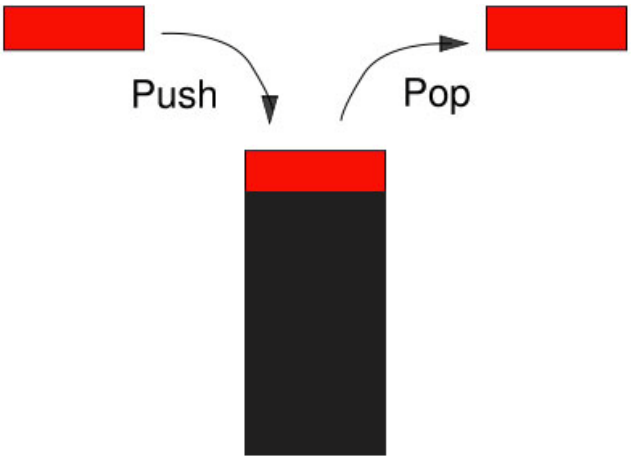
Stacks

- **Definitions:**
  - It is an LIFO ADT
  - It is a list of elements where a new element is inserted or pushed and deleted or popped only at the top of the list.
  - Only the element at the top of the list is accessed

- **Example:**



- **Operations:**
  - isEmpty: checks whether the stack is empty
  - Empty: empties the stack
  - Push: inserts an element at the top
  - Pop: deletes an element at the top
  - Top: Returns the value of the element at the top.
  - Etc.

- **Implementations:**
  - Arrays
  - Linked lists

Big-O Comparison of Stack Operations		
Operation	Array Implementation	Singly Linked Implementation
Class constructor	O(1)	O(1)
Empty The Stack	O(1)	O(N)
IsFull?	O(1)	O(1)
IsEmpty?	O(1)	O(1)
Push	O(1)	O(1)
Pop	O(1)	O(1)
Destructor	O(1)	O(N)

  - **Time Complexity**:

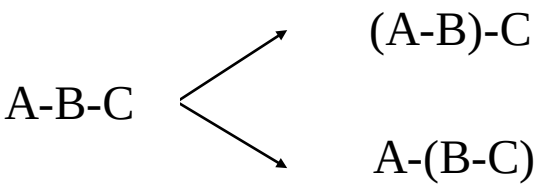
- **Application: Arithmetic Expression Evaluation**

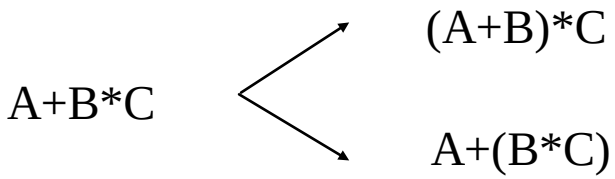
  - **Objective:**
    - Needs for a unique form of an expression
    - Compilers: simple expression evaluation procedures

  - **There are 3 types of representations of an expression:**
    - Infix notation
    - Postfix notations also reverse Polish notation (RPN)
    - Prefix notation also forward Polish notation (FPN)

  - **Infix Notation**

    - The operator is located between the operands to which it applies.
    - Example: A + B
    - Parentheses may be used to specify the order of operations.
    - Parentheses indicate the order in which operations are to be performed.
      - Example: A+(B*C)/(E+F)
    - Example:
      - In what order should we evaluate the expression?





- Associativity Rule:
  - When an expression or subexpression contains + or - , then it is evaluated from left to right. For example: (A-B)+C
  - Same for * and /. For example: (A*B)/C
- Precedence of Operators:
  - What happens if the expression contains different operators?
  - Solution: The operators are assigned priorities or precedence.

o **Postfix Notation**

- The operator is located after the operands to which it applies.
- No need for parentheses within an expression.
- Example: A + B ===> A B +

- **Infix to Postfix conversion algorithm:**
  - Step 1: Completely parenthesize the infix expression
  - Step 2: Move each operator to the space held by its corresponding closing parenthesis.
  - Step 3: Remove all parentheses
  - Example: A/B↑C+D*E-A*C
    - Step 1: (((A/(B↑C))+(D*E))-(A*C))
    - Step 2: (((A(BC↑/(DE*+(AC*-
    - Step 3: ABC↑/DE*+AC*-

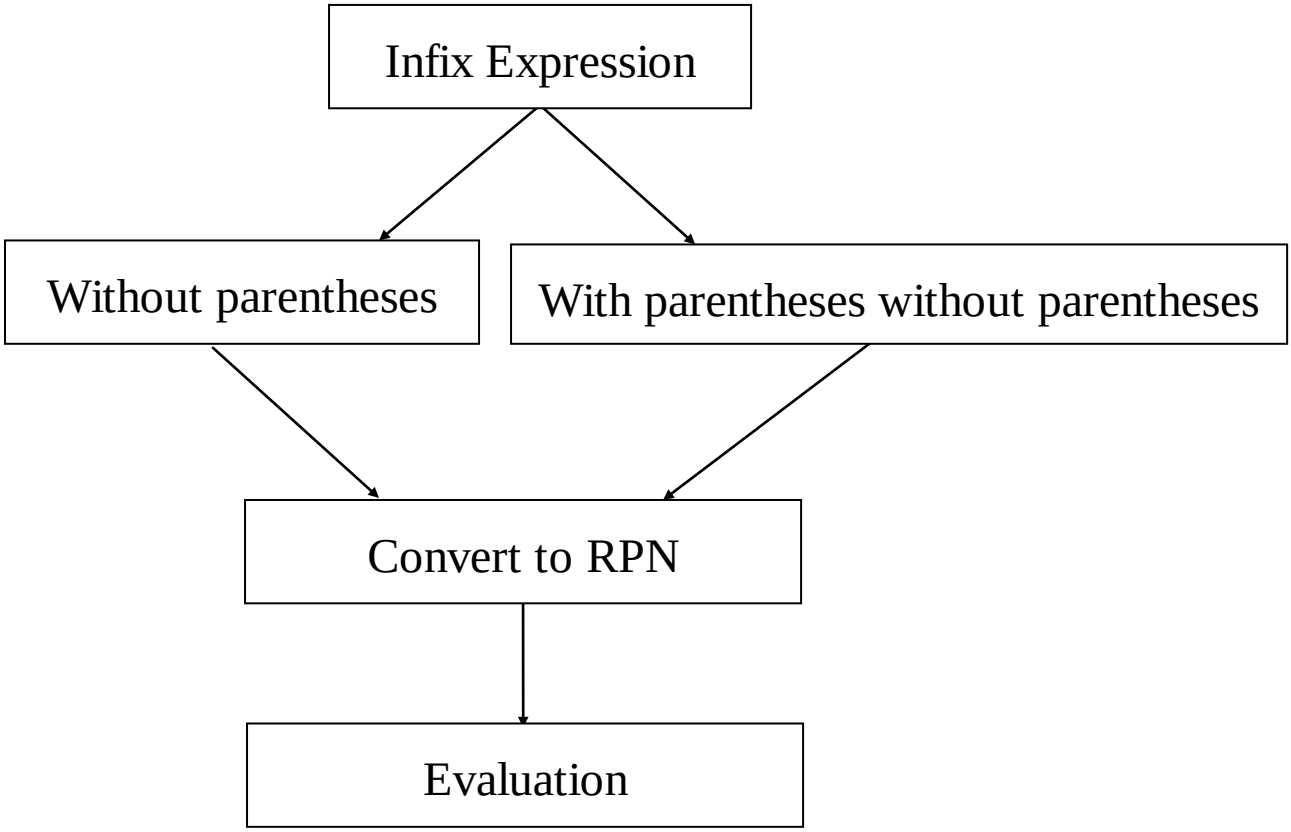
- **Prefix Notation**

  - The operator precedes its two operands.
  - No need for parentheses within an expression.
  - Example: A + B ===> + A B

- **Infix to Prefix conversion algorithm:**
  - Step 1: Completely parenthesize the infix expression
  - Step 2: Move each operator to the space held by its corresponding opening parenthesis.
  - Step 3: Remove all parentheses
  - Example: A/B↑C+D*E-A*C
    - Step 1: (((A/(B↑C))+(D*E))-(A*C))
    - Step 2: +/A↑BC))-*DE))*AC))
    - Step 3: +/A↑BC-*DE*AC

- **Programming Languages**:
  - Infix and Postfix notation are used in several interpreters & compilers
  - Lisp uses FPN.

- **Expression Evaluation**:



- **Parsing from Infix to Postfix**
  - Procedure RPN(string X; string result)

```
char C; string T; stack S;
begin
    X = T;  C = head(T);
    /* The infix exp. ends with a $ character */
    While (C !='$') do
    begin
        If C in (('A'..'Z') or ('a'..'z') or ('0'..'9'))
        then  result = result & C;
        else begin
            If C in ['+','-','*','/']
            then  if isempty(S)
                        then push (S,C);
                        else if priority(Top(S)<priority(C)
                                then push(S,C);
            else    begin
                    while (isempty(S) or (priority(Top(S)<priority(C))
                    begin
                        result = result & Top(S);
                        pop(S);
                    end;
                    push(S,C);
            end;
        end;
        C = head(T);
    end;
    while (!isempty(S)) do   begin
                result = result & Top(S);
                Pop(S);
        end;
end; /* procedure */
```

- **Example:**

2				2					
*				*					2
3				*					23
/				/					23*`
(				/	(				23*
2				/	(				23*2
-				/	(	-			23*2
1				/	(	-			23*21
)				/					23*21-
+				+					23*21-/
5				+					23*21-/5
*				+	*				23*21-/5
(				+	*	(			23*21-/5
4				+	*	(			23*21-/54
-				+	*	(	-		23*21-/54
1				+	*	(	-		23*21-/541
)				+	*				23*21-/541-
									23*21-/541-*+

- **Expression Evaluation**

```
int Function Eval_Exp(string X);
int V, Y, Z; char C; string T;
Stack S;
Begin
        T = X;
        C = head(T);
        while (C!='$') do
        begin
                If (C is a number)
                then  begin
                                V = convert_to_number(C);
                                push(S,V);
                        end;
                else  begin
                                Y = Top(S); Pop(S);
                                Z = Top(S); Pop(S);
                                Case C of
                                        '+':  Push(S,Z+Y);
                                        '-':  Push(S,Z-Y);
                                        '*':  Push(S,Z*Y);
                                        '/':  Push(S,Z/Y);
                                end;
                        end;
                C = head(T);
        end;
end;
```

- Lab Assignment:
  - Given the following array-based Stack class, complete the implementation of the missing methods.

```
public class Stack {
  // *** fields ***
    private static int INITSIZE;  // initial array size
    private int[] elements; // the elements in the stack
    private int numElements;   // the number of elements in the stack

  //*** methods ***

  // constructor
    public Stack(int s) {
      INITSIZE = s;
       elements = new int[INITSIZE];
       numElements = -1;

    }
   // add elements
    public void push(int element) {
        if (elements.length == numElements){
            System.out.println("StackOverflow");
            System.exit(0);
        }
        elements[++numElements] = element;
    }

  // remove element
    public int pop()   {
      //Implement this method.
      }

  // other methods
    public int top() {
      //Implement this method
    }
    public int size() {
            return numElements;

    }
    public boolean isEmpty() {
      //Implement this method
    }

    public void printStack() {
      if (isEmpty())
            System.out.println("Empty Stack");
      else{
            for(int i=0;i<=numElements; ++i){
                  System.out.println(elements[i]);
            }
      }
    }
}
```