STACKS AND QUEUES

Problem Solving with Computers-II

```
#include <iostream>
using namespace std;

int main(){
    cout<<"Hola Facebook!";
    return 0;
}
```

Stacks – container class available in the C++ STL

- Container class that uses the Last In First Out (LIFO) principle
- Methods

i. push() → insert

ii. pop() → remove

iii. top() //element on top of the stack

iv. empty()

//returns true if the stack is empty

① push(6)

② push(8)

③ push(15)

④ top() //15

⑤ pop()

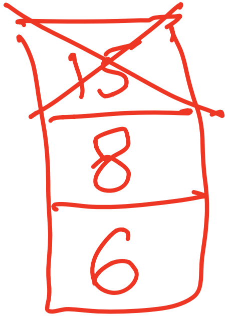
⑥ top() //8

⑦ empty() //false

⑧ pop()

⑨ pop()

⑩ empty() // true



Demo reversing a string

Notations for evaluating expression

operands

- Infix number operator number (7 + (3 * 5)) – (4 / 2)

operators

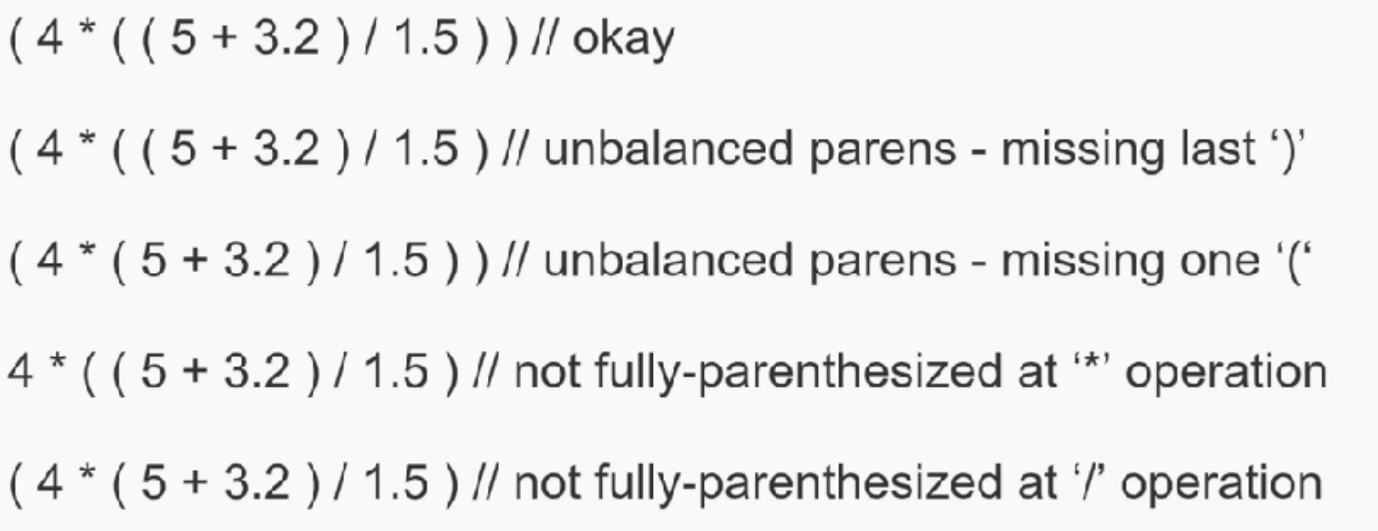
- Prefix operators precede the operands - + 7 * 3 5 / 4 2
- Postfix operators come after the operands 7 3 5 * + 4 2 / -

Used in calculators, easy to evaluate

3 * 5	* 3 5	3 5 *
infix	prefix	postfix

Lab05 – part 1: Evaluate a fully parenthesized infix expression

)4+3(Unbalanced parens ')' befor '('

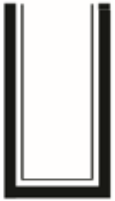


```
( 4 * ( ( 5 + 3.2 ) / 1.5 ) ) // okay

( 4 * ( ( 5 + 3.2 ) / 1.5 ) // unbalanced parens - missing last ')'

( 4 * ( 5 + 3.2 ) / 1.5 ) ) // unbalanced parens - missing one '('

4 * ( ( 5 + 3.2 ) / 1.5 ) // not fully-parenthesized at '*' operation

( 4 * ( 5 + 3.2 ) / 1.5 ) // not fully-parenthesized at '/' operation
```

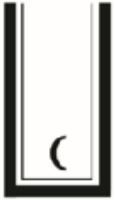
(4 * 5) +)

((2 * 2) + (8 + 4))

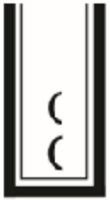
Initial empty stack



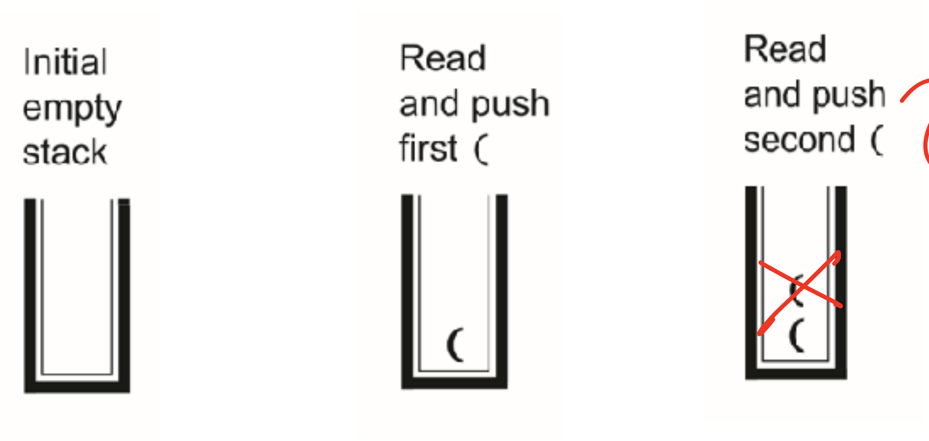
Read and push first (



Read and push second (



((2 * 2) + (8 + 4))



Initial empty stack

Read and push first (

Read and push second (

What should **be the next step after the first right parenthesis is encountered**?

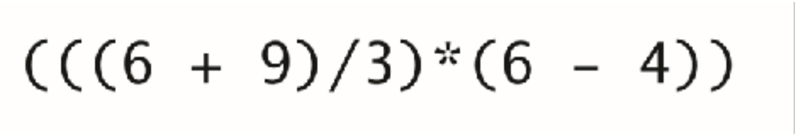
A. Push the right parenthesis onto the stack

[✓ B.] If the stack is not empty pop the next item on the top of the stack

C. Ignore the right parenthesis and continue checking the next character

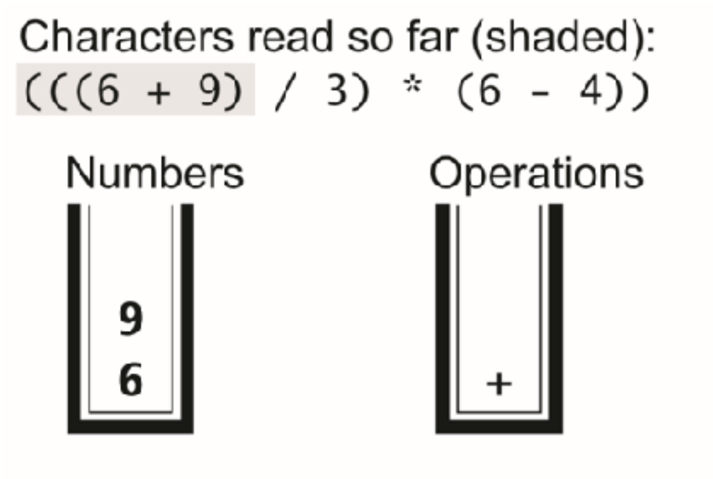
D. None of the above

Evaluating a fully parenthesized infix expression



```
(((6 + 9)/3)*(6 - 4))
```

Evaluating a fully parenthesized infix expression



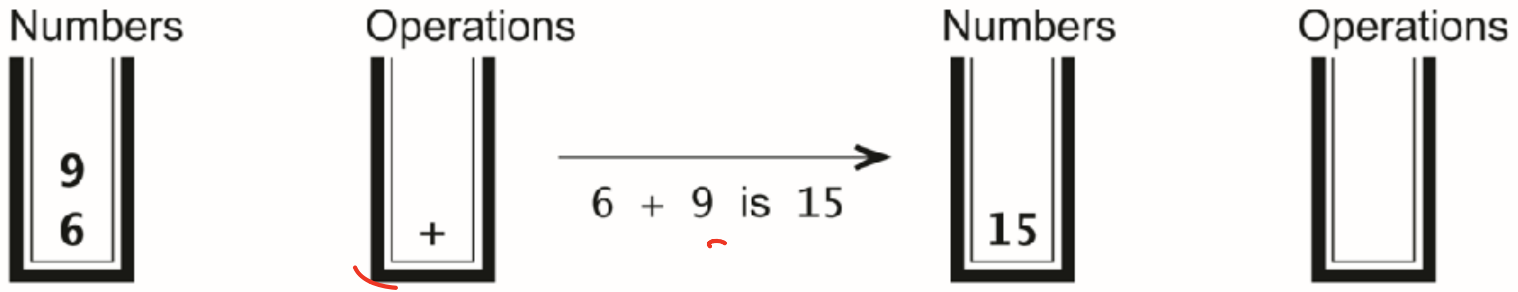
Characters read so far (shaded):

(((6 + 9) / 3) * (6 - 4))

Evaluating a fully parenthesized infix expression

Characters read so far (shaded):

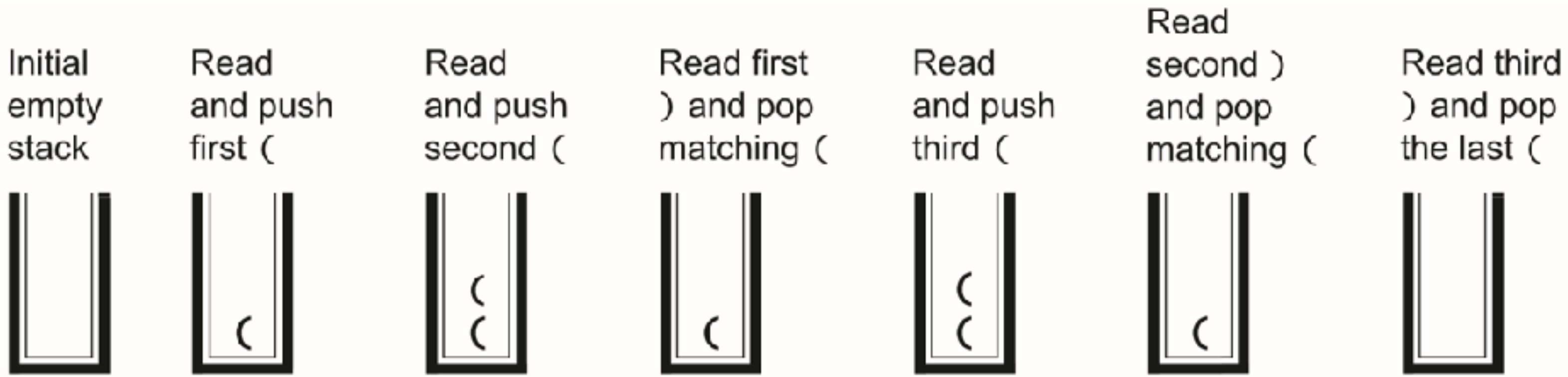
(((6 + 9) / 3) * (6 - 4))



Before computing 6 + 9

After computing 6 + 9

((2 * 2) + (8 + 4))



Initial empty stack

Read and push first (

Read and push second (

Read first) and pop matching (

Read and push third (

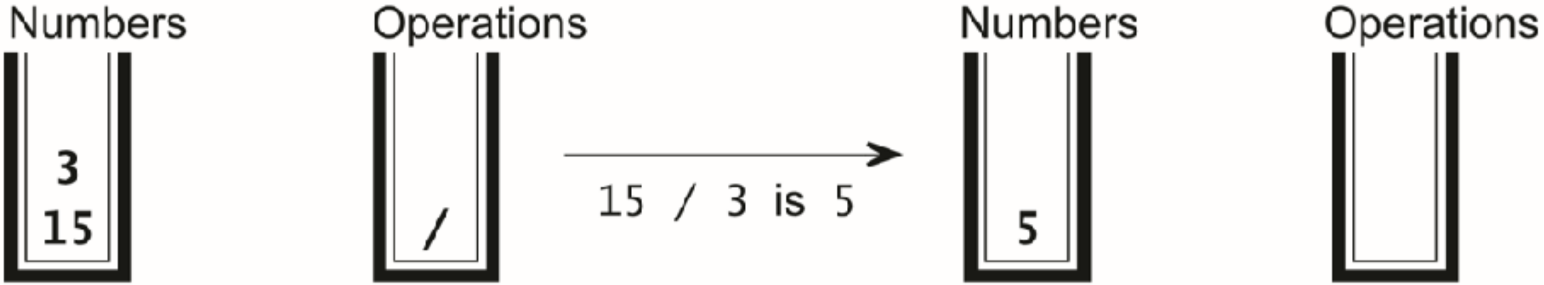
Read second) and pop matching (

Read third) and pop the last (

Evaluating a fully parenthesized infix expression

Characters read so far (shaded):

(((6 + 9) / 3) * (6 - 4))



Before computing 15/3

After computing 15/3

Lab 05, part2 :
Evaluating post fix expressions using a single stack

Postfix: 7 3 5 * + 4 2 / -

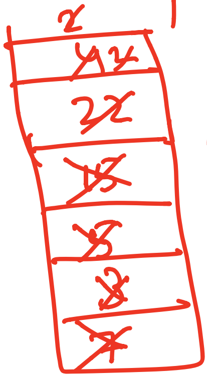
Infix: (7 + (3 * 5)) – (4 / 2)

Push operands onto the stack

When you encounter an operator, pop the two most recently seen operands (the two on top of the stack), perform the appropriate operation and push the result on the stack

$$(7+15) - 2 = 20$$

22



③ 22 − 2 = 20

② 15 + 7 = 22

① 5 * 3 = 15

After encountering a '*' pop 5 & 3 out of the stack, calculate 5*3 and push the result onto the stack

How would you design a Stack that in addition to push(), pop() provides the operation min()

All operations should have a complexity of O(1)

	min
~~7~~	4
~~4~~	4
~~10~~	6
~~6~~	6

Use two Stacks. – one that behaves like a regular stack, another that keeps track of the min of the elements pushed onto the stack (so, far)

See code from lec-13

Summary

- Like stacks, queues have many applications.
- Items enter a queue at the rear and leave a queue at the front.
- Queues can be implemented using an array or using a linked list.