

## Topic 15

# Implementing and Using Stacks

"stack n.

The set of things a person has to do in the future. "I haven't done it yet because every time I pop my stack something new gets pushed." If you are interrupted several times in the middle of a conversation, "My stack overflowed" means "I forget what we were talking about."

### -The Hacker's Dictionary

**Friedrich L. Bauer**  
German computer scientist  
who proposed "stack method  
of expression evaluation"  
in 1955.

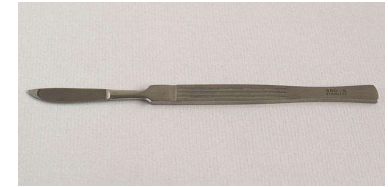


## Sharper Tools

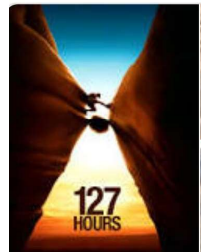


Lists

CS314



Stacks



Stacks

## Stacks

- Access is allowed only at one point of the structure, normally termed the *top* of the stack
  - access to the most recently added item only
- Operations are limited:
  - push (add item to stack)
  - pop (remove top item from stack)
  - top (get top item without removing it)
  - isEmpty
- Described as a "Last In First Out" (LIFO) data structure



3

CS314

Stacks

## Stack Operations

Assume a simple stack for integers.

```
Stack<Integer> s = new Stack<Integer>();  
s.push(12);  
s.push(4);  
s.push( s.top() + 2 );  
s.pop();  
s.push( s.top() );  
//what are contents of stack?
```

CS314

Stacks

4

# Stack Operations

Write a method to print out contents of stack in reverse order.



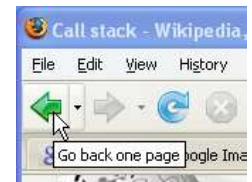
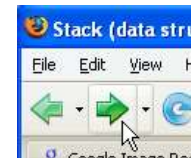
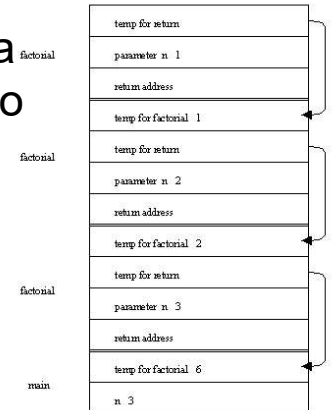
CS314

Stacks

5

# Uses of Stacks

- ▶ The runtime stack used by a process (running program) to keep track of methods in progress
- ▶ Search problems
- ▶ Undo, redo, back, forward



CS314

Stacks

6

## Clicker 1 - What is Output?

```
Stack<Integer> s = new Stack<>();
// put stuff in stack
for(int i = 0; i < 5; i++)
    s.push(i);
// Print out contents of stack
// while emptying it.
// Assume there is a size method.
for(int i = 0; i < s.size(); i++)
    System.out.print(s.pop() + " ");
```

- A 0 1 2 3 4      D 2 3 4  
 B 4 3 2 1 0      E No output due  
 C 4 3 2              to runtime error

CS314

Stacks

7

## Corrected Version

```
Stack<Integer> s = new Stack<Integer>();
// put stuff in stack
for(int i = 0; i < 5; i++)
    s.push(i);
// print out contents of stack
// while emptying it
int limit = s.size();
for(int i = 0; i < limit; i++)
    System.out.print(s.pop() + " ");
//or
// while(!s.isEmpty())
//     System.out.println(s.pop());
```

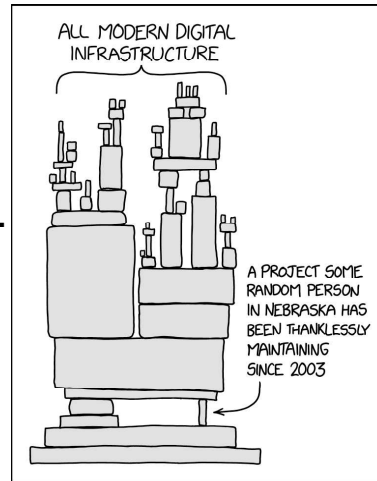
CS314

Stacks

8

## Implementing a stack

- ▶ need an underlying collection to hold the elements of the stack
- ▶ 2 obvious choices
  - native array
  - a list!!!
- ▶ Adding a *layer of abstraction*. A HUGE idea.
- ▶ array implementation
- ▶ linked list implementation



<https://xkcd.com/2347/>

9

CS314

Stacks

## Applications of Stacks

### Mathematical Calculations

- ▶ What does  $3 + 2 * 4$  equal?  
 $2 * 4 + 3?$   $3 * 2 + 4?$
- ▶ The precedence of operators affects the order of operations.
- ▶ A mathematical expression cannot simply be evaluated left to right.
- ▶ A challenge when evaluating a program.
- ▶ *Lexical analysis* is the process of interpreting a program.

What about  $1 - 2 - 4 ^ 5 * 3 * 6 / 7 ^ 2 ^ 3$

CS314

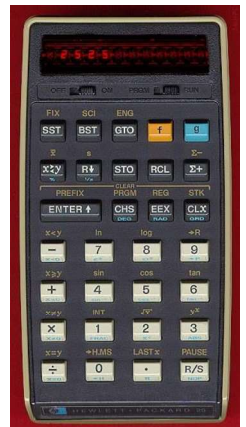
Stacks

11

### Infix and Postfix Expressions

- ▶ The way we use to writing expressions is known as infix notation
- ▶ Postfix expression does not
- ▶ require any precedence rules
- ▶  $3 2 * 1 +$  is postfix of  $3 * 2 + 1$
- ▶ evaluate the following postfix expressions and write out a corresponding infix expression:

$2 3 2 4 * + *$                        $1 2 3 4 ^ * +$   
 $1 2 - 3 2 ^ 3 * 6 / +$                $2 5 ^ 1 -$



CS314

Stacks

12

## Clicker Question 2

- What does the following postfix expression evaluate to?

6 3 2 + \*

- A. 11
- B. 18
- C. 24
- D. 30
- E. 36

CS314

Stacks

13

## Evaluation of Postfix Expressions

- Easy to do with a stack
- given a proper postfix expression:
  - get the next token
  - if it is an operand push it onto the stack
  - else if it is an operator
    - pop the stack for the right hand operand
    - pop the stack for the left hand operand
    - apply the operator to the two operands
    - push the result onto the stack
  - when the expression has been exhausted the result is the top (and only element) of the stack

CS314

Stacks

14

## Infix to Postfix

- Convert the following equations from infix to postfix:

$2^3 + 3 + 5 * 1$

$11 + 2 - 1 * 3 / 3 + 2^2 / 3$

Problems:

Negative numbers?

parentheses in expression

CS314

Stacks

15

## Infix to Postfix Conversion

- Requires operator precedence parsing algorithm
  - parse v. To determine the syntactic structure of a sentence or other utterance

Operands: add to expression

Close parenthesis: pop stack symbols until an open parenthesis appears

Operators:

Have an on stack and off stack precedence

Pop all stack symbols until a symbol of lower precedence appears. Then push the operator

End of input: Pop all remaining stack symbols and add to the expression

CS314

Stacks

16

## Simple Example

Infix Expression:  $3 + 2 * 4$

PostFix Expression:

Operator Stack:

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence
+	1	1
-	1	1
*	2	2
/	2	2
^	10	9
(	20	0

## Simple Example

Infix Expression:  $+ 2 * 4$

PostFix Expression: 3

Operator Stack:

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence
+	1	1
-	1	1
*	2	2
/	2	2
^	10	9
(	20	0

## Simple Example

Infix Expression:  $2 * 4$

PostFix Expression: 3

Operator Stack: +

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence
+	1	1
-	1	1
*	2	2
/	2	2
^	10	9
(	20	0

## Simple Example

Infix Expression:  $* 4$

PostFix Expression: 3 2

Operator Stack: +

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence
+	1	1
-	1	1
*	2	2
/	2	2
^	10	9
(	20	0

## Simple Example

Infix Expression: 4

PostFix Expression: 3 2

Operator Stack: + \*

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence
+	1	1
-	1	1
*	2	2
/	2	2
^	10	9
(	20	0

## Simple Example

Infix Expression:

PostFix Expression: 3 2 4

Operator Stack: + \*

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence
+	1	1
-	1	1
*	2	2
/	2	2
^	10	9
(	20	0

## Simple Example

Infix Expression:

PostFix Expression: 3 2 4 \*

Operator Stack: +

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence
+	1	1
-	1	1
*	2	2
/	2	2
^	10	9
(	20	0

## Simple Example

Infix Expression:

PostFix Expression: 3 2 4 \* +

Operator Stack:

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence
+	1	1
-	1	1
*	2	2
/	2	2
^	10	9
(	20	0

## Example

$$11 + 2^4 \cdot 3 - ((4 + 5) \cdot 6)^2$$

Show algorithm in action on above equation

## Balanced Symbol Checking

- ▶ In processing programs and working with computer languages there are many instances when symbols must be balanced  
 $\{ \}$ ,  $[ ]$ ,  $( )$

A stack is useful for checking symbol balance. When a closing symbol is found it must match the most recent opening symbol of the same type.

- ▶ Applicable to checking html and xml tags!

## Algorithm for Balanced Symbol Checking

- ▶ Make an empty stack
- ▶ read symbols until end of file
  - if the symbol is an opening symbol push it onto the stack
  - if it is a closing symbol do the following
    - if the stack is empty report an error
    - otherwise pop the stack. If the symbol popped does not match the closing symbol report an error
- ▶ At the end of the file if the stack is not empty report an error

## Algorithm in practice

- ▶  $list[i] = 3 * ( 44 - method( foo( list[ 2 * (i + 1) + foo( list[i - 1] ) ) / 2 * ) - list[ method(list[0]) ] );$
- ▶ Complications
  - when is it not an error to have non matching symbols?
- ▶ Processing a file
  - *Tokenization*: the process of scanning an input stream. Each independent chunk is a token.
- ▶ Tokens may be made up of 1 or more characters