Data Structure

Is about data organization and management, storage format that leads to efficent access & modification (operation performed on that)

Unordered array: 14 17 8 4 9 ← Linear Search.

Ordered array: 1 7 8 9 23 → Binary Search.

5 4 2 1 0

non-decreasing order: 4, 14, 14, 15, 16, 17, 17, 18

non-increasing order: 10, 9, 9, 8, 7, 7, 6, 5

$O(n)$ → Complexity.
- → size of array.
- → Order (Maximum value Notation)

$\Omega(1)$
- → no. of comparison
- → Minimum.

Array Address Calculation

BA = 1000. A [0 . . . 100]

lower Bound, Upper Bound.

Let size of each element = 2B

A[78] = ?.

★ The no. of element stored before A[78]

A[1] - - - A[77] = 77

∴ Address of A[78] = BA + no. of element stored × size of each element

= 1000 + 77 × 2

= 1154.

A [LB ---- UB]

Size → UB - LB + 1

no of element stored A[i]

= i - LB.

∴ BA + (i - LB) × S

2D array mapped as 1D array.

$A[1...3][1...3]$ → 1. Row major order.
2. Column major order.

A[1][1]	A[1][2]	A[1][3]
A[2][1]	—	—
A[3][1]	—	—

(1). Row major order (C lang)

A[1][1]	..	A[1][3]	A[2][1]	- - -	A[3][3]

1000.

Row 1

$A[3][2] = ?$

↳ 2 Part calculation.

Before 3rd Row, no. of rows arranged → 2.

no. of element in each row → 3.

∴ Total element = 3 × 2 = 6 element

In 3rd row, we are at 2nd column.

∴ no. of column completed in 3rd row (already) → 1

∴ Total element = 6+1 = 7.

∴ BA + 7 × size

1000 + 7 × 2 = 1014 ✓

$A[-5 \; --- \; 100]$
1000. 2Byte

$A[70] = ?$ ∴ $\underbrace{-5 \to -1}_{5}, \underbrace{0, 1 \; --- \; 69}_{70}$ = 75

∴ 1000 + 75 × 2 = 1150

∴ $\overset{ub}{70} - (\overset{lb}{-5}) = 75.$

$$BA + [(i - lb_1) \times (ub_2 - lb_2 + 1) + (j - lb_2)] \times size$$

Column major order,

A[1][1]	A[2][1]	A[3][1]	A[1,2]	- - -	A[3][3]
1000	1002	1004			

column - 1.

A[2][3] - ?

No. of colums completed. = 2. (3-1)

no. of element in each columns (no. of rows) = 3

∴ 2 × 3 = 6.

No. of rows completed in 3rd. colum = (2-1) = 1.

∴ 6 + 1 = 7 × 2 = 14

And 1000 + 14 = 1014.

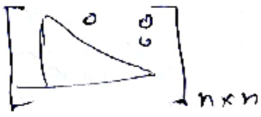
$$= BA + \left[(j - lb_2) \times (ub_1 - lb_1 + 1) + (i - lb_1) \right] \times size$$

Sparse Matrix

It are those matrix where relatively few entries are non-zero.

To store them in optimize way two store only non zero element.

Lower Triangular Matrix



= Square matrix

$j > i$. ⇒ $A[i][j] = 0$.

Row major

a[1][1]	a[2][1]	a[2][2]	a[3][1]	- - -
1000	1002			

$$= BA + \left[\frac{i(i-1)}{2} + (j-1) \right] \times size$$

Colum major

$$= BA + \left[n(j-1) - \frac{(j-1)(j-2)}{2} + (i-j) \right] \times size$$

Upper Triangular M :

Row major : $BA + \left[n(i-1) - \frac{(i-1)(i-2)}{2} + (j-1) \right] \times size$

Colum " : $BA + \left[\frac{j(j-1)}{2} + (i-1) \right] \times size$

Stack

↳ Applications

- Implementation of
  - ↳ function call, recursion, Divide & Conquer, Graph Traversal :- Depth First Search.
- Expression evaluation : In, pre, post fix.
- Permutation generator.
- PDA

Permutation Generator

with 'a' 'b' 'c' pushed in sequence :

a b c → [a] pop(a) [b] pop(b) [c] pop(c)

a c b → [a] pop(a) [c, b] pop(c) pop(b)

b a c → [b, a] pop() pop() [c] pop()

b c a

c a b ✗ → [a, b, c]

c b a

∴ n distinct character.

No. of permutation → $\frac{1}{n+1}\,{}^{2n}C_n$ Cartalan No

int a = 5;

'y.d', a - - - 2 → (a--) - 2
↳ 5 - 2 → 3

int b = 8 ;

'y.d', a - (-b) → 13

'y.d', a - - - (-b) → 13

'y.d', a - - - - 8 → 13

'y.d', a - - - - - b → -2

Let ↑ = power. Right to left

2 ↑ 3 ↑ 2
↳ 2 ↑ 9 → 512

Precedence

--, ++ Higher
-, +
& AND.
^ XOR
| OR
&& → logical
|| → logical

#2016

operator	precedence	Associativity
+	High	Left (starting side)
−	medium	right (starting side)
*	Low	Left (starting side)

there are some tokens

L ⟶ R

x = y

Right

Left to right → +, *

Right to left → −

$2 - 5 + 1 * -7 * 3$ = ?

$2 - 6 - 7 * 3$

$2 - (-1) * 3$

$3 * 3 = 9$

$2 - 4 - 3$

$2 - 1$

1

Post fix

$A - \underbrace{B/C}_{2} * \underbrace{D \uparrow E}_{1}$

3

4.

$A - B/C * DE\uparrow$

$A - BC/ * DE\uparrow$

$A - BC/DE\uparrow *$

$A\ BC/DE\uparrow * -$

Prefix

$A - B/C * \uparrow DE$

$A - /BC * \uparrow DE$

$A - * /BC \uparrow DE$

$- A * /BC \uparrow DE$

2

$(-a * b / c) + e$

Prefix.

$-a * b / c + e$

$* -ab / c + e$

$/ * -abc + e$

$+ / * -abce$

Postfix

$a - * b / c + e$

$a - b * / c + e$

$a - b * c / + e$

$a - b * c / e +$

#2004.

+ − * → left ass. ^ → right

^ * + − → High to low.

a + b × c − d ^ e ^ f

~~a + b × c − d ^ ^ ef~~ is a + b × c − d ^ ef ^

~~a + b × c − ^ d ^ ef~~ post a + b × c − def ^ ^

~~a + b × c d e f ^ ^~~ a + bc* − def^^

a b c * + − def ^ ^

a b c * + def ^ ^ −

Infix → Postfix

Stack

1. operator Stack
2. Scan input Left to right
3. printing output Left to right
4. if operand is Input print the operand.
5. if operator is input and stack is empty push the ~~operand~~ operator.
6. ★ if input is operator and having higher precedence operator on top of stack than push.

5 + 6 * 7.

① 5 — 5 [ ] ② + — 5 [+] ③ 6 — 5 6 [+] ④ * — 5 6 [* +]

⑤ 7 — 5 6 7 [* +] ⑥ 5 6 7 * +

★ 7. if input is operator and having lower precedance than operator on top of stack, than pop operators, Until input having lower precedence push the new operator.

5 * 6 − 7

− 5 6 [*] → 5 6 * [−] → 5 6 * 7 −

Infix $\xrightarrow[\text{stack}]{}$ Prefix

1. Scan Right to left.
2. print Right to Left
3. empty stack, operator input (same as Postfix)
4. Operand is input same as postfix
5. Operator with higher precedence than top of stack
6. Lower precedence operator than top of stack. (same as Postfix)

a + b * c

↳ c * b + a

cb*a+

↳ +a*bc

a+b*c

c

bc

*bc

+a*bc

precedence

In Infix multiple scanning is required

Prefix : Polish Notation

Postfix : Reverse Polish Notation.

> Single scan. to evaluate

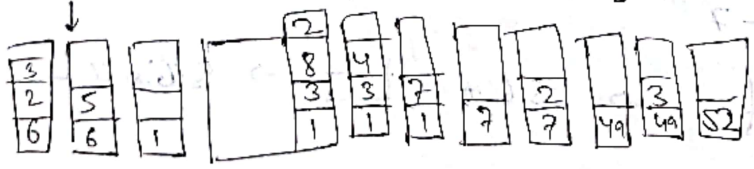
Evaluation of Postfix ex.

↳ operand stack

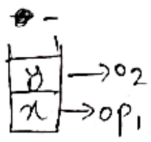
☆ Conversion

↳ operator stack

6 2 3 + − 3 8 2 / + * 2 ↑ 3 +



☆ −



→ x − y

What will be the top 2 values of stack after first *

8 3 2 ^ / 2 3 * + 5 1 -

↳ 6, 1

↑ more precedence than ↓ ↑ right associative

$a \uparrow b = a^b$ $a \downarrow b = \log_b a$ ↓ left "

$65536 \downarrow 2 \uparrow 4 \uparrow 2 \downarrow 2$

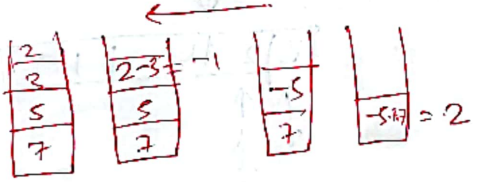
$65536 \downarrow 2 \uparrow 16 \downarrow 2$

$65536 \downarrow 65536 \downarrow 2$

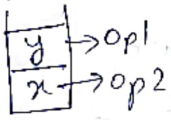
$\log_{65536} 65536 \downarrow 2$

$1 \downarrow 2 \rightarrow \log_2 1 \rightarrow 0$

☆ + * - 2 3 5 7



/



= op1 / op2
= y / x

☆

Postfix

$3 * \log(x+1) - a/2$

$3 * \log(x1+) - a/2$

$3 * (x1+)\log - a/2$

$3 (x1+)\log * a2/ -$

$3 (x1+)\log * a2/ -$

Unary precedence is more

log is unary