**ABSTRACT DATA TYPES**

**Stacks**

* Stack is an **Abstract Data Type** (ADT).
* ADT are data types which contain values, and we can apply certain special operations on it.
* It is called **abstract** as information are hidden from user.
* ADT is made with **primitive data types**.
* **peek()**: Returns top element without removing it.
* **isFull()**: Checks if stack is full.
* **isEmpty()**: Checks is stack is empty.
* In any **push()** or **pop()** operation, always print if stack is full/empty/successful/failure etc.

**Application Of Stacks**

* Recursion
* Polish expression
* Reverse Polish expression

**Recursion**

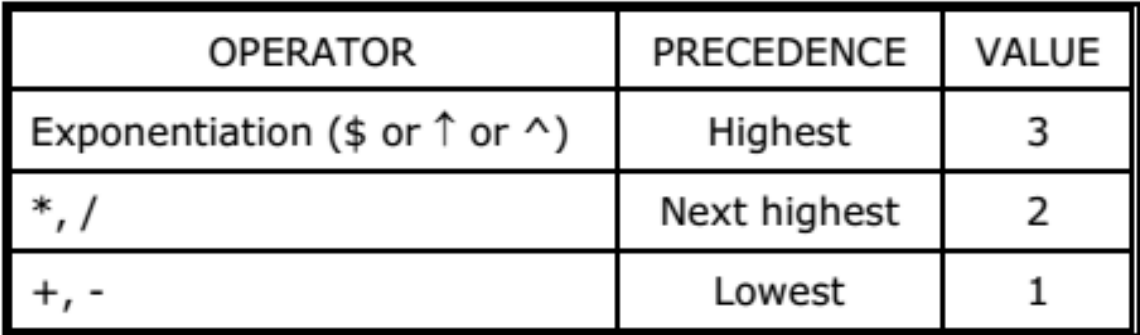
* **Base case** is the piece of code used to avoid infinite loops by terminating it.
* **Recursive call** is the piece of code executed during iterations.
* Two types:-
  + **Direct recursion:** Function calls itself inside it.
  + **Indirect recursion:** Two functions calling each other.

**Tower Of Henoi**

* Chinese/Japanese game resembling appearance of a stack.
* 3 rods are there, 1st with the rings (small, normal, large etc).
* 2nd rod for temporary storage.
* 3rd rod to prepare final result.
* Larger rings must be below and smaller rings must be above.
* Minimum number of moves: **(2n – 1)**

**Expression Parsing**

* The way to write arithmetic expression, or use arithmetic operators is known as a notation.
* 3 ways to write it without changing the output: Infix notation, prefix (polish), postfix (reverse polish).
* **Infix notation:** a+b (normal in between operators)
* **Prefix notation:** +ab (operator before expression)
* **Postfix notation:** ab+



**Infix To Postfix Conversion**

* Operators are symbols, operands are values involved.
* Use **push()** and **pop()** logics to store operators in stack and output operands.
* Go from **left to right** of expression.
* Lowest priority **((,))**, higher **(+,-)**, highest **(\*,/)**.
* Make two columns: **stack** and **postfix string**.
* Write **operands in postfix string** and **operators in stack** column one by one.
* Add **symbols to right** in both columns.
* Two **operators of same precedence can’t be together in postfix column**, so if such situation comes, **pop the operator on left side** out of the two from postfix, and **add it to the rightmost of stack**.
* If an operator comes between brackets, then **pop both the brackets and the operator** between it, and **add the operator between to the stack**.
* Our goal is to obtain the final postfix answer.

**Infix To Prefix Conversion**

* **Reverse** the given infix expression.
* Solve it by making: **stack** and **postfix column**
* Now **reverse the final postfix** solution expression, that’s our required prefix solution.

**Evaluation Of Postfix Expression**

* Make two columns: **symbol scanned** and **stack**
* Go **left to right**.
* If an operand is scanned, add it to the stack.
* If an operator is scanned, then **pop and perform** that symbol operation between the last two operands. **Replace** the last two operand with the result.

**Types Of Queues**

* Simple queue
* Circular queue
* Priority queue

**Queue:** Insertion in front, deletion in rear.

**Application Of Queue**

* Process scheduling, the **round robin algorithm**.
* **Spooling**, copying of data from one device to another.

**Operations On Queue**

* **initialize():** empties the queue.
* **empty():** gives answer in binary.
* **full():** \*same as above\*
* **print():** prints all queue elements.
* **enqueueF/enqueueR/dequeueF/dequeueR** (front/rear)
* If you delete any queue element from front, **nothing can occupy** the empty space, a **major disadvantage**.

**Circular queue:** Elements deleted in an order (right to left) and are added in the empty spaces, from left to right.

**Input restriction:** Enabling deletion at both ends in a queue, but input at one only.

**Output restriction:** \*Now you know\*

**Methods To Implement Dequeue**

* Circular array
* Singly linked list
* Doubly linked list
* Singly circular linked list
* Doubly circular linked list

**Priority queue:** Operations done on the basis of priority given in problem, also for multiple elements of same priority its **first come first serve**.

**Implementation Of Priority Queues**

* Unsorted -> **Insert:** O(1), **Delete:** O(n)
* Sorted -> **Insert:** O(n), **Delete:** O(1)