Saturday, 25 September 2021 8:33 PM

Stacks

:
20
10
5

LIFO data structure
→ last In first out

Operations

1. push (adding something to stack)
2. pop (removing from stack)

peek () → get the top element

↓ top
| 10 | 20 |

peek() → 20 ✓

Stack <Integer> stack = new Stack();
→ stack: name of stack

Implementations

1. Arrays
2. Linked list
3. Queues

Arrays

↓ top
| 20 | 40 | 50 |

top → [points to the topmost element in stack]

```
s. push (20);
s. push (30);
s. pop ();
s. push (40);
s. push (50);
s. push (60);
s. pop ();
```

1. stack overflow

| 10 | 20 | 30 | 40 | 50 |

2. stack underflow

Pros

→ Easy to implement
→ push | pop
  push → O(1), pop → O(1)

Cons

→ arrays are not dynamic nature

Stack using LinkedList

1. push → [add At head()]
2. pop → [remove At head()]

↓ head
(20) → (10)

→ push (10);
→ push (20);
→ push (30);
→ push (40);
→ push (50);
→ pop();
→ pop();
→ push (60);
→ pop ();
→ pop ();

push () → O(1)
pop() → O(1)

push:
1. new node
2. new node → head
3. head → new node

pop: head = head . next;

Pros:

1. Dynamic in nature
2. O(1) complexity for push/ pop

Cons:

1. More space required

Q Given a string (s) of just containing the characters (,), {, }, [,]. and determine if the string is valid.

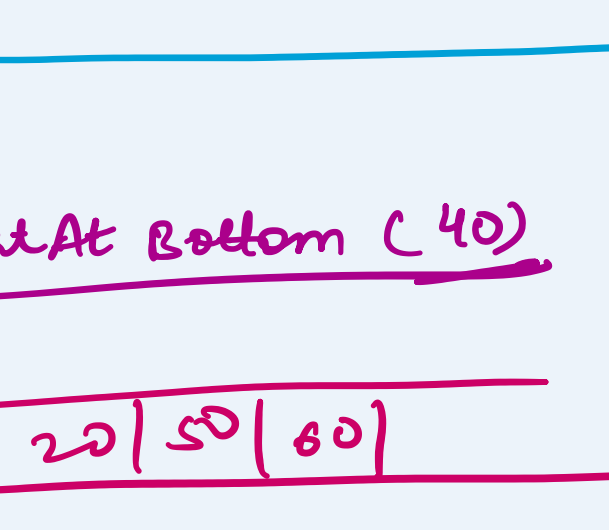
→ open brackets must be closed by same type of brackets

→ open brackets must be closed in correct order.

s =	( ) [ ] { }	true
s =	[( ]	false
s =	( ) [{ ( ) }]	true
s =	((( )]	false

1. { [[ ] { [ ] } ( ) ( )] }
expression → valid

subexpression → valid



{ [[ ] { [ ] } ( ) ( )] }

Q → InsertAtBottom (40)

| 10 | 20 | 50 | 60 |

| 40 | 10 | 20 | 50 | 60 |

InsertAtBottom (50)

| 40 | 30 | 20 | ch = s.pop() → 10

| 40 | 30 | ch = s.pop() → 20

| 40 | 30

| 50 | 40 | 40

| 50 | 40 | 30 | 20 | 10 |

```
InsertAtBottom (x) {
    if (s == empty)
        s.push (x);
    else {
        ch = s.pop ();
        IAB (x);
        s.push (ch);
}
```

Q Given a stack, reverse it.

| 10 | 20 | 30 | 40 | 50 |

| 50 | 40 | 30 | 20 | 10 |