ECE 264 Spring 2023

Advanced C Programming

Aravind Machiry
Purdue University

Homework 11-12

Shuffle Cards

yunglu@purdue.edu

Riffle Shuffling

- Many card games need to shuffle so that players do not know which card may appear next.
- Riffle shuffling is a popular method for shuffling

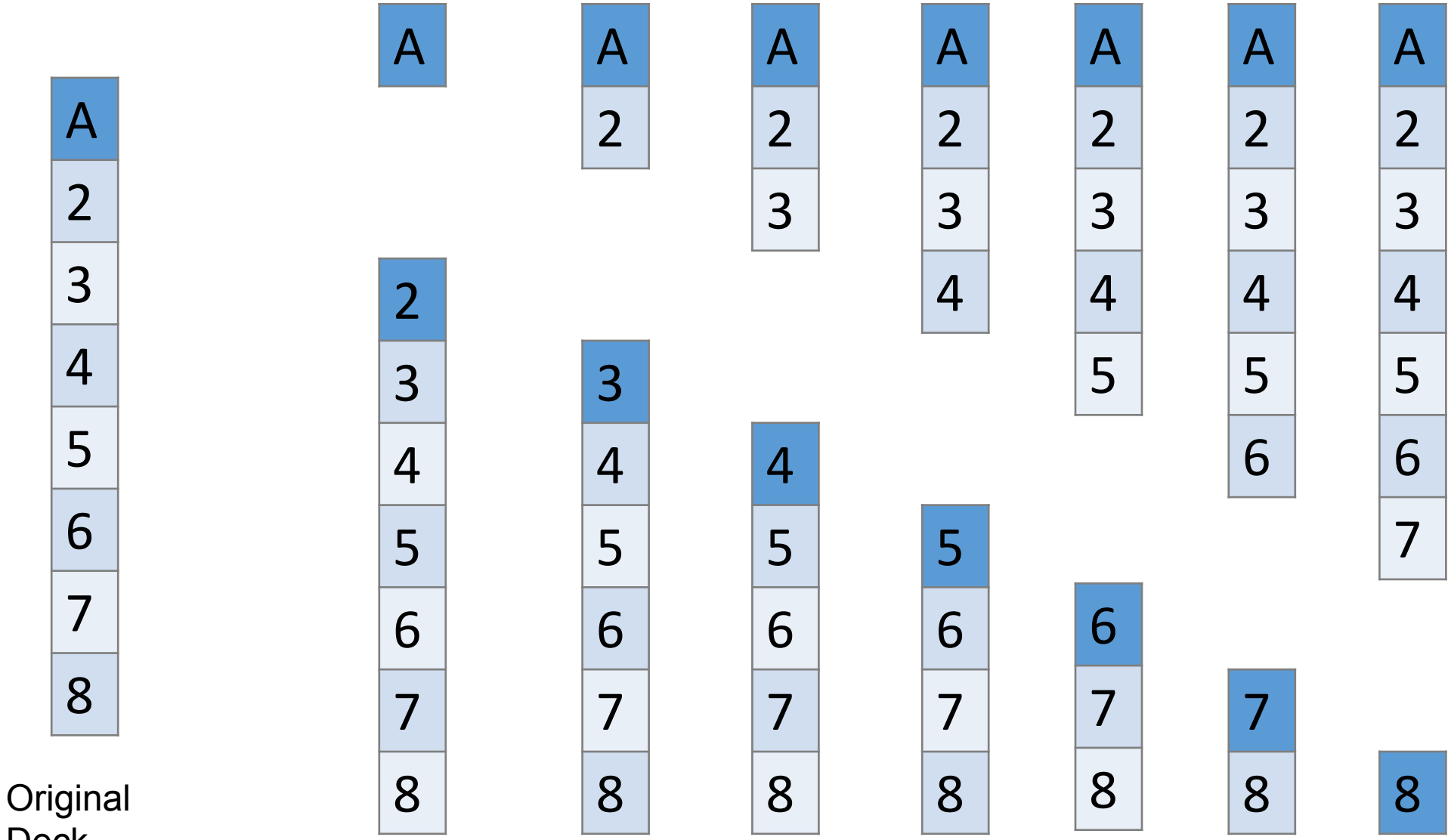
1. divide a deck of cards into two parts
2. hold the parts by right and left hands
3. interleave the cards

yunglu@purdue.edu

Homework 11-12

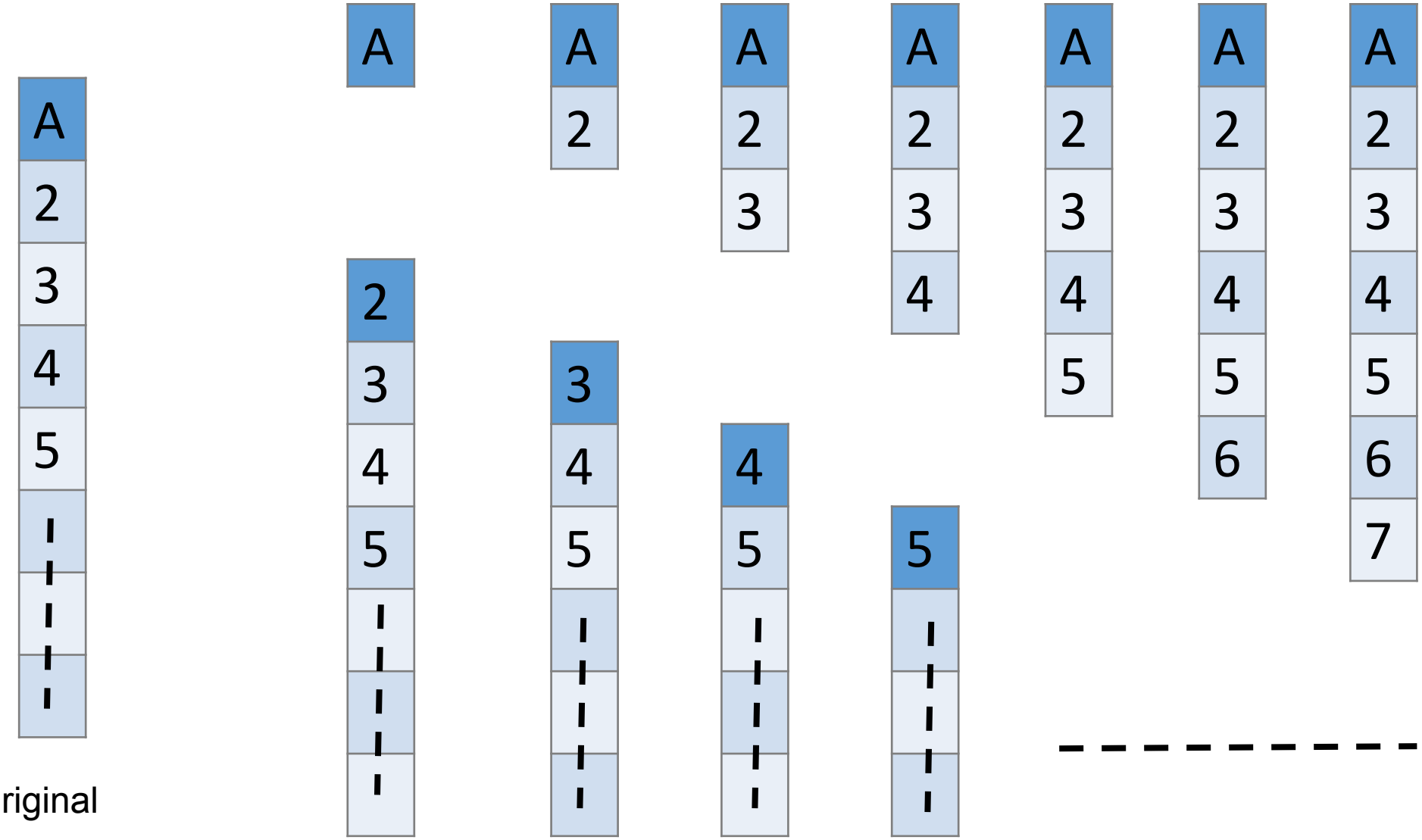
- Homework 11: Shuffle Once
- Homework 12: Shuffle Multiple Times
- The homework considers all possible scenarios under these restrictions:
  - Each (of the two) part has at least one card
  - If a card is above another card in one part, the order must be preserved in the interleave result

yunglu@purdue.edu



Different ways to divide the original deck into two parts

yunglu@purdue.edu



Original Deck ***n*** cards

Different ways to divide the original deck into two parts

The first part has ***1, 2, 3, …., n – 1*** cards

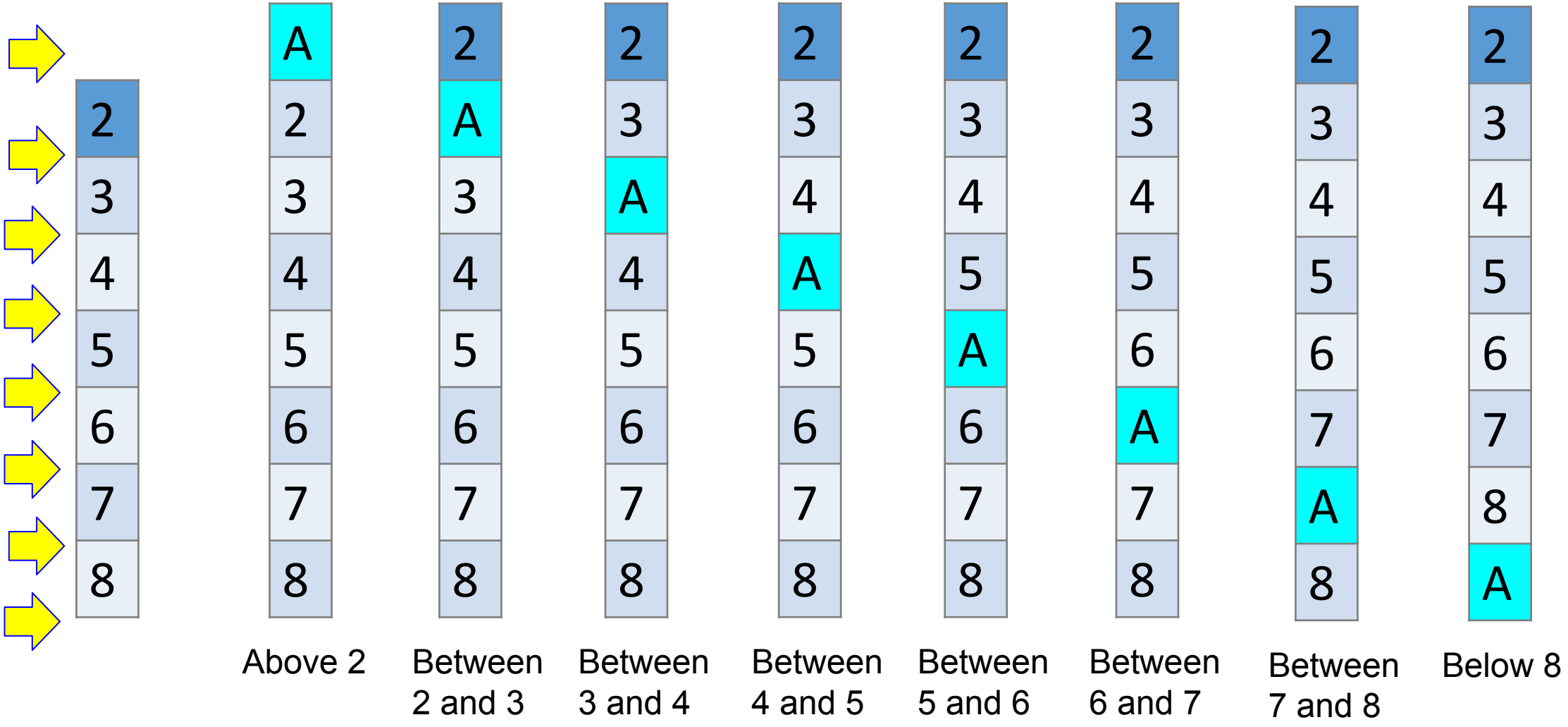
n – 1 ways to divide the cards

yunglu@purdue.edu

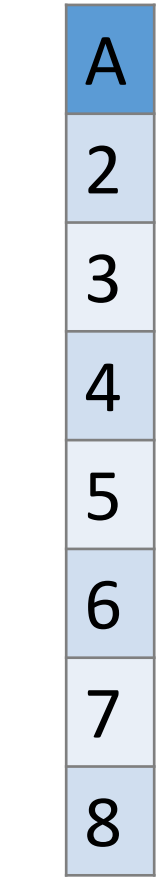
A

Where can A be placed?

The order 2, 3, 4, 5, 6, 7, 8 **will not change**



yunglu@purdue.edu



Original Deck

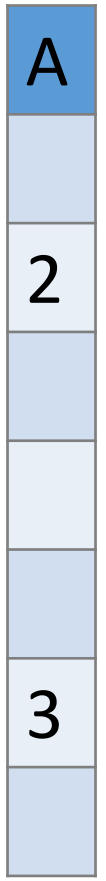
left hand

right hand

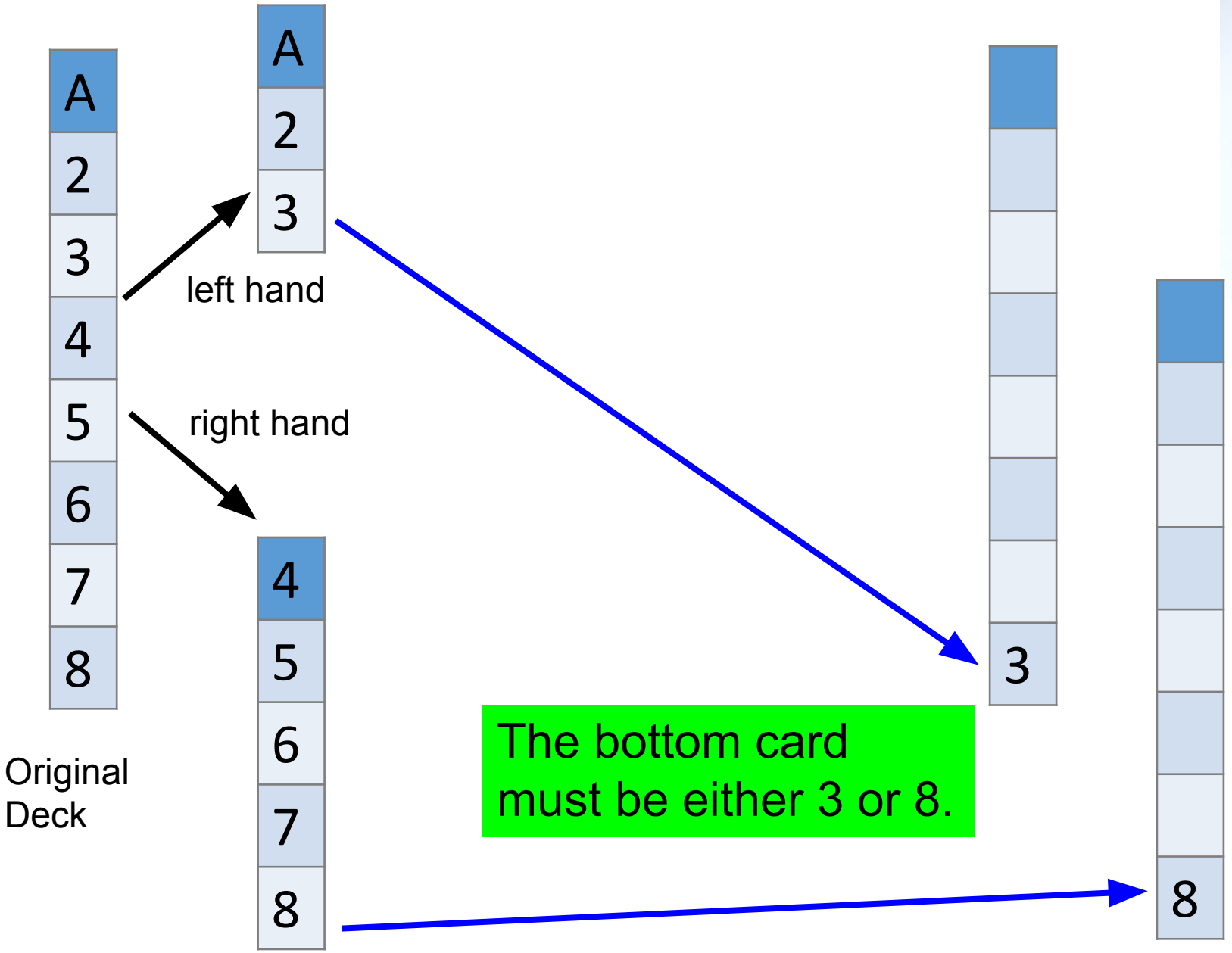
mix the cards

The order A, 2, 3 **will not change**

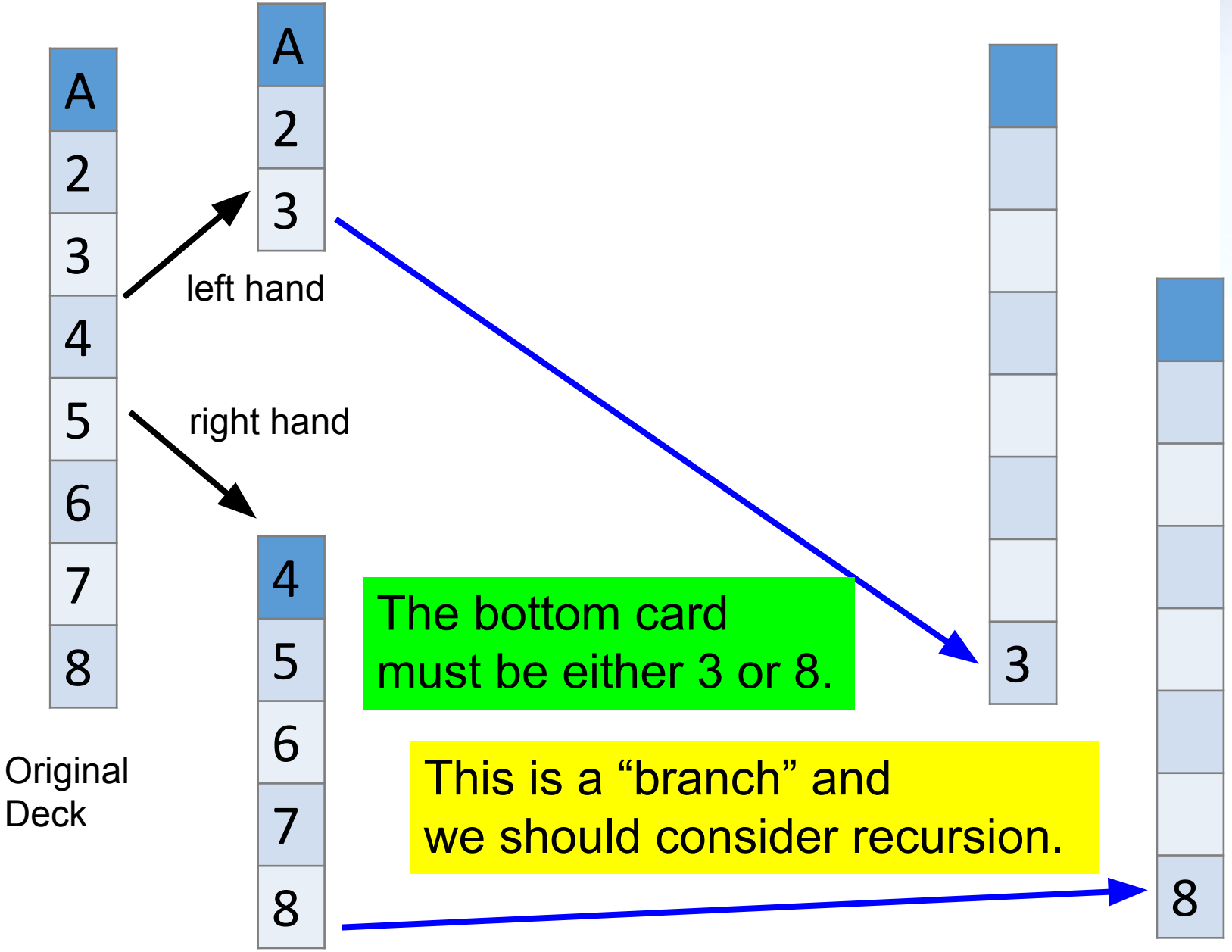
The order 4, 5, 6, 7, 8 **will not change**



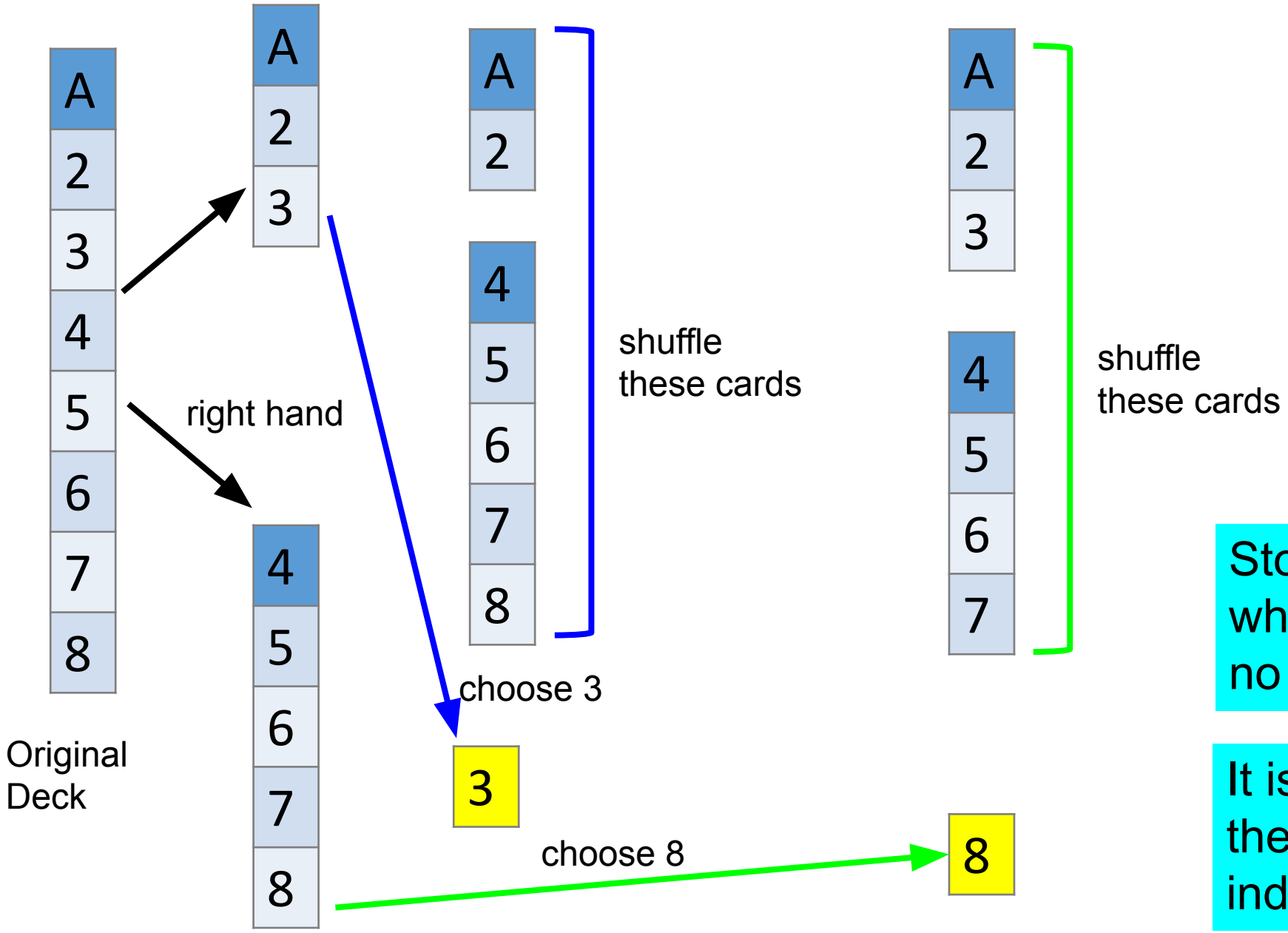
yunglu@purdue.edu



yunglu@purdue.edu



yunglu@purdue.edu



Stop condition: when one hand has no card left

It is easier to start from the bottom. The top's index must be zero

yunglu@purdue.edu

Homework 13-14

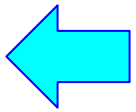
Arithmetic Operations

yunglu@purdue.edu

HW13 and HW 14

- Use linked list to handle arithmetic expressions
- Understand infix and postfix expressions.
- Convert infix to postfix expressions.
- Use linked list as stack

```
Node * List_insert(Node * h, int v)
{
  Node * p = Node_construct(v);
  p -> next = h;
  return p;
}
```



The new node (p) is
in front of the previous node

yunglu@purdue.edu

Infix and postfix expressions (for binary operations)

- Unary operations: an operator (such as negation) needs only one operand. Binary operations: an operator (such as +, -, *) needs two operands
- infix: operand operation operand, e.g. 4 + 9
- postfix: operand operand operation, e.g. 4 9 +
- postfix: easier for computers, no need of parenthesis
  - 3 5 * 6 + (postfix) means 3 * 5 + 6 = 15 + 6 = 21
  - 3 5 + 6 * means (3 + 5) * 6 = 8 * 6 = 48
  - 3 5 6 + * means 3 * (5 + 6) = 3 * 11 = 33

yunglu@purdue.edu

C Operator Precedence

The following table lists the precedence and associativity of C operators. Operators are listed top to bottom, in descending precedence.

Precedence	Operator	Description	Associativity

1	`++ --`
	`()`
	`[]`
	`.`
	`->`
	`(`*type*`){`*list*`}`
		Suffix/postfix increment and decrement
		Function call
		Array subscripting
		Structure and union member access
		Structure and union member access through pointer
		Compound literal(C99)
			Left-to-right

2	`++ --`
	`+ -`
	`! ~`
	`(`*type*`)`
	`*`
	`&`
	`sizeof`
	`_Alignof`
		Prefix increment and decrement[note 1]
		Unary plus and minus
		Logical NOT and bitwise NOT
		Cast
		Indirection (dereference)
		Address-of
		Size-of[note 2]
		Alignment requirement(C11)
			Right-to-left

3	`* / %`	Multiplication, division, and remainder	Left-to-right
4	`+ -`	Addition and subtraction	
5	`<< >>`	Bitwise left shift and right shift	
6	`< <=` `> >=`	For relational operators < and ≤ respectively For relational operators > and ≥ respectively	
7	`== !=`	For relational = and ≠ respectively	
8	`&`	Bitwise AND	
9	`^`	Bitwise XOR (exclusive or)	
10	`|`	Bitwise OR (inclusive or)	
11	`&&`	Logical AND	
12	`||`	Logical OR	
13	`?:`	Ternary conditional[note 3]	Right-to-left

14[note 4]	`=`
	`+= -=`
	`*= /= %=`
	`<<= >>=`
	`&= ^= |=`
		Simple assignment
		Assignment by sum and difference
		Assignment by product, quotient, and remainder
		Assignment by bitwise left shift and right shift
		Assignment by bitwise AND, XOR, and OR	

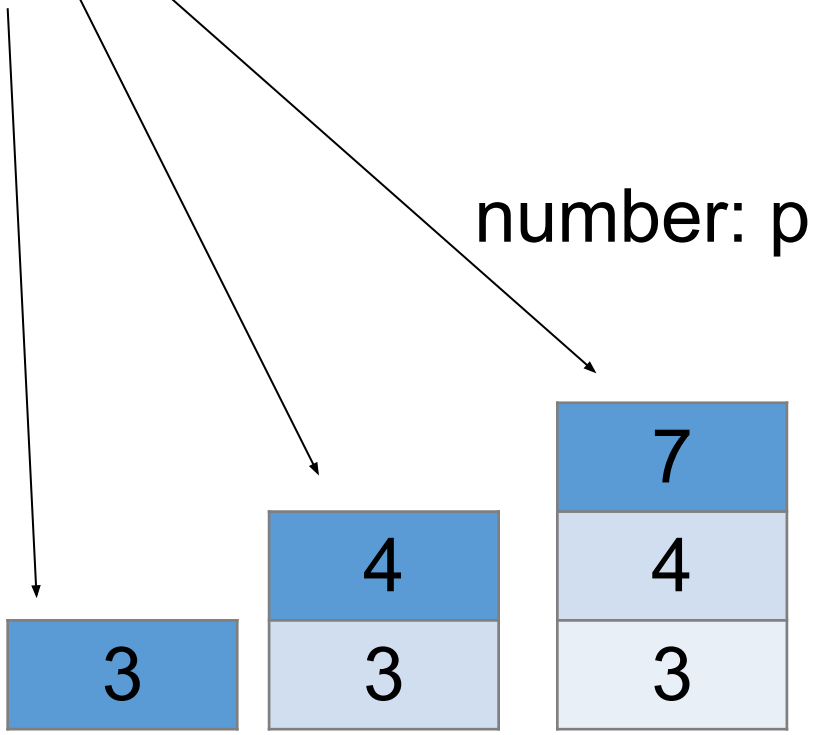
15	`,`	Comma	Left-to-right

Evaluate Postfix using a stack

3 4 7 * + 2 -

A valid must start with a number

number: push to stack

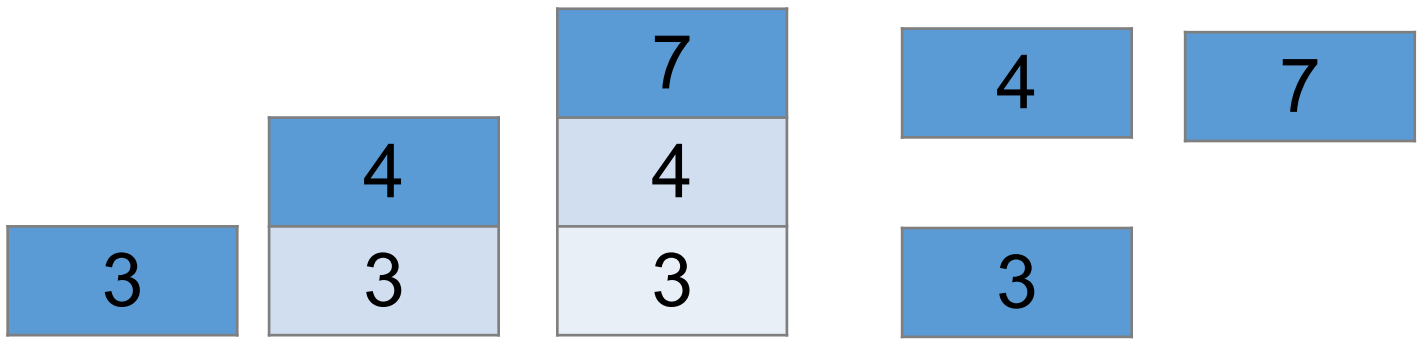


yunglu@purdue.edu

Evaluate Postfix using a stack

3 4 7 * + 2 -

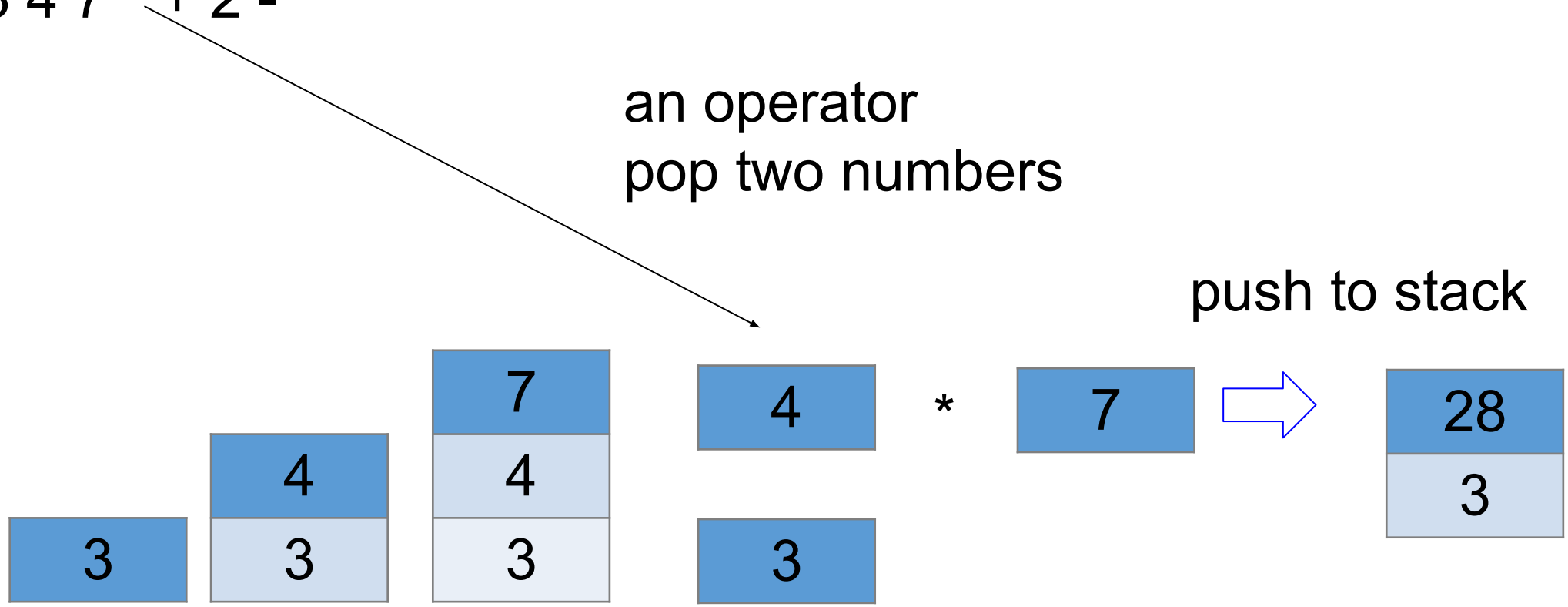
an operator
pop two numbers



yunglu@purdue.edu

Evaluate Postfix using a stack

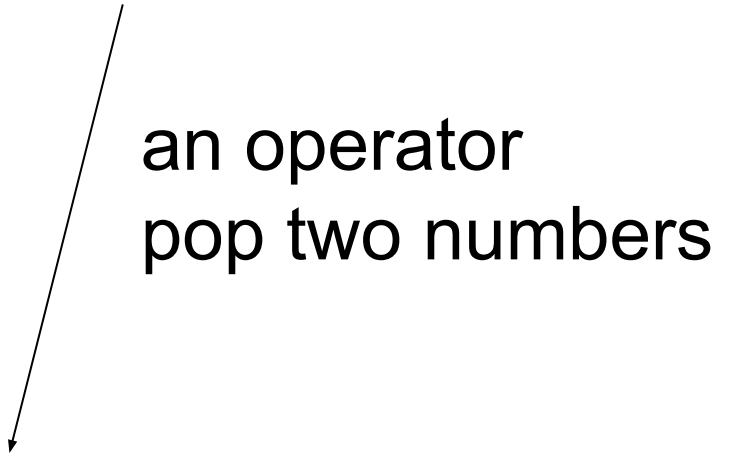
3 4 7 * + 2 -

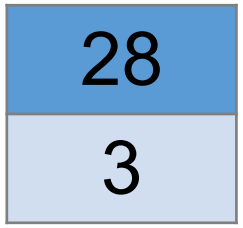


yunglu@purdue.edu

Evaluate Postfix using a stack

3 4 7 * + 2 -

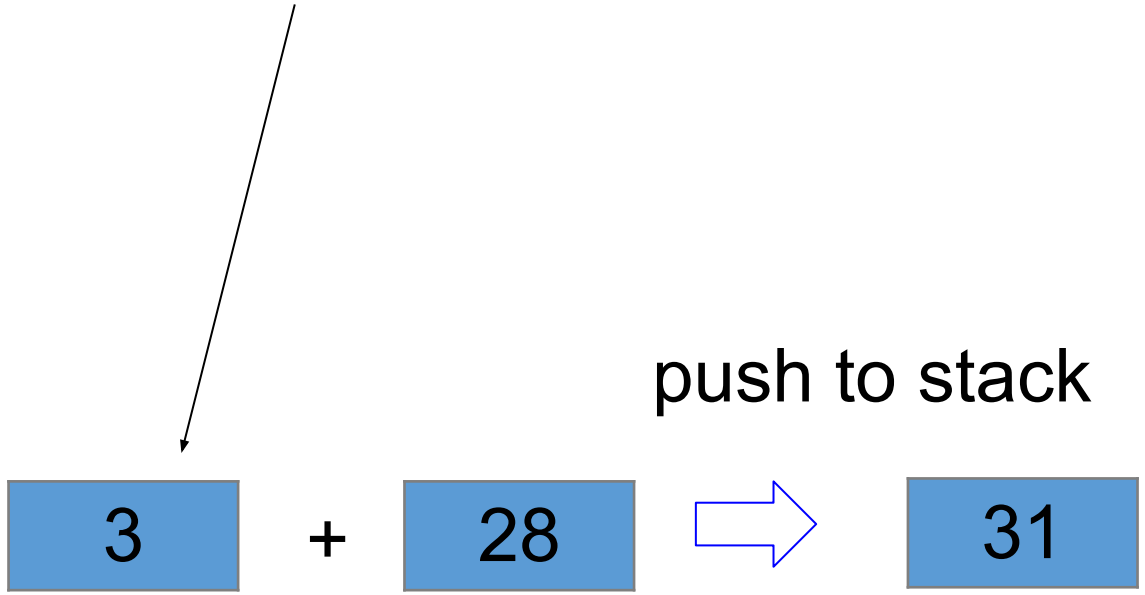




yunglu@purdue.edu

Evaluate Postfix using a stack

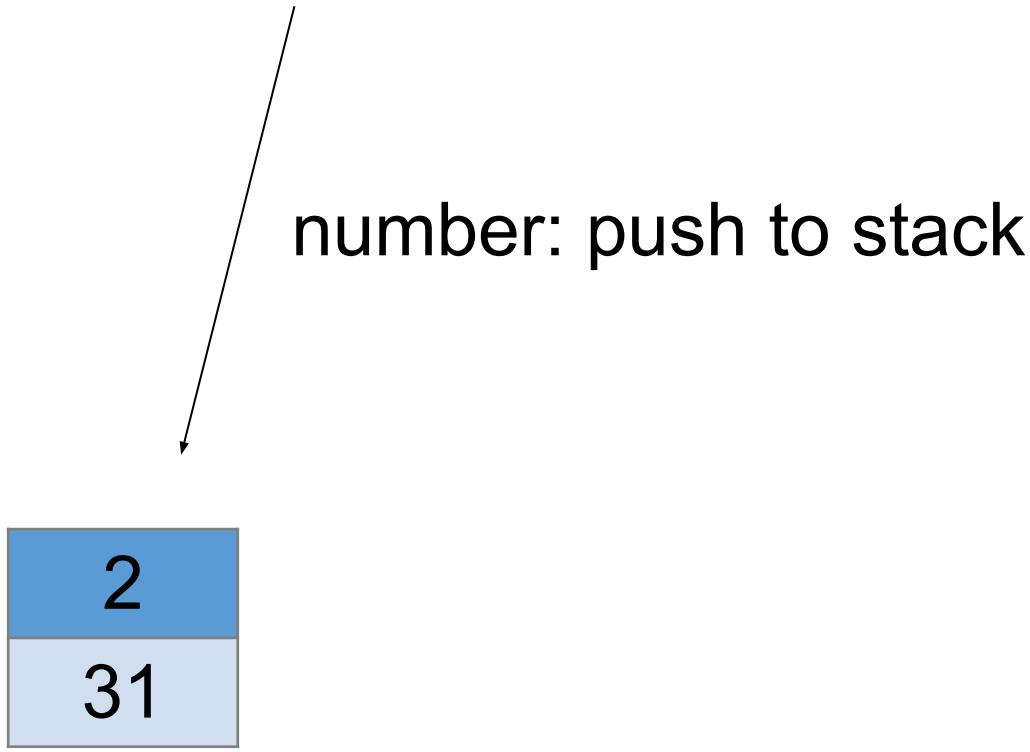
3 4 7 * + 2 -



yunglu@purdue.edu

Evaluate Postfix using a stack

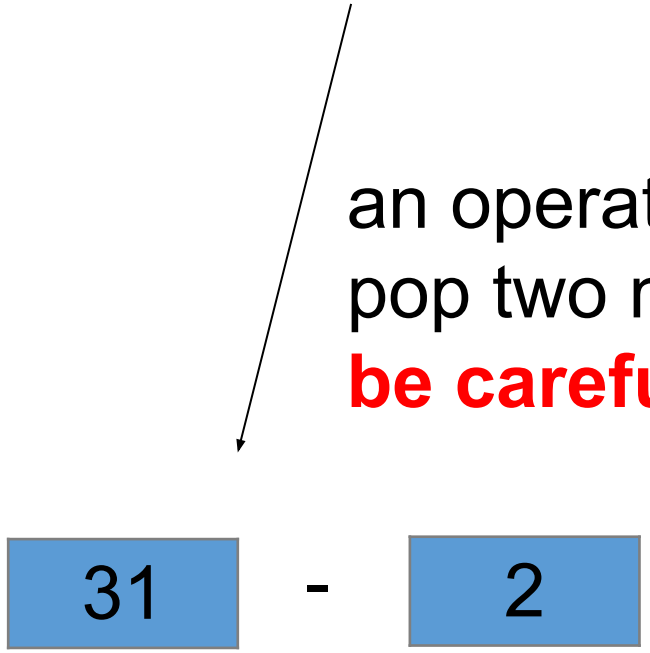
3 4 7 * + 2 -



yunglu@purdue.edu

Evaluate Postfix using a stack

3 4 7 * + 2 -



An operation is ***commutative*** if the order can be changed

a + b = b + a

a * b = b * a

a – b is different from b – a

yunglu@purdue.edu

Evaluate Postfix using a stack

3 4 7 * + 2 -

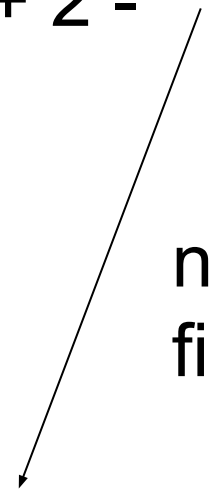
an operator
pop two numbers
be careful of the order

push to stack

yunglu@purdue.edu

Evaluate Postfix using a stack

3 4 7 * + 2 -



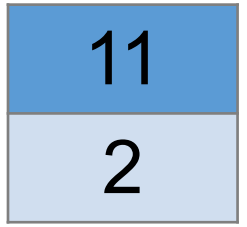
nothing left in the input
final result: 29

29

yunglu@purdue.edu

Handle Errors

- The stack must have two (or more) numbers when an operator is seen. 7 + is invalid. * (no number) is invalid
- After finishing the number, there should be exactly one number left. 2 8 3 + is invalid because the result is



yunglu@purdue.edu

Convert infix to postfix

Convert 6 * 4 + 3

Stack: first-in, last-out
Queue: first-in, first-out

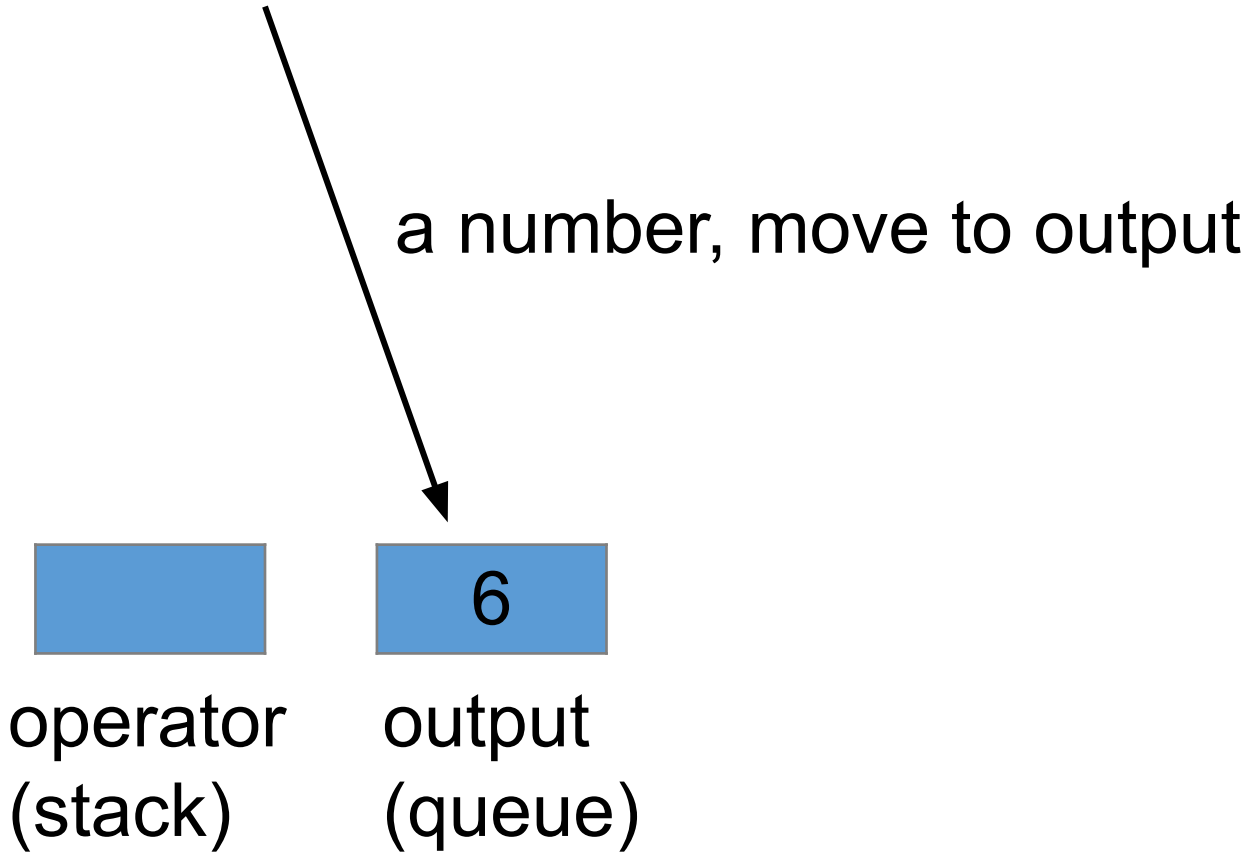
operator (stack)

output (queue)

yunglu@purdue.edu

Convert infix to postfix

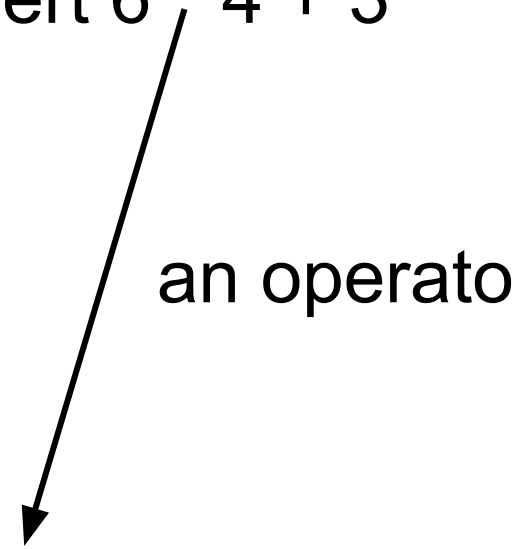
Convert 6 * 4 + 3



yunglu@purdue.edu

Convert infix to postfix

Convert 6 * 4 + 3



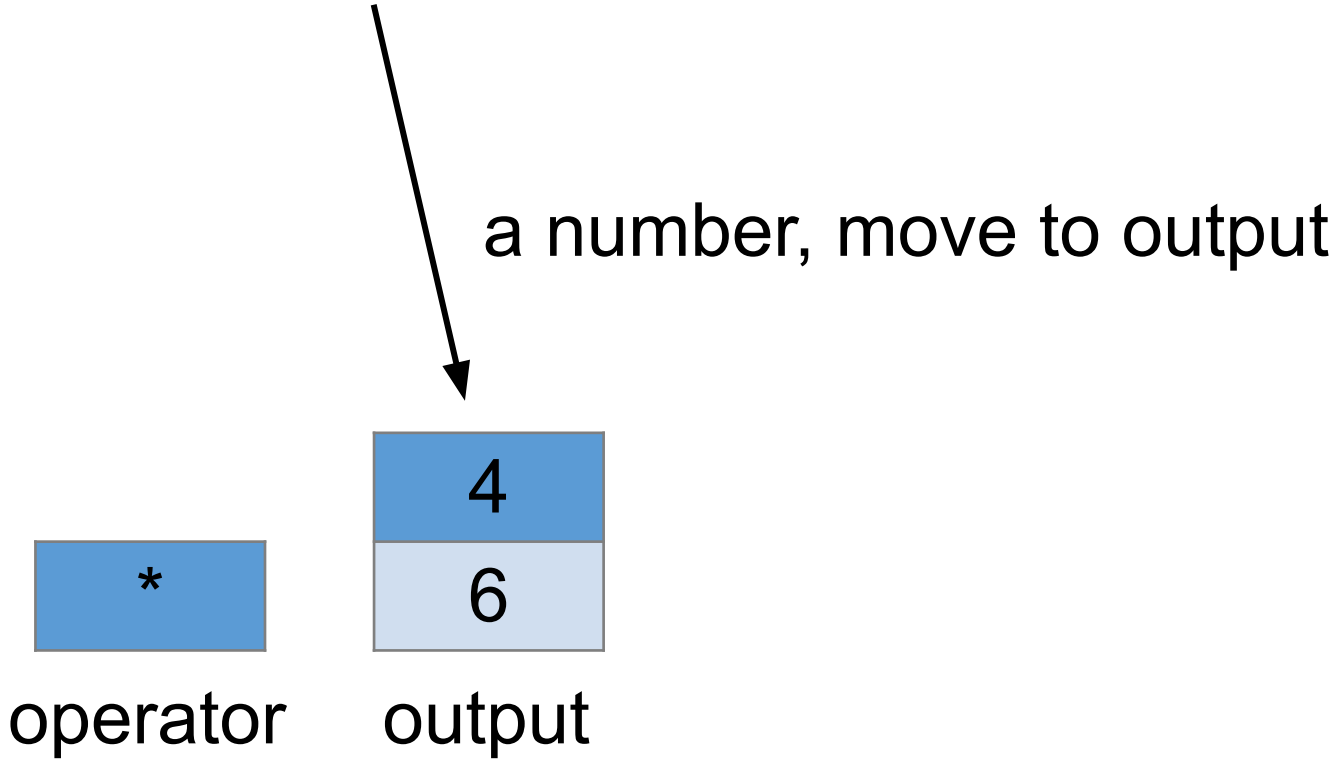
an operator, move to operator stack

operator output

yunglu@purdue.edu

Convert infix to postfix

Convert 6 * 4 + 3



yunglu@purdue.edu

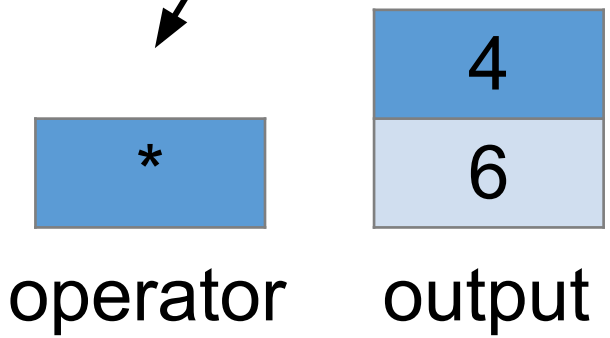
Convert infix to postfix

Convert 6 * 4 + 3

an operator, compare its precedence with *

* has higher precedence, pop and move to output

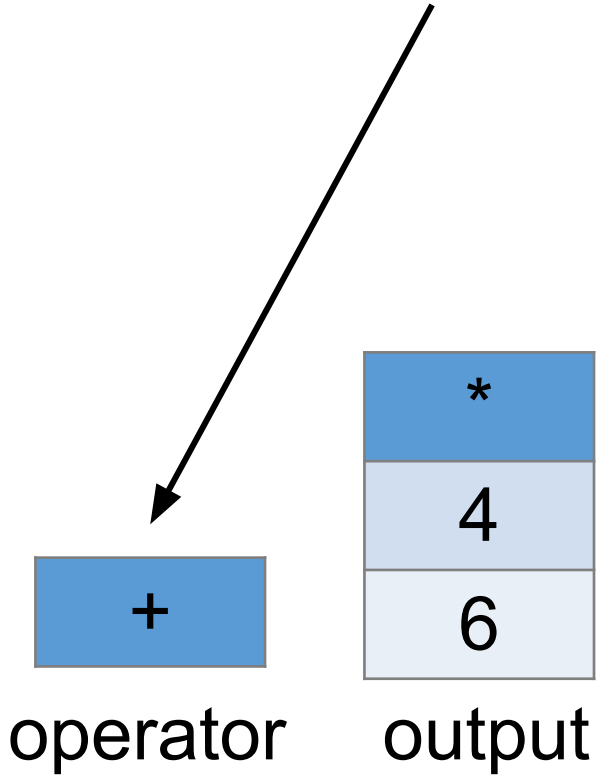
push + to operator



yunglu@purdue.edu

Convert infix to postfix

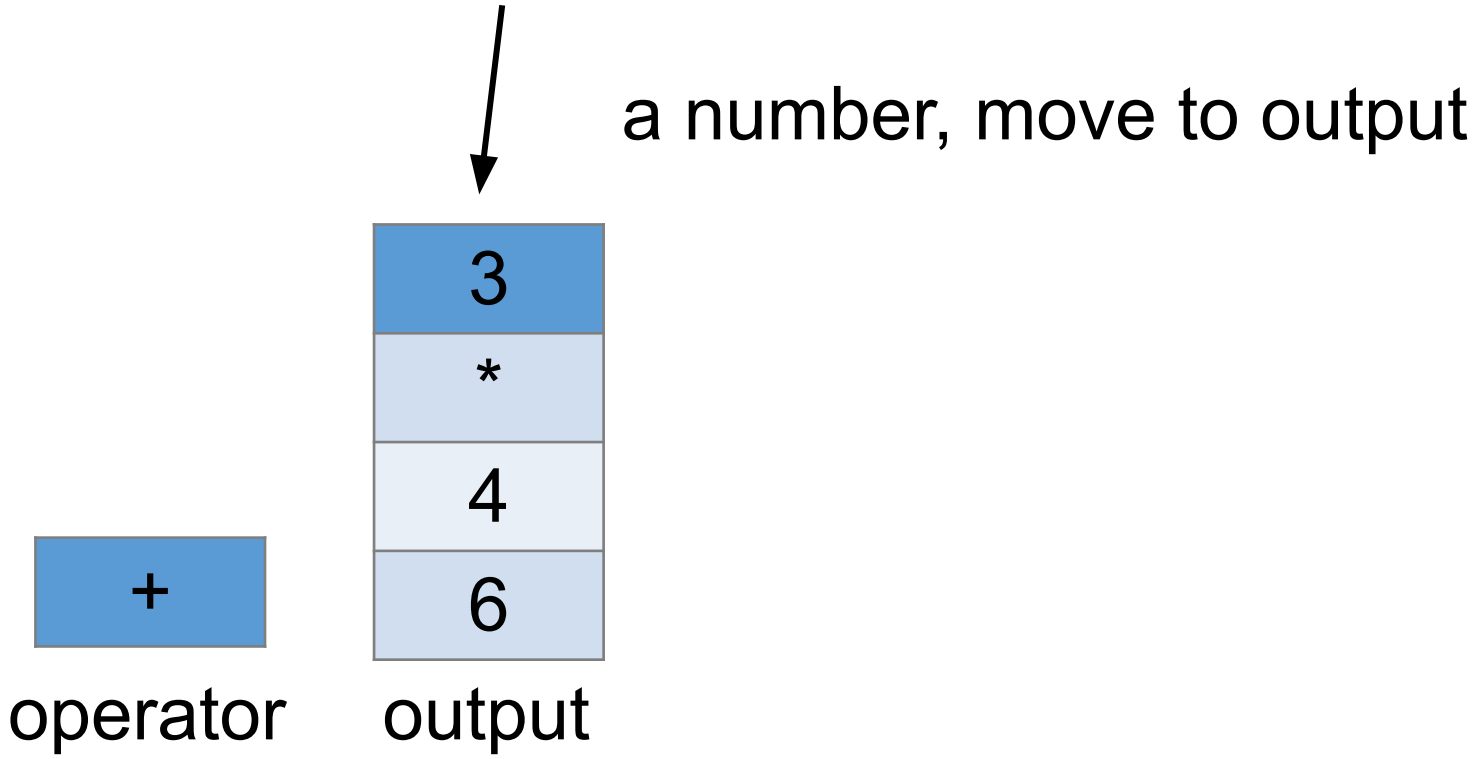
Convert 6 * 4 + 3



yunglu@purdue.edu

Convert infix to postfix

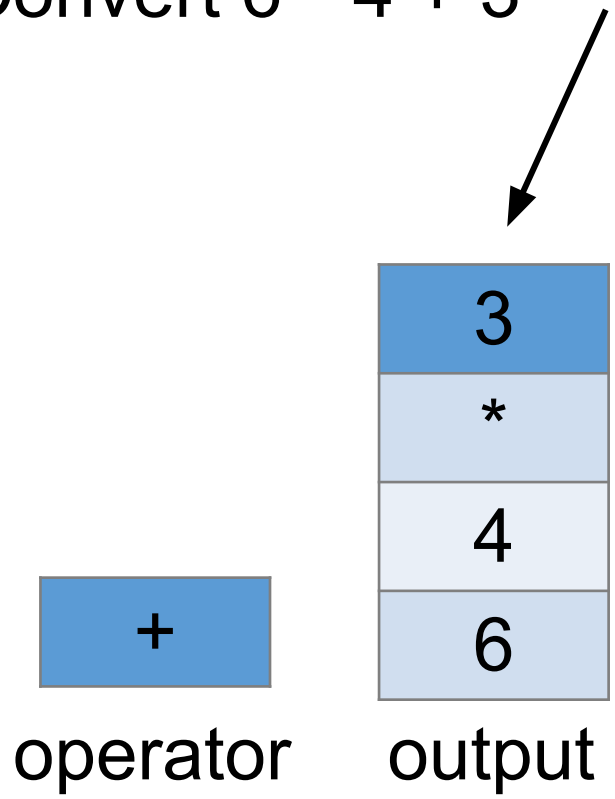
Convert 6 * 4 + 3



yunglu@purdue.edu

Convert infix to postfix

Convert 6 * 4 + 3

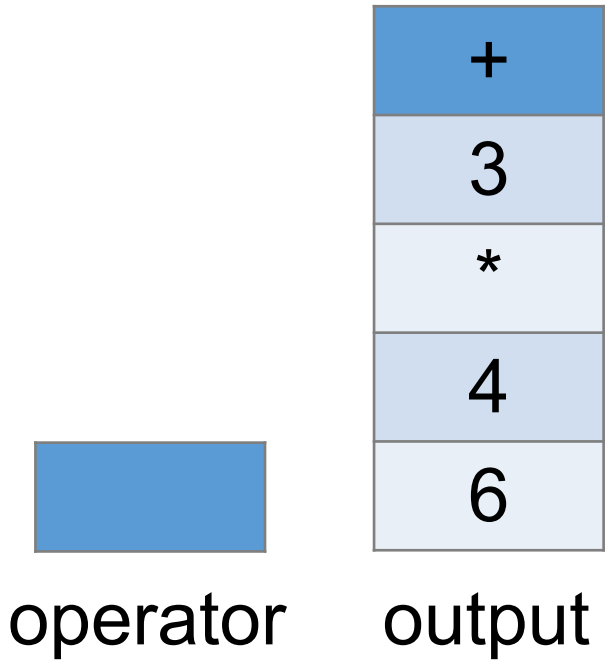


no more input
pop operator
push to output

yunglu@purdue.edu

Convert infix to postfix

Convert 6 * 4 + 3



postfix: 6 4 * 3 +

yunglu@purdue.edu

Convert infix to postfix

Convert 6 + 4 * 3 (different from the previous: **+ * exchanged**)

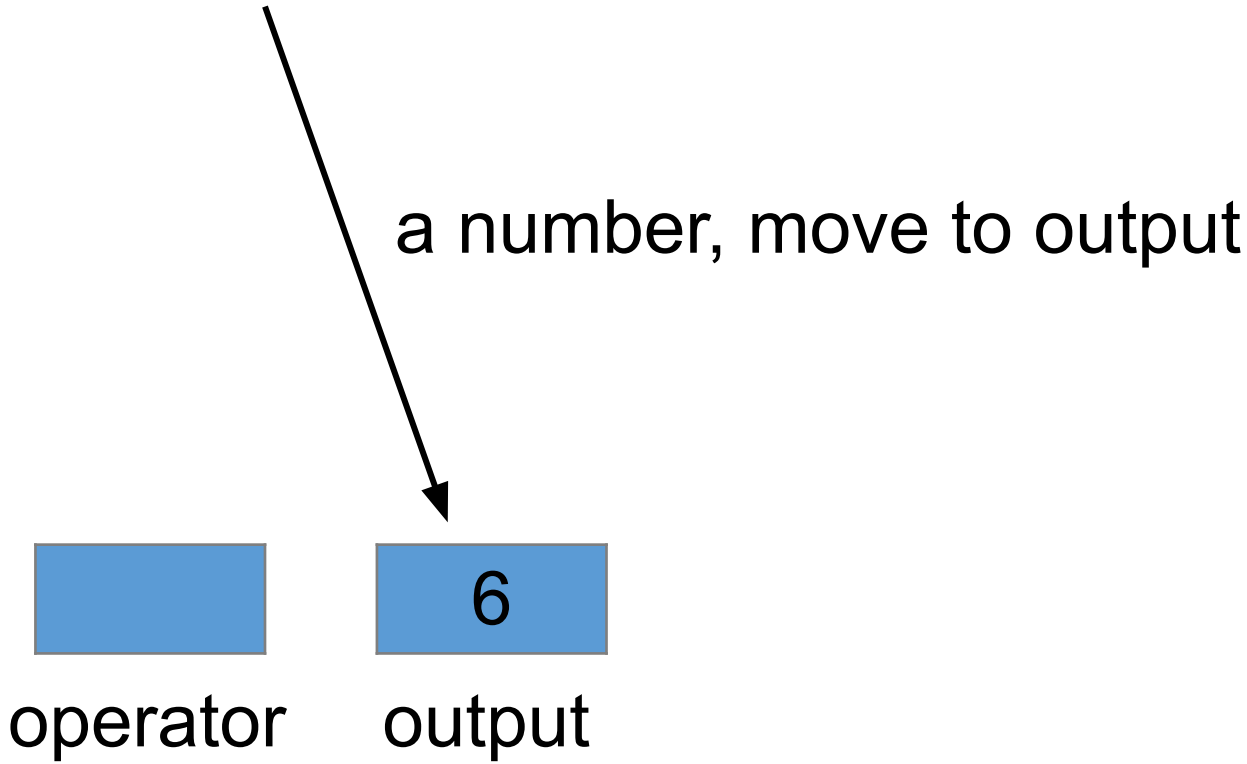
operator (stack)

output (queue)

yunglu@purdue.edu

Convert infix to postfix

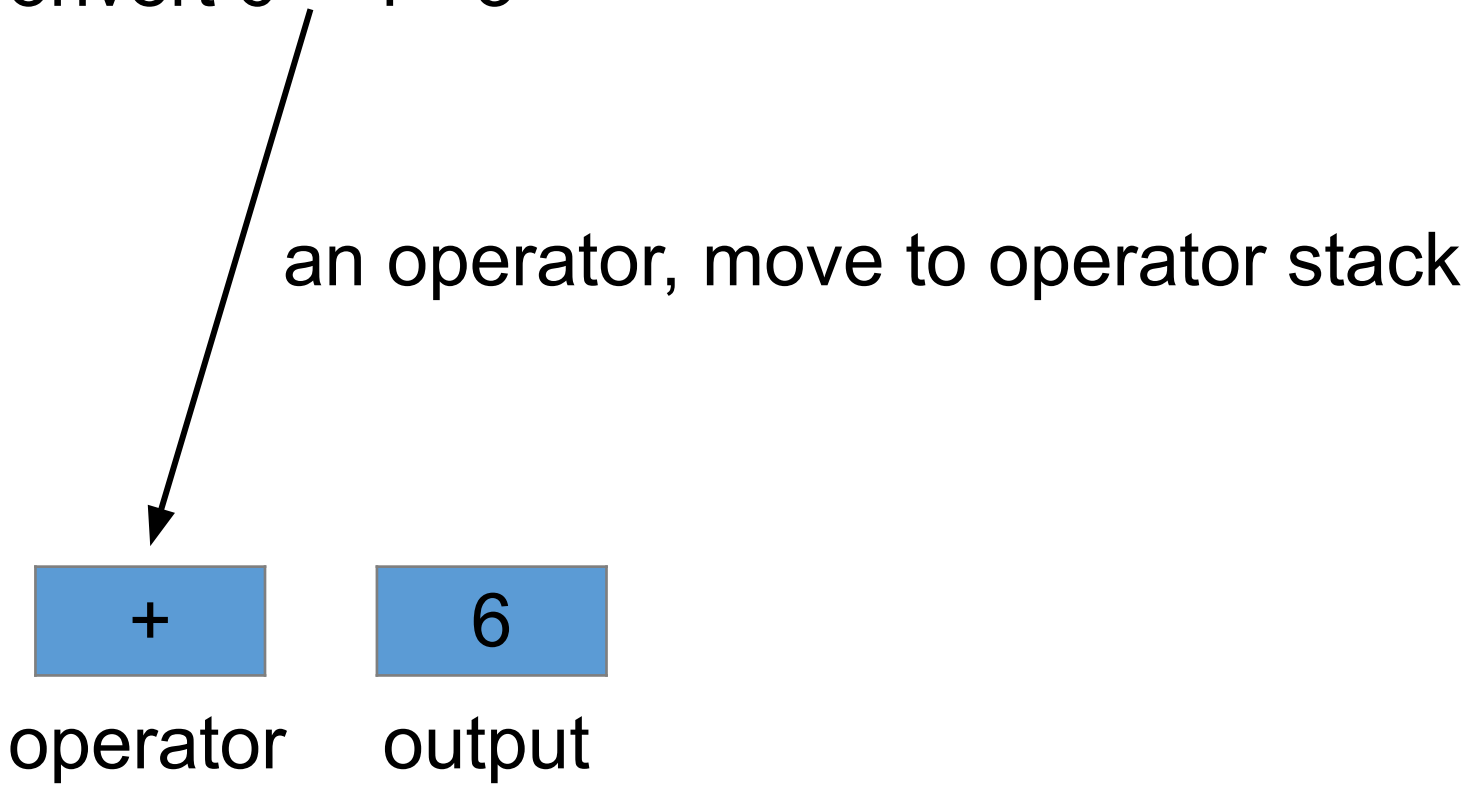
Convert 6 + 4 * 3



yunglu@purdue.edu

Convert infix to postfix

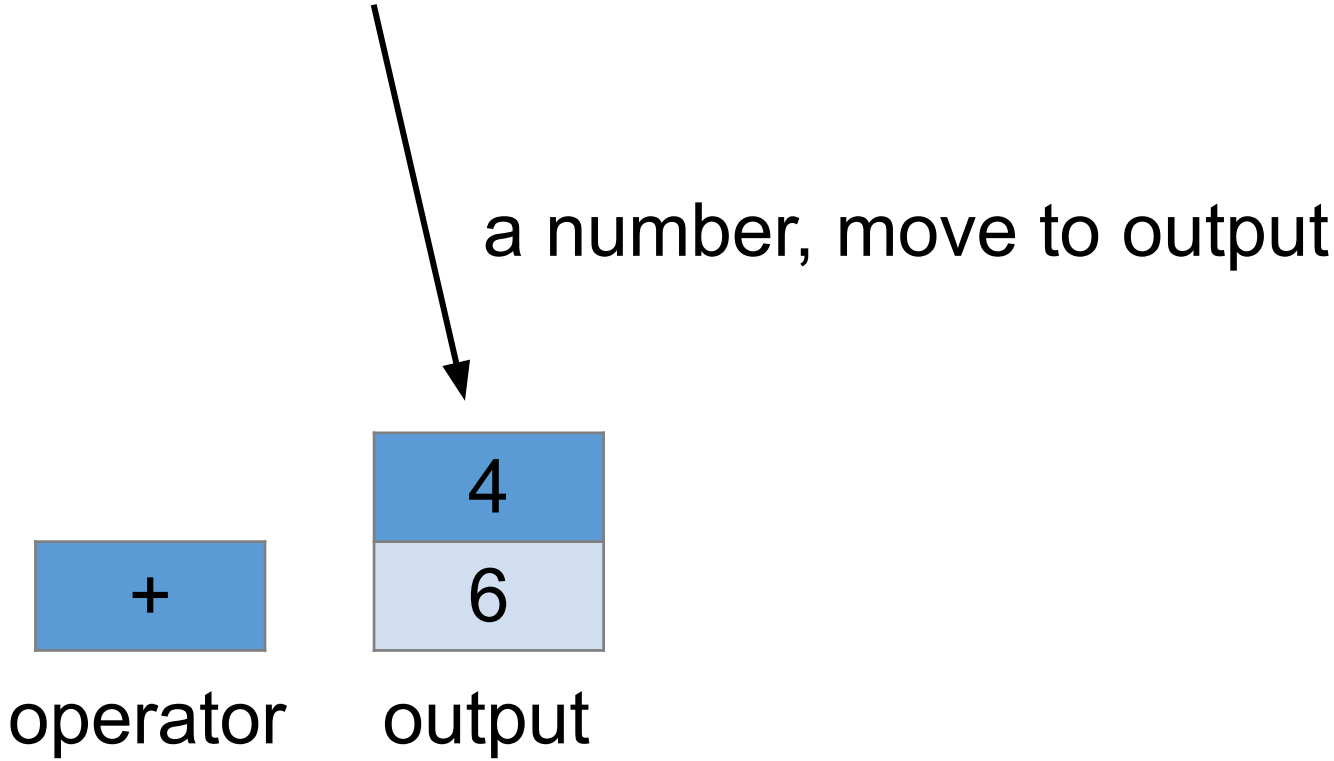
Convert 6 + 4 * 3



yunglu@purdue.edu

Convert infix to postfix

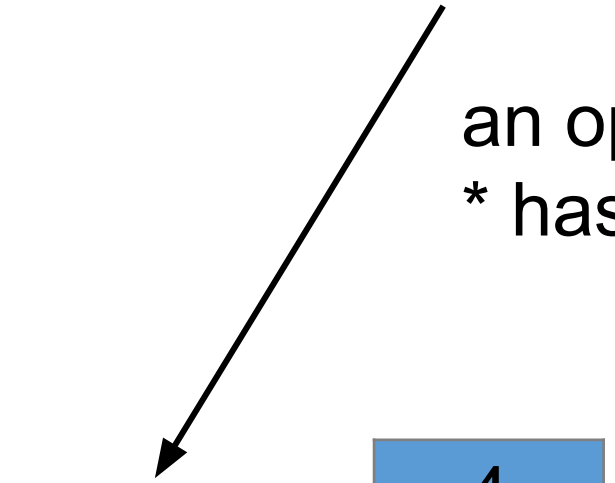
Convert 6 + 4 * 3



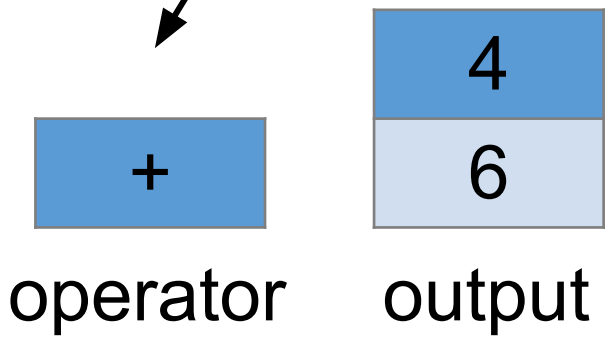
yunglu@purdue.edu

Convert infix to postfix

Convert 6 + 4 * 3



an operator, compare its precedence with +
* has higher precedence, **push * to operator**



yunglu@purdue.edu

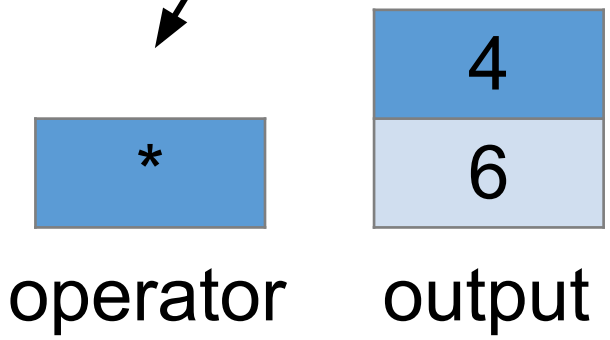
(Review) Convert infix to postfix

Convert 6 * 4 + 3

an operator, compare its precedence with *

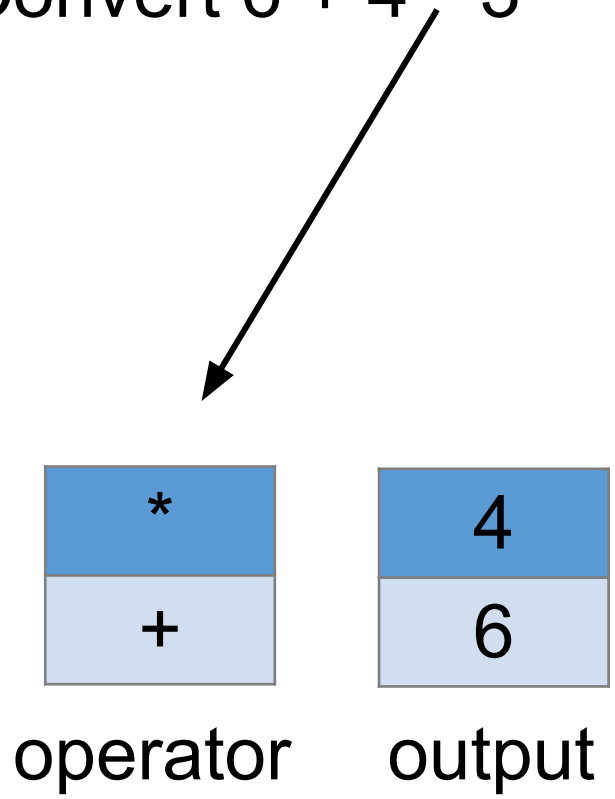
* has higher precedence, pop and move to output

push + to operator



Convert infix to postfix

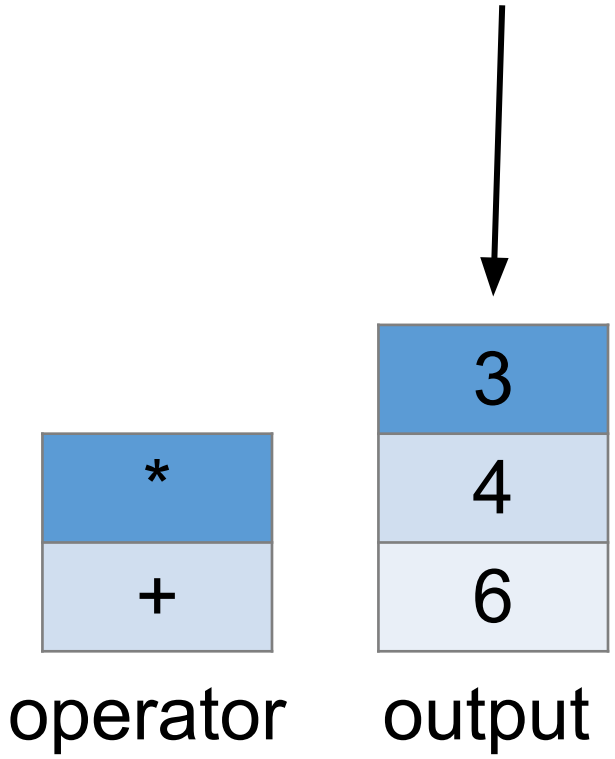
Convert 6 + 4 * 3



yunglu@purdue.edu

Convert infix to postfix

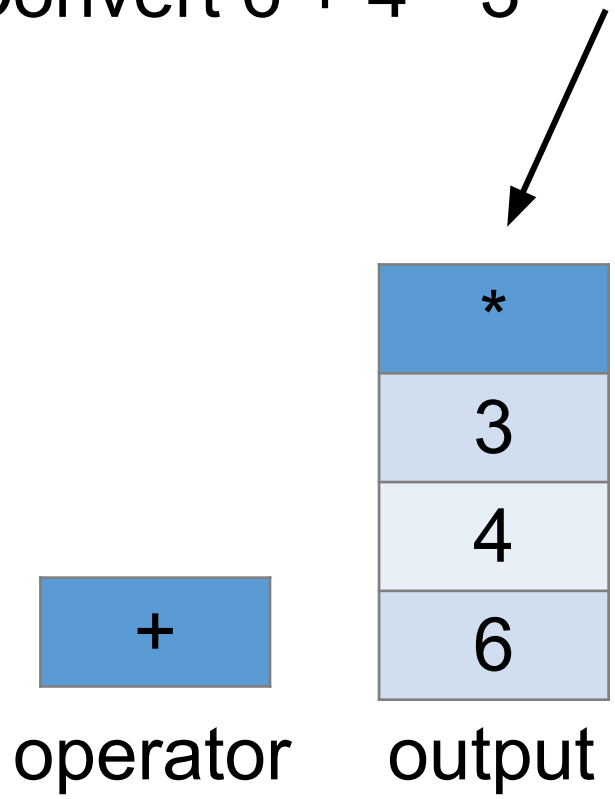
Convert 6 + 4 * 3



yunglu@purdue.edu

Convert infix to postfix

Convert 6 + 4 * 3

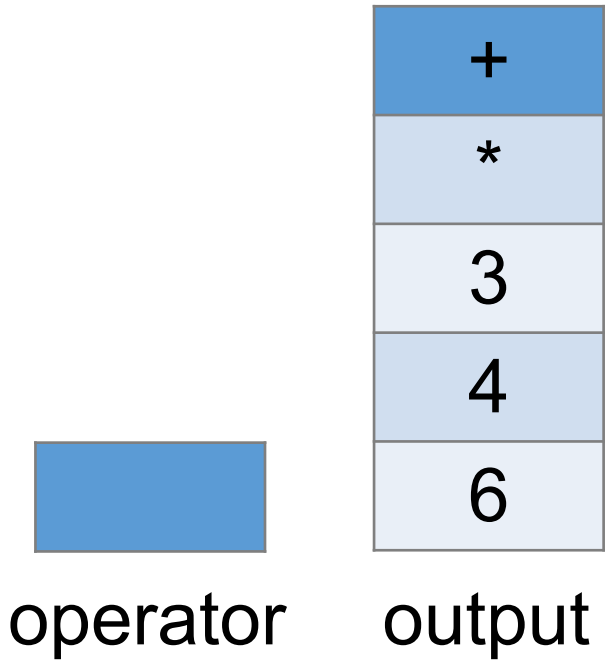


no more input
pop operator
push to output

yunglu@purdue.edu

Convert infix to postfix

Convert 6 + 4 * 3

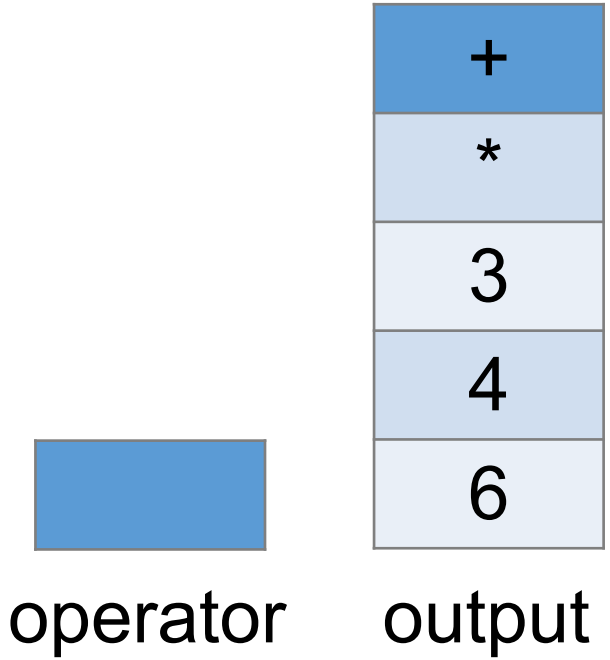


Postfix: 6 4 3 * +

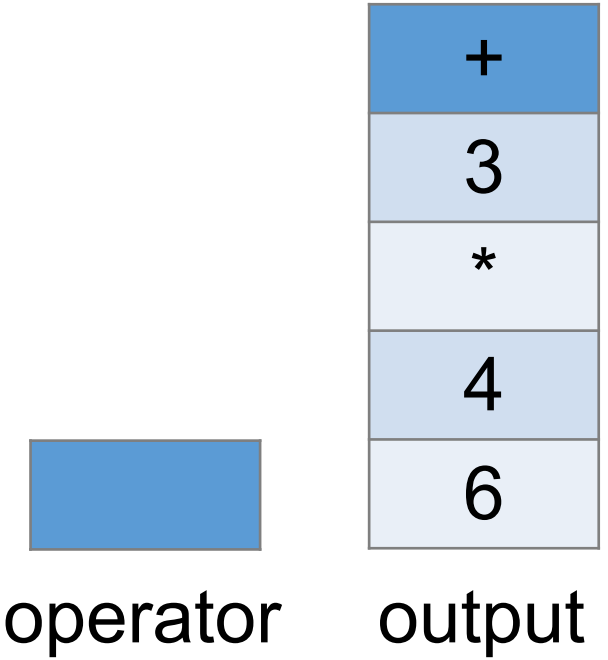
yunglu@purdue.edu

Convert infix to postfix

Convert 6 + 4 * 3



Convert 6 * 4 + 3



yunglu@purdue.edu

Convert infix to postfix

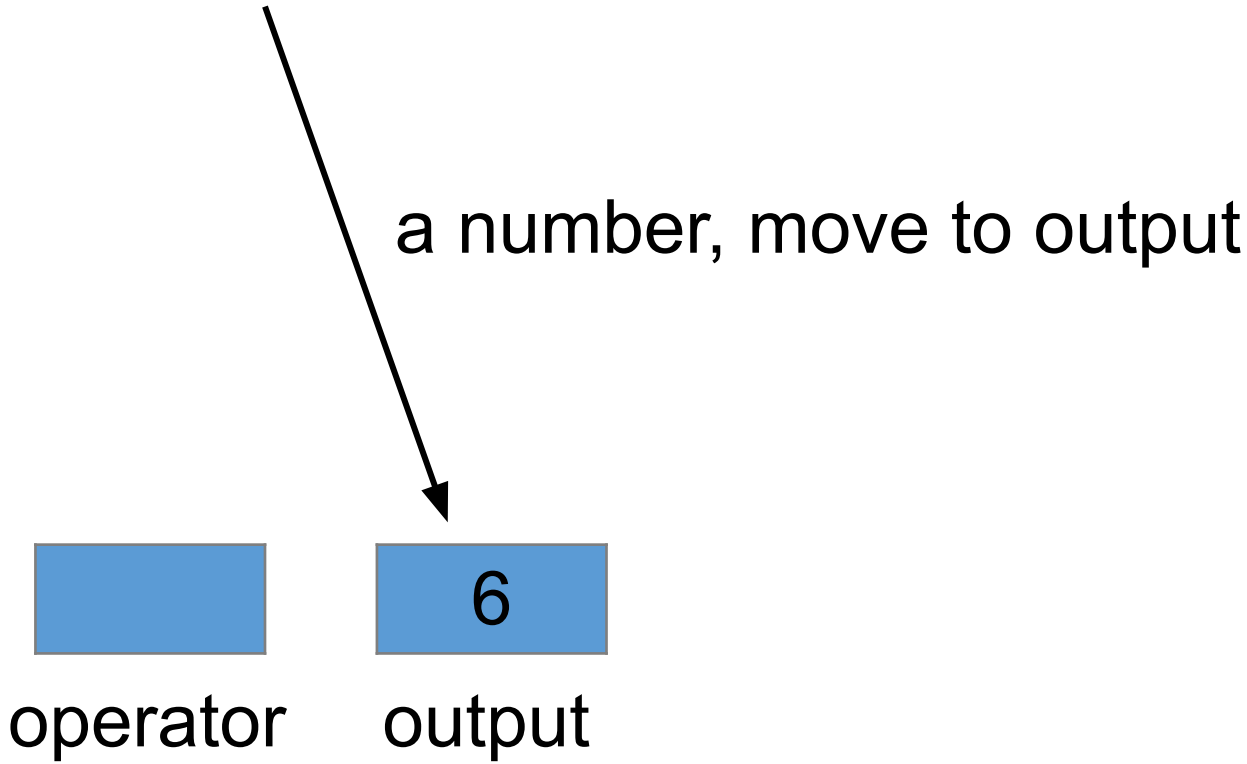
Convert 6 * (4 + 3)

operator output

yunglu@purdue.edu

Convert infix to postfix

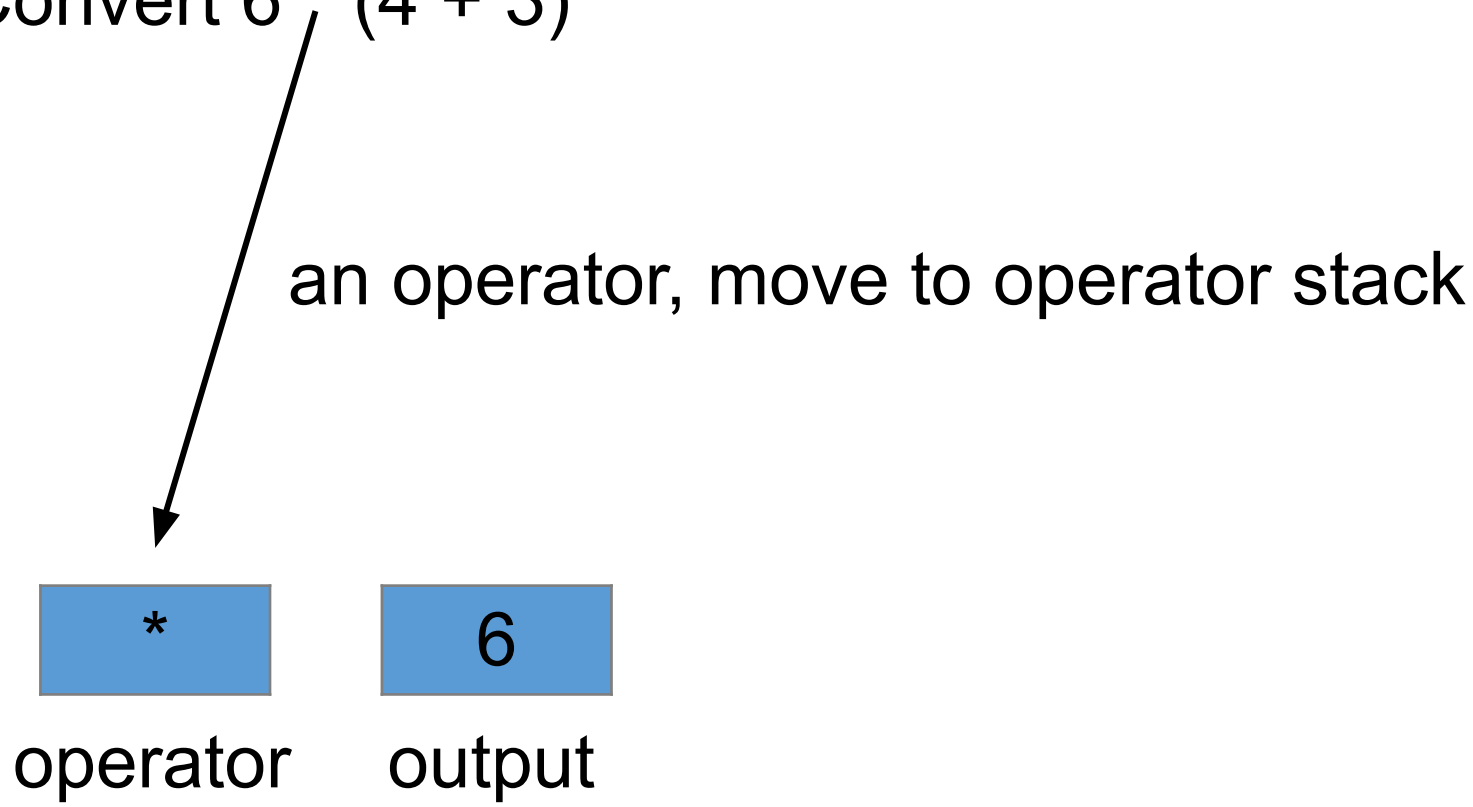
Convert 6 * (4 + 3)



yunglu@purdue.edu

Convert infix to postfix

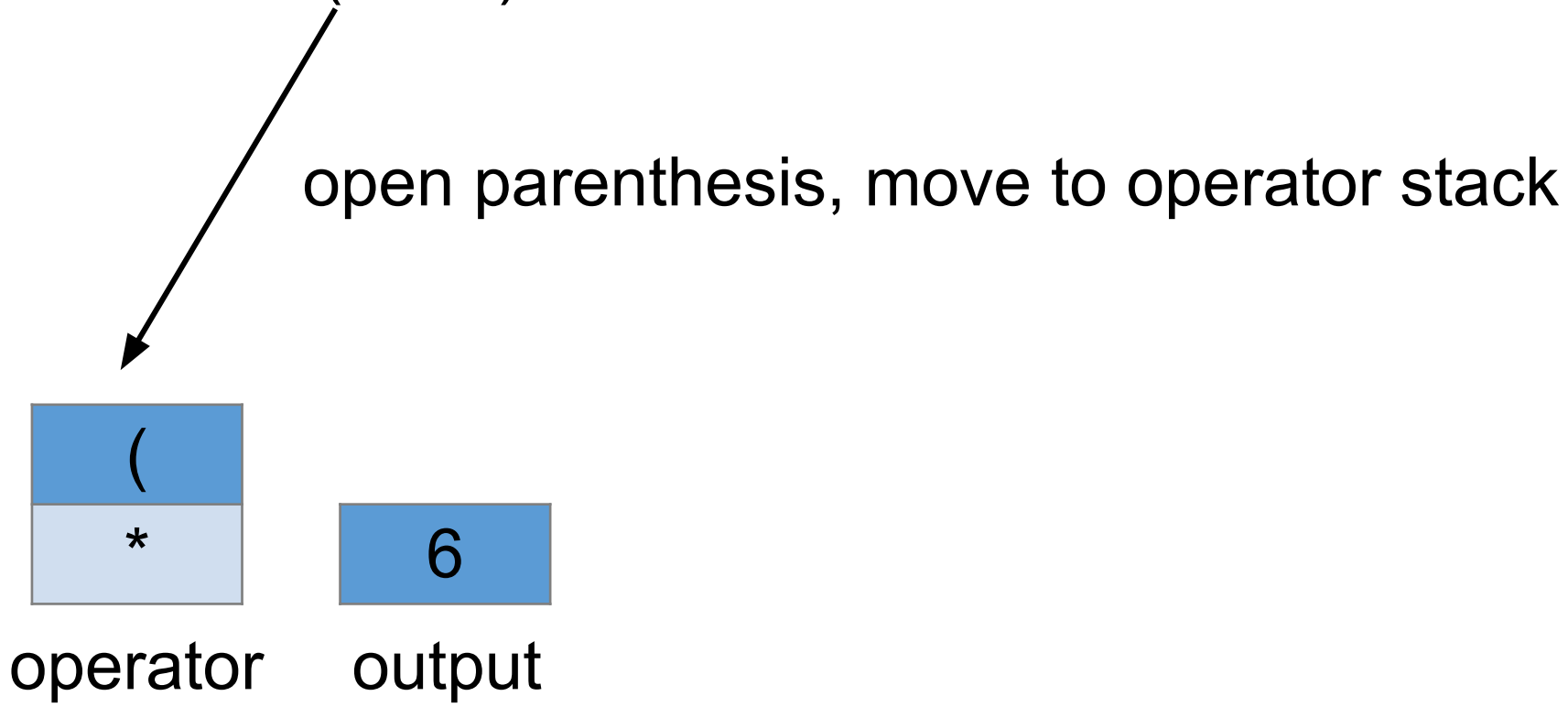
Convert 6 * (4 + 3)



yunglu@purdue.edu

Convert infix to postfix

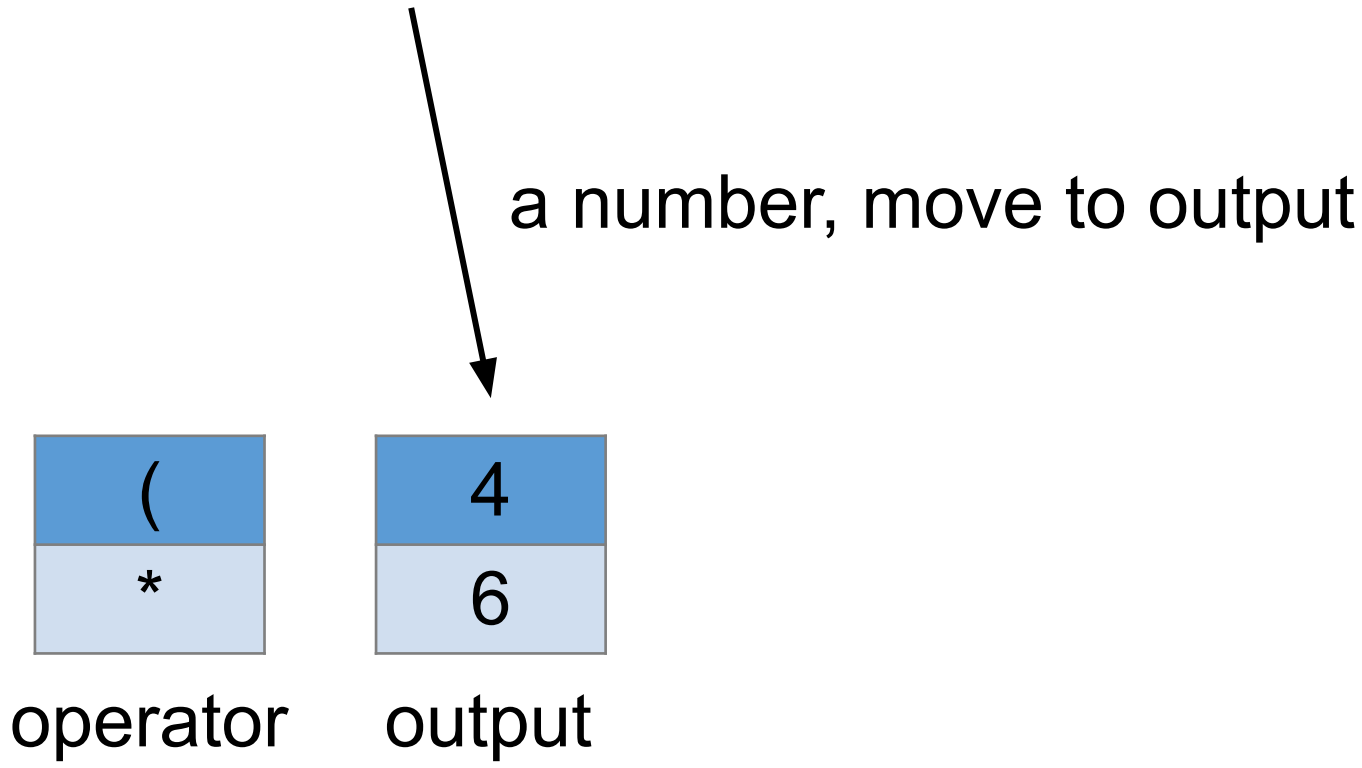
Convert 6 * (4 + 3)



yunglu@purdue.edu

Convert infix to postfix

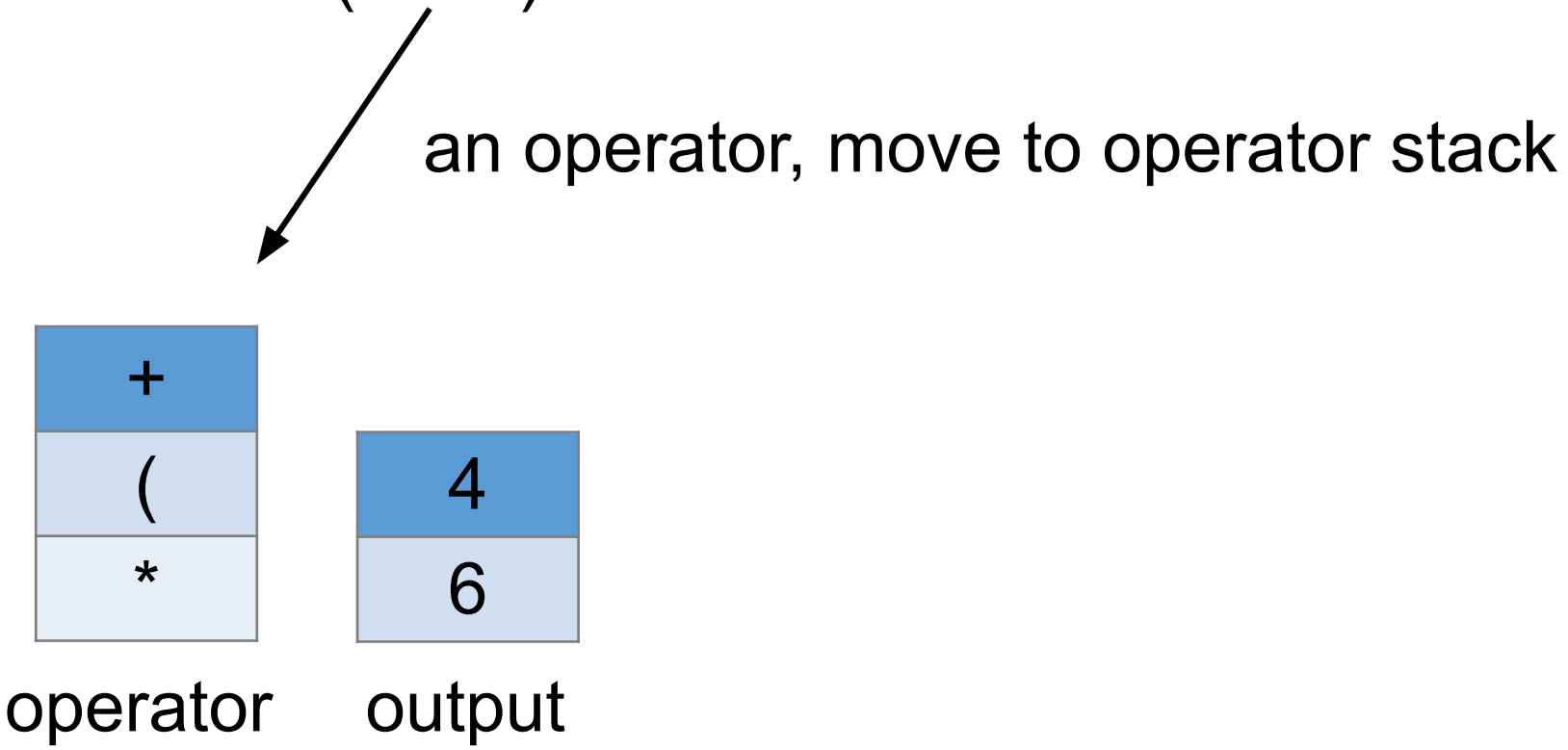
Convert 6 * (4 + 3)



yunglu@purdue.edu

Convert infix to postfix

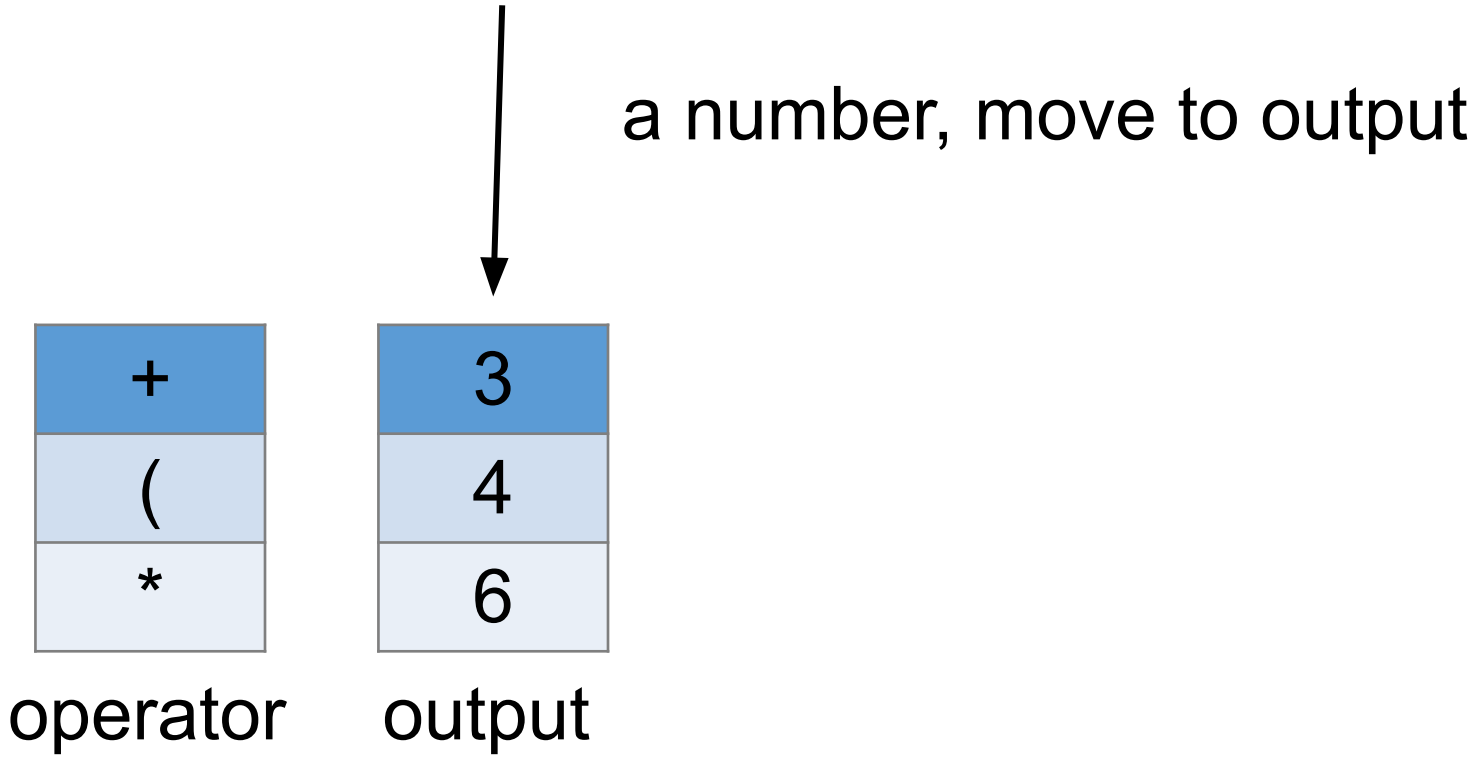
Convert 6 * (4 + 3)



yunglu@purdue.edu

Convert infix to postfix

Convert 6 * (4 + 3)

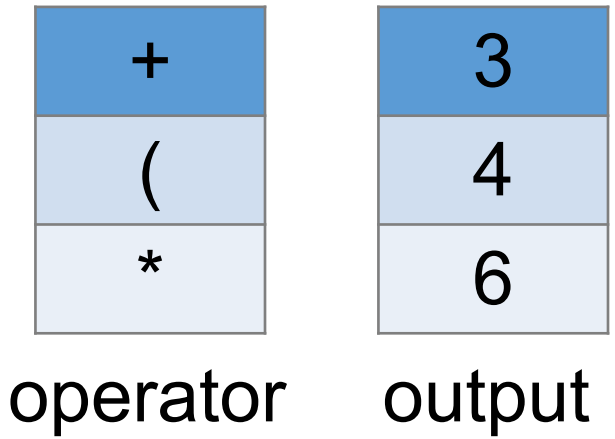


yunglu@purdue.edu

Convert infix to postfix

Convert 6 * (4 + 3)

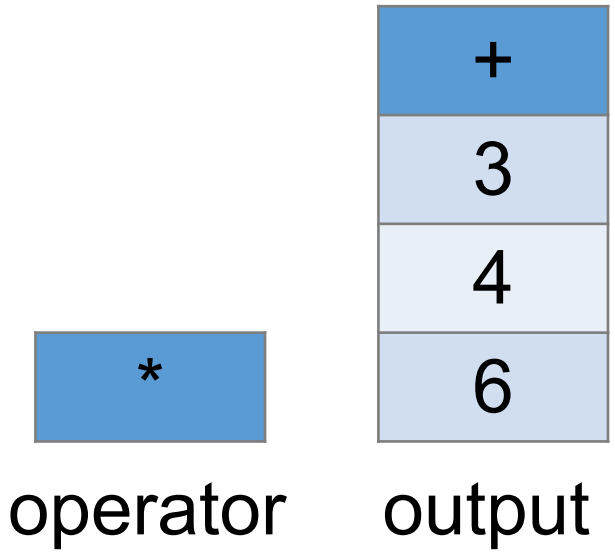
close parenthesis
pop until open parenthesis



yunglu@purdue.edu

Convert infix to postfix

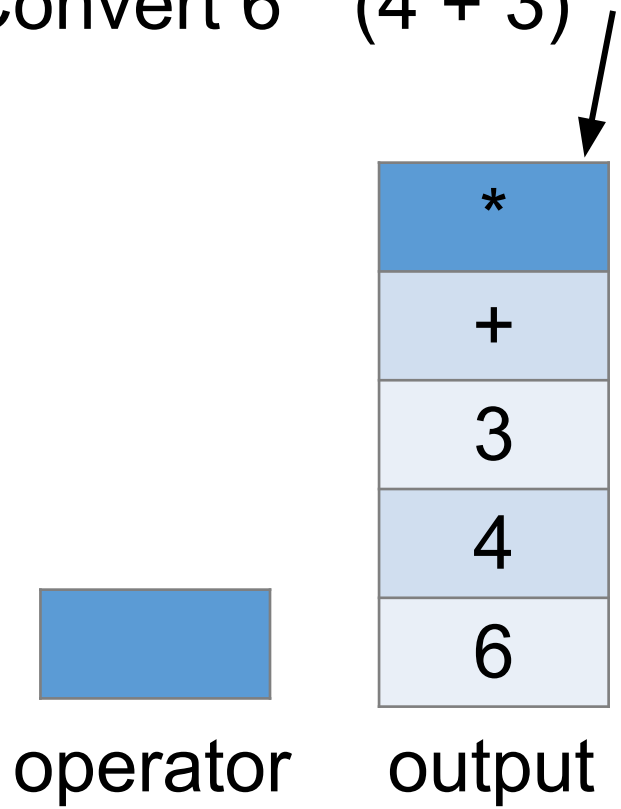
Convert 6 * (4 + 3)



yunglu@purdue.edu

Convert infix to postfix

Convert 6 * (4 + 3)



no more input
pop operator
push to output

postfix: 6 4 3 + *

yunglu@purdue.edu