CIS 351-Data Structure-Stack-Queue
Mar 12, 2020

Dr. Farzana Rahman

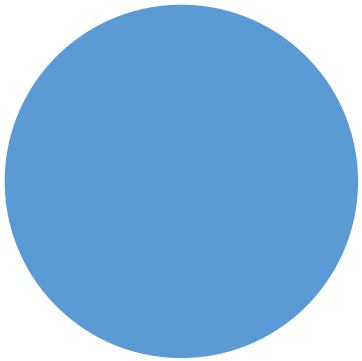
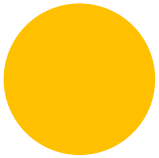
Syracuse University

Stack Applications

- Undo Operations
- Matching Parentheses
- Postfix expression evaluation...

Queues Application

- Simulation: a technique in which one system models the behavior of another system
- Queuing systems: computer simulations using queues as the data structure
  - Queues of objects are waiting to be served

Expression Evaluation with Stack

Prefix

- Prefix (Polish) notation: operators are written before the operands
  - Introduced by the Polish mathematician Jan Lukasiewicz in early 1920s
  - Parentheses can be omitted
  - Prefix: ***+ a b***

Regular **a + b**

Infix

- Infix notation: usual notation for writing arithmetic expressions
  - Operator is written between the operands
  - Example: ***a* + *b***
  - Evaluates from left to right
  - Operators have precedence
    - Parentheses can be used to override precedence

Postfix Expressions Calculator (cont'd.)

- Postfix notation has important applications in computer science
  - Many compilers first translate arithmetic expressions into postfix notation and then translate this expression into machine code

- **Evaluation algorithm:**
  - Scan expression from left to right
  - When an operator is found, back up to get operands, perform the operation, and continue

Postfix Expressions Calculator (cont'd.)

- Reverse Polish notation: operators *follow* the operands (postfix operators)
  - Proposed by Australian philosopher and early computer scientist Charles L. Hamblin in the late 1950s
  - Advantage: operators appear in the **order** required for computation
  - Example: ***a* + *b* * *c*** becomes ***a b c* * +**

Using a Stack to Process Algebraic Expressions

- Algebraic expressions composed of
  - Operands (variables, constants)
  - Operators (+, -, /, *, ^)
- Operators can be unary or binary
- Different precedence notations
  - Infix a + b
  - Prefix + a b
  - Postfix a b +

Infix to Postfix

- Manual algorithm for converting infix to postfix
  (a + b) * c
  - Write with parentheses to force correct operator precedence **((a + b) * c)**
  - Move operator to right inside parentheses
    ((a b +) c *)
  - Remove parentheses
    a b + c *

Infix to Postfix

- Stack is used to hold the operators
- Stack is to reverse the order of the operators in the expression.
- It also serves as a storage structure, since no operator can be printed until both of its operands have appeared.

Infix to Postfix Conversion

- We use a stack
- When an operand is read, output it
- When an operator is read
  - Pop until the top of the stack has an element of lower precedence
  - Then push it
- When) is found, pop until we find the matching (
- (has the lowest precedence when in the stack
- but has the highest precedence when in the input
- When we reach the end of input, pop until the stack is empty

- We will show this in a table with **three** columns.
- The **first** will show the symbol currently being read.
- The **second** will show what is on the stack
- The **third** will show the current contents of the postfix string.
- **The stack is written from left to right with 'bottom' of the stack to the left.**

A * B + C becomes A B * C +

	current symbol	operator stack	postfix string
1	A		A
2	*	*	A
3	B	*	A B
4	+	+	A B * {pop and print the '*' before pushing the '+'}
5	C	+	A B * C
6			A B * C +

A + B * C becomes A B C * +

	current symbol	operator stack	postfix string
1	A		A
2	+	+	A
3	B	+	A B
4	*	+ *	A B
5	C	+ *	A B C
6			A B C * +

A * (B + C) becomes A B C + *

A subexpression in parentheses must be done before the rest of the expression.

	current symbol	operator stack	postfix string
1	A		A
2	*	*	A
3	(	* (	A B
4	B	* (	A B
5	+	* (+	A B
6	C	* (+	A B C
7	)	*	A B C +
8			A B C + *

Postfix Expressions Calculator

- **+, -, *, and / are operators, require two operands**
  - Pop stack twice and evaluate expression
  - If stack has less than two elements → error
- **If symbol is =, expression ends**
  - Pop and print answer from stack
  - If stack has more than one element → error
- **If symbol is anything else**
  - Expression contains an illegal operator

Evaluate Postfix expression

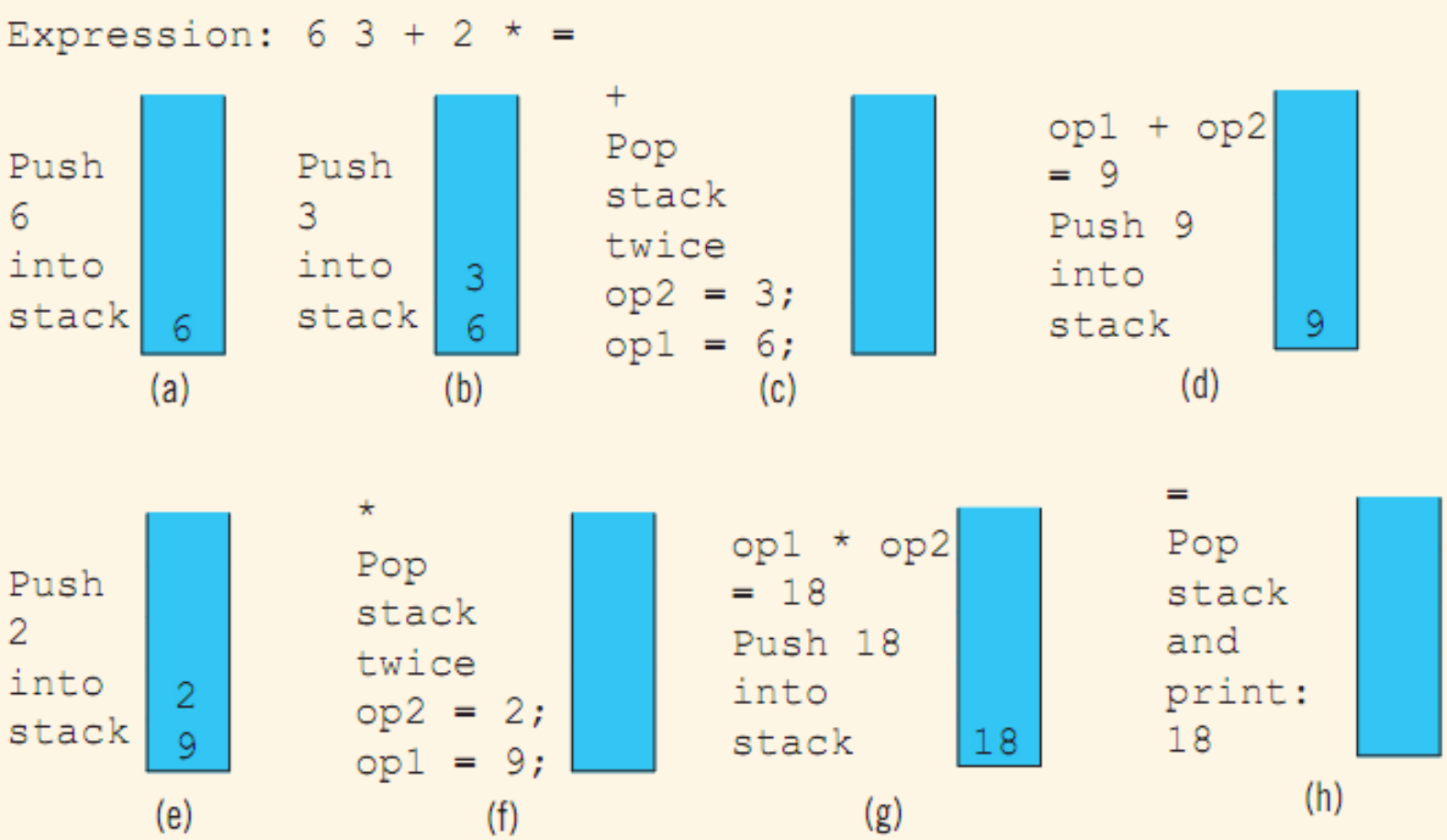


FIGURE 17-17 Evaluating the postfix expression: 6 3 + 2 * =

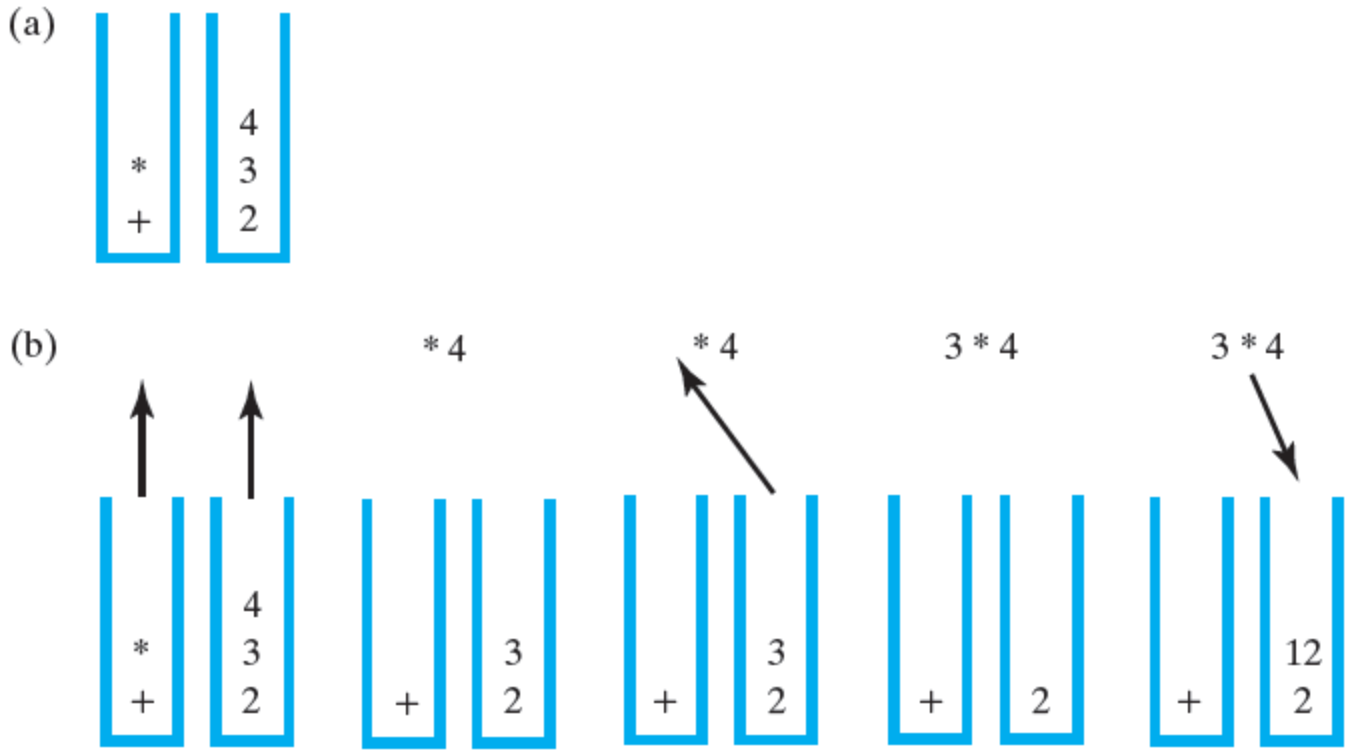
Evaluate Infix expression

We will use **two** stacks:
Operand stack: to keep values (numbers) and
Operator stack: to keep operators (+, -, *, . and ^)

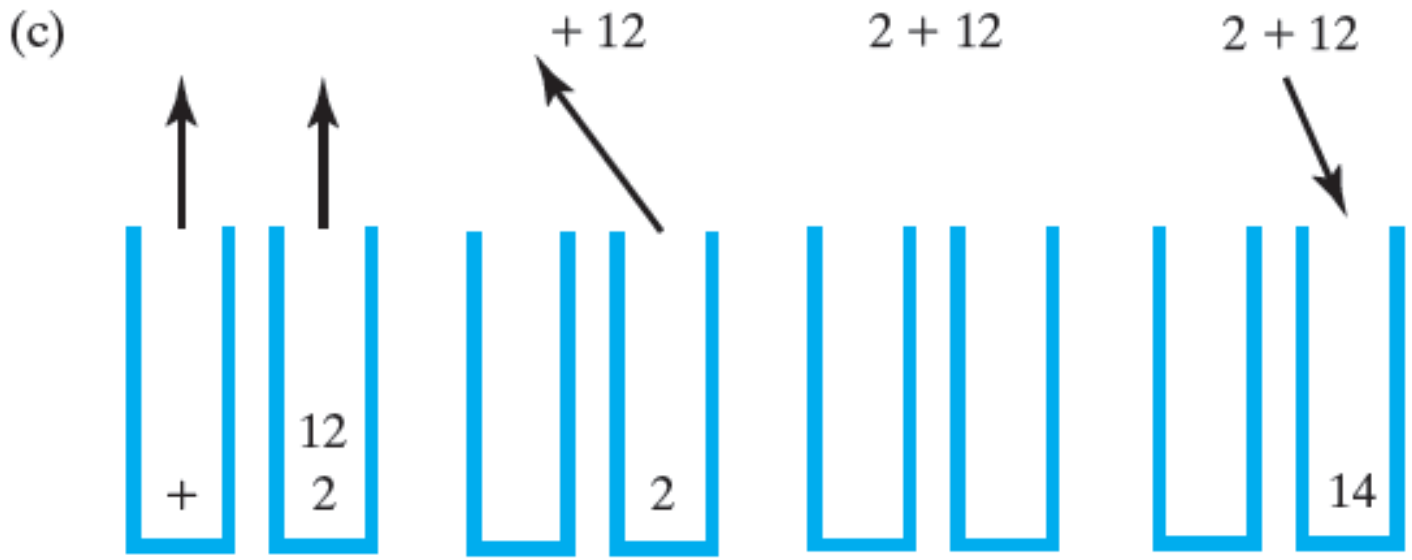
General algorithm

- Use two stacks:
  - one for operands and one for operators.
- When we encounter a right parenthesis
  - pop off one operator and two operands
  - perform the operation
  - and push the result back on the operand stack

Evaluating Infix Expressions



Two stacks during the evaluation of a + b * c when a is 2, b is 3, and c is 4:
(a) after reaching the end of the expression
(b) while performing the multiplication



Two stacks during the evaluation of a + b * c when a is 2, b is 3, and c is 4: (c) while performing the addition