University of Pittsburgh

Algorithms and Data Structures 1
CS 0445

Fall 2022

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS1501 slides.)

Announcements

- Upcoming Deadlines:
  - Homework 5: this Friday @ 11:59 pm
  - Lab 4: next Monday @ 11:59 pm
  - Programming Assignment 1: ~~Friday Oct. 7th~~ Monday Oct. 10th
- **Live Remote Support Session** for Assignment 1
  - Recording and slides on Canvas
- **Student Support Hours** of the teaching team are posted on the Syllabus page

Previous Lecture …

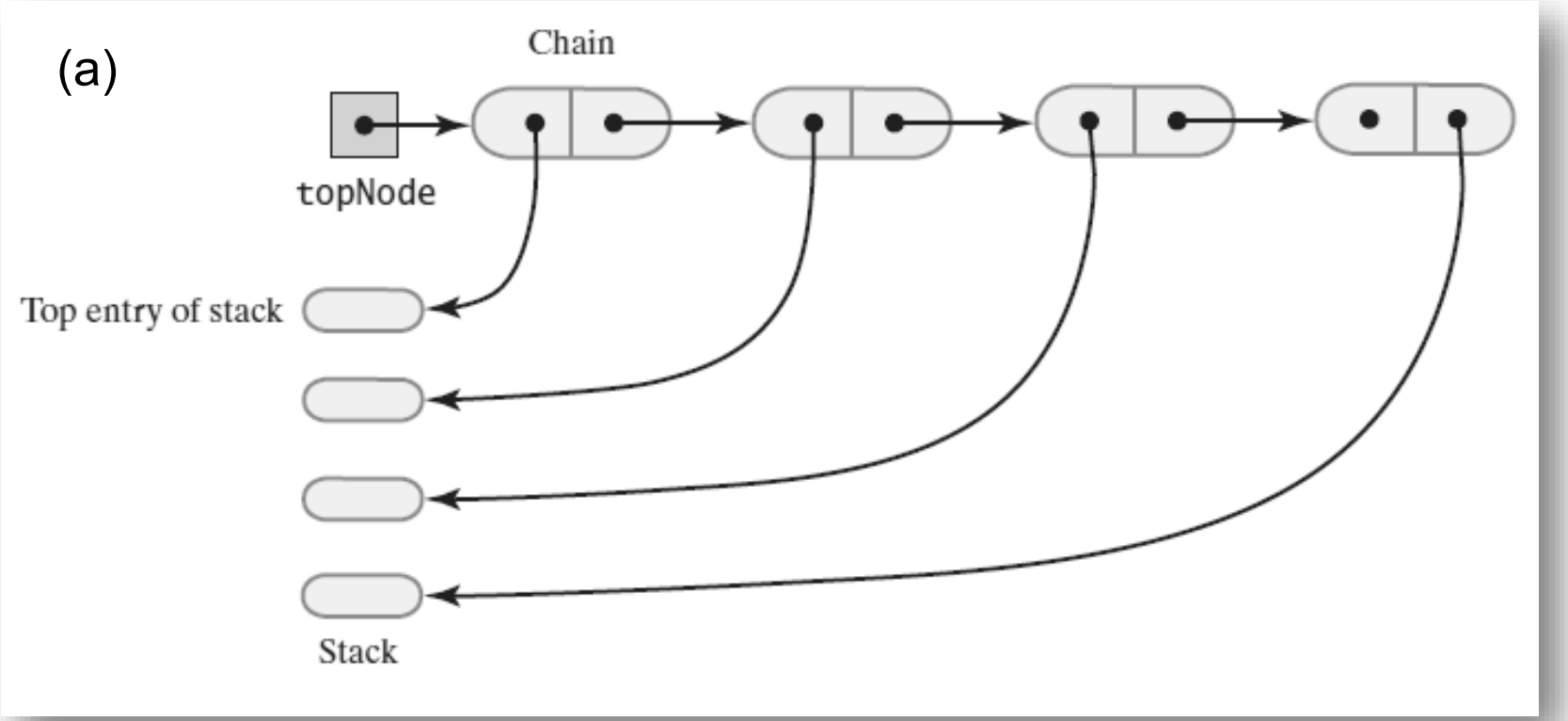
- ADT List
  - Refined Linked implementation with head and tail references
- ADT Stack
  - Array-based implementation
  - Linked implementation

Today …

- ADT Stack
  - Linked implementation
  - Implementation using ADT List
  - Application: Building a simple parser of Algebraic expressions

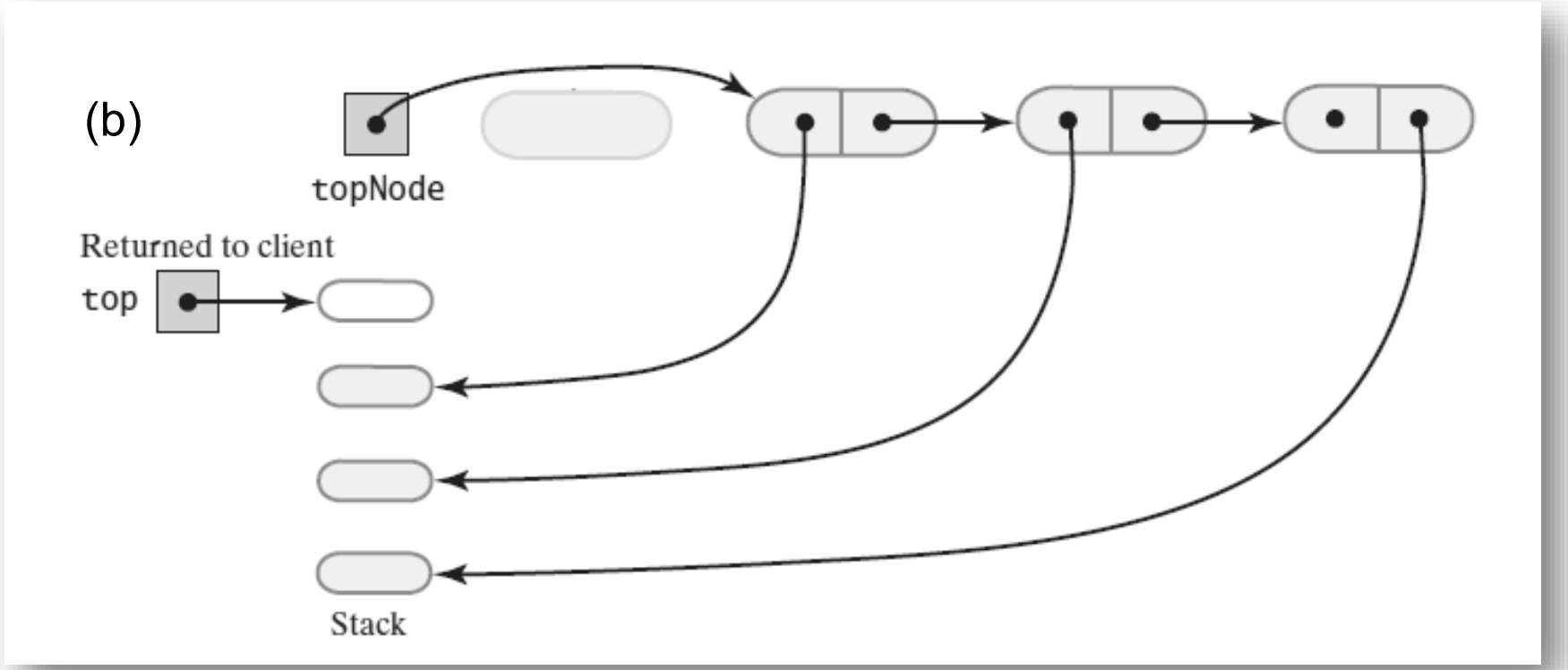
Linked Implementation

The stack before the
first node in the chain is deleted



Linked Implementation

The stack after the
first node in the chain is deleted



Linked Implementation

- Definition of `pop`

```
public T pop()
{
   T top = peek(); // Might throw EmptyStackException
   assert topNode != null;
   topNode = topNode.getNextNode();
   return top;
} // end pop
```

Linked Implementation

- Definition of rest of class.

```
public boolean isEmpty()
{
   return topNode == null;
} // end isEmpty

public void clear()
{
   topNode = null;
} // end clear
```

ADT Stack Application

Let's use the ADT Stack to design and implement a simple parser of Algebraic Expressions

Processing Algebraic Expressions

- Algebraic expressions can take different forms:
  - Infix: each binary operator appears between its operands $a + b$
  - Prefix: each binary operator appears before its operands $+ a b$
  - Postfix*:* each binary operator appears after its operands $a b +$
- Prefix and Postfix forms are easy to evaluate
  - no parentheses needed
  - no need for operator precedence rules while evaluating the Postfix expression
- But we have to make sure first that the expressions is balanced
  - parentheses paired correctly

Our Plan

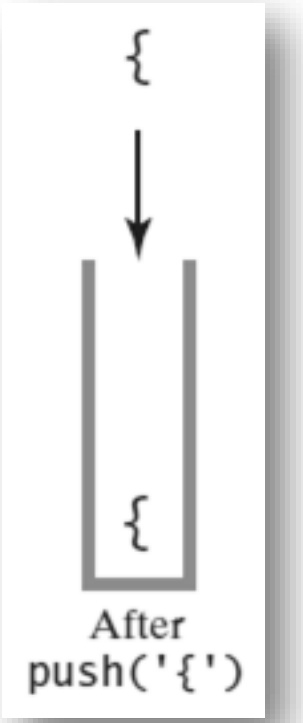
1. Check if input infix expression is balanced
2. Convert the expression from infix to postfix
3. Evaluate the postfix expression

Step 1: Balance Checking an Algebraic Expression

- Let's use a stack!
- initialize an empty Stack
- for each character in the input infix expressions
  - if an open parenthesis
    - push to Stack
  - if a closing parenthesis
    - pop from stack and compare
    - if a matching pair, continue
    - else, report unbalanced and stop
  - if the stack is not empty
    - report unbalanced and stop
  - report balanced

Balance Checking an Algebraic Expression

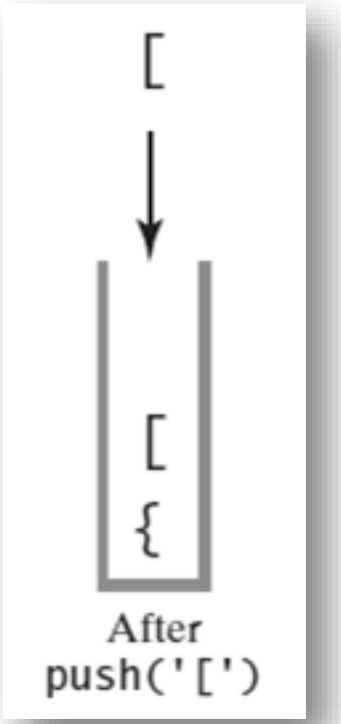
The contents of a stack during the scan of an expression that contains the balanced delimiters { [( )] }



{

Balance Checking an Algebraic Expression

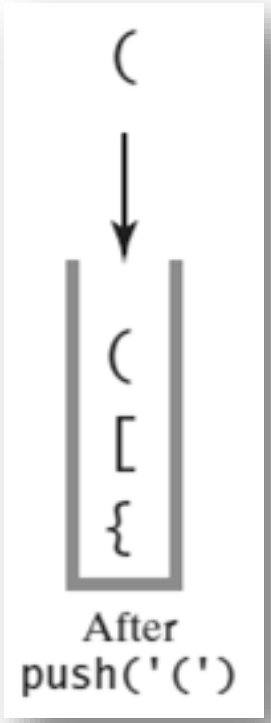
The contents of a stack during the scan of an expression that contains the balanced delimiters { [( )] }



[

Balance Checking an Algebraic Expression

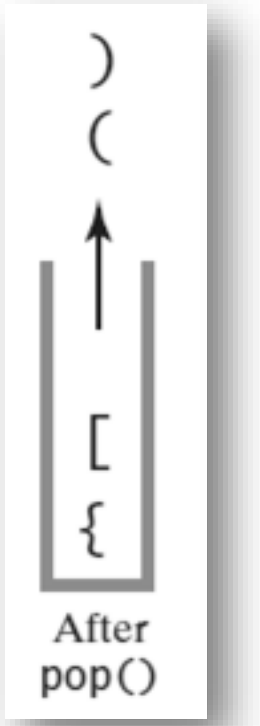
The contents of a stack during the scan of an expression that contains the balanced delimiters { [( )] }



(

Balance Checking an Algebraic Expression

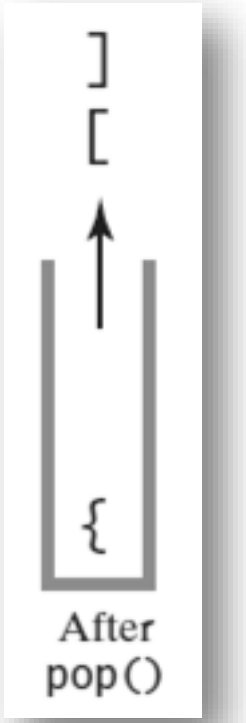
The contents of a stack during the scan of an expression that contains the balanced delimiters { [( )] }



)

Balance Checking an Algebraic Expression

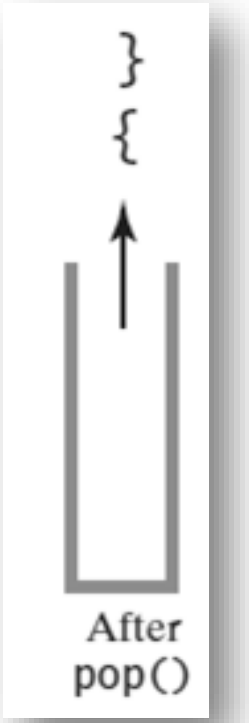
The contents of a stack during the scan of an expression that contains the balanced delimiters { [( )] }



]

Balance Checking an Algebraic Expression

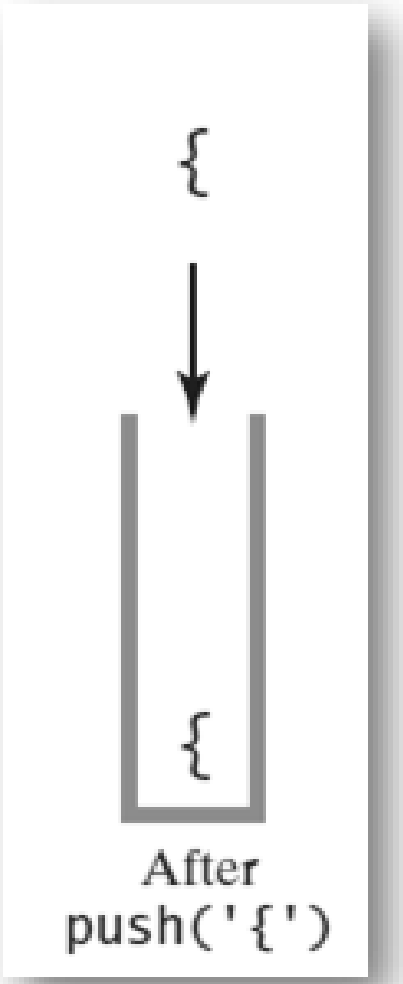
The contents of a stack during the scan of an expression that contains the balanced delimiters { [( )] }



}

Unbalanced Expression: Case 1

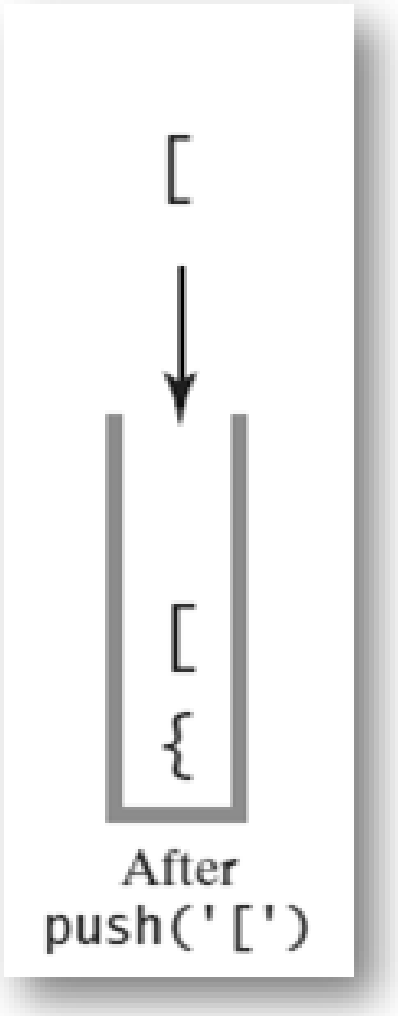
The contents of a stack during the scan of an expression that contains the unbalanced delimiters { [( ]) }



{

Unbalanced Expression: Case 1

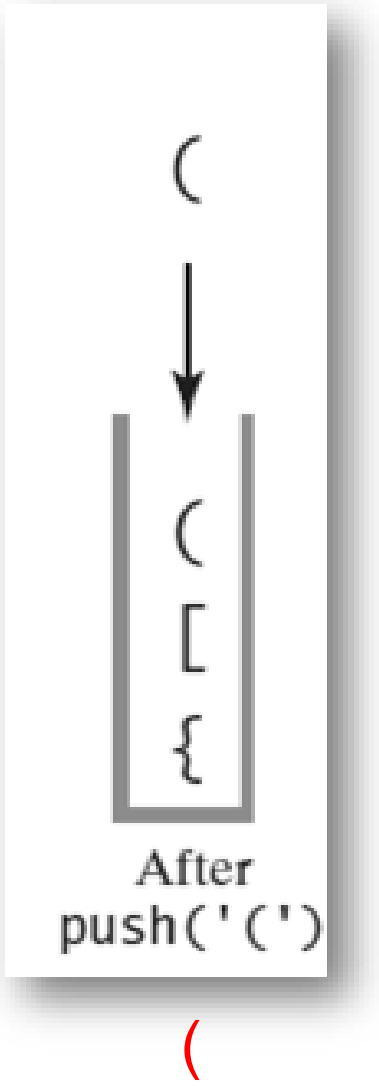
The contents of a stack during the scan of an expression that contains the unbalanced delimiters { [( ]) }



[

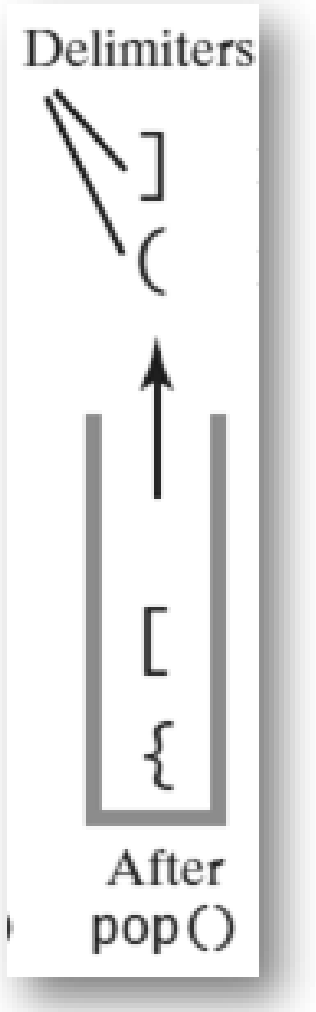
Unbalanced Expression: Case 1

The contents of a stack during the scan of an expression that contains the unbalanced delimiters { [( ]) }



Unbalanced Expression: Case 1

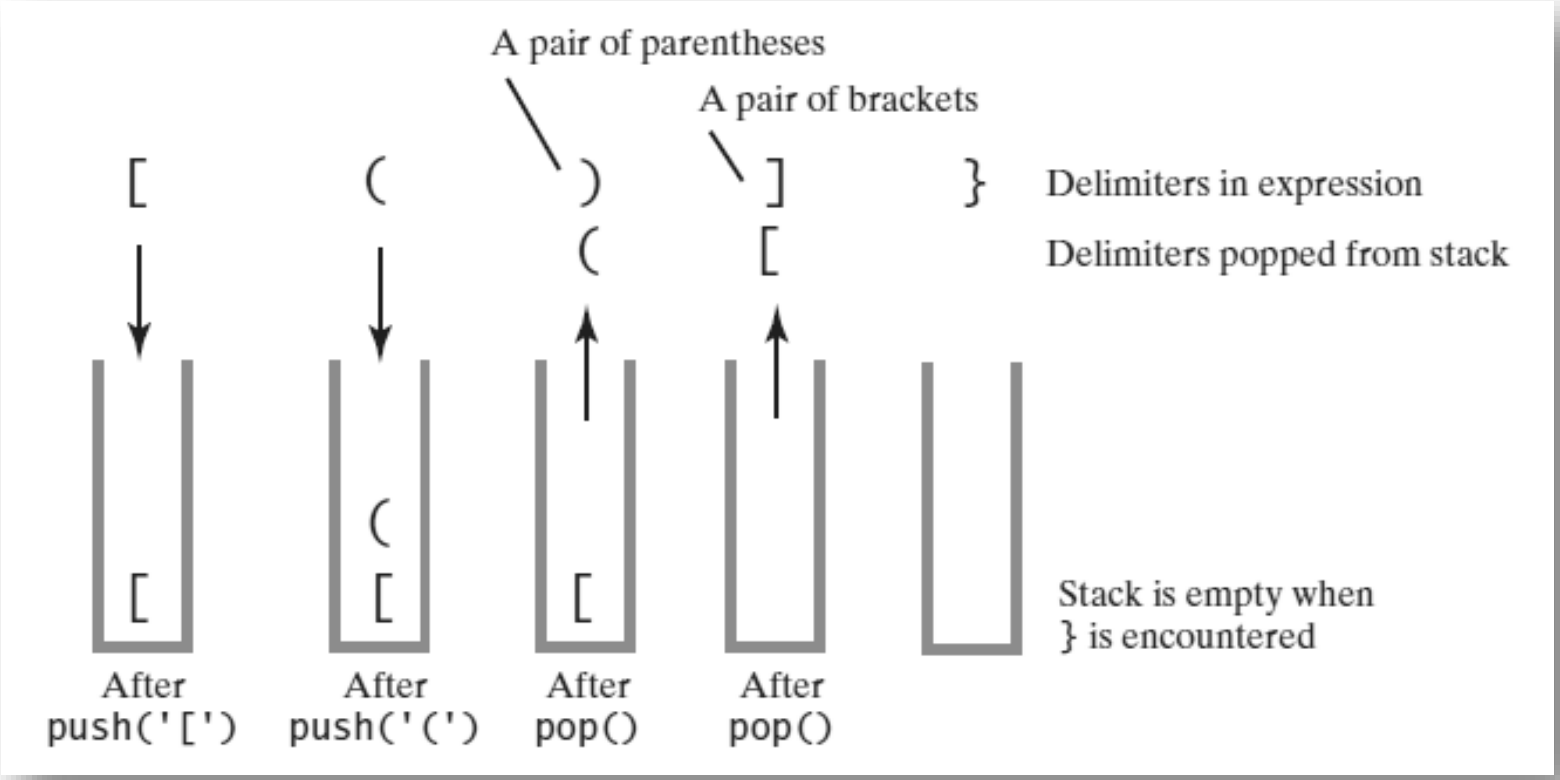
The contents of a stack during the scan of an expression that contains the unbalanced delimiters { [( ]) }



]

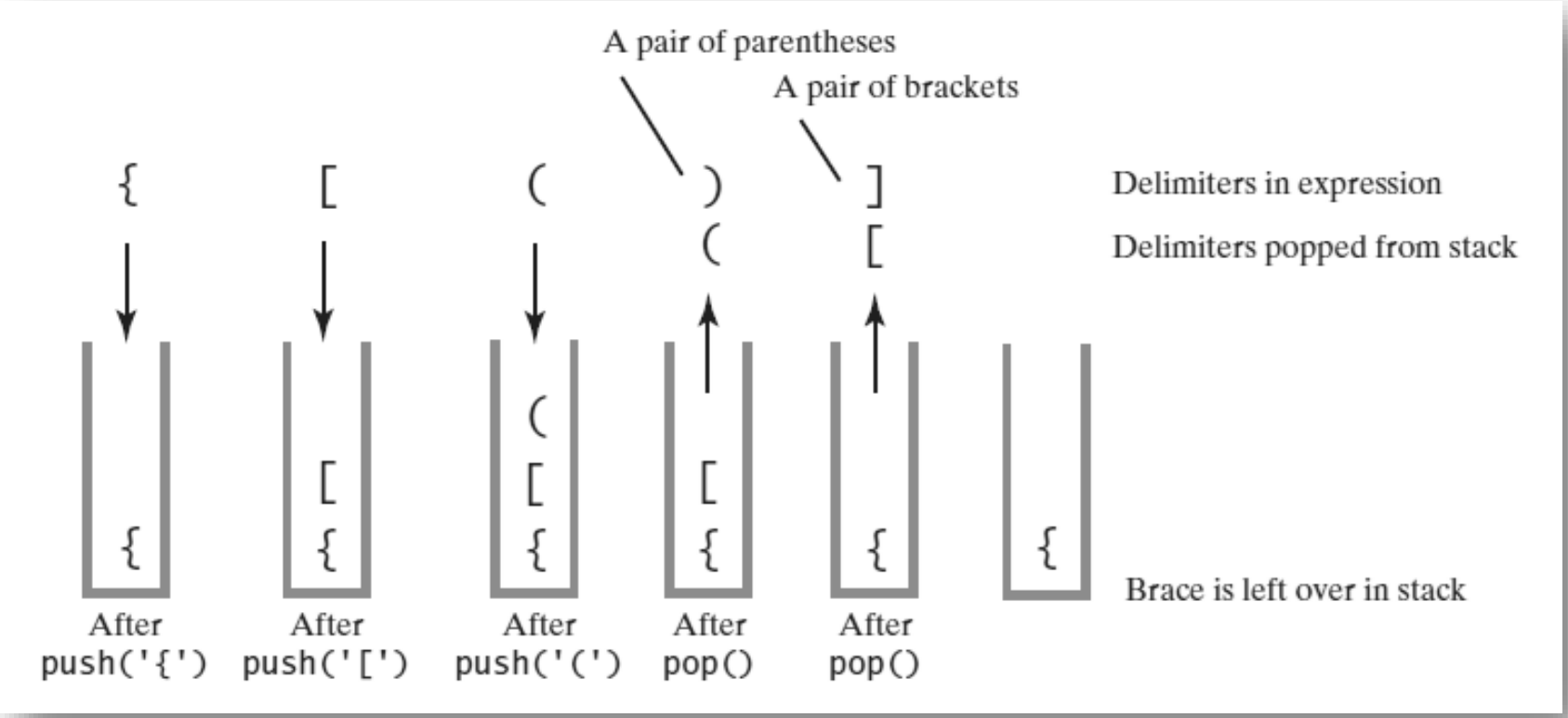
Unbalanced Expression: Case 2

The contents of a stack during the scan of an expression that contains the unbalanced delimiters [( )] }



Unbalanced Expression: Case 3

The contents of a stack during the scan of an expression that contains the unbalanced delimiters { [( )]



Processing Algebraic Expressions

Algorithm to check for balanced expression

```
Algorithm checkBalance(expression)
// Returns true if the parentheses, brackets, and braces in an expression are paired correctly.

isBalanced = true
while ((isBalanced == true) and not at end of expression)
{
   nextCharacter = next character in expression
   switch (nextCharacter)
   {
      case '(': case '[': case '{':
         Push nextCharacter onto stack
         break

      case ')': case ']': case '}':
         if (stack is empty)
            isBalanced = false
         else
```

Processing Algebraic Expressions

```
            case ')' : case ']' : case '}' :
               if (stack is empty)
                  isBalanced = false
               else
               {
                  openDelimiter = top entry of stack
                  Pop stack
                  isBalanced = true or false according to whether openDelimiter and
                               nextCharacter are a pair of delimiters
               }
               break
         }
      }

      if (stack is not empty)
         isBalanced = false
      return isBalanced
```

Java Implementation

```
3    /** Decides whether the parentheses, brackets, and braces
4        in a string occur in left/right pairs.
5        @param expression  A string to be checked.
6        @return  True if the delimiters are paired correctly. */
7    public static boolean checkBalance(String expression)
8    {
9       StackInterface<Character> openDelimiterStack = new OurStack<>();
10
11      int characterCount = expression.length();
12      boolean isBalanced = true;
13      int index = 0;
14      char nextCharacter = ' ';
15
16      while (isBalanced && (index < characterCount))
17      {
18         nextCharacter = expression.charAt(index);
19         switch (nextCharacter)
20         {
21            case '(': case '[': case '{':
```

Java Implementation

```
16        while (isBalanced && (index < characterCount))
17        {
18           nextCharacter = expression.charAt(index);
19           switch (nextCharacter)
20           {
21              case '(': case '[': case '{':
22                 openDelimiterStack.push(nextCharacter);
23                 break;
24              case ')': case ']': case '}':
25                 if (openDelimiterStack.isEmpty())
26                    isBalanced = false;
27                 else
28                 {
29                    char openDelimiter = openDelimiterStack.pop();
30                    isBalanced = isPaired(openDelimiter, nextCharacter);
31                 } // end if
```

Java Implementation

```
32                  break;
33               default: break; // Ignore unexpected characters
34            } // end switch
35            index++;
36         } // end while
37
38         if (!openDelimiterStack.isEmpty())
39            isBalanced = false;
40         return isBalanced;
41      } // end checkBalance
42
43      // Returns true if the given characters, open and close, form a pair
44      // of parentheses, brackets, or braces.
45      private static boolean isPaired(char open, char close)
46      {
47         return (open == '(' && close == ')') ||
48                (open == '[' && close == ']') ||
49                (open == '{' && close == '}');
50      } // end isPaired
51   } // end BalanceChecker
```

Step 2: Infix-to-postfix Conversion Algorithm

- for each character in the input expression

• Operand	Append each operand to the end of the output expression.

Infix-to-postfix Conversion Algorithm

- for each character in the input expression

• Operand	Append each operand to the end of the output expression.
• Operator ^	Push ^ onto the stack.

Infix-to-postfix Conversion Algorithm

- for each character in the input expression

• Operand	Append each operand to the end of the output expression.
• Operator ^	Push ^ onto the stack.
• Operator +, -, *, or /	Pop operators from the stack, appending them to the output expression, until the stack is empty or its top entry has a lower precedence than the new operator. Then push the new operator onto the stack.

Infix-to-postfix Conversion Algorithm

- for each character in the input expression

• Operand	Append each operand to the end of the output expression.
• Operator ^	Push ^ onto the stack.
• Operator +, -, *, or /	Pop operators from the stack, appending them to the output expression, until the stack is empty or its top entry has a lower precedence than the new operator. Then push the new operator onto the stack.
• Open parenthesis	Push (onto the stack.

Infix-to-postfix Conversion Algorithm

- for each character in the input expression

• Operand	Append each operand to the end of the output expression.
• Operator ^	Push ^ onto the stack.
• Operator +, -, *, or /	Pop operators from the stack, appending them to the output expression, until the stack is empty or its top entry has a lower precedence than the new operator. Then push the new operator onto the stack.
• Open parenthesis	Push (onto the stack.
• Close parenthesis	Pop operators from the stack and append them to the output expression until an open parenthesis is popped. Discard both parentheses.

Infix to Postfix: Example 1

Converting the infix expression *a* + *b* * *c* to postfix form

Next Character in Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	→

Infix to Postfix: Example 1

Converting the infix expression *a* + *b* * *c* to postfix form

Next Character in Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	→
+	*a*	+

Infix to Postfix: Example 1

Converting the infix expression *a* + *b* * *c* to postfix form

Next Character in Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	⟶
+	*a*	+
b	*a b*	+

Infix to Postfix: Example 1

Converting the infix expression *a* + *b* * *c* to postfix form

Next Character in Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	→
+	*a*	+
b	*a b*	+
*	*a b*	+ *

Infix to Postfix: Example 1

Converting the infix expression *a* + *b* * *c* to postfix form

Next Character in Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	⟶
+	*a*	+
b	*a b*	+
*	*a b*	+ *
c	*a b c*	+ *

Infix to Postfix: Example 1

Converting the infix expression *a* + *b* * *c* to postfix form

Next Character in Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	→
+	*a*	+
b	*a b*	+
*	*a b*	+ *
c	*a b c*	+ *
	a b c *	+

Infix to Postfix: Example 1

Converting the infix expression *a* + *b* * *c* to postfix form

Next Character in Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	⟶
+	*a*	+
b	*a b*	+
*	*a b*	+ *
c	*a b c*	+ *
	a b c *	+
	a b c * +	

Example with Successive Operators with Same Precedence

Converting an infix expression
to postfix form: *a - b + c*

Next Character in Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	

Example with Successive Operators with Same Precedence

Converting an infix expression to postfix form: *a - b + c*

Next Character in Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
–	*a*	–

Example with Successive Operators with Same Precedence

Converting an infix expression
to postfix form: *a - b + c*

Next Character in Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
–	*a*	–
b	*a b*	–

Example with Successive Operators with Same Precedence

Converting an infix expression to postfix form: *a - b + c*

Next Character in Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
–	*a*	–
b	*a b*	–
+	*a b* –	

Example with Successive Operators with Same Precedence

Converting an infix expression to postfix form: *a - b + c*

Next Character in Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
–	*a*	–
b	*a b*	–
+	*a b* –	
	a b –	+

Example with Successive Operators with Same Precedence

Converting an infix expression to postfix form: *a - b + c*

Next Character in Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	a	
$-$	a	$-$
b	$a\ b$	$-$
$+$	$a\ b\ -$	
	$a\ b\ -$	$+$
c	$a\ b\ -\ c$	$+$

Example with Successive Operators with Same Precedence

Converting an infix expression to postfix form: *a - b + c*

Next Character in Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	a	
$-$	a	$-$
b	$a\ b$	$-$
$+$	$a\ b\ -$	
	$a\ b\ -$	$+$
c	$a\ b\ -\ c$	$+$
	$a\ b\ -\ c\ +$	

Another Example with Successive Operators with Same Precedence

Converting an infix expression
to postfix form: *a* ^ *b* ^ *c*

Next Character in Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	

Another Example with Successive Operators with Same Precedence

Converting an infix expression
to postfix form: *a* ^ *b* ^ *c*

Next Character in Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
^	*a*	^

Another Example with Successive Operators with Same Precedence

Converting an infix expression
to postfix form: *a* ^ *b* ^ *c*

Next Character in Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
^	*a*	^
b	*a b*	^

Another Example with Successive Operators with Same Precedence

Converting an infix expression
to postfix form: *a* ^ *b* ^ *c*

Next Character in Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
^	*a*	^
b	*a b*	^
^	*a b*	^ ^

Another Example with Successive Operators with Same Precedence

Converting an infix expression
to postfix form: *a* ^ *b* ^ *c*

Next Character in Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
^	*a*	^
b	*a b*	^
^	*a b*	^ ^
c	*a b c*	^ ^

Another Example with Successive Operators with Same Precedence

Converting an infix expression to postfix form: *a* ^ *b* ^ *c*

Next Character in Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
^	*a*	^
b	*a b*	^
^	*a b*	^ ^
c	*a b c*	^ ^
	a b c ^	^

Another Example with Successive Operators with Same Precedence

Converting an infix expression to postfix form: *a* ^ *b* ^ *c*

Next Character in Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
^	*a*	^
b	*a b*	^
^	*a b*	^ ^
c	*a b c*	^ ^
	a b c ^	^
	a b c ^ ^	

Infix to Postfix: Larger Example

The steps in converting the infix expression
*a / b * (c + (d - e))* to postfix form

Next Character from Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	

Infix to Postfix: Larger Example

The steps in converting the infix expression
*a / b * (c + (d - e))* to postfix form

Next Character from Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
/	*a*	*/*

Infix to Postfix: Larger Example

The steps in converting the infix expression
*a / b * (c + (d - e))* to postfix form

Next Character from Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
/	*a*	*/*
b	*a b*	*/*

Infix to Postfix: Larger Example

The steps in converting the infix expression
*a / b * (c + (d - e))* to postfix form

Next Character from Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
/	*a*	/
b	*a b*	/
*	*a b /*	

Infix to Postfix: Larger Example

The steps in converting the infix expression
*a / b * (c + (d - e))* to postfix form

Next Character from Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
/	*a*	/
b	*a b*	/
*	*a b /*	
	a b /	*

Infix to Postfix: Larger Example

The steps in converting the infix expression
*a / b * (c + (d - e))* to postfix form

Next Character from Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
/	*a*	/
b	*a b*	/
*	*a b /*	
	a b /	*
(	*a b /*	* (

Infix to Postfix: Larger Example

The steps in converting the infix expression
*a / b * (c + (d - e))* to postfix form

Next Character from Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
/	*a*	/
b	*a b*	/
*	*a b /*	
	a b /	*
(	*a b /*	* (
c	*a b / c*	* (

Infix to Postfix: Larger Example

The steps in converting the infix expression
*a / b * (c + (d - e))* to postfix form

Next Character from Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
/	*a*	/
b	*a b*	/
*	*a b /*	
	a b /	*
(	*a b /*	* (
c	*a b / c*	* (
+	*a b / c*	* (+

Infix to Postfix: Larger Example

The steps in converting the infix expression
*a / b * (c + (d - e))* to postfix form

Next Character from Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
/	*a*	/
b	*a b*	/
*	*a b /*	
	a b /	*
(	*a b /*	* (
c	*a b / c*	* (
+	*a b / c*	* (+
(	*a b / c*	* (+ (

Infix to Postfix: Larger Example

The steps in converting the infix expression
*a / b * (c + (d - e))* to postfix form

Next Character from Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
/	*a*	/
b	*a b*	/
*	*a b /*	
	a b /	*
(	*a b /*	* (
c	*a b / c*	* (
+	*a b / c*	* (+
(	*a b / c*	* (+ (
d	*a b / c d*	* (+ (

Infix to Postfix: Larger Example

The steps in converting the infix expression
*a / b * (c + (d - e))* to postfix form

Next Character from Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
/	*a*	/
b	*a b*	/
*	*a b /*	
	a b /	*
(	*a b /*	* (
c	*a b / c*	* (
+	*a b / c*	* (+
(	*a b / c*	* (+ (
d	*a b / c d*	* (+ (
−	*a b / c d*	* (+ (−

Infix to Postfix: Larger Example

The steps in converting the infix expression
*a / b * (c + (d - e))* to postfix form

Next Character from Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
/	*a*	/
b	*a b*	/
*	*a b /*	
	a b /	*
(	*a b /*	* (
c	*a b / c*	* (
+	*a b / c*	* (+
(	*a b / c*	* (+ (
d	*a b / c d*	* (+ (
−	*a b / c d*	* (+ (−
e	*a b / c d e*	* (+ (−

Infix to Postfix: Larger Example

The steps in converting the infix expression
*a / b * (c + (d - e))* to postfix form

Next Character from Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
/	*a*	/
b	*a b*	/
*	*a b /*	
	a b /	*
(	*a b /*	* (
c	*a b / c*	* (
+	*a b / c*	* (+
(	*a b / c*	* (+ (
d	*a b / c d*	* (+ (
−	*a b / c d*	* (+ (−
e	*a b / c d e*	* (+ (−
)	*a b / c d e −*	* (+ (
	a b / c d e −	* (+

Infix to Postfix: Larger Example

The steps in converting the infix expression
*a / b * (c + (d - e))* to postfix form

Next Character from Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
/	*a*	/
b	*a b*	/
*	*a b /*	
	a b /	*
(	*a b /*	* (
c	*a b / c*	* (
+	*a b / c*	* (+
(	*a b / c*	* (+ (
d	*a b / c d*	* (+ (
−	*a b / c d*	* (+ (−
e	*a b / c d e*	* (+ (−
)	*a b / c d e −*	* (+ (
	a b / c d e −	* (+
)	*a b / c d e − +*	* (

Infix to Postfix: Larger Example

The steps in converting the infix expression
*a / b * (c + (d - e))* to postfix form

Next Character from Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
/	*a*	/
b	*a b*	/
*	*a b /*	
	a b /	*
(	*a b /*	* (
c	*a b / c*	* (
+	*a b / c*	* (+
(	*a b / c*	* (+ (
d	*a b / c d*	* (+ (
−	*a b / c d*	* (+ (−
e	*a b / c d e*	* (+ (−
)	*a b / c d e −*	* (+ (
	a b / c d e −	* (+
)	*a b / c d e − +*	* (
	a b / c d e − +	*

Infix to Postfix: Larger Example

The steps in converting the infix expression
*a / b * (c + (d - e))* to postfix form

Next Character from Infix Expression	Postfix Form	Operator Stack (bottom to top)
a	*a*	
/	*a*	/
b	*a b*	/
*	*a b /*	
	a b /	*
(	*a b /*	* (
c	*a b / c*	* (
+	*a b / c*	* (+
(	*a b / c*	* (+ (
d	*a b / c d*	* (+ (
−	*a b / c d*	* (+ (−
e	*a b / c d e*	* (+ (−
)	*a b / c d e −*	* (+ (
	a b / c d e −	* (+
)	*a b / c d e − +*	* (
	a b / c d e − +	*
	*a b / c d e − + **	

Infix-to-postfix Algorithm

```
Algorithm convertToPostfix(infix)
// Converts an infix expression to an equivalent postfix expression.

operatorStack = a new empty stack
postfix = a new empty string
while (infix has characters left to parse)
{
    nextCharacter = next nonblank character of infix
    switch (nextCharacter)
    {
        case variable:
            Append nextCharacter to postfix
            break

        case '^' :
            operatorStack.push(nextCharacter)
            break

        case '+' : case '-' : case '*' : case '/' :
```

Infix-to-postfix Algorithm

```
case '+' : case '-' : case '*' : case '/' :
   while (!operatorStack.isEmpty() and
          precedence of nextCharacter <= precedence of operatorStack.peek())
   {
      Append operatorStack.peek() to postfix
      operatorStack.pop()
   }
   operatorStack.push(nextCharacter)
   break
case '( ' :
   operatorStack.push(nextCharacter)
   break
case ')' : // Stack is not empty if infix expression is valid
   topOperator = operatorStack.pop()
   while (topOperator != '(')
   {
```

Infix-to-postfix Algorithm

```
                Append topOperator to postfix
                topOperator = operatorStack.pop()
            }
            break
        default: break // Ignore unexpected characters
    }
}
while (!operatorStack.isEmpty())
{
    topOperator = operatorStack.pop()
    Append topOperator to postfix
}
return postfix
```