

1.3 BAGS, QUEUES, AND STACKS



- ▶ stacks
- ▶ resizing arrays
- ▶ queues
- ▶ generics
- ▶ iterators
- ▶ applications

Arithmetic expression evaluation

Goal. Evaluate infix expressions.

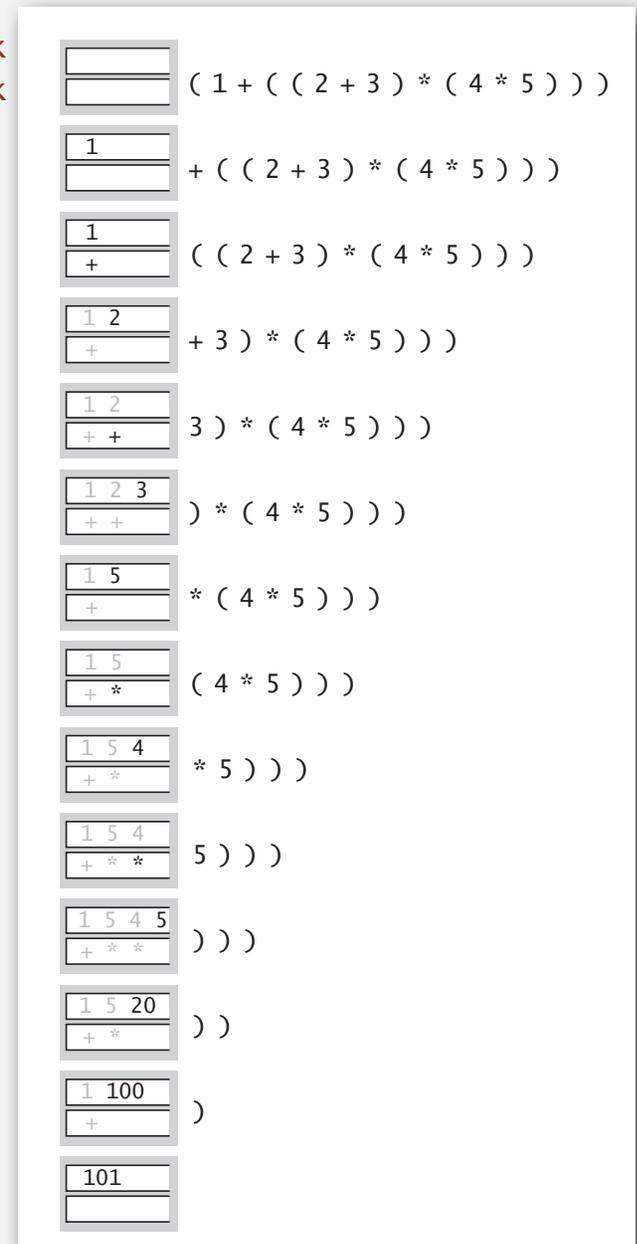


Two-stack algorithm. [E. W. Dijkstra]

- Value: push onto the value stack.
- Operator: push onto the operator stack.
- Left parenthesis: ignore.
- Right parenthesis: pop operator and two values; push the result of applying that operator to those values onto the operand stack.

Context. An interpreter!

value stack
operator stack



Arithmetic expression evaluation

```
public class Evaluate
{
    public static void main(String[] args)
    {
        Stack<String> ops = new Stack<String>();
        Stack<Double> vals = new Stack<Double>();
        while (!StdIn.isEmpty()) {
            String s = StdIn.readString();
            if (s.equals("("))
                ops.push(s);
            else if (s.equals("+")) ops.push(s);
            else if (s.equals("*")) ops.push(s);
            else if (s.equals(")")
            {
                String op = ops.pop();
                if (op.equals("+")) vals.push(vals.pop() + vals.pop());
                else if (op.equals("*")) vals.push(vals.pop() * vals.pop());
            }
            else vals.push(Double.parseDouble(s));
        }
        StdOut.println(vals.pop());
    }
}
```

```
% java Evaluate
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
101.0
```

Correctness

Q. Why correct?

A. When algorithm encounters an operator surrounded by two values within parentheses, it leaves the result on the value stack.

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```

as if the original input were:

```
( 1 + ( 5 * ( 4 * 5 ) ) )
```

Repeating the argument:

```
( 1 + ( 5 * 20 ) )  
( 1 + 100 )  
101
```

Extensions. More ops, precedence order, associativity.

Stack-based programming languages

Observation 1. The 2-stack algorithm computes the same value if the operator occurs **after** the two values.

```
( 1 ( ( 2 3 + ) ( 4 5 * ) * ) + )
```

Observation 2. All of the parentheses are redundant!

```
1 2 3 + 4 5 * * +
```



Jan Lukasiewicz

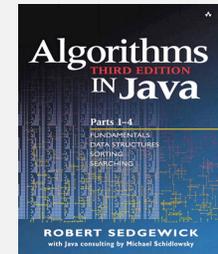
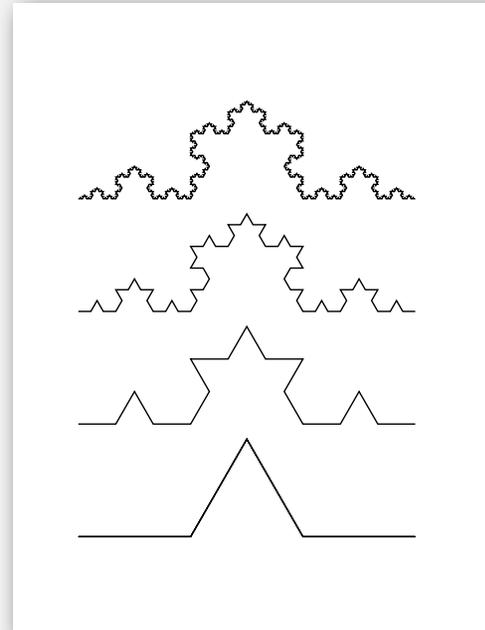
Bottom line. Postfix or "reverse Polish" notation.

Applications. Postscript, Forth, calculators, Java virtual machine, ...

PostScript applications

Algorithms, 3rd edition. Figures created directly in PostScript.

```
%!  
72 72 translate  
  
/kochR  
{  
  2 copy ge { dup 0 rlineto }  
  {  
    3 div  
    2 copy kochR 60 rotate  
    2 copy kochR -120 rotate  
    2 copy kochR 60 rotate  
    2 copy kochR  
  } ifelse  
  pop pop  
} def  
  
0 0 moveto 81 243 kochR  
0 81 moveto 27 243 kochR  
0 162 moveto 9 243 kochR  
0 243 moveto 1 243 kochR  
stroke
```



see page 218

Algorithms, 4th edition. Figures created using enhanced version of `stdDraw` that saves to PostScript for vector graphics.

