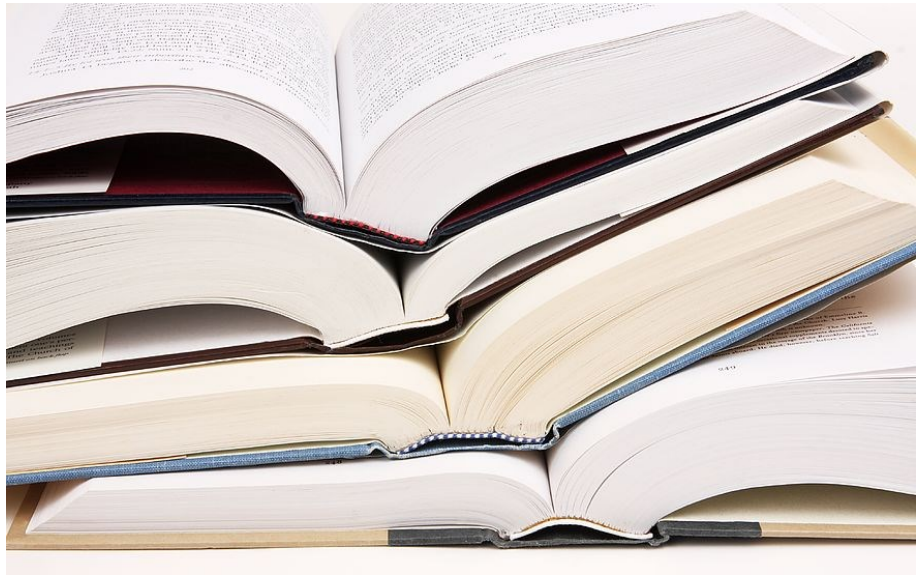
# Data Structure

# Stack

Shin Hong

27 Mar 2023



DS&A. Chapter 5.1. Stacks

# Stack

- A stack is a container of objects that are inserted and removed according to Last-In-First-Out (LIFO) principle
  - the top of the stack holds the latest value

- Basic operations
  - push(): add a new element to the top (i.e., writing)
  - pop(): retrieve the element at the top (i.e., reading)

- A stack can be implemented with a list
  - push: insert a new element as the first element
  - pop: remove the first element

# Stack Abstract Data Type

- Operations
  - *push*(*e*): insert element e at the top of the slack
  - *pop*(): remove the top element from the stack
  - *top*(): get a reference to the top element
  - *size*(): return the number of elements in the stack

| *Operation* | *Output* | *Stack Contents* |
|---|---|---|
| push(5) | – | (5) |
| push(3) | – | (5,3) |
| pop() | – | (5) |
| push(7) | – | (5,7) |
| pop() | – | (5) |
| top() | 5 | (5) |
| pop() | – | () |
| pop() | *"error"* | () |
| top() | *"error"* | () |
| empty() | *true* | () |
| push(9) | – | (9) |
| push(7) | – | (9,7) |
| push(3) | – | (9,7,3) |

# Stack and Recursive Algorithm

- A recursive algorithm divides a problem into sub-problems and defines the solution of the problem by combining the solutions of sub-problems
  - solve a problem by dividing it into many sub-problems of the same structure
  - the same logic should be applied for a super-problem and its sub-problems

- A stack can be used for holding solutions of sub-problems derived by a recursive algorithm
  - a stack can be used as the memory of a hierarchical problem-solving process

# Ex. Postfix Expression Evaluation

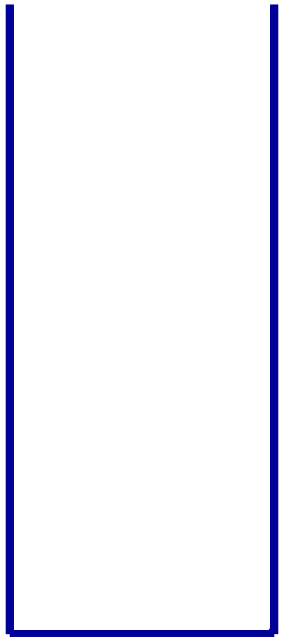
- An arithmetic expression is a value, or two expressions (i.e., operands) connected with an operator
  - operator: +, -, /, *

- Postfix notation represents an arithmetic expression by placing the operands (if exists) before than the operator
  - all terms (e.g., operators, operands) are separated with whitespace
  - ex: `3 6 + 2 4 - * 7 + = -11`

# Ex. Postfix Expression Evaluation

- Store the two operands in the stack for the next operator
  - Top-most and 2nd top-most elements are the operands

- Push the evaluation result back to the stack for the next operator
  - Ex. 3 6 + 2 4 - * 7 +



# Ex. Matching Parentheses

- We call a string of parentheses well-formed (or matching) if each opening symbol is matched with its corresponding closing symbol
  - parenthesis: “(“ and “)”
  - braces: “{“ and “}”
  - brackets: “[“ and “]”
  - angle brackets “<“ and “>”

- A string is well-formed if and only if:
  - an empty string is well-formed
  - the pair of opening and closing parenthesis may surround nothing, or another well-formed string
  - a well-formed string may be followed by another well-formed string

# Ex. Maze Pathfinding

- Find a path that consists of vertical and/or horizontal lines from the top-left corner to the bottom-right corner
  - a player can move up, down, left or right to an empty cell
  - assume that no cycle exists
- Store the current path in a stack
  - each stack element represents the exploration status at a cell