CS24 - Problem Solving with Computers II

Stacks and Queues

Announcements

Hot Mic…

Lab 04 and PA02 are out. Lab 04 is due next Wednesday, PA02 is due Aug 18th (3 weeks)

Lots of requests for last minute PA01 help… Please use the three weeks wisely

Discussion forum: method to the madness + OFFICE HOURS

Stacks

Stacks (as a data structure) operate in a Last in First Out (LIFO) pattern (just like stack memory)

Like stack memory, you can think of it as a stack of books, the last one you add to the pile has to be taken off before accessing the ones previously added

We won't have to build a stack ourselves… C++ STL (standard template library) has many useful data structures built in (array, vector, list, set, stack, queue, priority_queue, deque, plus many more)

C++STL also has iterators (to search containers) and algorithms built in

Stacks

STL stack class has five methods:

- push() - Pushes a new element onto the stack
- pop() - Removes the last added element from the stack
- top() - Gets the last added element from the stack
- empty() - Returns whether or not the stack is empty
- size() - Returns the size of the stack

Stacks

STL stack class has five methods:

- push() - Pushes a new element onto the stack
- pop() - Removes the last added element from the stack
- top() - Gets the last added element from the stack
- empty() - Returns whether or not the stack is empty
- size() - Returns the size of the stack

All operations are O(1)!

Stacks

Stacks use a template class for the type of information they’re storing (see week 1 for a review of template)

This means when initializing a stack, you must specify the type of data it’s storing (ie. stack<int> myIntStack;   stack<string> myStringStack; etc)

Can be useful for organizing data that’s entered sequentially (lab 5, parenthesis)

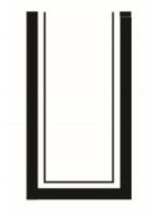
Stacks

Initial empty stack

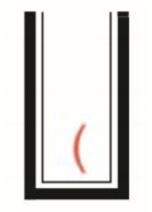
((2 * 2) + (8 + 4))

Stacks

Initial
empty
stack



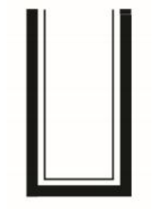
Push
first
parenthesis



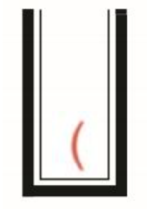
((2*2)+(8+4))

Stacks

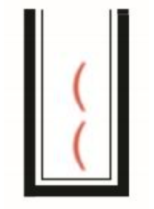
Initial empty stack

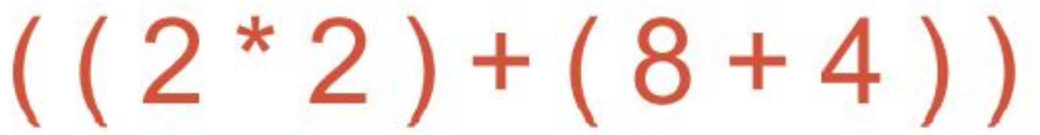


Push first parenthesis



Push second parenthesis





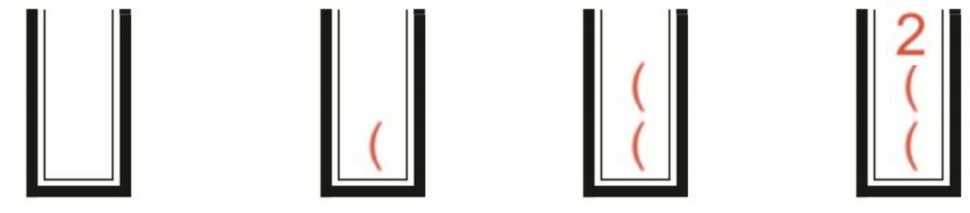
Stacks

Initial empty stack

Push first parenthesis

Push second parenthesis

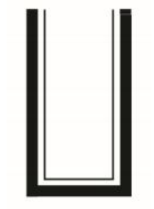
Push 2



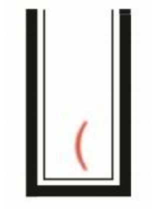
$((2*2)+(8+4))$

Stacks

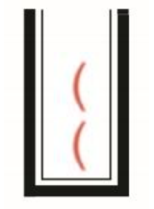
Initial empty stack



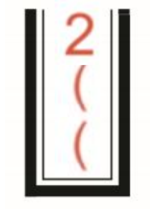
Push first parenthesis



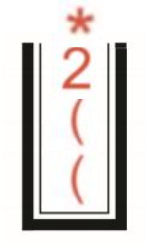
Push second parenthesis



Push 2

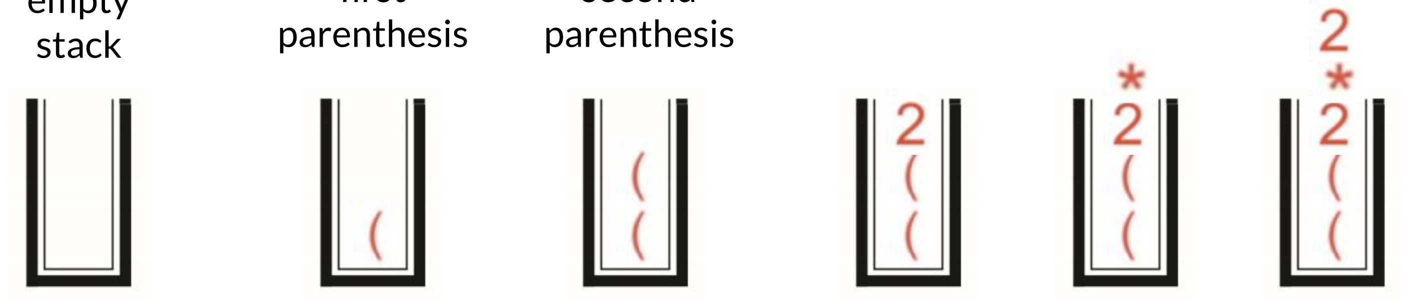


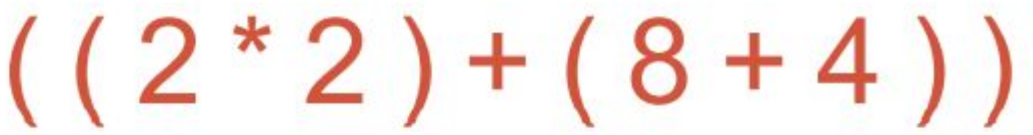
Push *



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

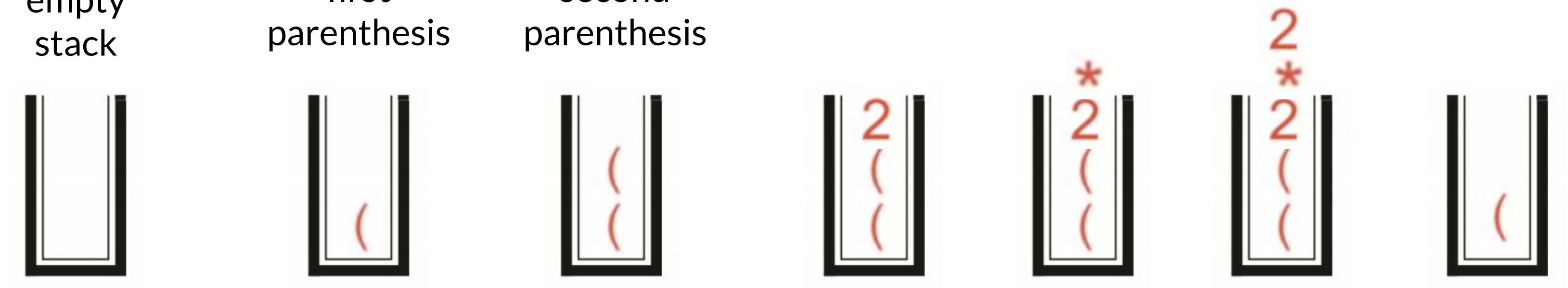
Stacks


Initial empty stack
Push first parenthesis
(
Push second parenthesis
(
(
Push 2
2
(
(
Push *
*
2
(
(
Push 2
2
*
2
(
(


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

Stacks

Initial empty stack | Push first parenthesis | Push second parenthesis | Push 2 | Push * | Push 2 | Push)



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

Queues

Also available in STL library

No lab assignment will use queues, BUT THEY WILL BE ON THE FINAL

Operate like the Stack, but as First In First Out (FIFO)

Queues

To implement a queue, we also must declare the type (ie. queue<int> myQueue;)

Important queue functions:

push(#) - pushes a value to the back of the queue (enqueue)
pop() - pops off the node at the front of the queue (dequeue)
front() - returns the value at the front of the queue
back() - returns the value at the back of the queue
size() - returns the size of the queue
empty() - returns a boolean value representing if it is empty

Queues

Useful for:

> Anything you want where data is processed in the order it is inputted (important in multi-threaded applications)
>
> Things like print queues or typing on the keyboard

Challenge

Go through each data structure we’ve learned, understand the time complexity of each operation… Think about things like min/max operations and search… Is it possible to search a stack/queue, how? What about keeping track of the min/max values? What would the runtime be? What are extra costs of implementing these operations? Is there could be a way to optimize them? Typically optimization results in runtime of O(1) (sometimes log(n), rare cases it’s higher) but at the expense of storing more data.

Why would you choose a binary search tree vs a stack or queue. Why would you want to use a Linked List? We will talk about these questions at next week’s live lecture!

Next week...

Heaps!