Stacks

- Introduction: List and Array representations,
- Operations on stack (traversal, push and pop)
- Arithmetic expressions: polish notation, evaluation and transformation of expressions.

Introduction to Stacks

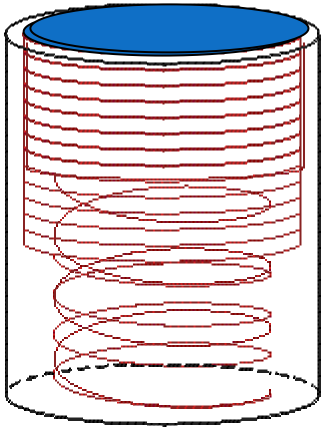
- Consider a card game with a discard pile
  - Discards always placed on the top of the pile
  - Players may retrieve a card only from the top

What other examples can you think of that are modeled by a stack?

- We seek a way to represent and manipulate this in a computer program
- This is a stack

Introduction to Stacks

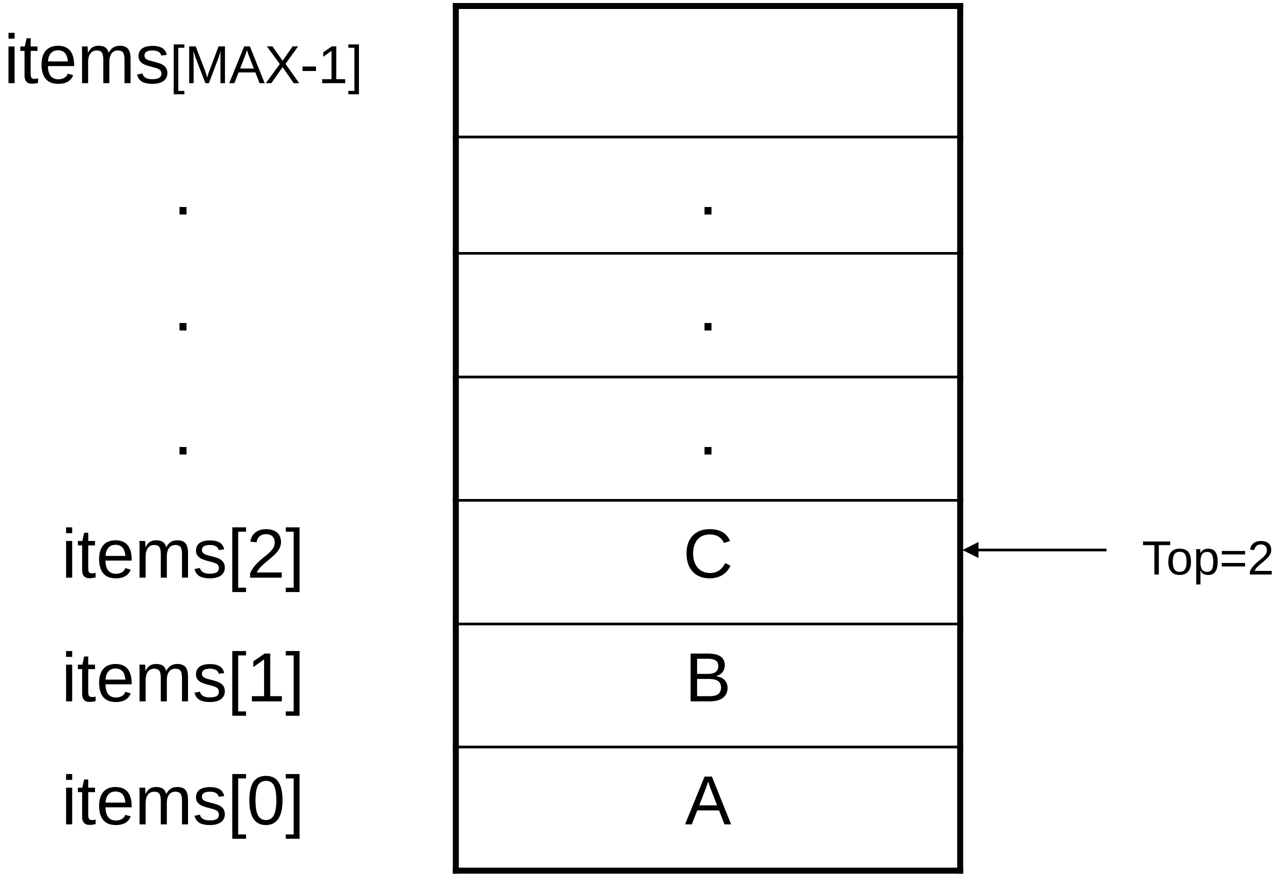
- A stack is a last-in-first-out (LIFO) data structure
- Adding an item
  - Referred to as pushing it onto the stack
- Removing an item
  - Referred to as popping it from the stack



Stack

- Definition:
  - An ordered collection of data items
  - Can be accessed at only one end (the top)
- Operations:
  - construct a stack (usually empty)
  - check if it is empty
  - Push: add an element to the top
  - Top: retrieve the top element
  - Pop: remove the top element

Stack



items[MAX-1]

.

.

.

items[2] C

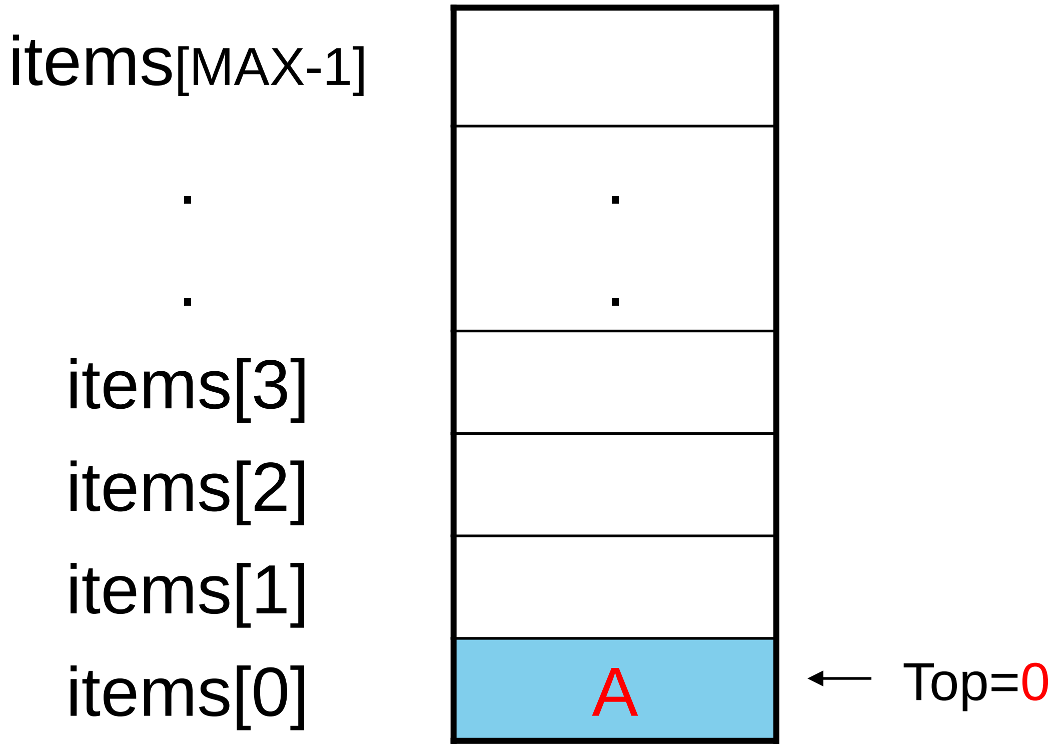
items[1] B

items[0] A

Top=2

Insert an item A

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



items[MAX-1]

.

.

items[3]

items[2]

items[1]

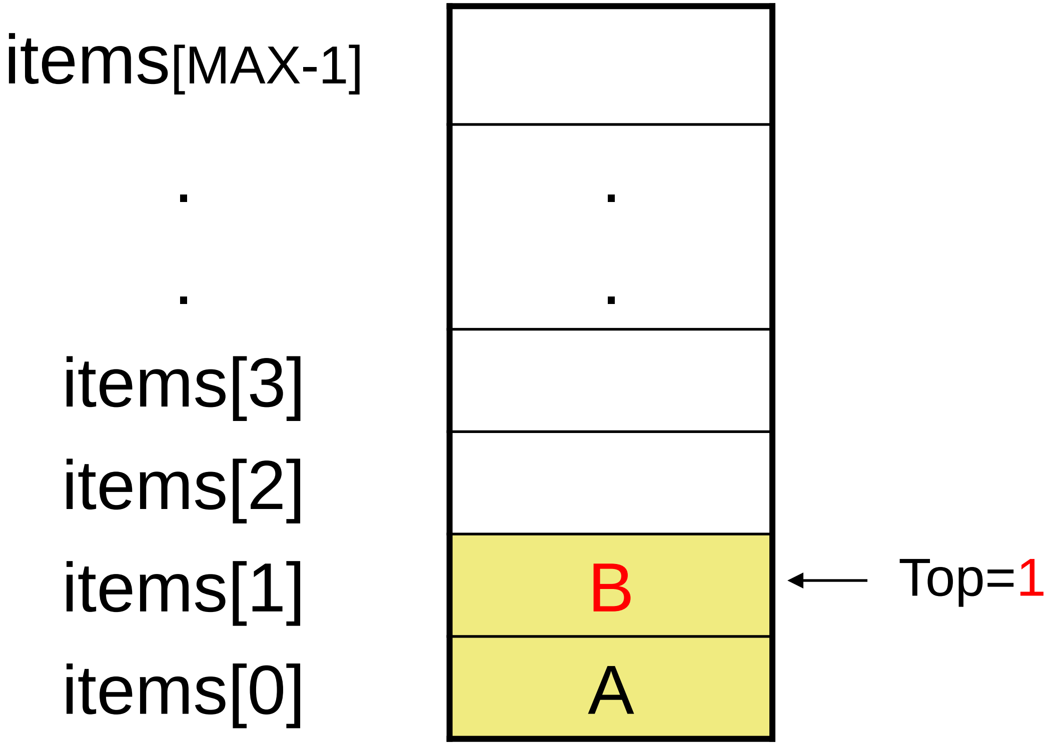
items[0]

A

Top=0

Insert an item B

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



items[MAX-1]

.

.

.

.

items[3]

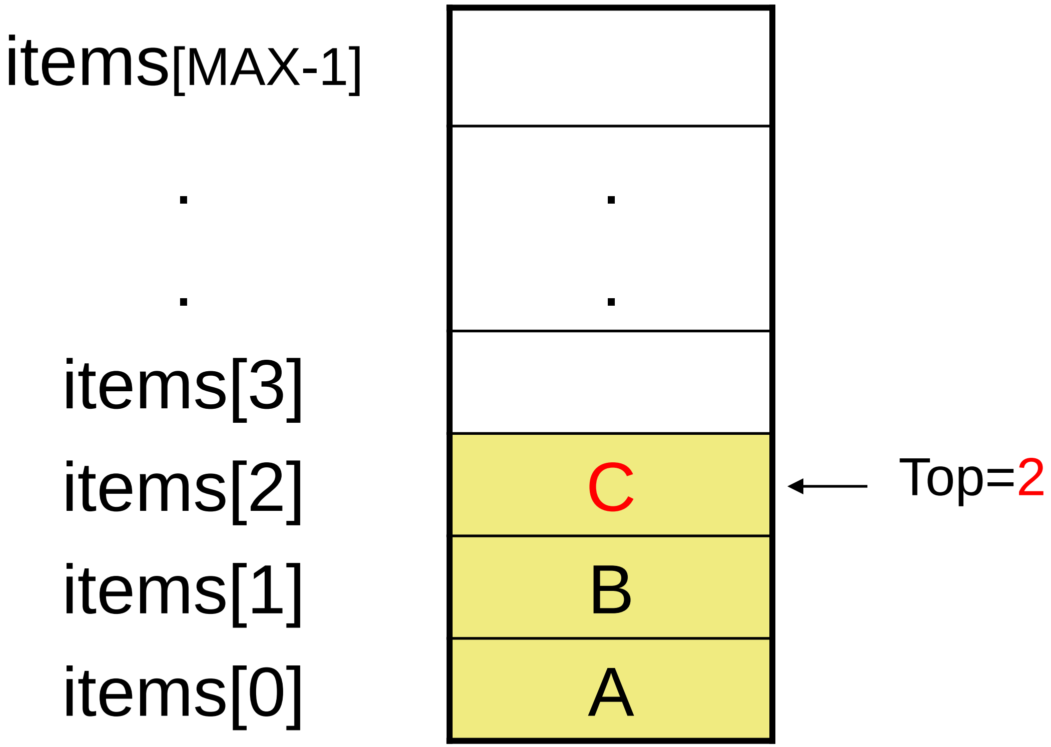
items[2]

items[1] B ← Top=1

items[0] A

Insert an item C

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



items[MAX-1]

.

.

items[3]

items[2]

items[1]

items[0]

.

.

C

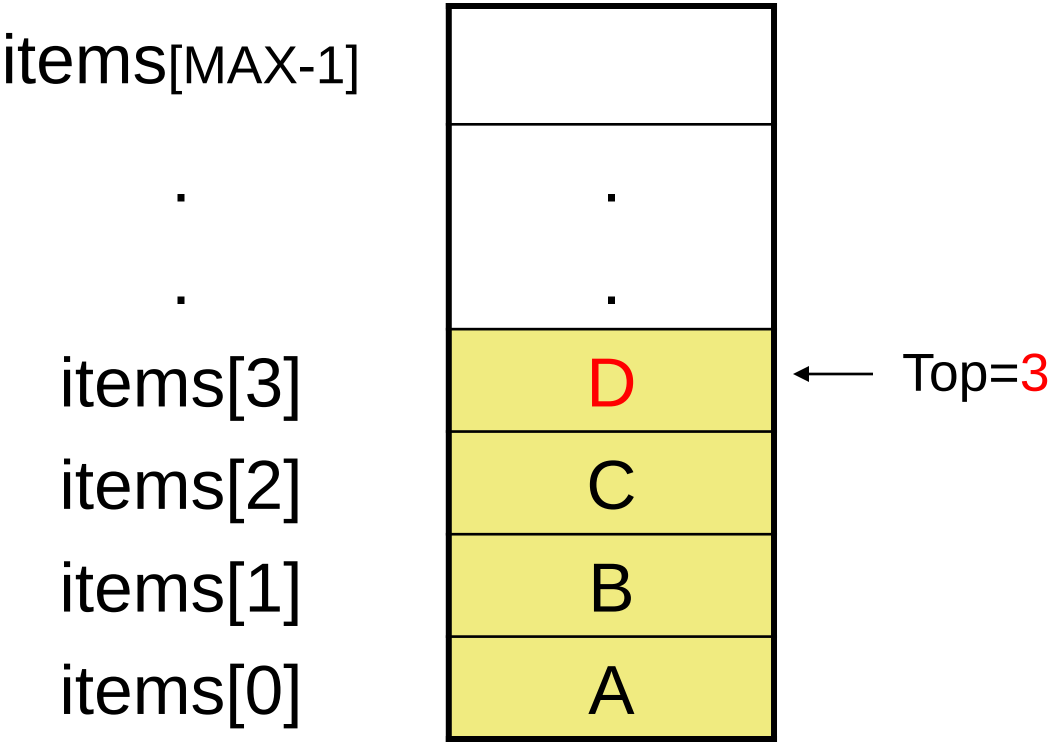
B

A

Top=2

Insert an item D

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



items[MAX-1]

.

.

items[3] D ← Top=3

items[2] C

items[1] B

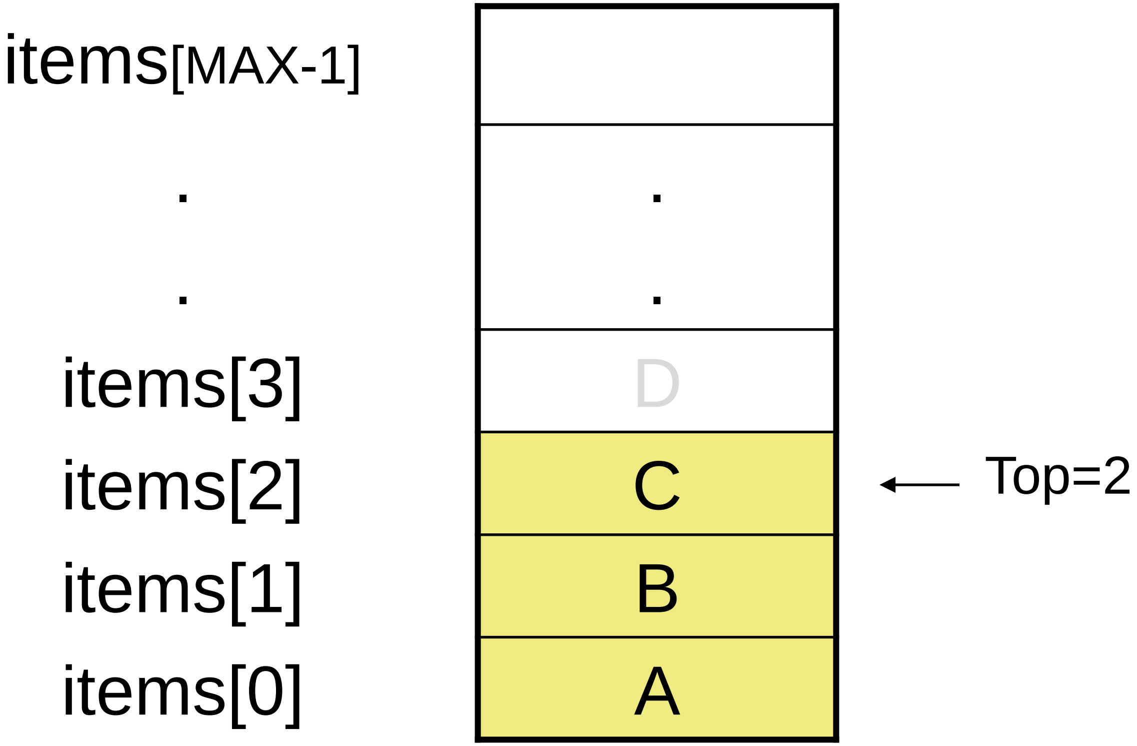
items[0] A

Insert Operation(Array)

```
PUSH(STACK, N, TOP, ITEM)
1.  If TOP=N:
        write OVERFLOW, and Return.
2.  Set TOP := TOP + 1.
3.  Set STACK[TOP]:= ITEM.
4.  Return.
```

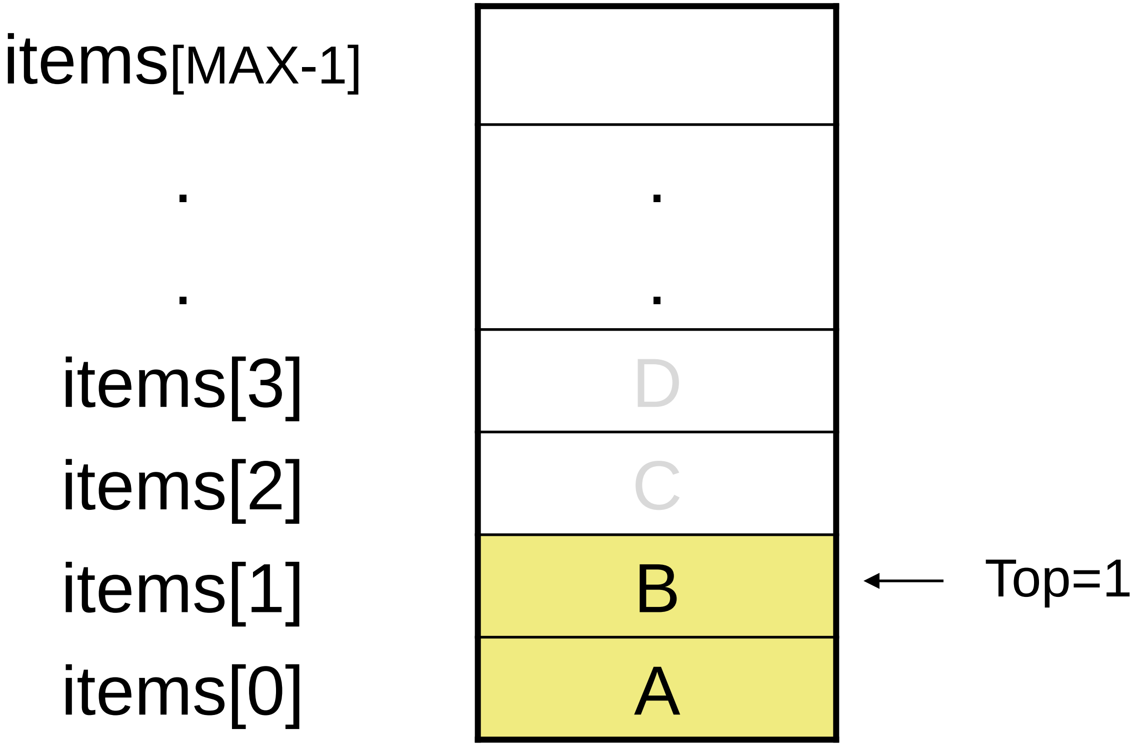
Delete D

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



Delete C

- **an item (*C*) is deleted from the *Top* of the stack.**



items[MAX-1]

.

.

items[3] D

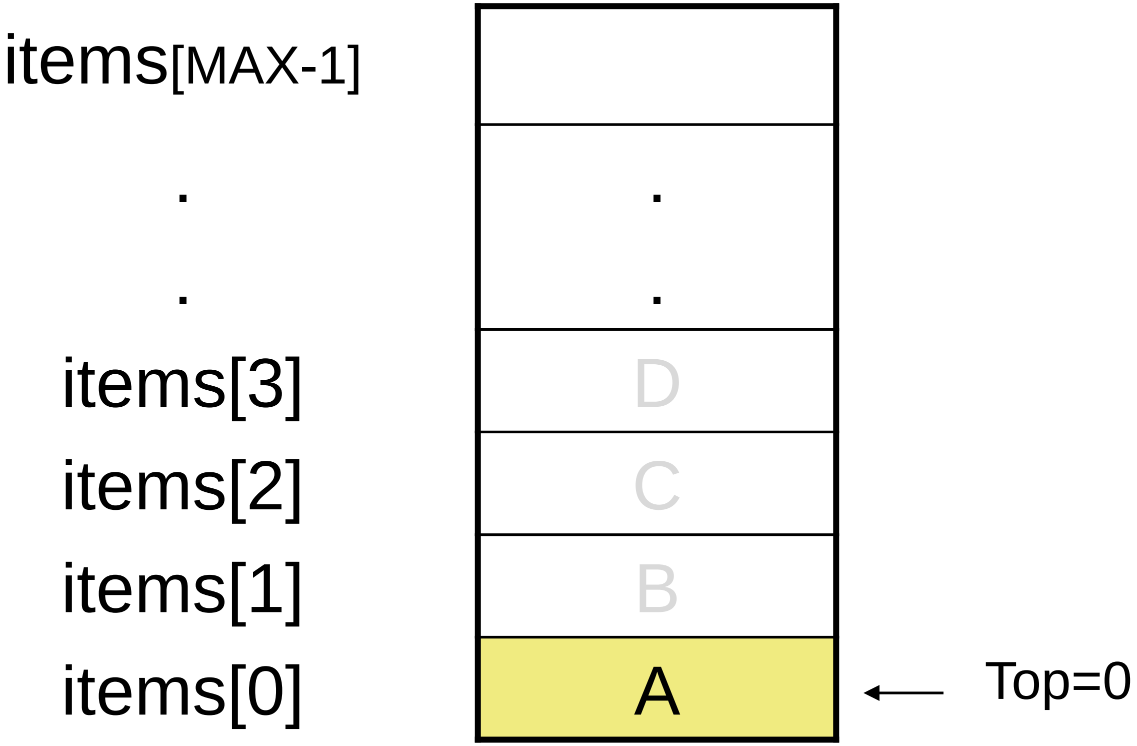
items[2] C

items[1] B ← Top=1

items[0] A

Delete B

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



items[MAX-1]

.

.

items[3] D

items[2] C

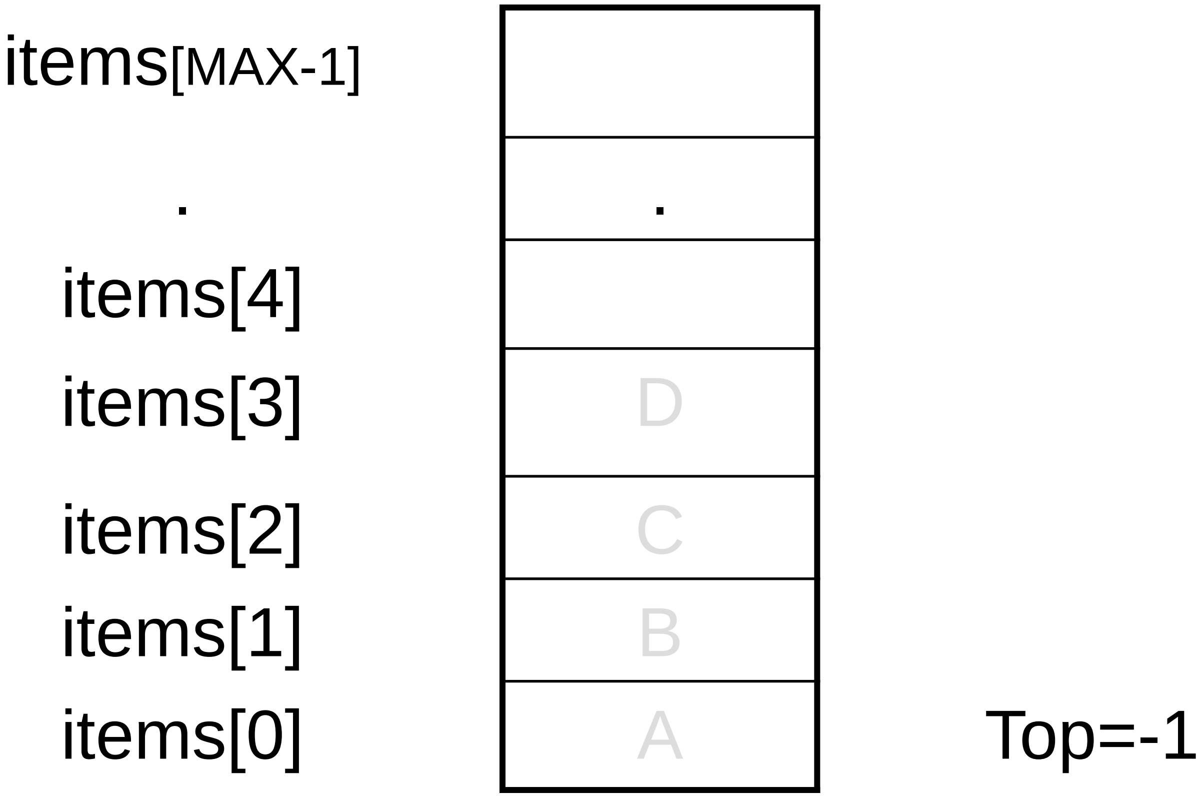
items[1] B

items[0] A

Top=0

Delete A

- **an item (*A*) is deleted from the *Top* of the stack.**



items[MAX-1]

.

items[4]

items[3]

items[2]

items[1]

items[0]

.

D

C

B

A

Top=-1

Delete Operation(Array)

```
POP(STACK, N, TOP, ITEM)
1.  If TOP = NULL then :
        write: UNDERFLOW, and Return.
2.  Set ITEM := STACK [TOP].
3.  Set TOP := TOP - 1.
4.  Return.
```

Insert Operation(LL)

```
Step 1: Allocate memory for the new node and name it as NEW_NODE

Step 2: SET NEW_NODE->DATA = VAL

Step 3: IF TOP = NULL
                SET NEW_NODE->NEXT = NULL
                SET TOP = NEW_NODE
            ELSE
                SET NEW_NODE->NEXT = TOP
                SET TOP = NEW_NODE
            [END OF IF]
Step 4: END
```

Delete Operation(LL)

```
POP(INFO, LINK, TOP, AVAIL, ITEM)
1.  If TOP = NULL, then :
        write UNDERFLOW, and Exit.
2.  Set TEMP := TOP and Set ITEM :=INFO[TEMP]
3.  TOP :=LINK[TOP]
4.  LINK[TEMP] = AVAIL  and AVAIL = TEMP
5.  Exit
```

Postfix Notation(RPN)

- Polish notation, also known as prefix 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***
  - the operators are written after the operands
- So a * b + c becomes a b * c +
- Advantage:
  - expressions can be written without parentheses

Postfix and Prefix 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 <u>before</u> the operands

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:

$$2\ \ 3\ \ 4\ \ +\ \ 5\ \ 6\ \ -\ \ -\ \ \ast$$

$$\rightarrow\ 2\ \ \underline{3\ \ 4\ \ +}\ \ 5\ \ 6\ \ -\ \ -\ \ \ast$$

$$\rightarrow\ 2\ \ 7\ \ 5\ \ 6\ \ -\ \ -\ \ \ast$$

$$\rightarrow\ 2\ \ 7\ \ \underline{5\ \ 6\ \ -}\ \ -\ \ \ast$$

$$\rightarrow\ 2\ \ 7\ \ -1\ \ -\ \ \ast$$

$$\rightarrow\ 2\ \ \underline{7\ \ -1\ \ -}\ \ \ast\ \ \rightarrow\ 2\ \ 8\ \ \ast\ \ \rightarrow\ \underline{2\ \ 8\ \ \ast}\ \ \rightarrow\ 16$$

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.
```

Evaluating RPN Expressions

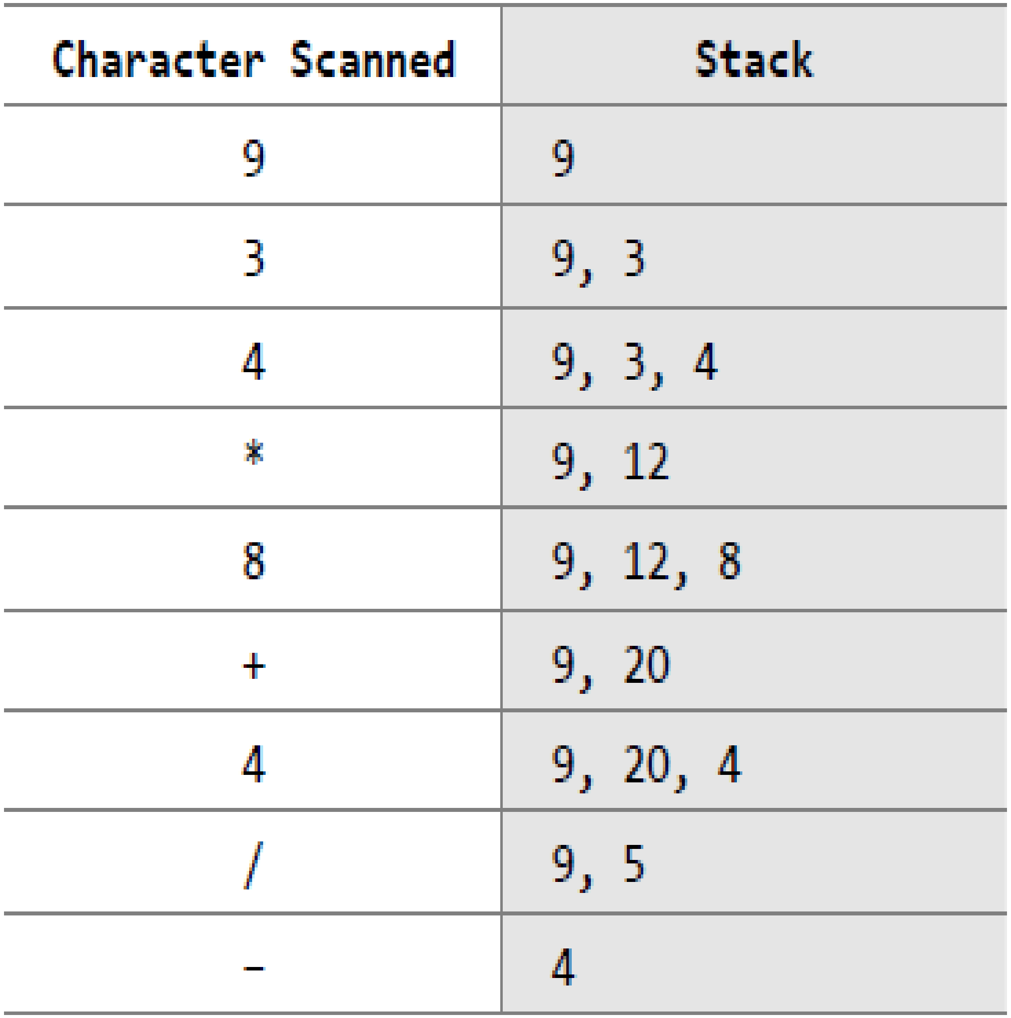
- Note the changing status of the stack
- infix expression

  9 – ((3 * 4) + 8) / 4

  Equivalent,

  Postfix Notation

  9 3 4 * 8 + 4 / –



Character Scanned	Stack
9	9
3	9, 3
4	9, 3, 4
*	9, 12
8	9, 12, 8
+	9, 20
4	9, 20, 4
/	9, 5
-	4

Transforming Infix into Postfix

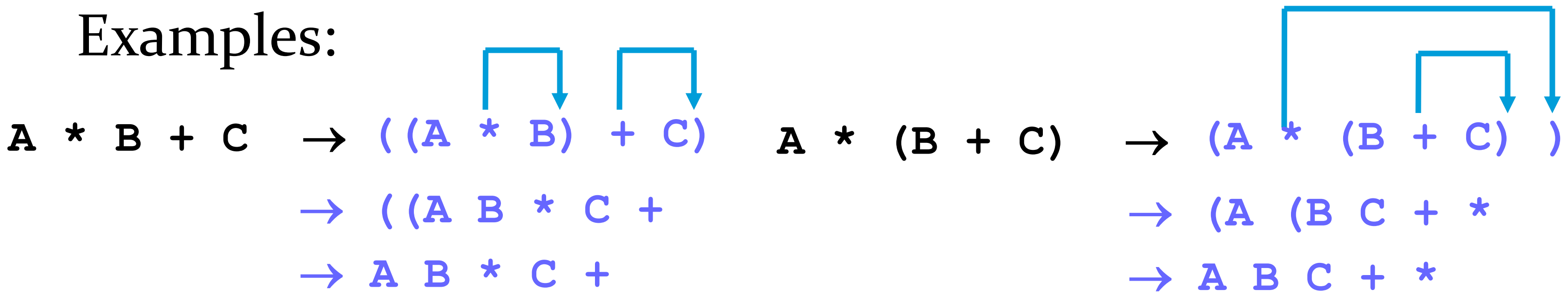
By hand: "Fully parenthesize-move-erase" method:

1. Fully parenthesize the expression.

2. Replace each right parenthesis by the corresponding operator.

3. Erase all left parentheses.

Examples:



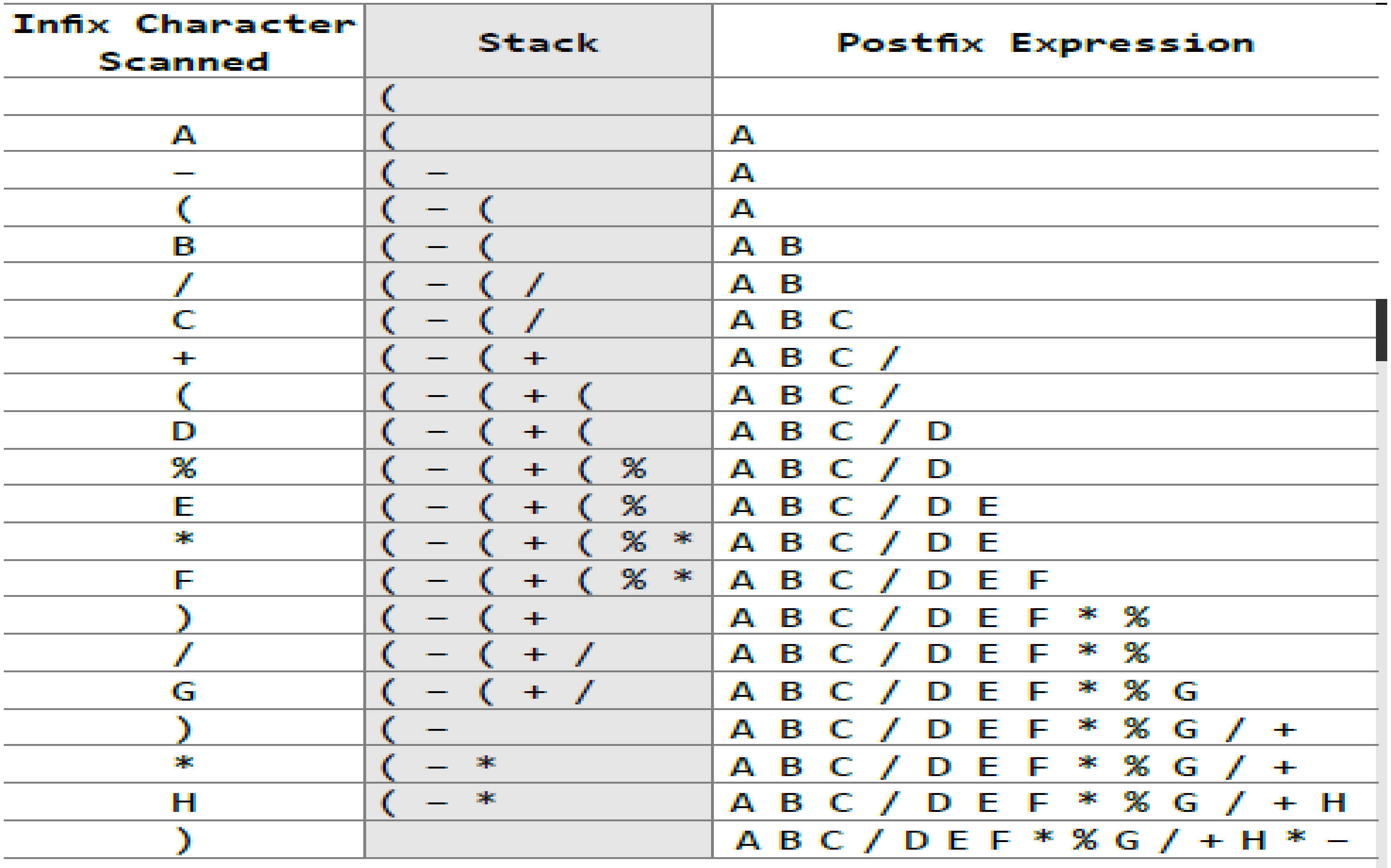
```
A * B + C  → ((A * B) + C)        A * (B + C)  → (A * (B + C) )
           → ((A B * C +                       → (A (B C + *
           → A B * C +                         → A B C + *
```

Stack Algorithm

```
POLISH (Q, P)
1.  PUSH "(" on to STACK and add ")" to the end of Q.
2.  Scan Q from left to right and Repeat steps 3 to 6 for each element of Q
    until the STACK is empty:
3.      If an operand is encountered, add it to P.
4.      If a left parenthesis is encountered, push it onto STACK.
5.      If an operator is encountered, then:
            (a) Repeatedly POP from STACK and add to P each operator
    (On the TOP of STACK) which has the same precedence as or higher
    precedence than @.
            (b) Add @ to STACK.
    [End of If structure.]
6.  If a right parenthesis is encountered, then:
        (a) Repeatedly POP from STACK and add to P each operator (On the
    TOP of STACK.) until a left parenthesis is encountered.
        (b) Remove the left parenthesis. [Don't add the left parenthesis to P.]
    [End of If Structure.]
    [End of step 2 Loop.]
7.  Exit.
```

(a) A – (B / C + (D % E * F) / G)* H

(b) A – (B / C + (D % E * F) / G)* H)



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 * –

Transforming Infix into prefix

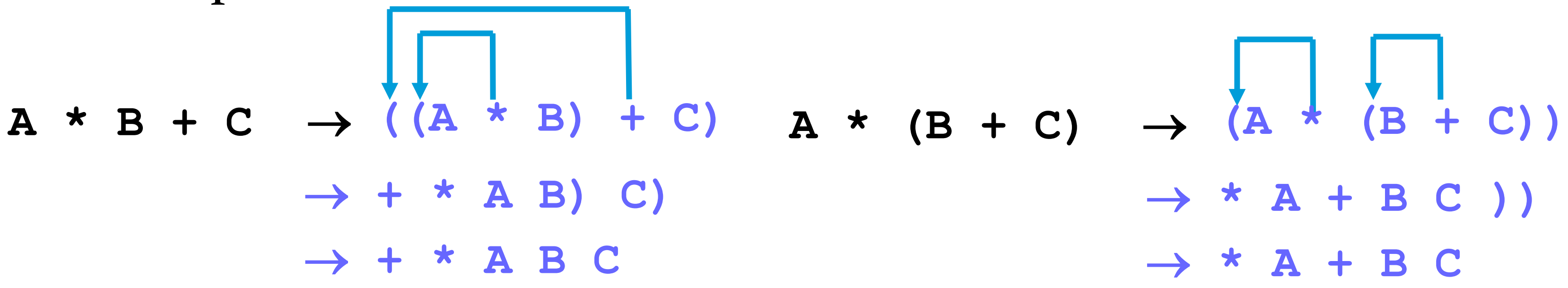
By hand: "Fully parenthesize-move-erase" method:

1. Fully parenthesize the expression.

2. Replace each left parenthesis by the corresponding operator.

3. Erase all right parentheses.

Examples:



```
A * B + C    → ((A * B) + C)        A * (B + C)    → (A * (B + C))
             → + * A B) C)                         → * A + B C ))
             → + * A B C                           → * A + B C
```

Thank You