Data Structures and Algorithms

STACK

PRASHANT HEMRAJANI

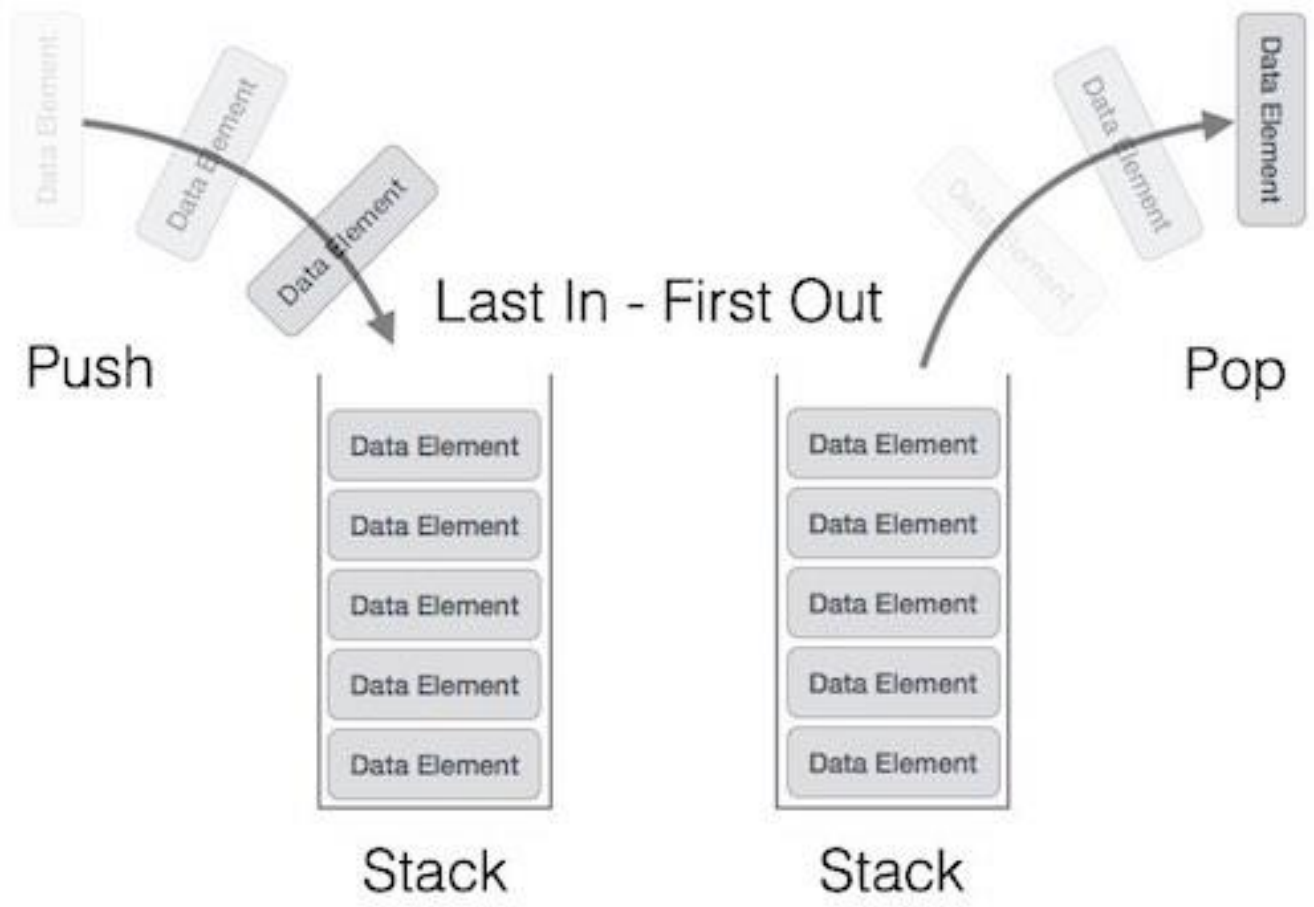
Assistant Professor

(Computer and Communication Engineering)

Introduction

- A stack is one of the most commonly used data structure.
- A stack also called a Last In First Out (LIFO) System is a linear list in which insertions and deletions can take place only at one end called the top.
- The insertion and deletion operations in stack terminology are known as push and pop operations.

Representation



Operations

1. **Push()**
   - It is the term used to insert an element into a stack.
2. **Pop()**
   - It is the term used to delete an element from a stack.
3. **peek()**
   - Get the top data element of the stack, without removing it.
4. **isFull()**
   - Check if stack is full.
5. **isEmpty()**
   - Check if stack is empty.

Applications

1. Expression Evaluation
2. Backtracking (game playing, finding paths, exhaustive searching)
3. Memory management, run-time environment for nested language features.
4. Language processing:
   - Space for parameters and local variables is created internally using a stack.
   - Compiler's syntax check for matching braces is implemented by using stack.
   - Support for Recursion (E.g. Tower of Hanoi etc.)
5. Depth-First Search in Tree

Push (Array Representation)

PUSH (STACK, TOP, MAXSTK, ITEM)

This procedure pushes an ITEM onto a stack.

Step 1 : [Stack already filled?]

If TOP= MAXSTK,

then: Print: “OVERFLOW”, and Exit.

Step 2 : Set TOP: = TOP +1. [Increases TOP by 1.]

Step 3 : Set STACK [TOP]: = ITEM.

[Inserts ITEM in new TOP position.]

Step 4 :Exit

Pop (Array Representation)

POP (STACK, TOP, ITEM)

This procedure deletes the top element of STACK and assigns it to the variable ITEM.

Step 1 : [Stack has an item to be removed?]
IF TOP = 0,
then: Print: “UNDERFLOW”, and Exit

Step 2 : Set ITEM: = STACK [TOP].
[Assigns TOP element to ITEM.]

Step 3 : Set TOP: = TOP- 1. [Decreases TOP by 1.]

Step 4 : Exit

Push (Linked Representation)

PUSH (TOP, AVAIL, LINK, INFO, ITEM)

This procedure adds a new element at the TOP of a STACK using the link list.

Step I: - [OVERFLOW?]

If AVAIL = NULL, then: Write : OVERFLOW and EXIT.

Step II: - [Remove the first node from AVAIL list.]

Set NEW:= AVAIL, AVAIL:= LINK [AVAIL]

Step III: -Set INFO [NEW]:= ITEM

Step IV: -Set LINK [NEW]:= TOP.

Step V: - Set TOP:= NEW.

Step VI: -Exit.

Pop (Linked Representation)

POP (TOP, LINK, INFO, ITEM)

This procedure removes the element form the TOP of a stack using the link list.

Step I: - [UNDERFLOW?]

If TOP= NULL, then : write; UNDERFLOW and Exit.

Step II: - Set PTR:= TOP

Step III: -Set TOP = LINK [TOP]

Step IV: - Set ITEM:= INFO [PTR]

Step V: - Set LINK [PTR]: = AVAIL.

Step VI: -Set AVAIL: = PTR.

Step VII: -Exit.

Expression Parsing

- The way to write arithmetic expression is known as a **notation**.
- An arithmetic expression can be written in three different but equivalent notations, i.e., without changing the essence or output of an expression.
- These notations are −
  1. Infix Notation
  2. Prefix (Polish) Notation
  3. Postfix (Reverse-Polish) Notation

Infix Notation

- We write expression in **infix** notation, e.g. a - b + c, where operators are used **in**-between operands.
- It is easy for us humans to read, write, and speak in infix notation but the same does not go well with computing devices.
- An algorithm to process infix notation could be difficult and costly in terms of time and space consumption.

Prefix and Postfix Notations

Prefix Notation

- In this notation, operator is **prefix**ed to operands, i.e. operator is written ahead of operands.
- For example, **+ab**. This is equivalent to its infix notation **a + b**. Prefix notation is also known as **Polish Notation**.

Postfix Notation

- This notation style is known as **Reversed Polish Notation**. In this notation style, the operator is **postfix**ed to the operands i.e., the operator is written after the operands.
- For example, **ab+**. This is equivalent to its infix notation **a + b**.

Infix to Prefix and Postfix Conversion

Sr. No.	Infix Notation	Prefix Notation	Postfix Notation
1	a + b	+ a b	a b +
2	(a + b) ∗ c	∗ + a b c	a b + c ∗
3	a ∗ (b + c)	∗ a + b c	a b c + ∗
4	a / b + c / d	+ / a b / c d	a b / c d / +
5	(a + b) ∗ (c + d)	∗ + a b + c d	a b + c d + ∗
6	((a + b) ∗ c) - d	- ∗ + a b c d	a b + c ∗ d -

Infix to Postfix

INTOPOST(Q, P)

Suppose Q is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression P.

1. Push "(" onto 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 then *.
   (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. [Do not add the left parenthesis to P].
   [End of if structure]
   [End of step 2 loop]
7. Exit.

Infix to Prefix

INTOPRE (Q, P)

Suppose Q is an arithmetic expression written in infix notation. This algorithm finds the equivalent prefix expression P.

1. Push “)” onto STACK, and add “(” to the beg of Q.
2. Scan Q from right to left 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 an right 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 higher precedence then *.
   (b) Add * to STACK.
   [End of If structure]
6. IF a left parenthesis is encountered, then:
   (a) Repeatedly pop from STACK and add to P each operator (on the top of STACK) until a right parenthesis is encountered.
   (b) Remove the right parenthesis. [Do not add the right parenthesis to P].
   [End of If structure.]
   [End of step 2 loop].
7. Reverse P
8. Exit.

Evaluation of Postfix Notation

EVALPOST (P, VALUE)

This algorithm finds the VALUE of an arithmetic expression P written in postfix notation.

1. Add a right parenthesis ")" at the end of P. [This acts as a sentinel].
2. Scan P from left to right and Repeat Steps 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 element 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 step 2 loop]
5. Set VALUE equal to the top element on STACK.
6. Exit.

Evaluation of Prefix Notation

EVALPRE (P, VALUE)

This algorithm finds the VALUE of an arithmetic expression P written in prefix notation.

1. Add a left parenthesis “(” at the beg of P. [This acts as a sentinel.]
2. Scan P from right to left and Repeat Steps 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 element 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 2 loop.]
5. Set VALUE equal to the top element on STACK.
6. Exit.

Any Queries
????