CS2700 : PROGRAMMING & DATA STRUCTURES

WEEK 5 : STACK ADT

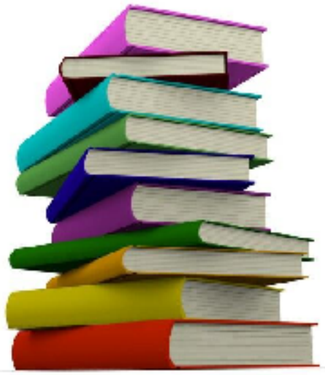
Goals for Today :

(1) STACK ADT

(2) IMPLEMENTING STACK ADT.

(3) USING STACK ADT

SOME REAL WORLD STACKS.



LAST IN FIRST OUT (LIFO)

STACK : A restricted list (yet very useful)

LIST ADT

insert (val)

delete (val)

size ()

find (val)

print ()

STACK ADT

push (val)

pop ()

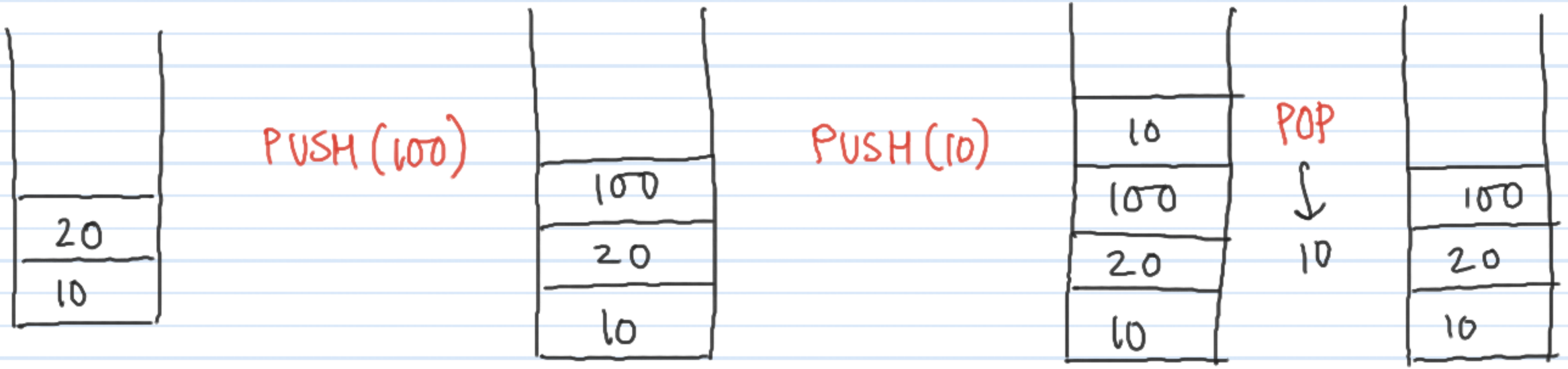
isempty ()

top ()

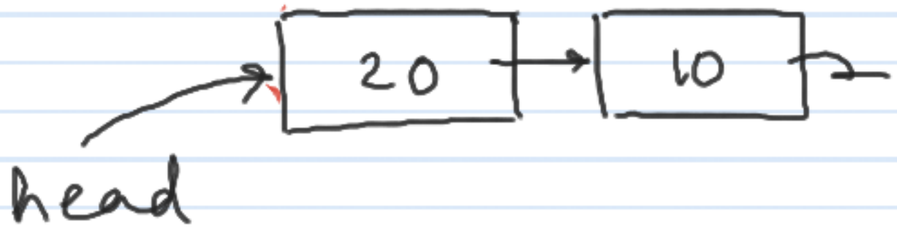
~~find~~ , ~~print~~

What is the running time of these operations?

IMPLEMENTING A STACK ADT



LINKED LIST IMPLEMENTATION

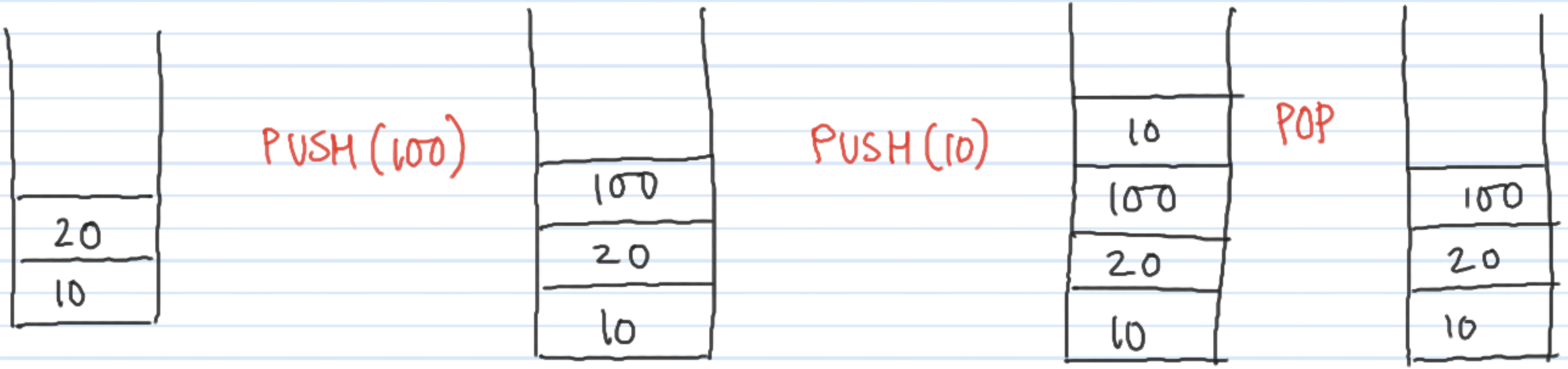


push(val) ≈ insertToFront(val)

pop() ≈ deleteFromFront()

Both are O(1) operations.

IMPLEMENTING A STACK ADT



ARRAY BASED IMPLEMENTATION

- maintain an index TOP into the array
- TOP = -1 ⇒ stack is empty.

Balanced parenthesis

( ) { } [ ]

INPUT : A STRING CONTAINING ABOVE CHARACTERS (ONLY)

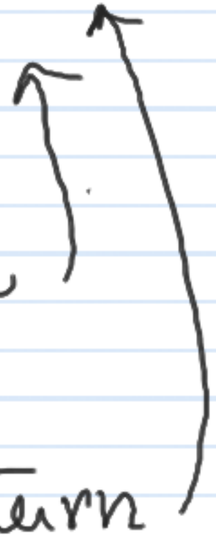
EXAMPLE : (((){}[{}]){{}}) : A

(((){}[)])) : B

GOAL : DECIDE WHETHER INPUT IS VALID

CHECKING BALANCED PARENTHESIS

```
init an empty stack S
for each symbol c in input {
    if c is opening parenthesis
        S.push(c);
    else {
        if (S.isempty()) return "unbalanced"
        else {
            c' = S.pop();
            if c and c' dont match return
        }
    }
}
if (S.isempty()) return "balanced" else return
```



APPLICATIONS OF STACK (continued)

Evaluating an expression

2 + 3 * 9 + 8 − 2 — infix expression

((2 + (3 * 9)) + 8) − 2

---

2 3 9 * + 8 + 2 − — postfix expression

[consider operators
+ , - , * , / for now]

Infix expression

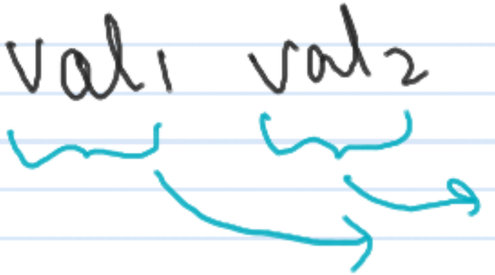
val : base case

val1 op val2 → infix expressions

Postfix expression

val : base case

val1 val2 op → postfix expressions.



INFIX VERSUS POST FIX

E1: (7+9) * (2-4)

7 9 + 2 4 - *

E2: 7+(9*(2-4))

7 9 2 4 - * +

E3: (7+9) * 2 - 4

7 9 + 2 * 4 -

E4: 7+(9*2) -4

7 9 2 * + 4 -

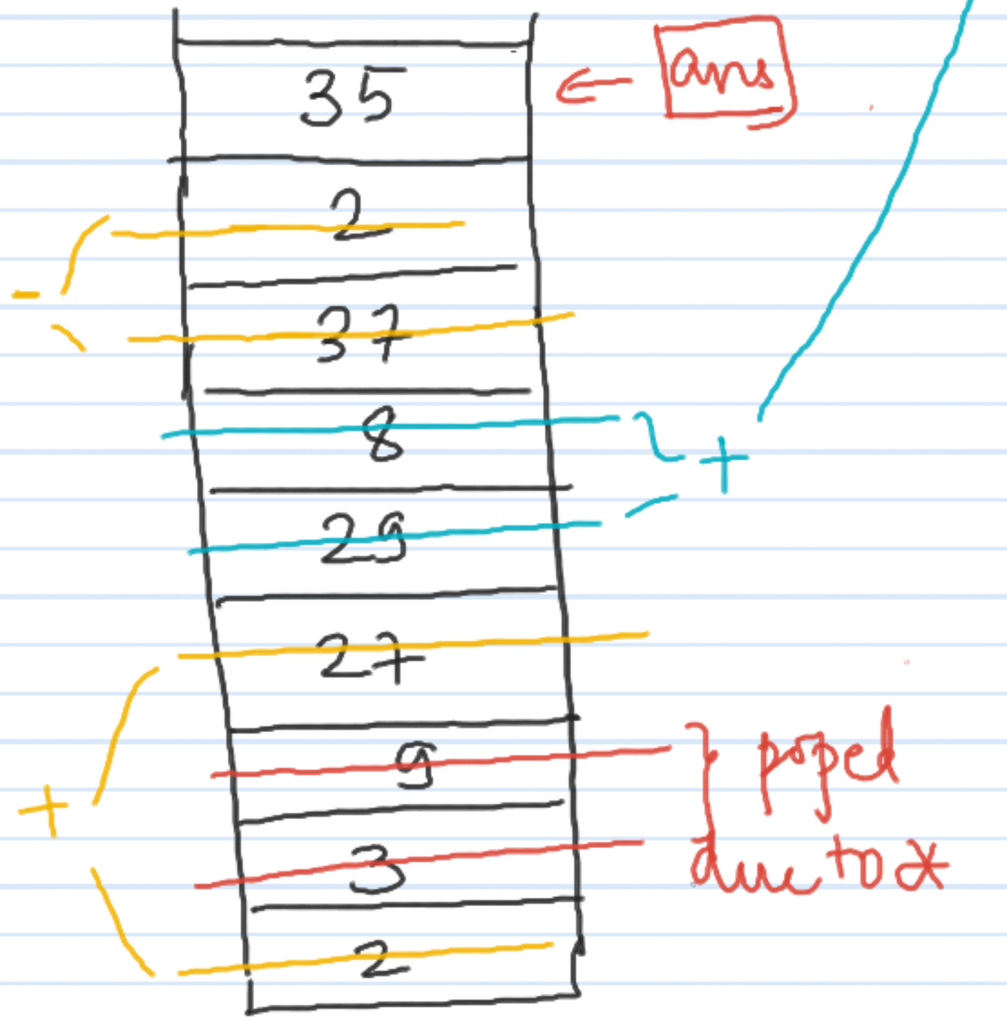
Note:

(1) Between infix and postfix operand order remains unchanged.

(2) Operator order does change.

Evaluating a postfix expression

2 3 9 * + 8 + 2 -



init an empty stack S

- when you encounter an operand val

  S.push(val)

- when you encounter an operator op

```
val2 = S.pop();
val1 = S.pop();
val3 = val1 op val2;
S.push(val3);
```

Converting Infix to Postfix

$2 + (3 * 9) + 8 - 2$

Scan string from left to right

(1) symbol is operand val

cout val

(2) symbol is (

push (on stack

---

(5) After reading whole input

cout all symbols on S in stack order.

(3) symbol is)

```
op = S.top();
while ( op != '(' ) {
    op = S.pop(); cout op;
    op = S.top();
} s.pop(); // remove '(' from S
```

(4) symbol `c` $\in \{+, -, *, /\}$

```
op = S.top()
while( priority(op) >= priority(c) {
    op = S.pop(); cout op;
    op = S.top();
}
                    // add current
s.push(c) // symbol c to S.
```