10B

8

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

Fall 2021

Stacks and Queues - CS50.10B

9

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



Fall 2021

Stacks and Queues - CS50.10B

11

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*



Fall 2021

Stacks and Queues - CS50.10B

12

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

Fall 2021

Scenario 1: Data - CS50.100

13

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

Fall 2021

Scenario 1: Data - CS50.100

14

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 /

Fall 2021

Scenario 1: Data - CS50.100

15

15

Where are My Parenthesis?

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

Fall 2021

Scenario 1: Data - CS50.100

16

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



Fall 2021

Scenario 1: Data - CS50.100

17

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



Fall 2021

Scenario 1: Data - CS50.100

18

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
    
```

Fall 2021

Scenario 3: Stack - CS51B

19

19

Compute Postfix Demo

Input Queue



24 / (10 - 7) + 34

Stack



Fall 2021

Scenario 3: Stack - CS51B

20

20

Compute Postfix Demo

Input Queue



Stack



Fall 2021

Scenario 3: Stack - CS51B

21

21

Compute Postfix Demo

Input Queue



Stack



Fall 2021

Scenario 3: Stack - CS51B

22

22

Compute Postfix Demo

Input Queue



Stack



Fall 2021

Scenario 3: Stack - CS51B

23

23

Compute Postfix Demo

Input Queue



Stack

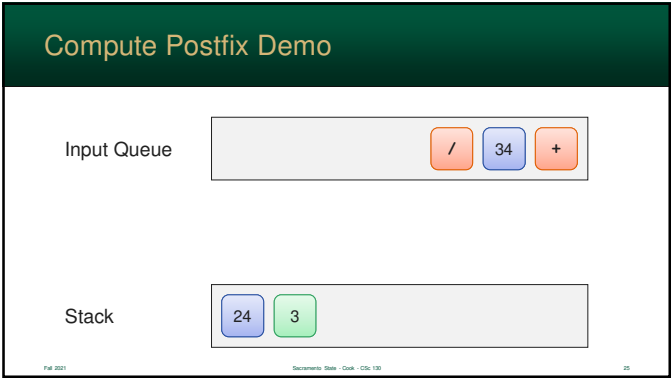


Fall 2021

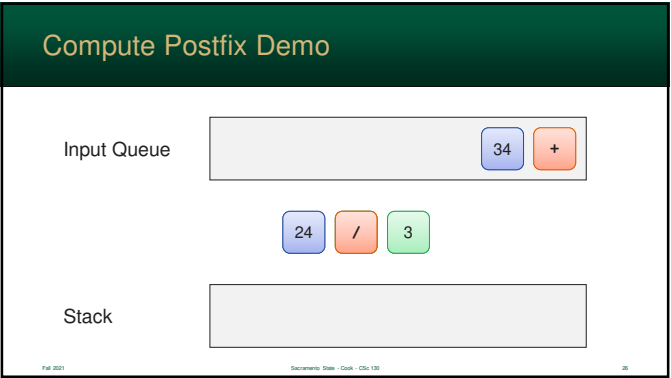
Scenario 3: Stack - CS51B

24

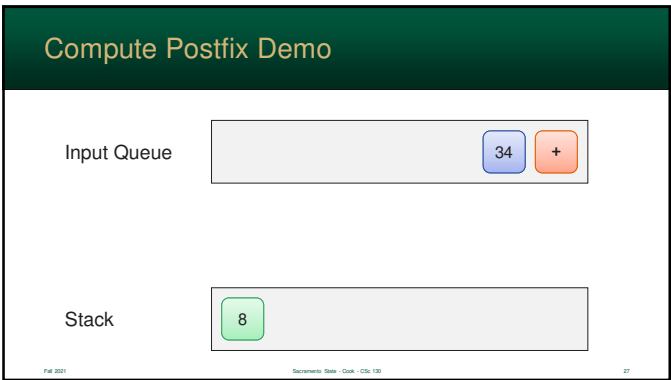
24



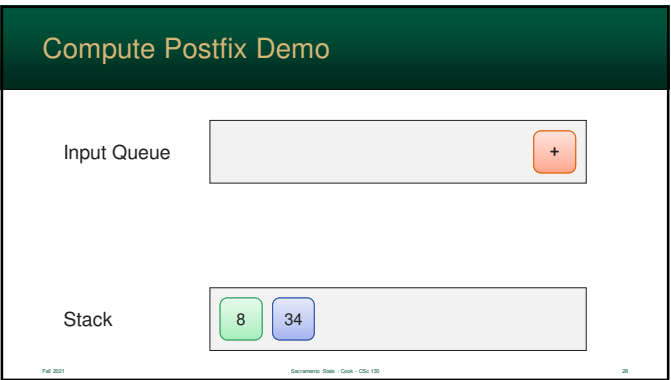
25



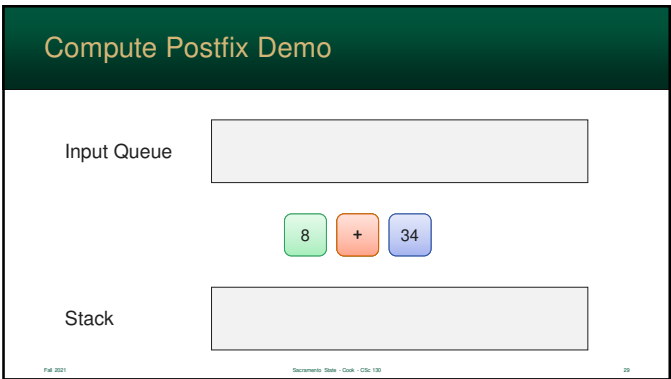
26



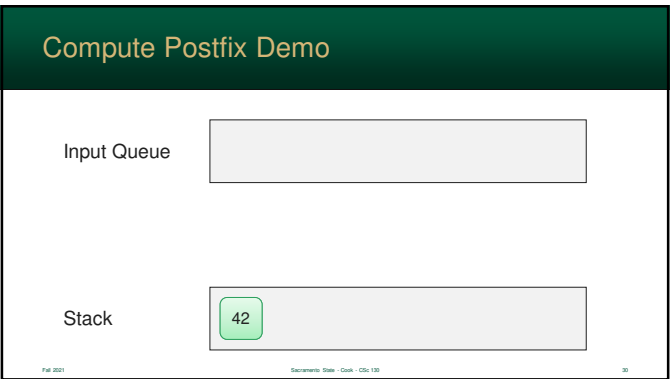
27



28



29



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

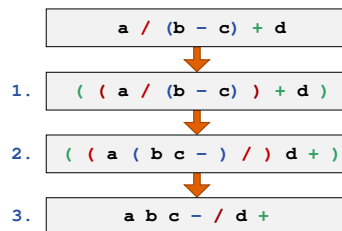
31

Convert Infix to Prefix / Postfix

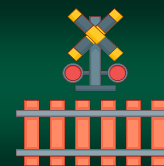
1. Make it a *Fully Parenthesized Expression (FPE)* - one pair of parentheses enclosing each operator and its operands
2. Move the operators to the start (prefix) or end (postfix) of each sub-expression
3. Finally, remove all the parenthesis

32

Infix to Postfix



33



Infix to Postfix Algorithm

Let the computer do the work...

34

Edsger Dijkstra

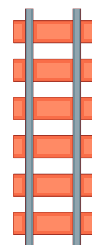
- *Edsger Dijkstra* is a World-famous computer scientist
- He invented a wealth of algorithms
- For his contributions, he received the Turing Award



35

Infix to Postfix Algorithm

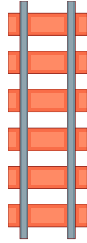
- Infix expressions need to be converted to postfix to be evaluated
- *Dijkstra's Shunting-yard algorithm* performs this task



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



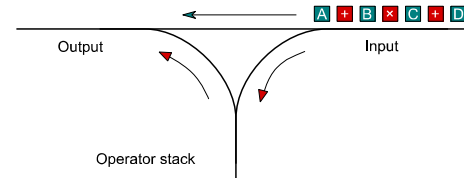
Fall 2021

Shunting-yard Algorithm - Dijkstra - CS161

37

37

Shunting-yard Algorithm



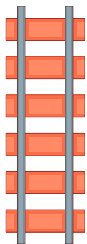
Fall 2021

Shunting-yard Algorithm - Dijkstra - CS161

38

38

Shunting-yard Algorithm



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

Fall 2021

Shunting-yard Algorithm - Dijkstra - CS161

39

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
```

Fall 2021

FPE Shunting-yard Algorithm - Dijkstra - CS161

40

40

FPE Shunting-yard Algorithm

Input Queue

((a * (b + c)) / d)

Operator Stack



Output Queue



Fall 2021

FPE Shunting-yard Algorithm - Dijkstra - CS161

41

41

FPE Shunting-yard Algorithm

Input Queue



Operator Stack



Output Queue

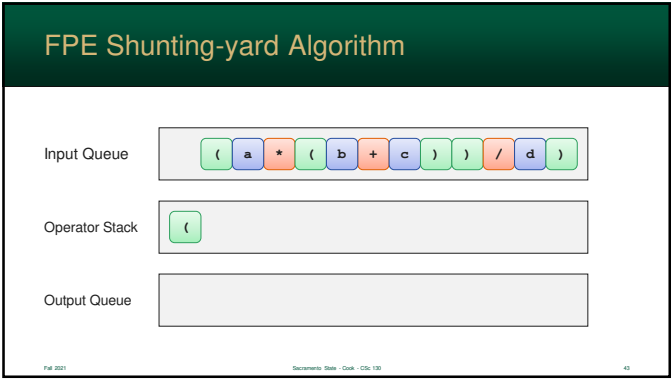


Fall 2021

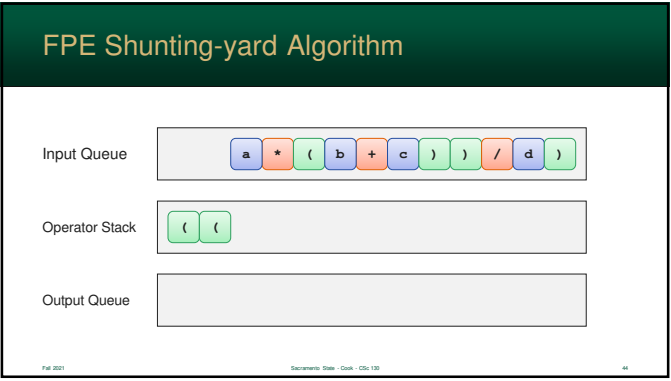
FPE Shunting-yard Algorithm - Dijkstra - CS161

42

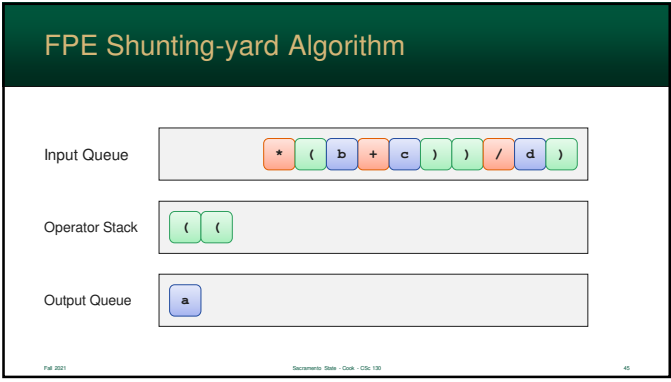
42



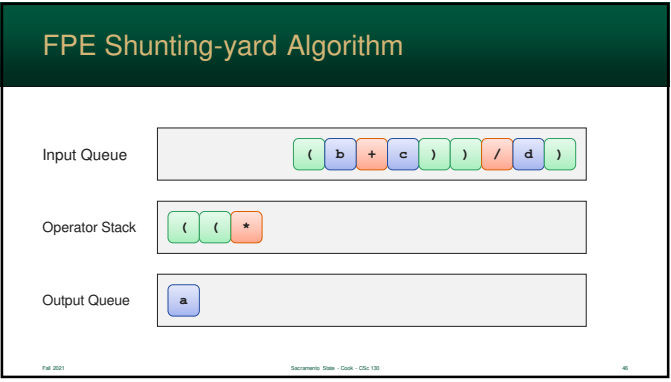
43



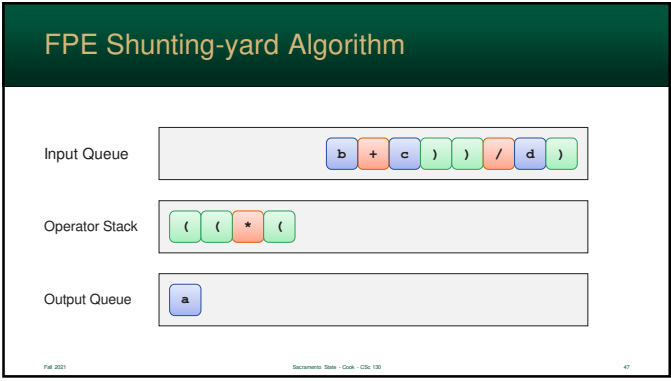
44



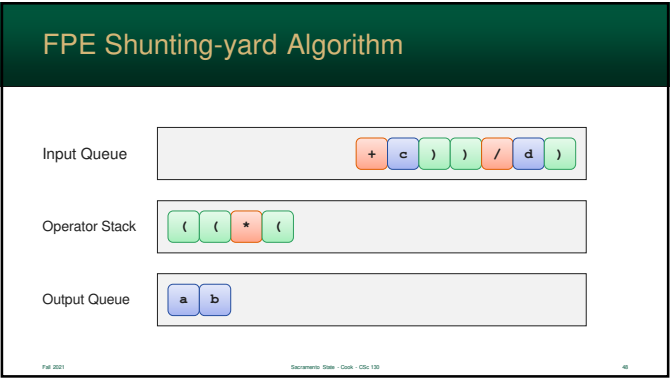
45



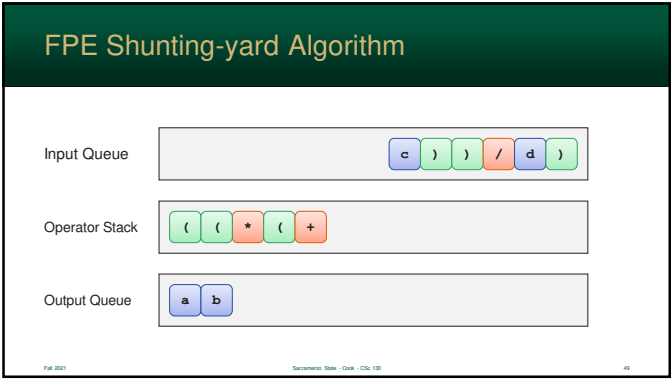
46



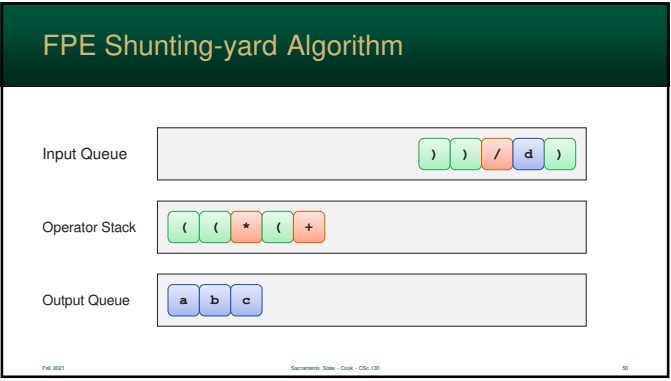
47



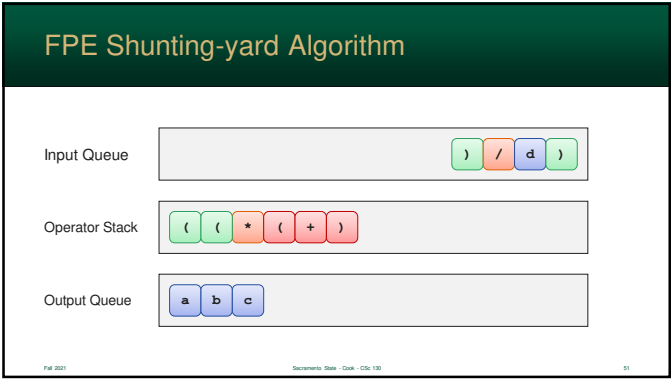
48



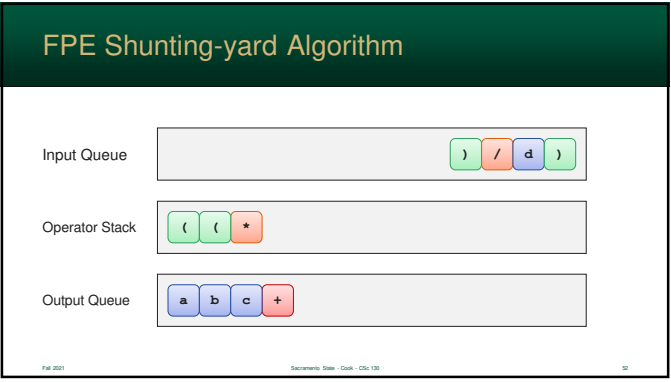
49



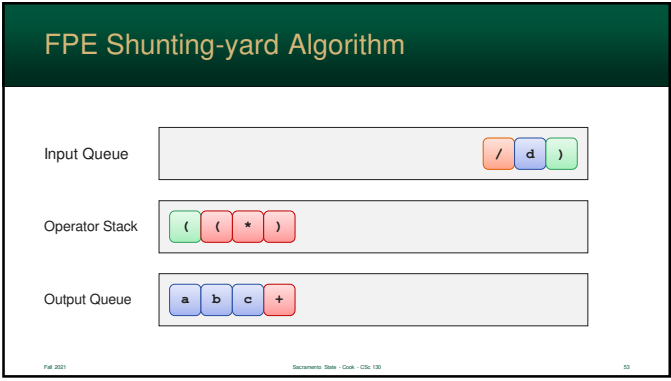
50



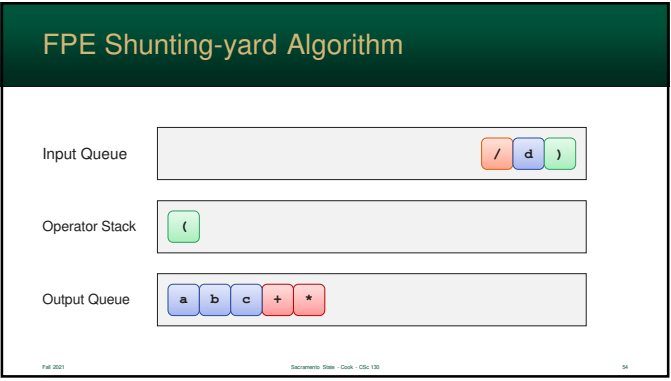
51



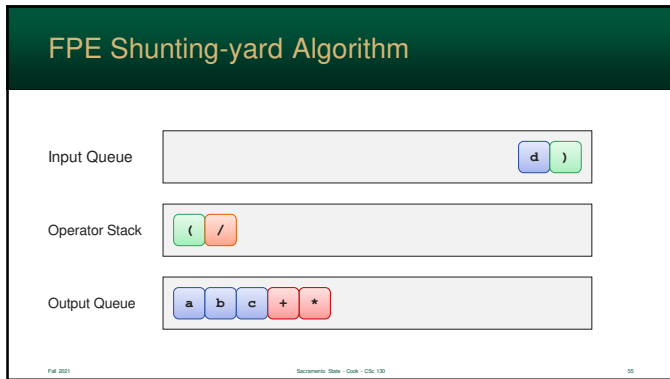
52



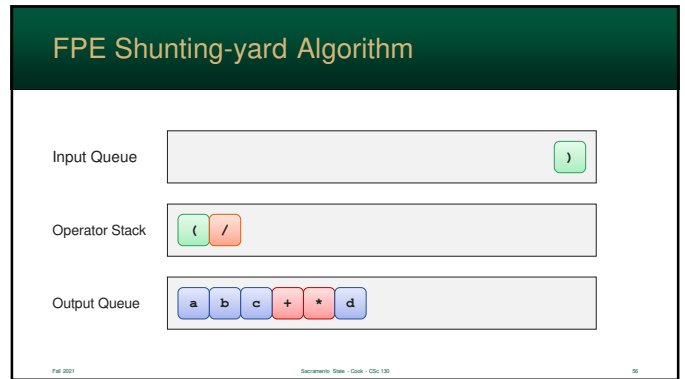
53



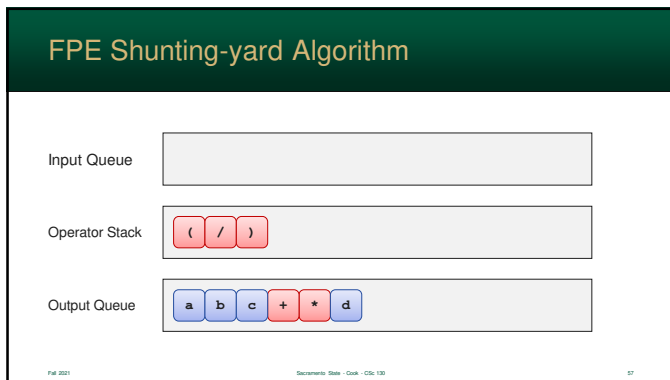
54



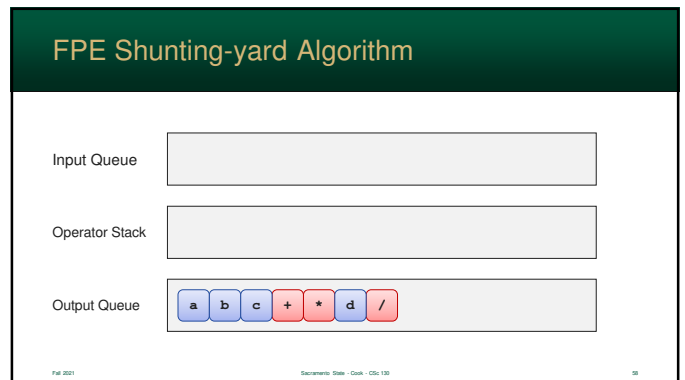
55



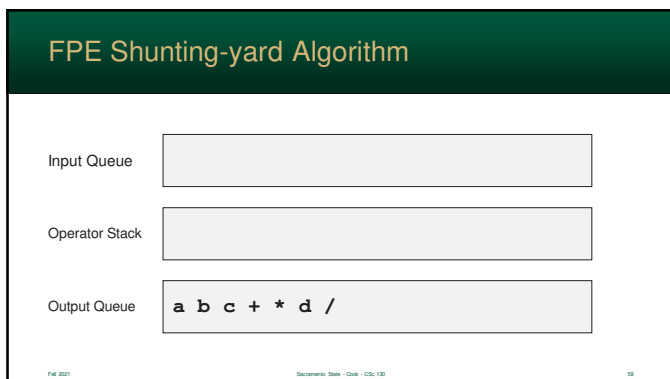
56



57




58



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!

Fall 2021 Sacramento State - CS&E - CS&E 130 60

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 Associativity

Operator	Associativity
+ - * /	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

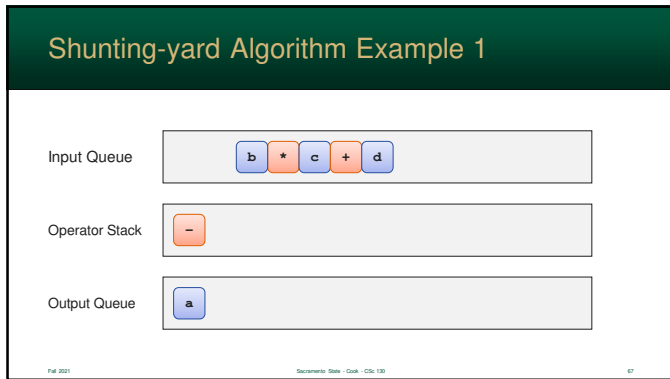
- b * c + d

Operator Stack

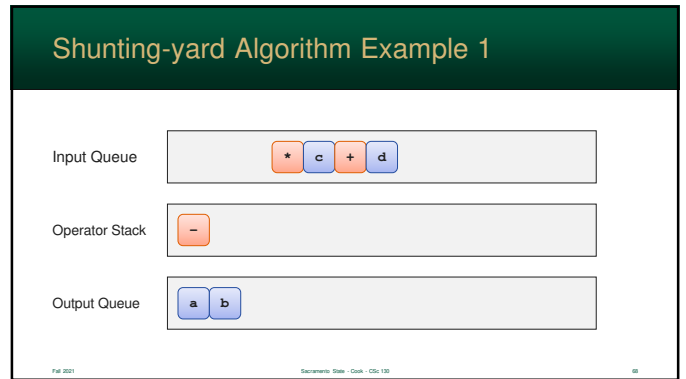
Output Queue

a

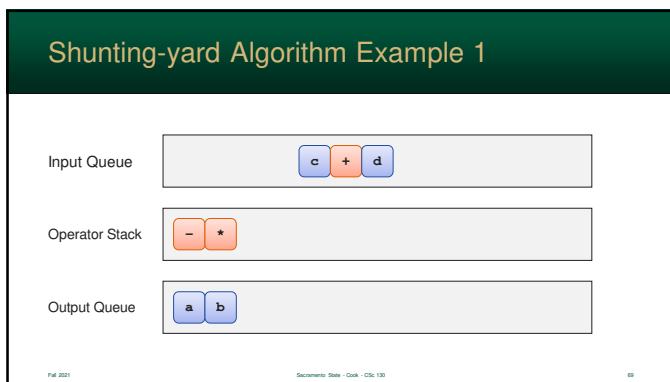
66



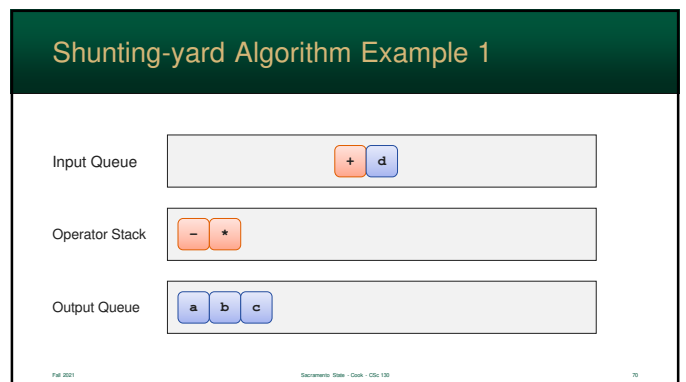
67



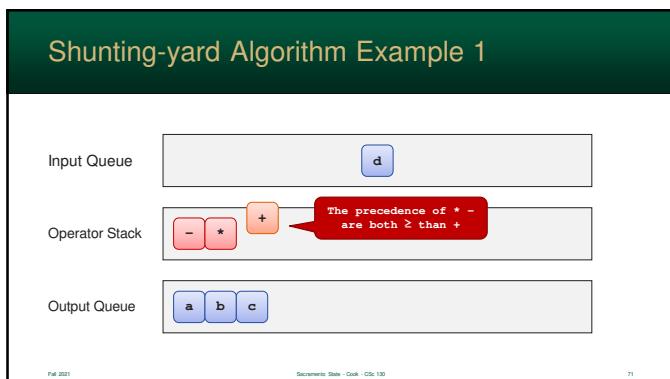
68



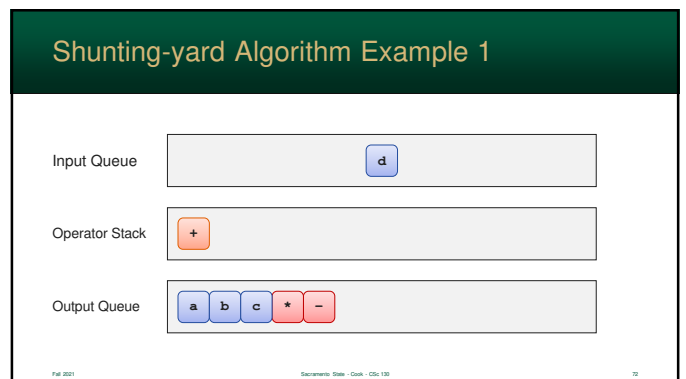
69



70



71



72

Shunting-yard Algorithm Example 1

Input Queue

Operator Stack

+

Output Queue

a

b

c

*

-

d

Remaining stack items pop'd

Fall 2021

Shunting-yard Algorithm - CS61B

73

73

Shunting-yard Algorithm Example 1

Input Queue

Operator Stack

Output Queue

a

b

c

*

-

d

+

Fall 2021

Shunting-yard Algorithm - CS61B

74

74

Shunting-yard Algorithm Example 1

Input Queue

Operator Stack

Output Queue

a

b

c

*

-

d

+

Fall 2021

Shunting-yard Algorithm - CS61B

75

75

Shunting-yard Algorithm Example 2

Input Queue

a

+

(

b

-

c

*

d

)

/

e

-

f

Operator Stack

Output Queue

Fall 2021

Shunting-yard Algorithm - CS61B

76

76

Shunting-yard Algorithm Example 2

Input Queue

a

+

(

b

-

c

*

d

)

/

e

-

f

Operator Stack

Output Queue

Fall 2021

Shunting-yard Algorithm - CS61B

77

77

Shunting-yard Algorithm Example 2

Input Queue

+

(

b

-

c

*

d

)

/

e

-

f

Operator Stack

Output Queue

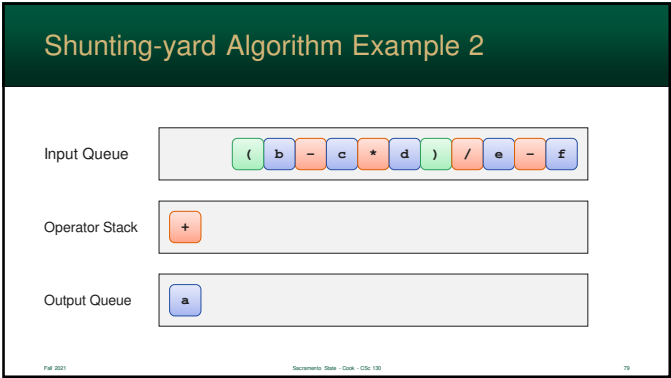
a

Fall 2021

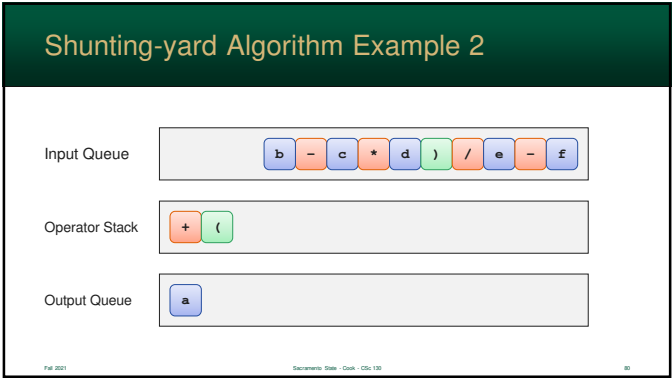
Shunting-yard Algorithm - CS61B

78

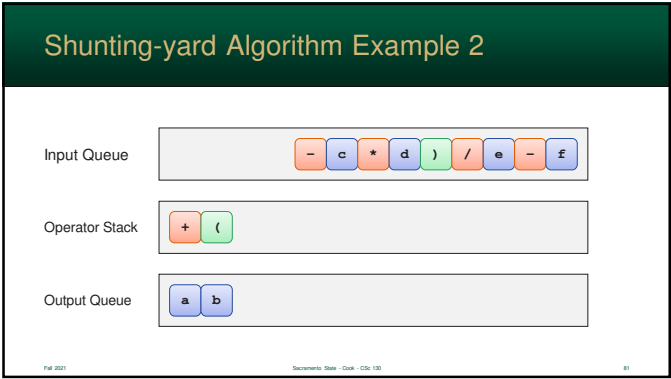
78



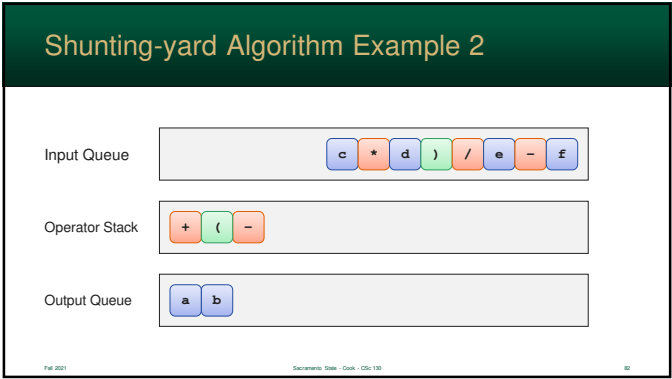
79



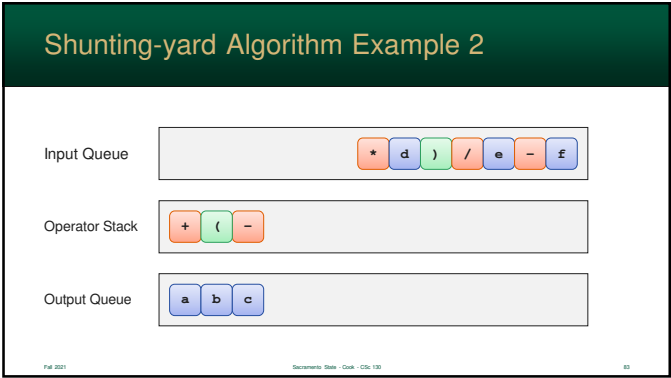
80



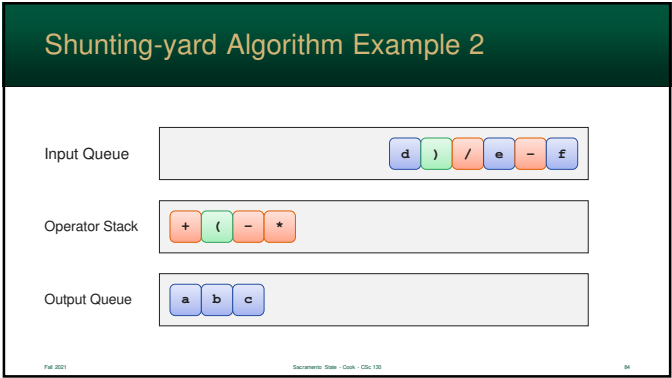
81



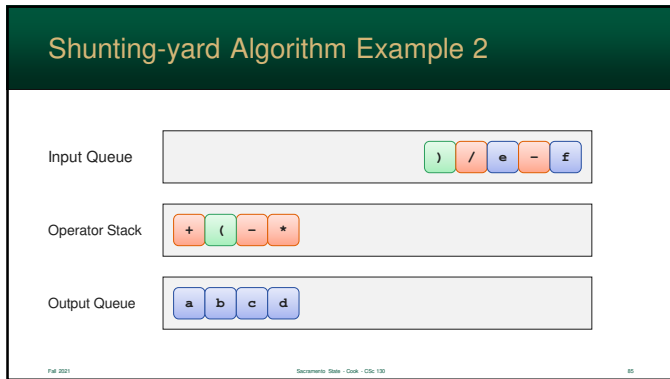
82



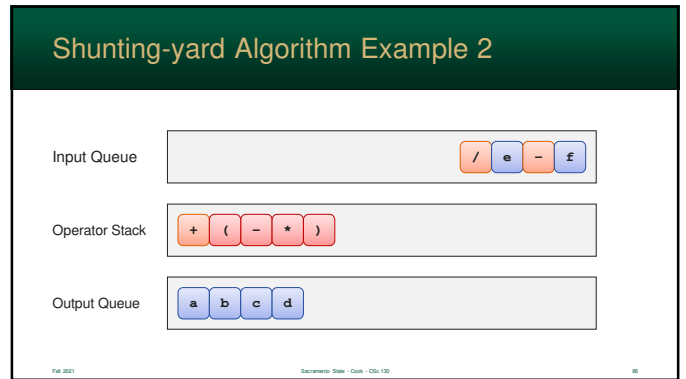
83



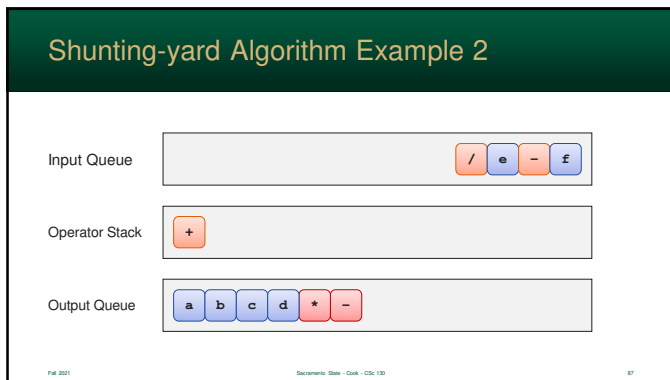
84



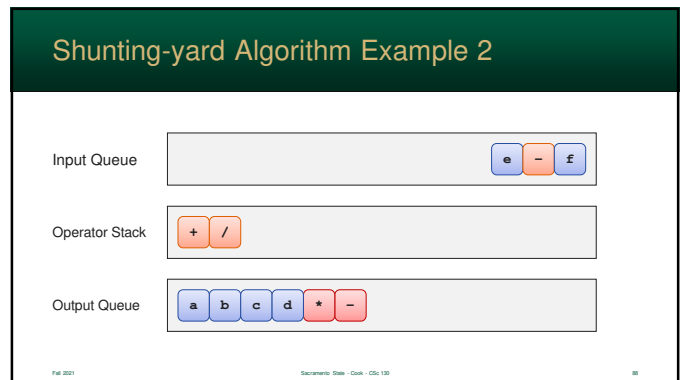
85



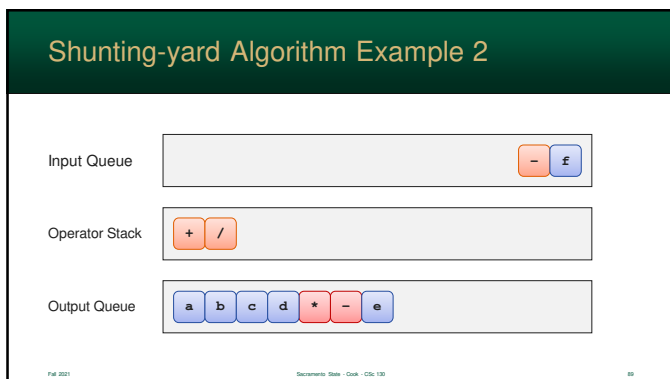
86



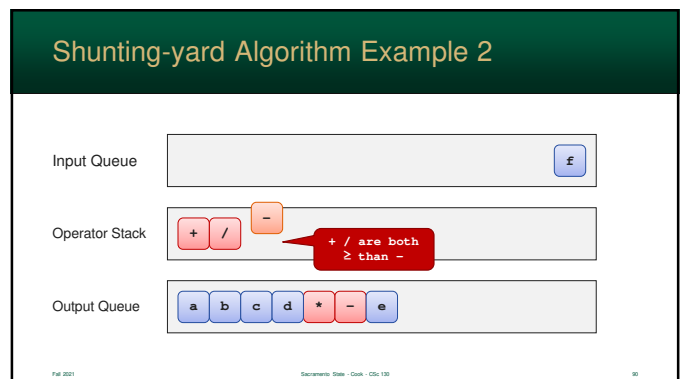
87



88



89



90

Shunting-yard Algorithm Example 2

Input Queue: f

Operator Stack: -

Output Queue: a b c d * - e / +

91

Shunting-yard Algorithm Example 2

Input Queue:

Operator Stack: - Remaining stack items pop'd

Output Queue: a b c d * - e / + f

92

Shunting-yard Algorithm Example 2

Input Queue:

Operator Stack:

Output Queue: a b c d * - e / + f -

93

Shunting-yard Algorithm Example 2

Input Queue:

Operator Stack:

Output Queue: a b c d * - e / + f -

94

Testing Our Result

$a + (b - c * d) / e - f$

1. $((a + ((b - (c * d)) / e)) - f)$

2. $((a ((b (c d *) -) e /) +) f -)$

3. $a b c d * - e / + f -$

95