

# STACK AND QUEUE

---

Problem Solving with Computers-II

C++

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

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

# C++STL

- The C++ Standard Template Library is a very handy set of three built-in components:
  - Containers: Data structures
  - Iterators: Standard way to search containers
  - Algorithms: These are what we ultimately use to solve problems

# C++ STL container classes

```
array  
vector  
forward_list  
list  
set  
stack  
queue  
priority_queue  
multiset (non unique keys)  
deque  
unordered_set  
map  
unordered_map  
multimap  
bitset
```

# Stacks – container class available in the C++ STL

- Container class that uses the Last In First Out (LIFO) principle
- Methods
  - i. `push()`
  - ii. `pop()`
  - iii. `top()`
  - iv. `empty()`

## Lab05: Evaluate a fully parenthesized infix expression

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

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

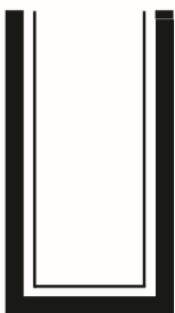
$(4 * (5 + 3.2) / 1.5))$  // unbalanced parens - missing one '('

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

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

# Checking if the parenthesis are balanced

Initial  
empty  
stack



Read  
and push  
first (



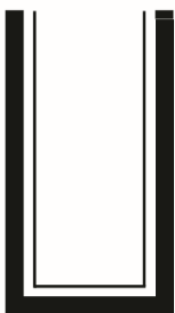
Read  
and push  
second (



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

# Checking if the parenthesis are balanced

Initial  
empty  
stack



Read  
and push  
first (



Read  
and push  
second (

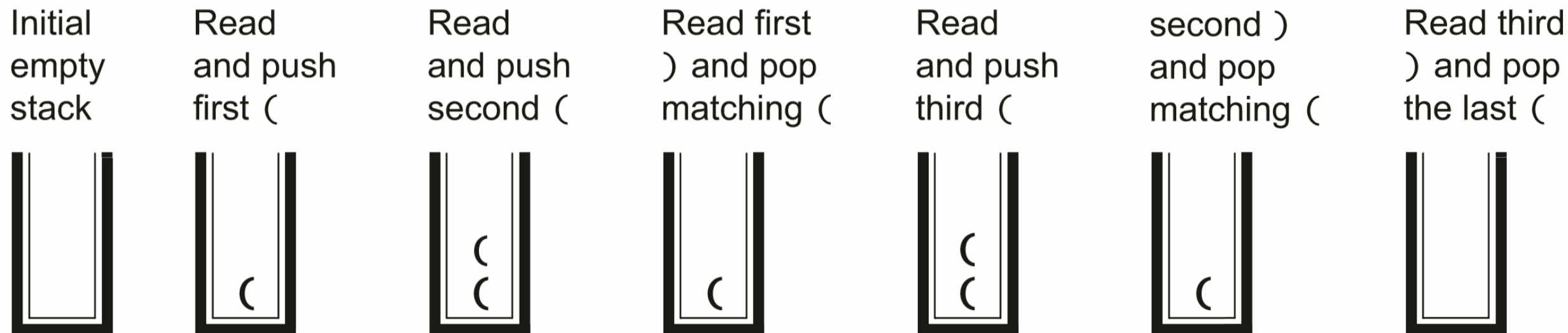


$((2 * 2) + (8 + 4))$

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

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





## Evaluating a fully parenthesized infix expression

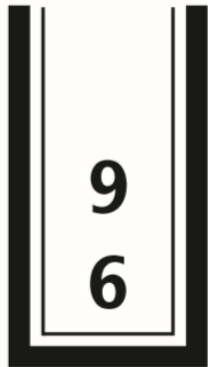
$$(((6 + 9)/3) * (6 - 4))$$

# Evaluating a fully parenthesized infix expression

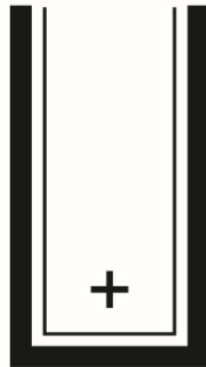
Characters read so far (shaded):

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

Numbers



Operations

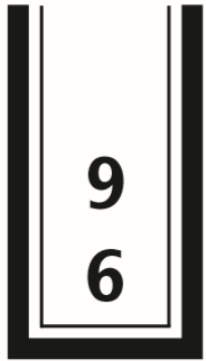


# Evaluating a fully parenthesized infix expression

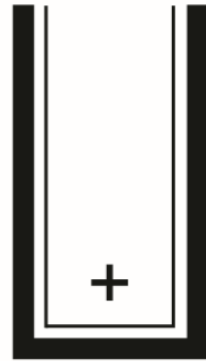
Characters read so far (shaded):

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

Numbers



Operations

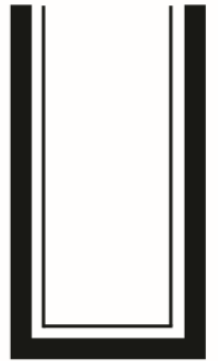


6 + 9 is 15

Numbers



Operations



Before computing 6 + 9

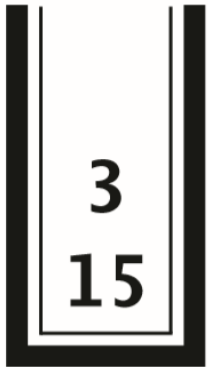
After computing 6 + 9

# Evaluating a fully parenthesized infix expression

Characters read so far (shaded):

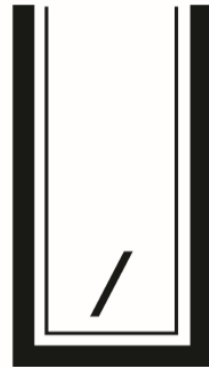
`((6 + 9) / 3) * (6 - 4)`

Numbers



Before computing 15/3

Operations



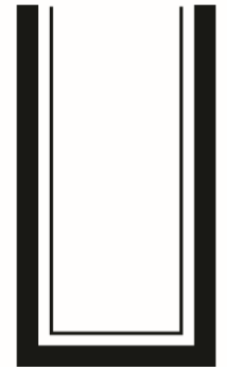
15 / 3 is 5

Numbers



After computing 15/3

Operations



# Notations for evaluating expression

- Infix    number operator number
- (Polish) Prefix   operators precede the operands
- (Reverse Polish) Postfix operators come after the operands

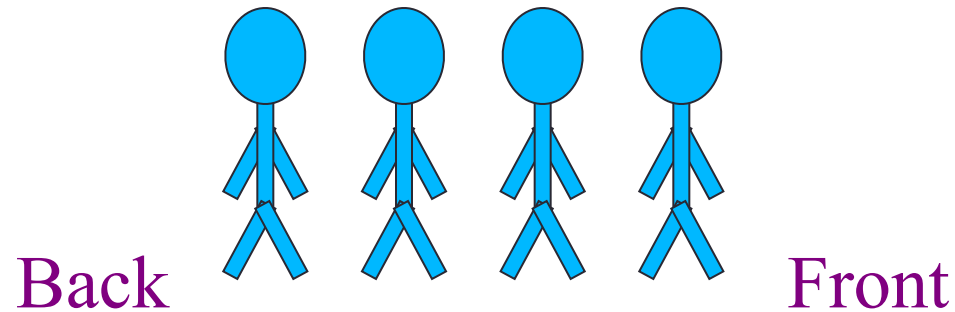
# Evaluating post fix expressions using a single stack

Postfix:

Infix:  $((6 + 9) / 3) * (6 - 4)$

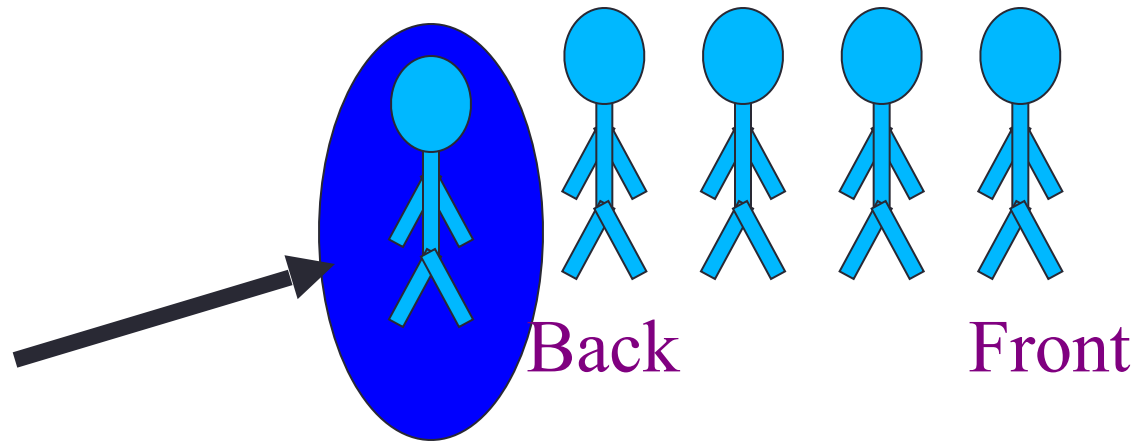
# Queue Operations

- A queue is like a queue of people waiting to be serviced
- The queue has a front and a back.



# Queue Operations

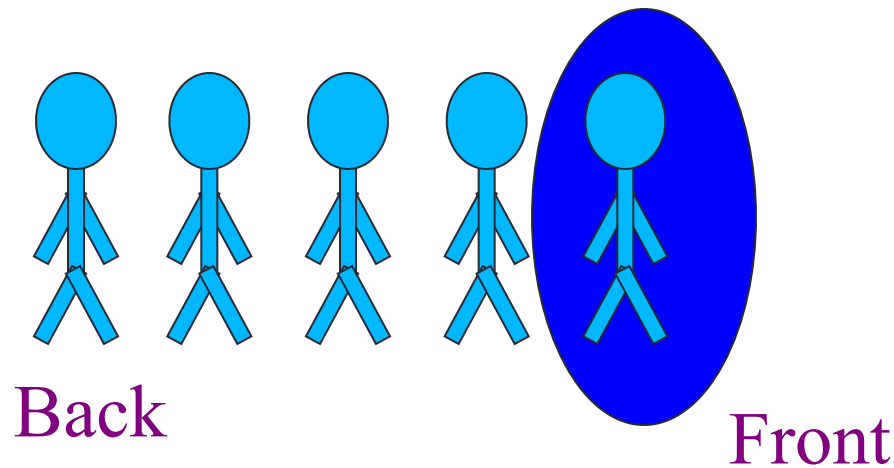
- New people must enter the queue at the back. The C++ queue class calls this a push, although it is usually called an enqueue operation.





# Queue Operations

- When an item is taken from the queue, it always comes from the front. The C++ queue calls this a pop, although it is usually called a dequeue operation.

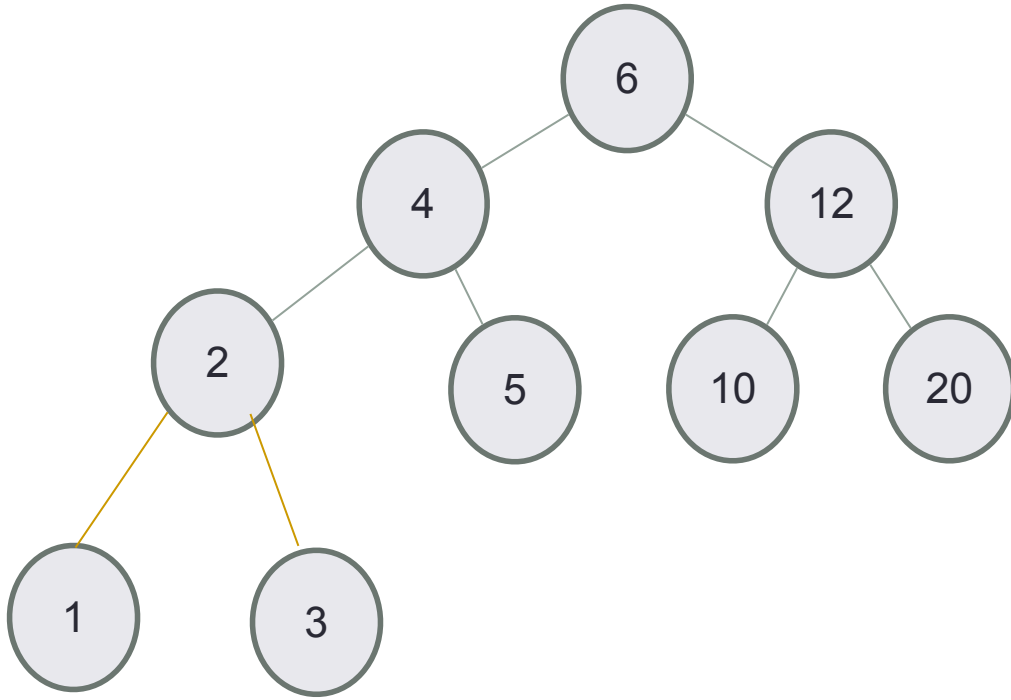


# Queue Class

- The C++ standard template library has a queue template class.
- The template parameter is the type of the items that can be put in the queue.

```
template <class Item>  
class queue<Item>  
{  
public:  
    queue( );  
    void push(const Item& entry);  
    void pop( );  
    bool empty( ) const;  
    Item front( ) const;  
    ...
```

# Breadth first traversal



- Take an empty Queue.
- Start from the root, insert the root into the Queue.
- Now while Queue is not empty,
  - Extract the node from the Queue and insert all its children into the Queue.
  - Print the extracted node.

# Summary of operations

Operation	Sorted Array	Binary Search Tree	Linked List
Min			
Max			
Median			
Successor			
Predecessor			
Search			
Insert			
Delete			