CHAPTER 2: STACK AND RECURSION

BIBHA STHAPIT
ASST. PROFESSOR
IoE, PULCHOWK CAMPUS

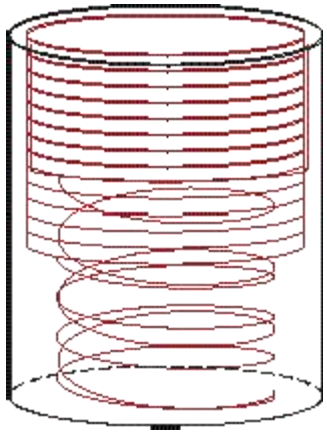
Stack : Introduction

- A ***stack*** is an ordered collection of items into which new items may be inserted and from which items may be deleted at one end called the ***top*** of the stack.

- The first inserted element can be removed only at last and the last inserted element first. This condition of operation is known as ***Last-In-First-Out (LIFO)***.

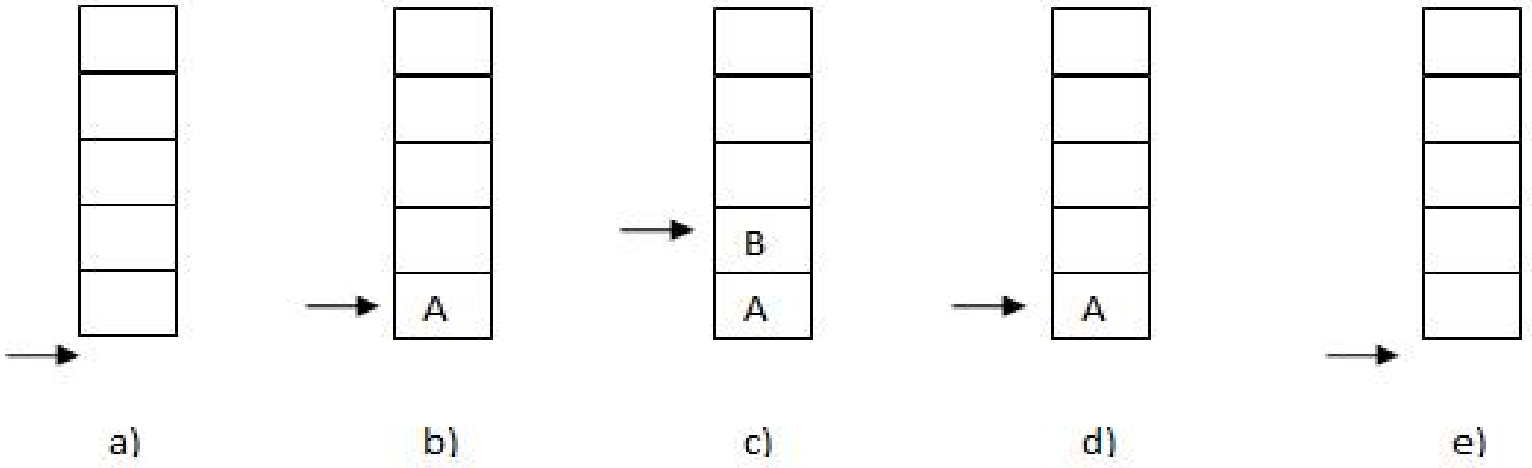
- Stacks are linear data structures and

hold objects, usually all of the same type.



Stack : Introduction

Example:



a) Stack is empty
b) Insert item A in stack
c) Insert item B in stack
d) Remove item B from stack
e) Remove Item A from stack
→ Top pointer

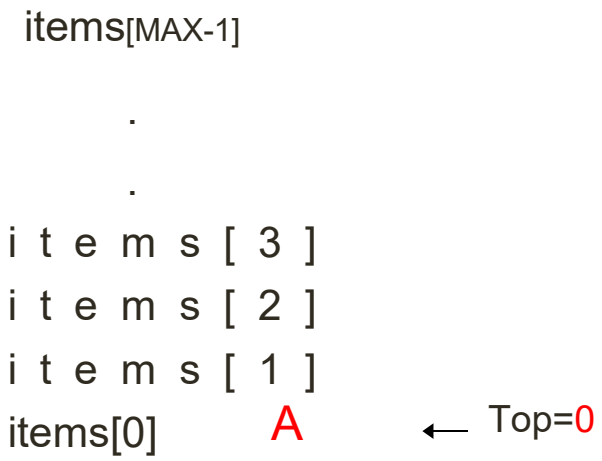
Operations on Stack

- Push
  - Adding an item
- Pop
  - Removing an item
- Peek
  - Returns top element of the stack
- Display
  - Displays all elements of the stack

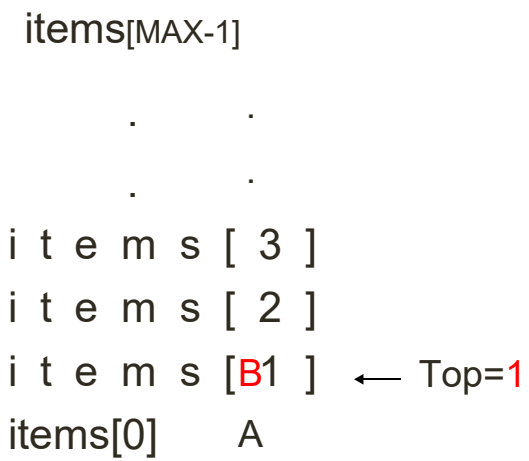
- **Overflow and Underflow conditions:**
- When we try to insert an item in a full stack, stack is ***overflow*** and the result of an illegal attempt to pop an item from an empty stack is known as ***underflow***.

Push Operation

- A new item (*A*) is *inserted* at the *Top* of the stack

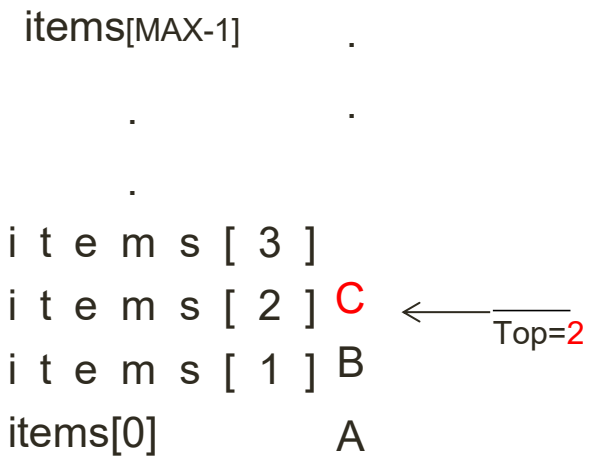


- A new item (*B*) is *inserted* at the *Top* of the stack

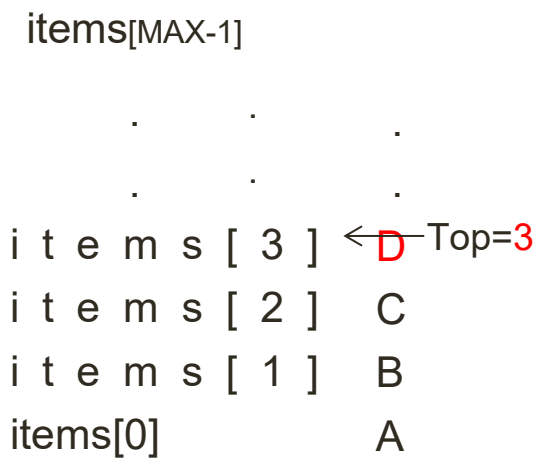


Push Operation

- A new item (*C*) is *inserted* at the *Top* of the stack



- A new item (*D*) is *inserted* at the *Top* of the stack



Push Operation

- Array implementation

- Initial condition : TOP= -1

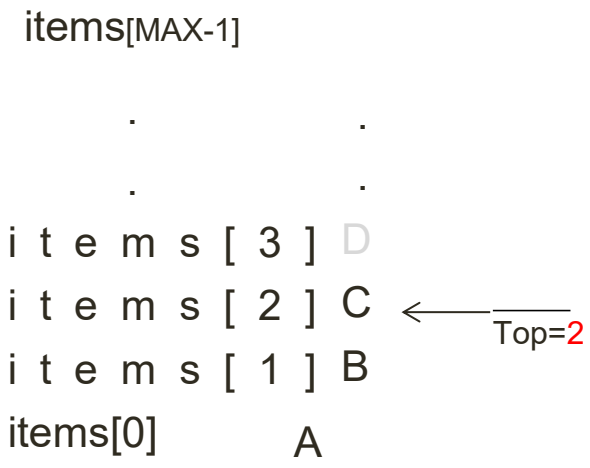
PUSH(STACK, N, TOP, ITEM)

1. If TOP=N-1:
   write OVERFLOW, and Return.

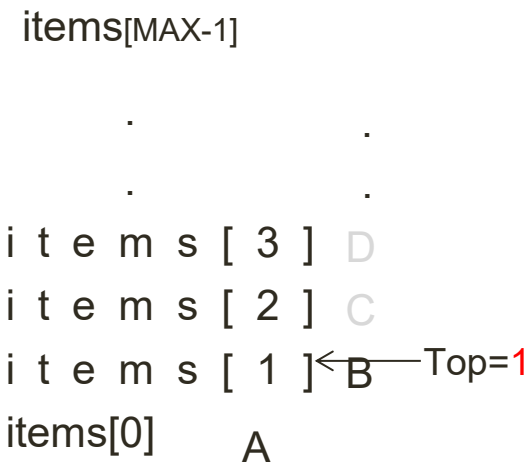
2. Set TOP := TOP + 1.
3. Set STACK[TOP]:= ITEM.
4. Return.

Pop Operation

- a n item (*D*) is deleted from the *Top* of the stack

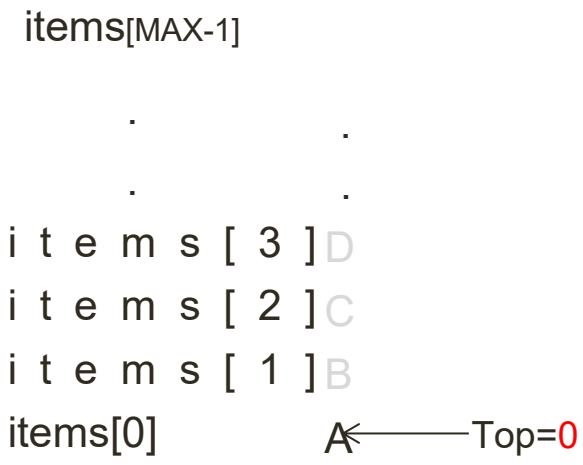


- a n item (*C*) is deleted from the *Top* of the stack

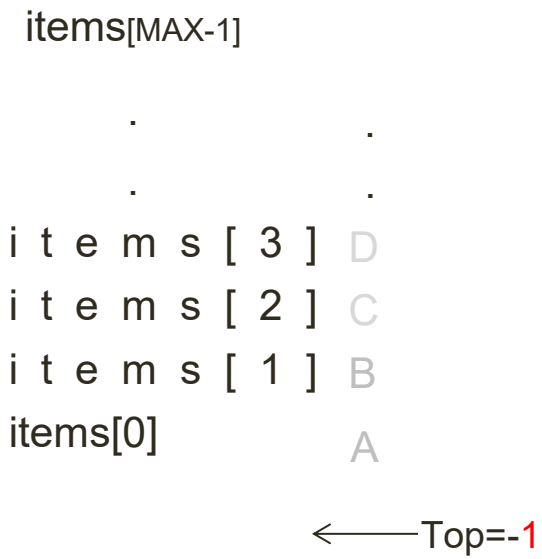


Pop Operation

- a n item (*B*) is deleted from the *Top* of the stack



- a n item (*A*) is deleted from the *Top* of the stack



Pop Operation

- Array Implementation

POP(STACK, N, TOP, ITEM)

1. If TOP = -1 then :
   write: UNDERFLOW, and Return.
2. Set ITEM := STACK [TOP].
3. Set TOP := TOP - 1.
4. Return.

```
void StackType::Push(ItemType newItem)
{
if (IsFull()) throw FullStack();
top++;
items[top] =newItem;
 }

ItemType StackType::Pop()
{
if(IsEmpty()) throw EmptyStack();
return items[top];
top--;
}
```

Prefix and postfix notations

- Polish notation, also known as prefix notation.
- RPN=Reverse Polish Notation=Postfix notation

- It is a symbolic logic invented by **Polish** mathematician **Jan Lukasiewicz** in the 1920's.

  - Most compilers convert an expression in infix notation to ***postfix*** where the operators are written after the operands

- So a * b + c becomes ab * c +
- Advantage: expressions can be written without parentheses

Prefix and postfix examples

INFIX	POSTFIX	PREFIX
A + B	A B +	+ A B
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

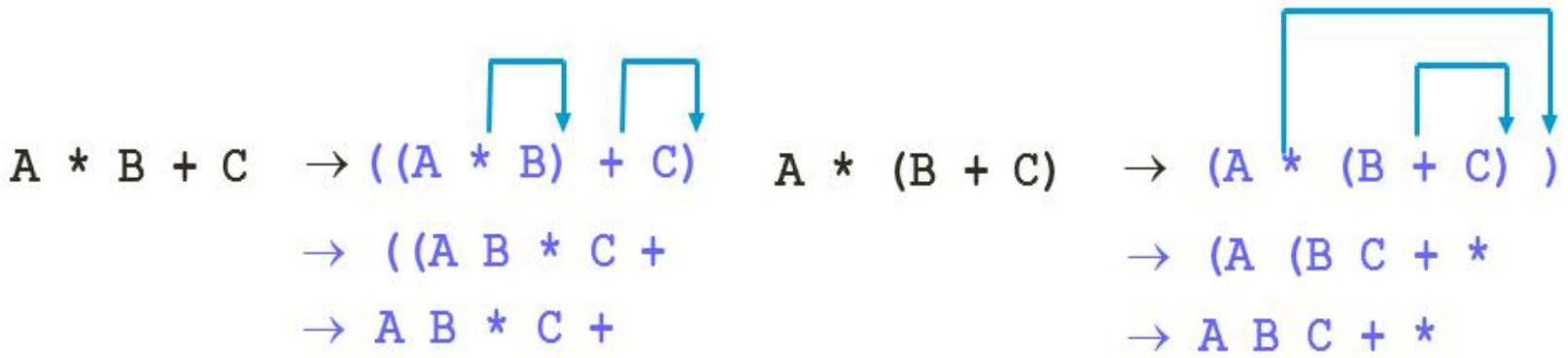
Prefix : Operators come before the operands

Transforming infix to postfix

- *By hand:* "Fully parenthesize-move-erase" method:

1. Fully parenthesize the expression.
2. Replace each right parenthesis by the corresponding operator.
3. Erase all left parentheses.

Examples:

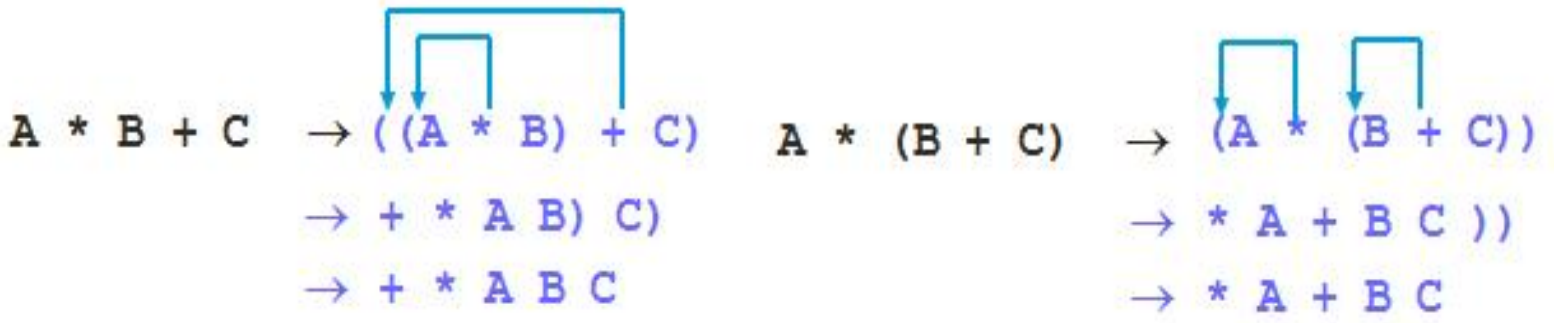


Transforming infix to prefix

- *By hand:* "Fully parenthesize-move-erase" method:

1. Fully parenthesize the expression.
2. Replace each left parenthesis by the corresponding operator.
3. Erase all right parentheses.

Examples:



Infix to postfix using stack

- 1. Scan the infix expression from left to right
- 2. If the scanned character is operand, then it will be added to postfix expression
- 3. If the scanned character is operator, then,
    - 3.a. If stack is empty or top of stack is left parenthesis, then, push scannedo perator to stack.
    - 3.b. If the priority of scanned operator is greater than that of stack operator, then the scanned operator is also pushed to stack
    - 3.c. If the priority of scanned operator is less than or equal to that of stack operator, then the stacked operator is popped from stack and added to expression while scanned operator is pushed to stack until stack is empty or left parenthesis is encountered
- 4. If the scanned character is parenthesis, then,
    - 4.a. If it is left parenthesis, then, push it to stack
    - 4.b. If it is right parenthesis, then, operators from stack are popped to postfix expression until left parenthesis is not encountered.
    - NOTE: use parenthesis for matching only, do not add it to expression
- 5. After end of expression, operators from stack are popped to postfix expression until stack is empty.

Infix Character Scanned	STACK	Postfix Expression
	(	
A	(	A
-	(-	A
(	(- (	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
F	(- (+ (% *	A B C / D E F
)	(- (+	A B C / D E F * %
/	(- (+ /	A B C / D E F * %
G	(- (+ /	A B C / D E F * % G
)	(-	A B C / D E F * % G / +
*	(- *	A B C / D E F * % G / +
H	(- *	A B C / D E F * % G / + H
)		A B C / D E F * % G / + H * -

Infix to Prefix using stack

- 1. Reverse the given expression
- Replace “(“ with “)” and vice versa
- 2. Scan the reversed expression from left to right
- 3. If the scanned character is operand, then it will be added to expression1
- 3. If the scanned character is operator, then,
    - 3.a. If stack is empty or top of stack is left parenthesis, then, push scanned operator to stack.
    - 3.b. If the priority of scanned operator is greater or equal than that of stack operator, then the scanned operator is also pushed to stack
    - 3.c. If the priority of scanned operator is less than that of stack operator, then the stackedoperator is popped from stack and added to expression1 while scanned operator is pushed to stack until stack is empty or left parenthesis is encountered
- 4. If the scanned character is parenthesis, then,
    - 4.a. If it is left parenthesis, then, push it to stack
    - 4.b. If it is right parenthesis, then, operators from stack are popped to expression1 until left parenthesis is not encountered.
    - NOTE: use parenthesis for matching only, do not add it to expression1
- 5. After end of reversed expression, operators from stack are popped to expression1 until stack is empty.
- 6. Reverse the expression1 to get the prefix expression.

Evaluating RPN expressions

- *"By hand" (Underlining technique)*:

1. Scan the expression from left to right to find an operator.
2. Locate ("underline") the last two preceding operands and combine them using this operator.
3. Repeat until the end of the expression is reached.

- Example:

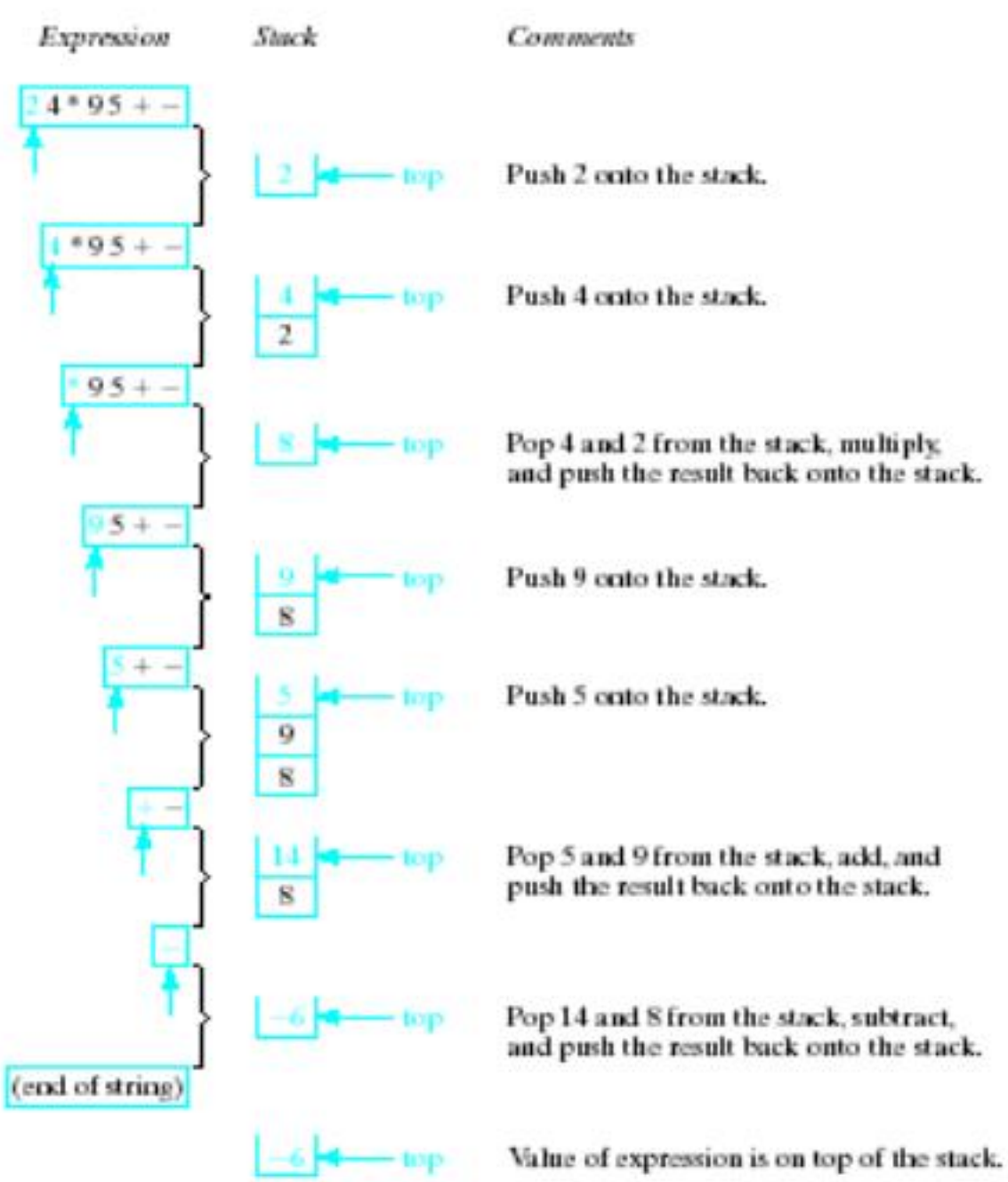
```
• 2 3 4 + 5 6 - - *
→   2 3 4 + 5 6 - - *
 → 2 7 5 6 - - *
 → 2 7 5 6 - - *
 → 2 7 -1 - *
 → 2 7 -1 - *   → 2 8 * → 2 8 * → 16
```

Evaluating RPN expressions

- P is an arithmetic expression in Postfix Notation.

1. Add a right parenthesis ")" at the end of P.
2. Scan P from left to right and Repeat Step 3 and 4 for each element of P until the sentinel ")" is encountered.
3. If an operand is encountered, put it on STACK.
4. If an operator @ is encountered, then:
   (a) Remove the two top elements of STACK, where A
   - is the top element and B is the next to top element.

   (b) Evaluate B @ A.
   (c) Place the result of (b) back on STACK.
   [End of if structure.]
   - [End of step 2 Loop.]
5. Set VALUE equal to the top element on STACK.
6. Exit.


Expression
Stack
Comments
2 4 * 9 5 + –
top
Push 2 onto the stack.
4 * 9 5 + –
4
2
top
Push 4 onto the stack.
* 9 5 + –
8
top
Pop 4 and 2 from the stack, multiply, and push the result back onto the stack.
9 5 + –
9
8
top
Push 9 onto the stack.
5 + –
5
9
8
top
Push 5 onto the stack.
+ –
14
8
top
Pop 5 and 9 from the stack, add, and push the result back onto the stack.
–
–6
top
Pop 14 and 8 from the stack, subtract, and push the result back onto the stack.
(end of string)
–6
top
Value of expression is on top of the stack.

Evaluate postfix expression:
5,6,2,+,*,12,4,/,-

	Symbol Scanned	Stack	Operation (B op A)
(1)	5	5	
(2)	6	5, 6	
(3)	2	5, 6, 2	
(4)	+	5, 8	[6+2] (A=2, B=6)
(5)	*	40	[5*8] (A=8, B=5)
(6)	12	40, 12	
(7)	4	40, 12, 4	
(8)	/	40, 3	[12/4] (A=4, B=12)
(9)	-	37	[40-3] (A=3, B=40)

Evaluating Prefix expressions

- P is an arithmetic expression in Prefix Notation.

1. Scan P from right to left and repeat step 2 and 3 for each element of P until the end of expression P.

2. If an operand is encountered, put it on STACK.

3. If an operator @ is encountered, then:

    (a)Remove the two top elements of STACK, where A is the top element and B is the next to top element.

    (b)Evaluate A @ B.

    (c)Place the result of (b) back on STACK. [End of if structure.]

[End of step 1 Loop.]

4. Set VALUE equal to the top element on STACK.

5. Exit.

Matching the nested parentheses

- Checking the validity od arithmetic expressions by matching parentheses.

- Parentheses are nested correctly if:
  - There equal number of right and left parentheses.
  - Every right parentheses is preceded by a matching left parentheses.

- *Example:*
- (a+b*(c+d)-e *invalid*
- a+((c+d)-e*f)) *invalid*
-)a+b(*invalid*
- (a+b))-c+d *invalid*

Matching the nested parentheses : Algorithm

- Input the arithmetic expression.
- Scan the expression left to right character-by-character.
- During your scanning,
  - if you found a left parenthesis, push it onto the stack and continue scanning
  - if you found a right parenthesis, examine the status of the stack,
    - if the stack is empty, then the right parenthesis does not have a matching left parenthesis.
    - if the stack is non-empty, pop the stack item and continue scanning.
- if expression end, examine the status of the stack,
  - if the stack is empty, then the expression is correct.
  - otherwise one or more left parentheses have been opened and have not been closed

End of Part A-Stack