Data Structures

Prof. Anand Jain, anand.jain@pccoepune.org

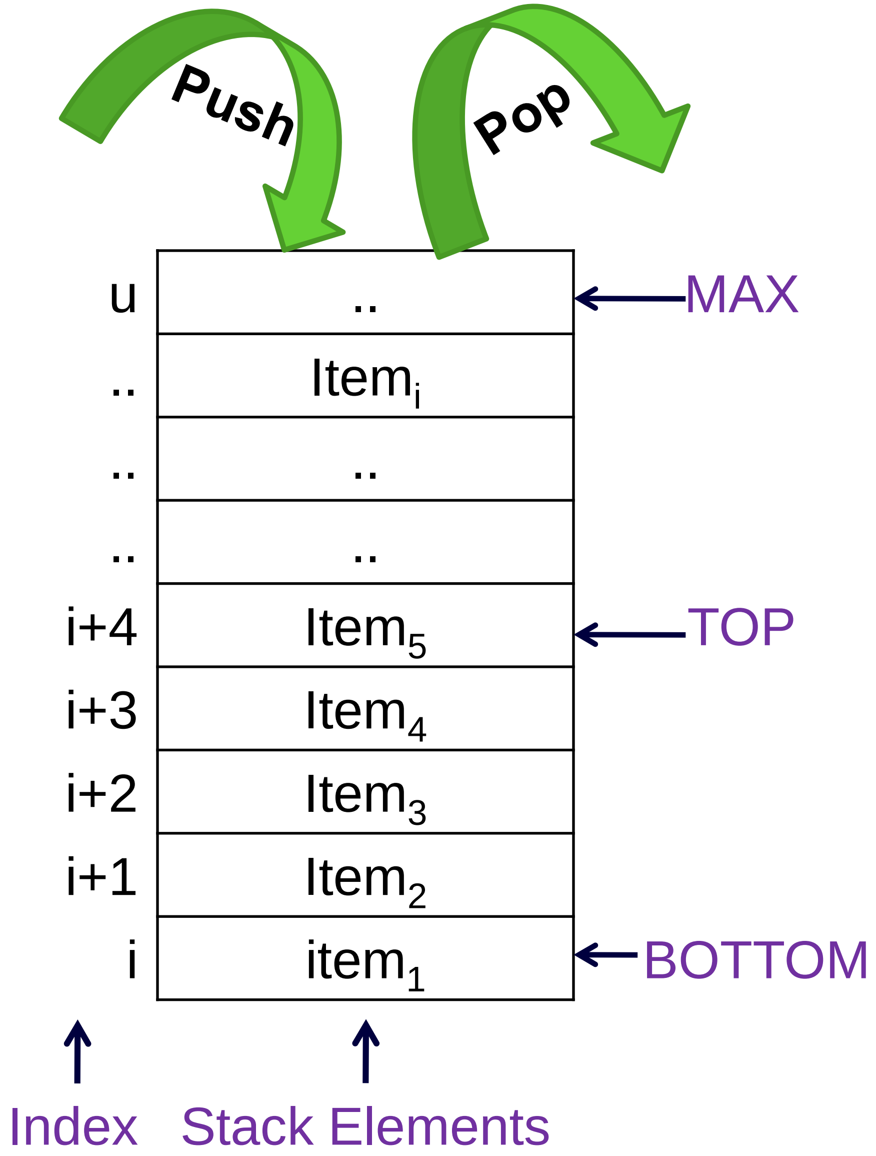
Unit 02 : Stack

Stack

- **Stack Introduction**
  - A linear list data structure which permits insertion or deletion of an element to occur only at one end is called as Stack.
  - Insert operation is referred as Push, Delete operation is referred as Pop
  - The most and least accessible elements of stack are referred as TOP and BOTTOM of stack respectively. Push and Pop are performed at TOP
  - Since insertion and deletion are allowed to be performed from one end only, the elements can be removed in opposite order from that they were added. Because of this, stack is also called as LIFO (Last In First Out) list.

Stack

- **Array Representation of Stack**
  - $Item_i$ denotes the ith item in the stack
  - i and u denotes index range
  - TOP is the stack position which is used to perform push and pop operations
  - BOTTOM indicates first element of the stack
  - MAX indicates maximum capacity of stack
- Array representation is **easy and convenient to implement**
- Array allows **representation of only fixed sized stack**



Stack

- **Linked List Representation of Stack**

Stack

- **Stack Operations**
  - Push and Pop are the primary operations of stack
    - Push : Pushing (storing) an element on the stack
    - Pop : Removing an element from the stack
  - We need to check the status of stack to make an efficient use of it
    - Peek : Get the top element of stack, without removing it
    - isFull : Check if stack is full. It is necessary before pushing an element
    - isEmpty : Check if stack is empty. This is necessary before performing pop operation

Stack

❖ **Stack Operations**

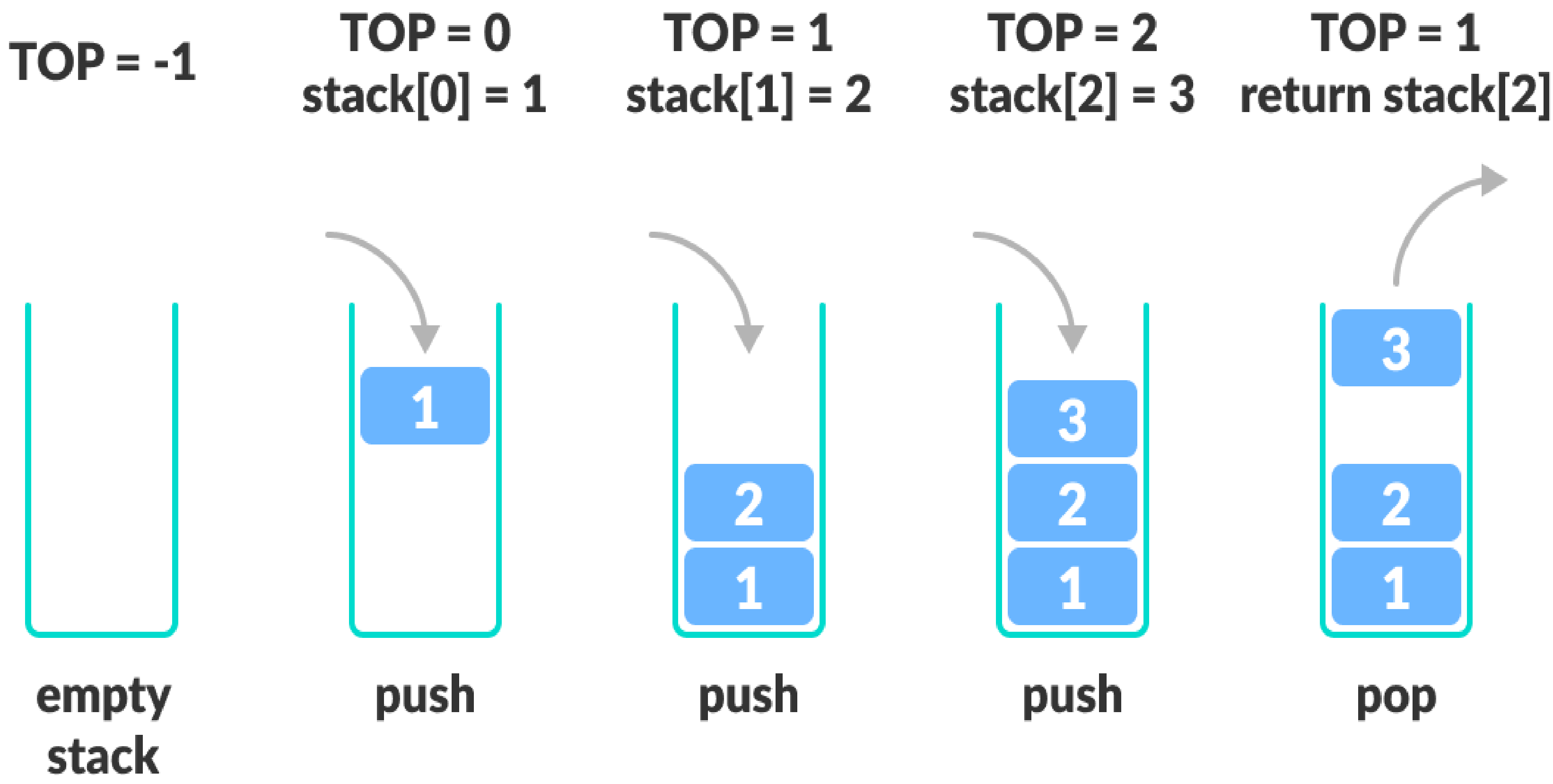
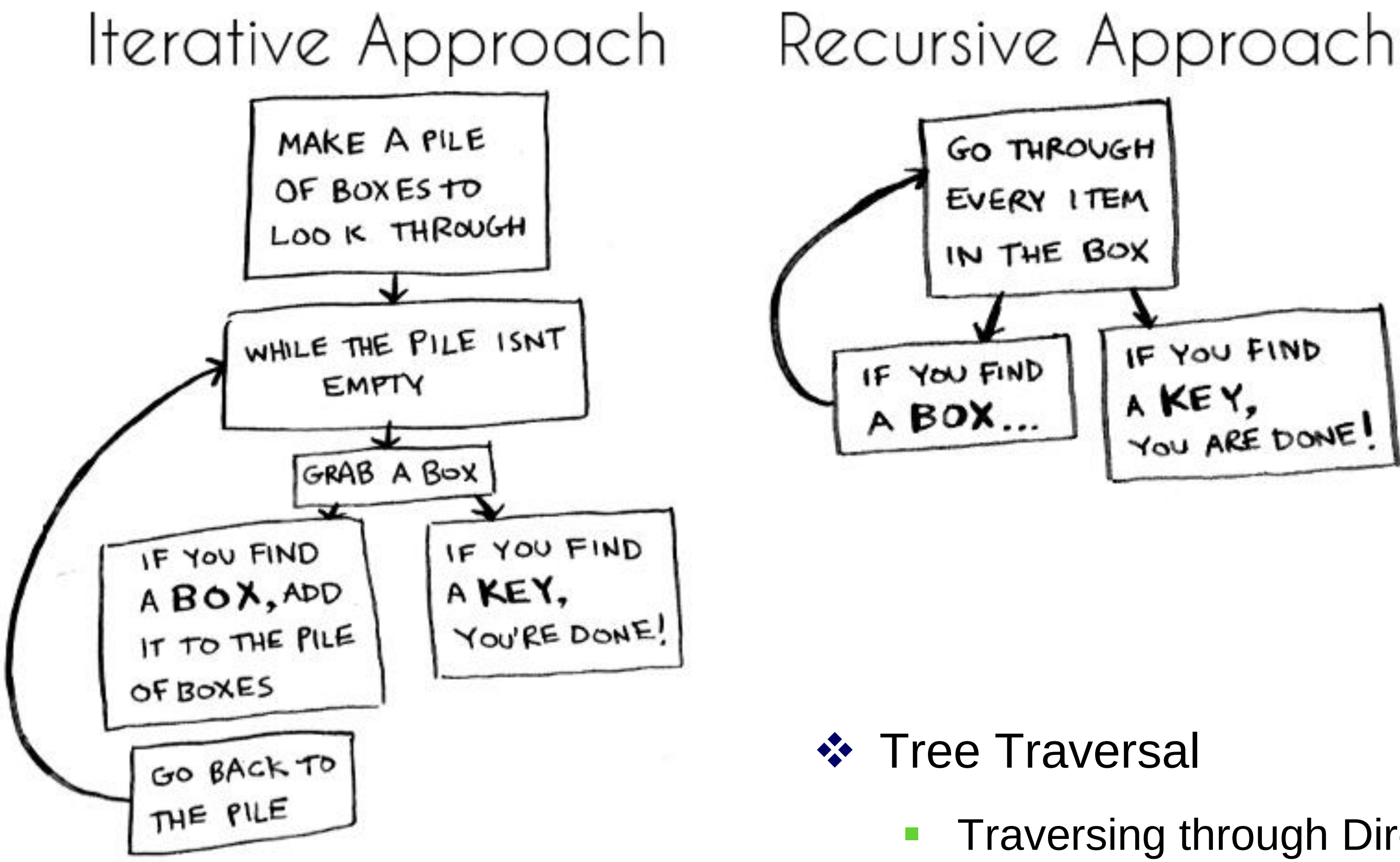


Image Source : https://www.programiz.com/dsa/stack

Applications of Stack

- **Recursion**
  - Recursion is a method of solving problems based on the divide and conquer mentality. The basic idea is that you take the original problem and divide it into smaller (more easily solved) instances of itself, solve those smaller instances (usually by using the same algorithm again) and then reassemble them into the final solution.
  - Recursion is especially good for working on things that have many possible branches and are too complex for an iterative approach.
  - Recursion is useful for the languages (For Ex. : Clojure) that do not support loop statements.
  - Recursion is a useful tool, but it can increase memory usage; and may crash the program due to stack overflow

Applications of Stack

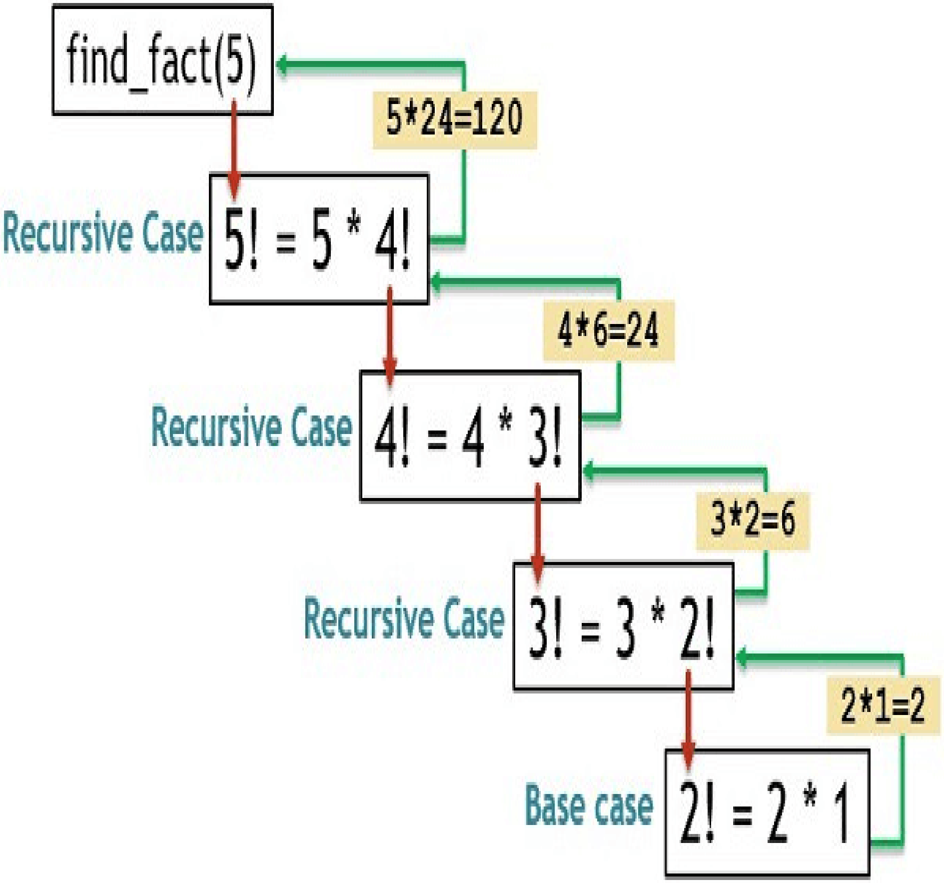


- ❖ Tree Traversal
  - Traversing through Directories

Applications of Stack

❖ **Recursion to Find Factorial of a Number**

```
int find_fact(unsigned int i)
{
 if(i <= 1)
 {
  return 1;
 }
 return i * find_fact (i - 1);
}
```



Applications of Stack

- **Evaluation of Arithmetic Expressions**
  - Operands are variables or constants
  - Operators indicate the operations to be performed on the operands
  - Operator Precedence and Operator Associativity

Level	Operator	Associativity
1.	! unary - ++ --	Right to left
2.	* / %	Left to right
3.	+ -	Left to right
4.	< <= > >=	Left to right
5.	== !=	Left to right
6.	&&	Left to right
7.	||	Left to right
8.	= += -= *= /=	Right to left

Applications of Stack

- **Notations for Arithmetic Expressions**
  - Infix Notation : When operator appears in between the operands then the expression is called as an infix expression. For Ex, A + B
  - Prefix Notation : When operator appears before the operands then the expression is called as a prefix expression. For Ex, +AB. Prefix notation is introduced by Polish mathematician Jan Lukasiewicz and hence also termed as Polish Notation
  - Postfix Notation : When operator appears after the operands then the expression is called as a postfix expression. For Ex, AB+. It is also termed as Reversed Polish Notation

Applications of Stack

- **Why Prefix/Postfix Notations**
  - Infix notation is easy to understand/read for humans; but additional load of operator precedence is associated with it. Prefix/Postfix notation is easier to parse for a machine, and there is no question of operator precedence.
  - Time Complexity
    - Evaluation of Infix Notation → O(n^2)
    - Infix to Postfix → O(n)
    - Postfix Evaluation → O(n)
    - O(n) + O(n) = O(n)

Applications of Stack

- Infix to Postfix Conversion using Stack

Rules for Infix to postfix using stack DS -

1) Scan Expression from Left to Right
2) Print OPERANDs as the arrive
3) If OPERATOR arrives & Stack is empty, push this operator onto the stack
4) IF incoming OPERATOR has HIGHER precedence than the TOP of the Stack, push it on stack
5) IF incoming OPERATOR has LOWER precedence than the TOP of the Stack, then POP and print the TOP. Then test the incoming operator against the NEW TOP of stack.
6) IF incoming OPERATOR has EQUAL precendence with TOP of Stack, use ASSOCIATIVITY Rules.
7) For ASSOCIATIVITY of LEFT to RIGHT - POP and print the TOP of stack, then PUSH the incoming OPERATOR
8) For ASSOCIATIVITY of RIGHT to LEFT - PUSH incoming OPERATOR on stack.
9) At the end of Expression, POP & print all OPERATORs from the stack


Thank You !