# 8 - Stack

- A **stack** is a **linear data structure** that follows the **Last In, First Out (LIFO)** principle, where the last element inserted is the first to be removed.
- An element is inserted or deleted only at one end, called the top of the stack.

**Operations:**

- **Push :** to insert an element into a stack.
- **Pop :** delete an element from a stack.

**Use cases:**

- Postponed Decisions: You are working on Project A. To complete A, you must first complete Project B. You “postpone” finishing A and push it onto a stack. Then you complete B. Once B is done, you pop A from the stack and continue its processing.
- **Function calls in programming**.
- **Undo operations**.

## Array Representation of Stack

- Keep track of the top index. To push an element, increment the top index and store the element at that position. To pop an element, access the element at the top index and then decrement the top index.

**Algorithm:**

- **Push:**
  1. If $TOP = MAXSTK$, then: Print $\text{OVERFLOW}$ and return.
  2. Set $TOP := TOP + 1$.
  3. Set $STACK[TOP] := ITEM$.
  4. return.
- **Pop:**
  1. If $TOP = 0$, then Print: $\text{UNDERFLOW}$ and return.
  2. Set $ITEM := STACK[TOP]$.
  3. Set $TOP := TOP - 1$.
  4. return.

### Minimizing Stack Overflow

- Programmers cannot directly control overflow, but they choose the memory space reserved for each stack.
- Time-space tradeoff:
  - Reserving more space reduces overflow frequency but may waste memory.
  - Reserving less space saves memory but increases overflow occurrences and handling costs.
- Techniques exist to efficiently manage multiple stacks in a single array to use space more effectively.

## Linked List Representation

Used Singly Linked List.

**Algorithm:**

- **Push**
- **Pop**

# Application: Arithmetic Expression

## Polish Notation

- Different levels of precedence:

| | |
|---|---|
| **Highest** | **Exponentiation**$(\uparrow)$ |
| **Next highest** | **Multiplication**$(*)$ **and Division**$(/)$ |
| **Lowest** | **Addition**$(+)$ **and Subtraction**$(-)$ |

### Infix Notation:

- Operator position is between the operands.

### Prefix Notation:

- Operator is placed before its two operands.

$$A + (B + C) = A + [*BC] = +A * BC$$

$$(A + B)/(C - D) = [+AB]/[-CD] = / + AB - CD$$

### Postfix Notation / Suffix Notation:

- Operator is placed after its two operands.

$$A + (B * C) = A + [BC*] = A + BC*$$

$$(A + B)/(C - D) = [AB+]/[CD-] = AB + CD - /$$

## Infix to Postfix notation:

**Algorithm:**

1. Push "( " into $STACK$, and add ")" to the end of $Q$. $[\text{Q is the input expression}]$
2. Scan $Q$ from left to right and repeat Steps $3$ to $6$ for each element of $Q$ until the $STACK$ is empty:
    1. If an operand is encountered, add it to $P$.
    $[\text{P is the postfix expression}]$
    3. If a left parenthesis is encountered, push it onto $STACK$.
    4. If an operator $\oplus$ is encountered, then:
        1. Repeatedly pop from $STACK$ and add to $P$ each operator which has the same precedence as.
        2. Add $\oplus$ to $STACK$.
    5. If a right parenthesis is encountered then:
        1. Repeatedly pop from $STACK$ and add to $P$ each operator until a left Parenthesis is encountered.
        2. Remove the left parenthesis.
6. Exit.

**Bangla vashate:**

- shurute stack e "(" push korte hobe r $Q$ er sheshe ")" push korte hobe.
- then $Q$ ke left to right check dite hobe,
    - jodi kono operand pai $P$ te push kore dibo
    - jodi kono operator pai tahole stack er operator pop korte thakbo jotokhon current tar theke higher or same precedence er kono operator pai $P$ te push korte thakbo. then current ta $P$ te push korbo.
    - jodi "(" pai tahole stack e push kore dibo
    - jodi ")" pai tahole stack theke "(" pop kore felbo.
- stack empty hoye gele exit korbo.

## Evaluate the value of an arithmetic expression $P$ written in postfix notation

**Algorithm:**

1. Add ")" at the end of P.
2. Scan $P$ from left to right and repeat steps $3$ and $4$ for each element of $P$ until the sentinel ") " encountered.
3. If an operand is encountered, put it on $STACK$.
4. If an operator $\oplus$ is encountered, then:
    1. Remove the two top element of $STACK$, where $A$ is the top element and $B$ is the next-to-top element.
    2. Evaluate $B \oplus A$.
    3. Place the result of $(2)$ back on $STACK$.
5. Set $VALUE$ equal to the top element on $STACK$.
6. Exit.