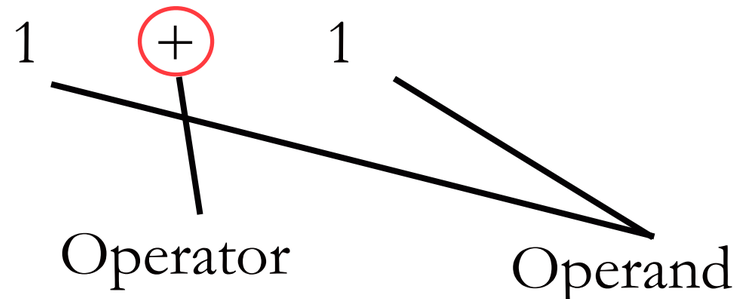
CS4221 - Computer Science

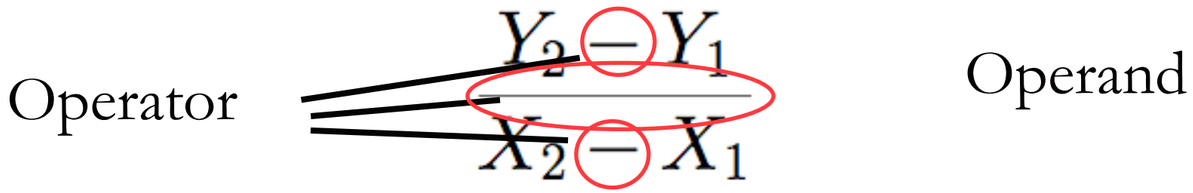
Lecture Set 1: Expressions

Expressions

- **“Mathematical phrase”**



- **Slope of a line (X1, Y1) and (X2, Y2)**



- **Evaluating expressions**
  - All numbers?
  - Some (or all) variables?
    - e.g. Y2 – Y1, what are Y2 and Y1?
- **Order of evaluation:**
  - 1+2*3=?
    - 3 * 3 = 9
    - 1 + 6 = 7

- **If operators are different?**
  - Give each a precedence
- **Standard precedence:**
  - ( )
  - *, /
  - +, -
  - Draw? Go left to right

Associative
  (a + b) + c = a + (b + c)
Commutative
  a * b = b * a
  a / b ≠ b / a
Example
  3 * 4 = 12
  4 * 3 = 12
  6 / 2 = 3
  2 / 6 = ⅓

- **Note, all of these operators are associative, makes no difference**

- Example

$$2 * 3 + 4 / 2 + 2$$

$$6 + 4 / 2 + 2$$

$$6 + 2 + 2$$

- Different kinds of notation:
  - So far, have used (mainly) infix notation
  - i.e. operators come between operands

- Other notations:
  - Prefix
    - Before operand
    - $\sqrt{4}$ -3, cos 45 etc.
  - Super-fix
    - Above (and usually after) operand
    - 3^2, x^y
  - Sub-fix
    - Underneath
    - $\log_x Y$, $\frac{2}{3}$

Syntax
How to write the notation

Semantics
What the notation **means**

- Postfix
  - After operand
  - 3++
- Why so many?
- Consider
  - All four
- Notice
  - Size of square root sign
  - Length of line over 2a
  - No multiplication sign
  - *ac* right up together

Quadratic Formula

$ax^2 + bx + c = 0$

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Reminder - what is this course about?
  - Phrasing things unambiguously
  - Start where?
  - Mix of notation
- Model Driven Development
  - Design
- Imperative Programming
  - Learn language
- Computer Organisation
  - Build hardware
- Computer Science
  - Step back and SOLVE the problem

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$(-b \pm \sqrt{b^2 - 4 * a * c})/(2 * a)$$

- Problems?
  - Square root sign
    - Can't be typed
    - Variable length
    - Variable scope
      - Area in which something takes effect
    - Aside: X√ would be more convenient
    - $(b^2\text{-}4ac)$√
    - Still pretty ugly...

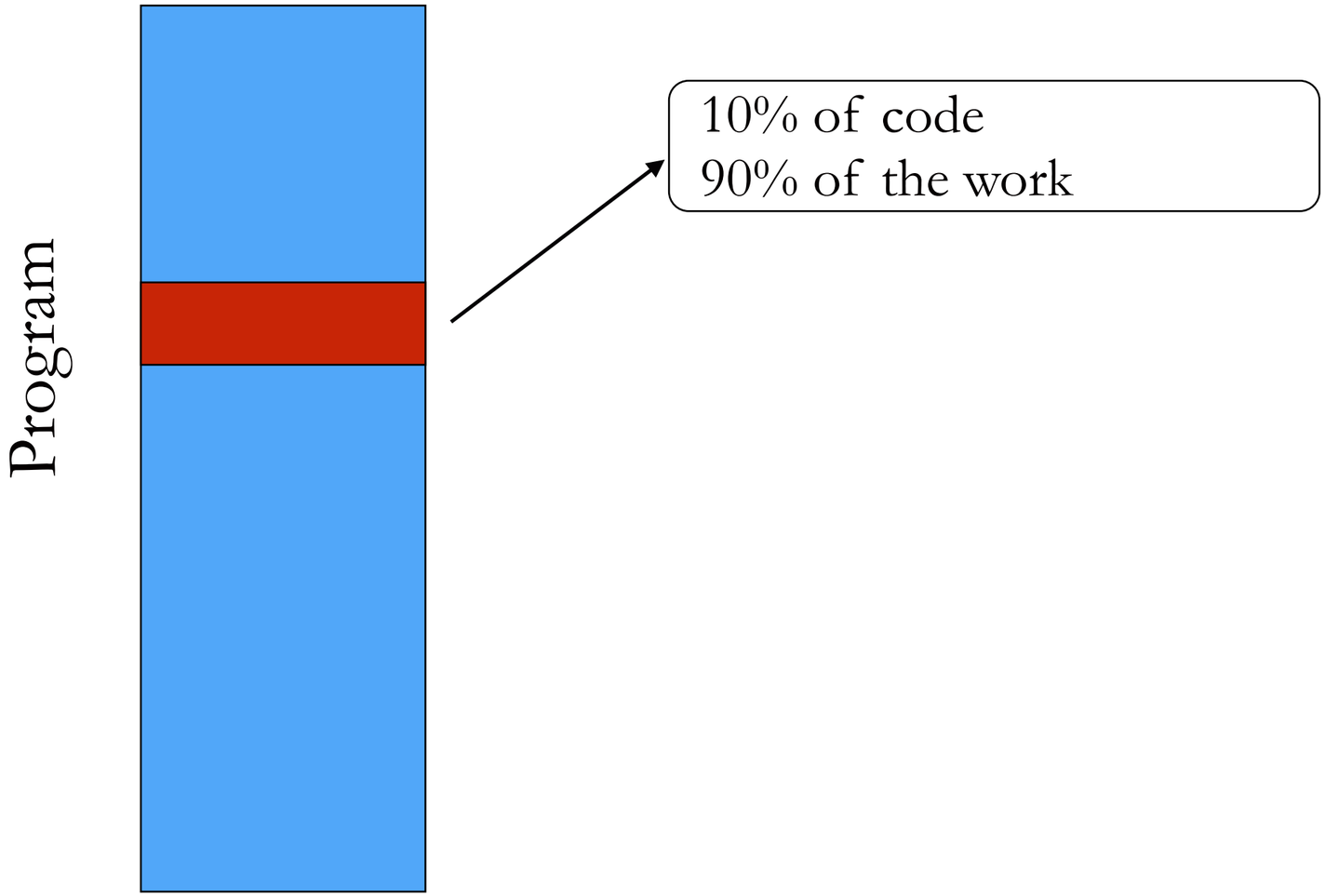
- (b^2-4ac)
- Read b
  - Meaning?
  - Not clear until after squared sign
- **$(b^2-$**
  - Ambiguity
    - Subtract next thing?
    - Evaluate next sub-expression?

- Better if there was a single notation
  - No ambiguity
  - Everything evaluated the same way
- Separate the “what” from
  - Make no comment on h
    - e.g. how to add numbe
  - Worry about implement
- Prefix problem
  - Don’t know how to deal with a character (or number)..
  - Until after (at least) the next one is read

Predictive text

F U C H S I A

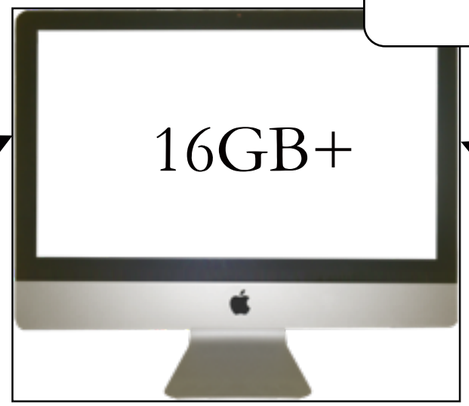
Locality of Reference



Locality of Reference

10% of cod
90% of the

Cache
Fast, expensive, small
RAM
Quick, dear, medium
Disk
Slow, cheap, big



CPU (cores)

60 nanoseconds

Hard disk 2TB
Approx. 7 microseconds

(1 ms = 1000 ns)

Cache memory 8MB (0.25 nanoseconds)

Internet of Things (IoT)

Low powered devices with small resources

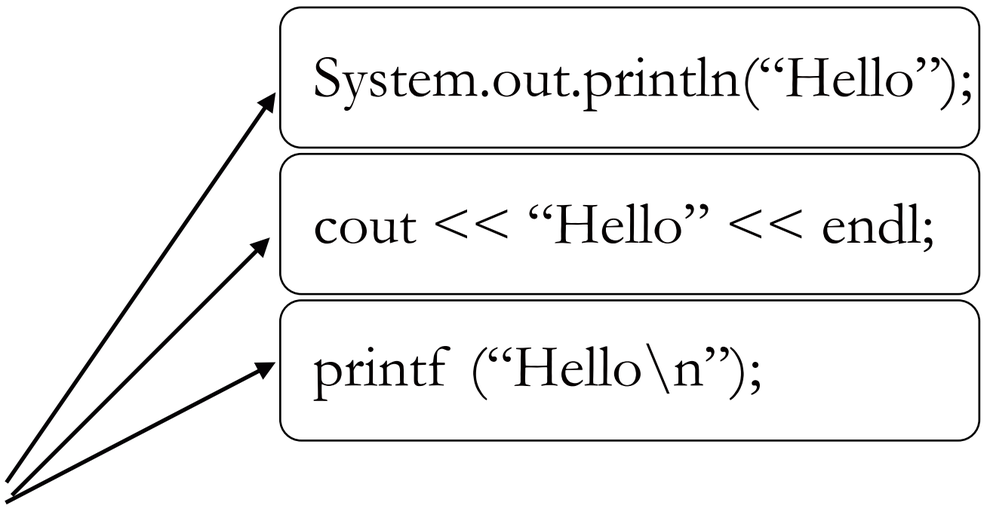
Require efficient programs

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Writing fast programs
  - Small (fit in the cache)
  - Reuse functionality (stay in the cache)
  - **Often faster to do one thing many times than several things once**
  - **Often faster to do one thing many times than several things once**

- Prefix Notation
  - Operator goes before operands
  - (+ 2 3)
  - “Apply plus operator to 2 and 3”
  - “Apply operator to next two items”
  - “.. to the next two arguments”
- Definitions
  - Syntax
    - *Representation of data/code*

- **Semantics**
  - *Meaning of syntax*
- **Abstract Syntax**
  - Representation that is **independent** of language



- **Semantics**
  - *Meaning of syntax*
- **Abstract Syntax**
  - Representation that is **independent** of language

System.out.println(“Hello”);

cout << “Hello” << endl;

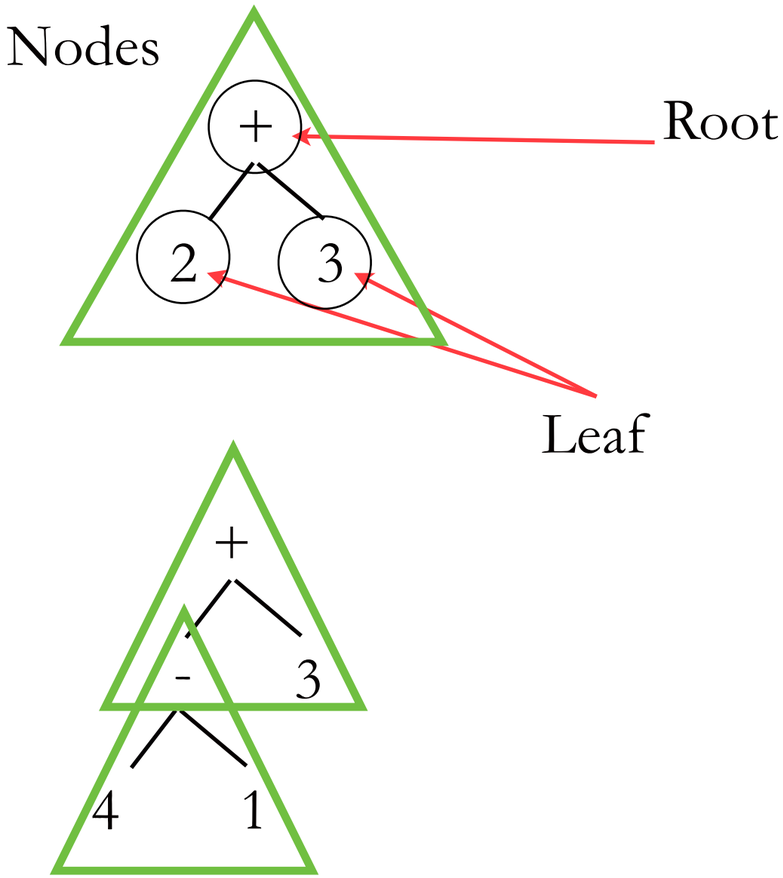
printf (“Hello\n”);

Design will work with any language

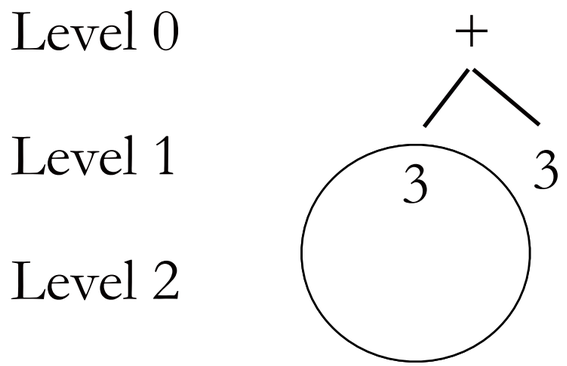
After a translation process

Design once, deploy many times

- Abstract Syntax Tree (AST)
  - Diagram of expression
  - Shows **what** expression does.
  - **(+ 2 3)**
- More complex tree
- AST
  - Convenient graphical notation

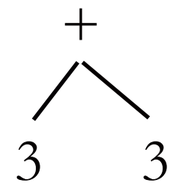


- Evaluation
  - Evaluate deepest operator
  - Repeat until no operators are left

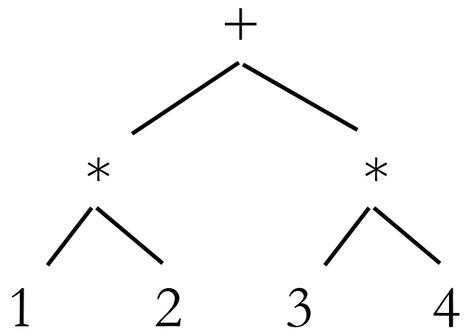


  - (- 4 1)
  - 3

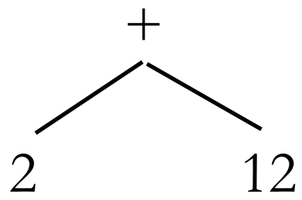
- Nothing on level 1
- (+ 3 3) = 6



- More complex tree



- Order?



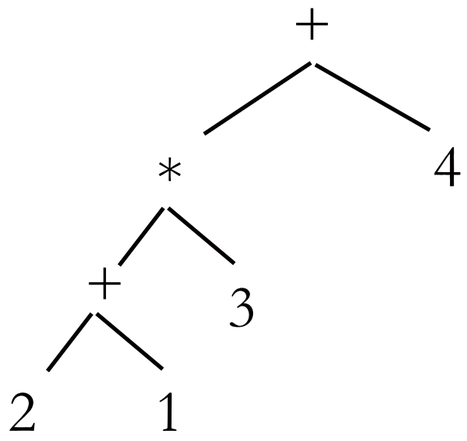
- Infix?
  - (1 * 2) + (3 * 4)
  - 1 * 2 + 3 * 4
  - (+ (* 1 2) (* 3 4))

- Evaluate (+ (* 1 2) (* 3 4))
  - Read +
    - Means?
      - Get first argument
      - Get second argument – Add them
  - First argument?
    - Another expression, evaluate it first
  - Read *
    - Means?
      - Get first argument
      - Get second argument – Multiply them
  - What next?

- Draw AST from prefix notation
  - First item (always an operator) in ( ) is a parent
    - Second is left child
    - Third is right child
  - Notes
    - A child can be the parent of another child
    - i.e. the start of another sub-tree
    - Children often called arguments, rather than operands

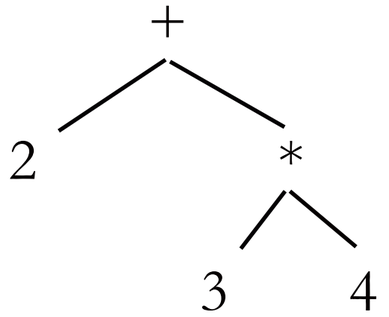
(+ (* 1 2) (* 3 4))

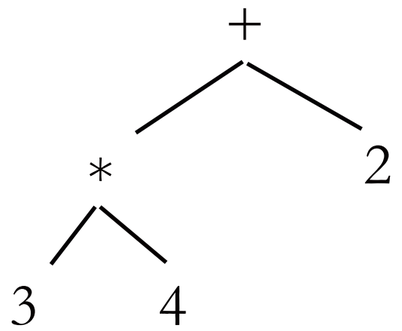
- Evaluating prefix?
  - Evaluate most deeply nested first
- (+ (* (+ 2 1) 3) 4)
- (+ (* 3 3) 4)
- AST?
  - Parse expression:
    - Read +
    - **Evaluate** (*..
    - Read *
    - **Evaluate** (+..
    - Read +
    - Read 2
    - Read 1
    - Add them



(+ (* 3 ..

- (+ 2 (* 3 4))
  - (+ 2 12)
- (+ (* 3 4) 2)
  - (+ 12 2)





- **Question**
  - If ASTs are representation independent, can any notation or representation be converted to one?

- Question
  - If ASTs are representation independent, can any notation or representation be converted to one?
  - Fortunately for us, yes.
  - Convert infix to AST
    - First item on left
    - Second becomes parent
    - Third on right

1 + 2

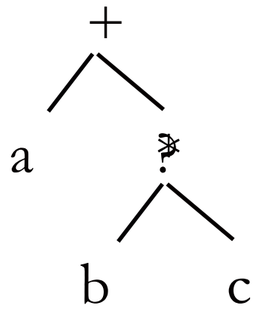
Why is this important?
Because we're scientists, not programmers!

Note: Assumes all operators are *binary*.

Binary: Take two operands.

This is why all trees (we've seen) have parent + two children

- AST for 1 + 2 + 3
  - Two operators, three operands
  - (1 + 2) + 3
  - Consider a + b * c
  - (a + (b * c))

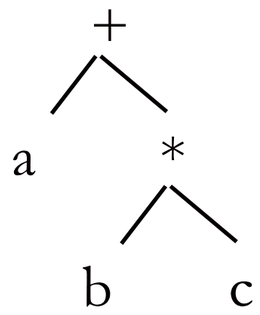


Binary: Take two operands.

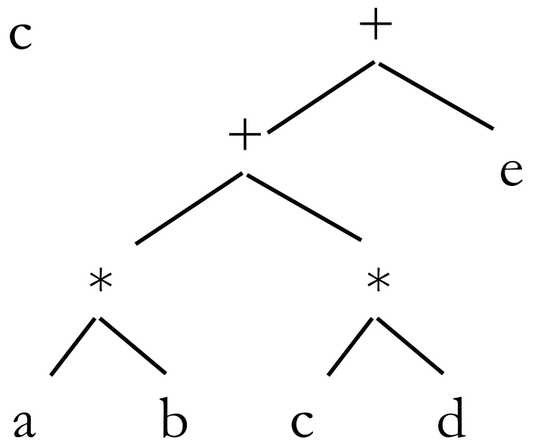
This is why all trees (we've seen) have parent + two children

All signs now binary

- (a + (b * c))



- a * b + c * d + e
- Group by precedence
  - (a * b) + (c * d) + e
- Make binary
  - (((a * b) + (c * d)) + e)
- Top operator?
- Most deeply nested operator goes to bottom of the tree
  - The **first thing evaluated**

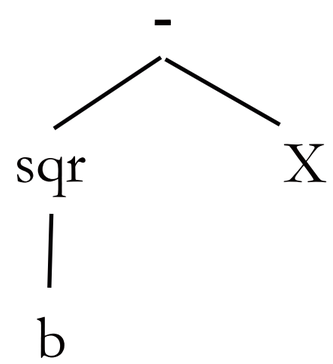


- How to write Sin X in infix?
  - Can’t -- must be prefix.
  - Infix often contains other representations
- What have we achieved?
  - Language independent representation for expressions (ASTs)
  - Prefix notation
    - Machine independent
    - Machine readable
    - Consistent

- Rewrite as prefix…
  - ± … not an operator
    - use two different expressions
  - b^2
    - (sqr b)
    - Is this fair?
      - Consistent with prefix
      - Unary argument

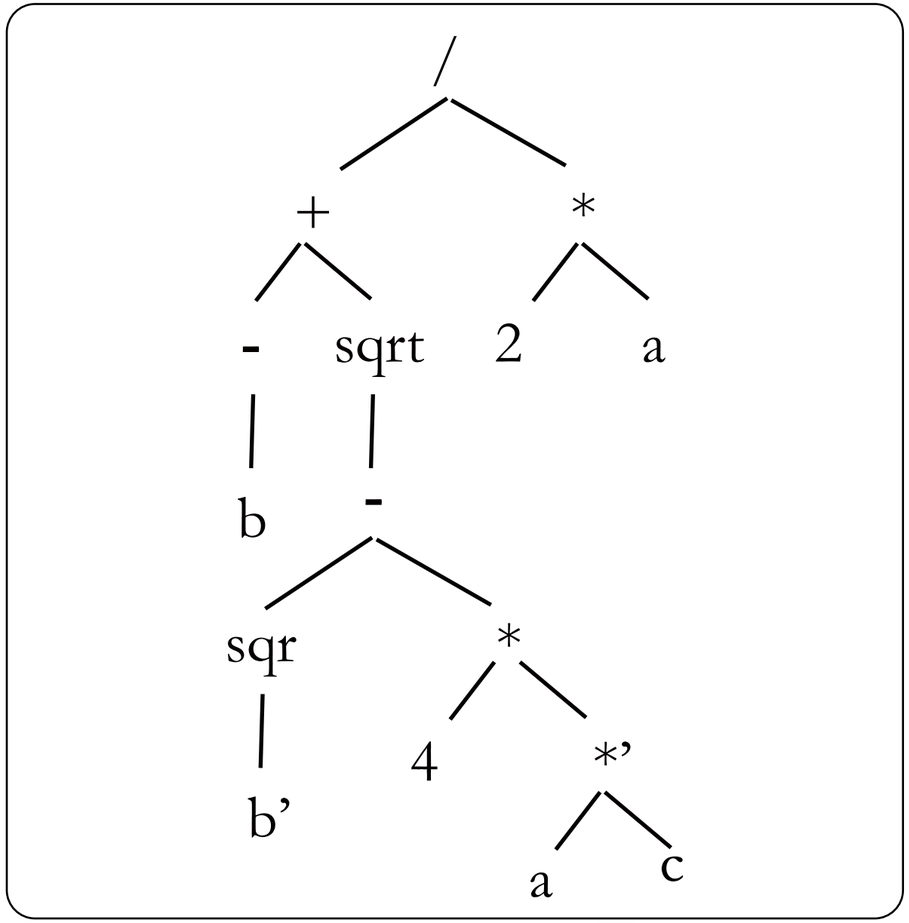
$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- b^2-X
- (sqr b) - X
- (- (sqr b) X)



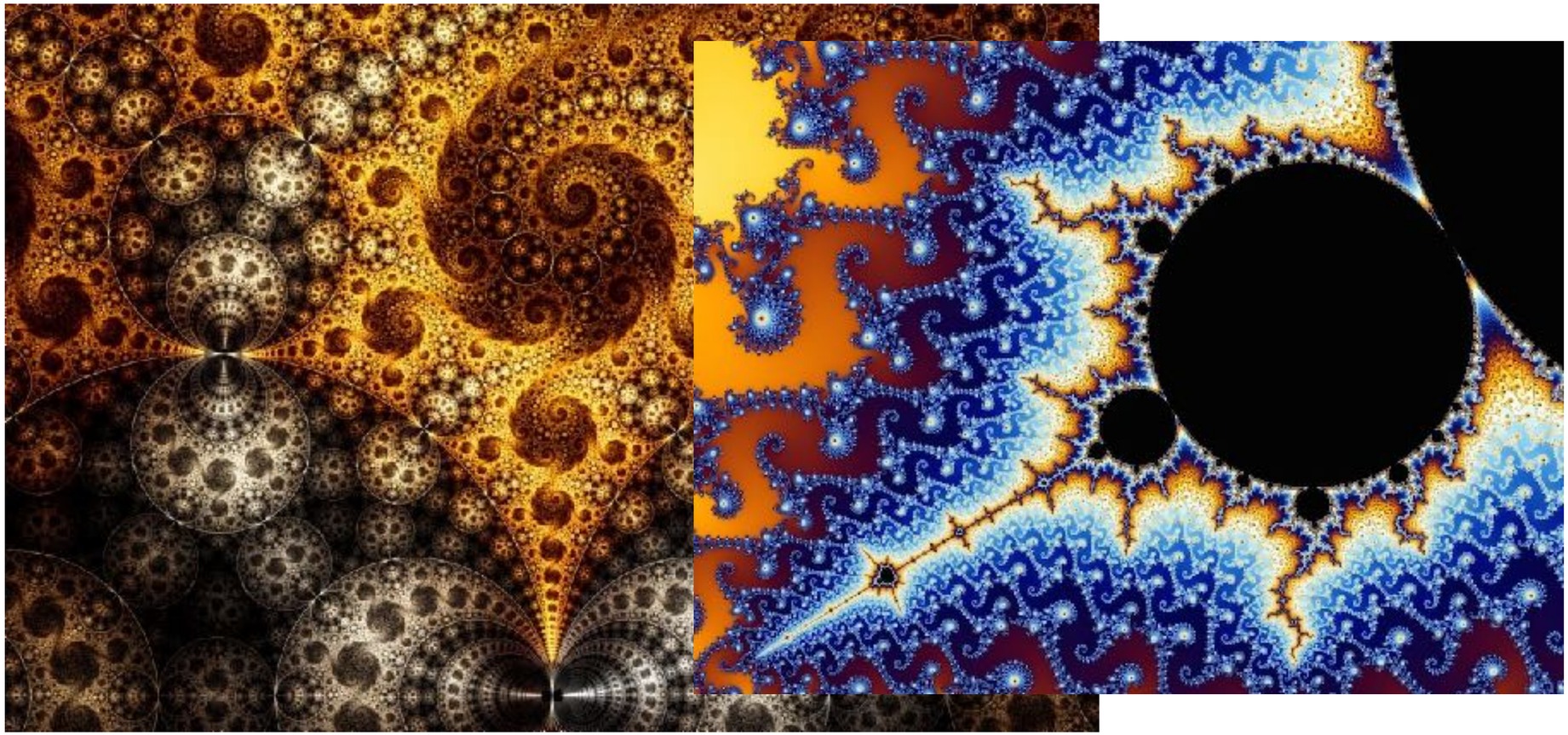
- Square root
  - Number of arguments? One
- Treat same as sqr
  - (sqrt x)
  - (sqrt (- a b))

```
(/ ( + ( - b)
       (sqrt
           (- (sqr b')
              (* (4 ( *' a c)))))
   (* 2 a))
```

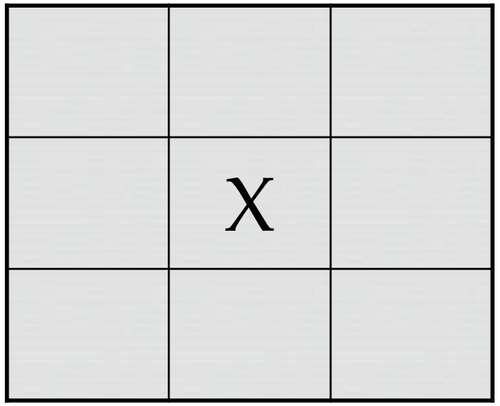


$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Global complexity through local interactions
  - Repeat the same **simple** action many times



Conway's Game of Life



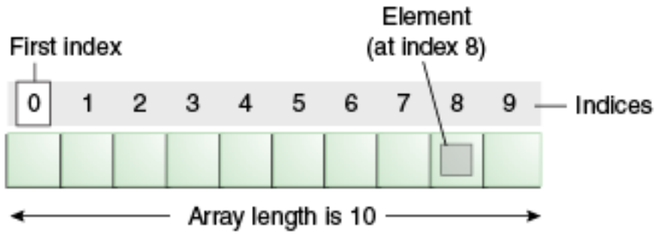
- Less than two neighbours the cell dies of loneliness
- Two or three neighbours, the cell stays alive
- More than three neighbours and the cell dies from overcrowding
- Dead cell with three neighbours becomes alive
- https://bitstorm.org/gameoflife/

Converting from infix to prefix

- Need a simple algorithm to convert any infix expression to corresponding prefix one
- (2 + (3 * 4))
- + 2 * 3 4
- (+ 2 (* 3 4))
- Respects the order of evaluation

Stacks

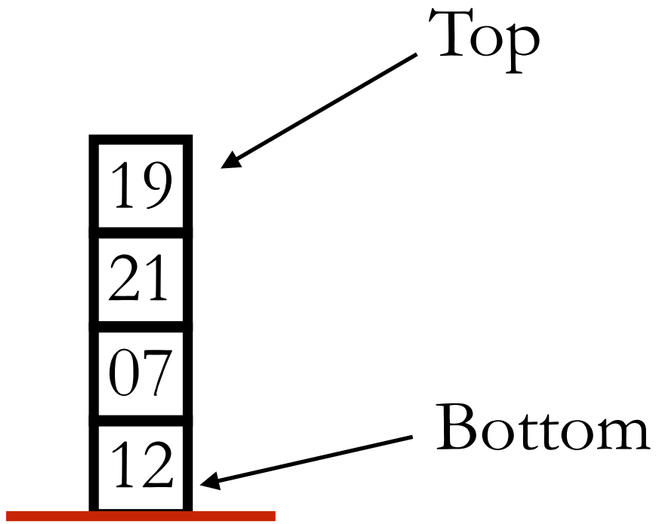
- *Data Structure*
  - Way of organising data in computer
- Operations
  - Add item to the data
  - Look at item
  - Remove item



Arrays
i[0]:=3;
x=i[1]+i[2];
Fixed length

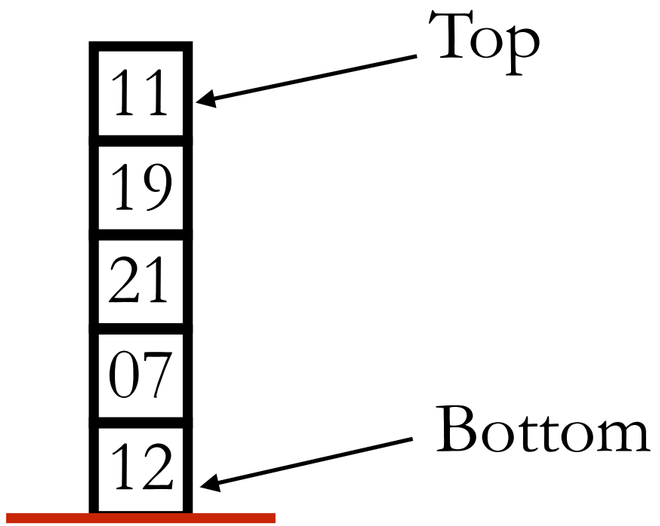
Stacks

- *Push* item on
  - PUSH 11



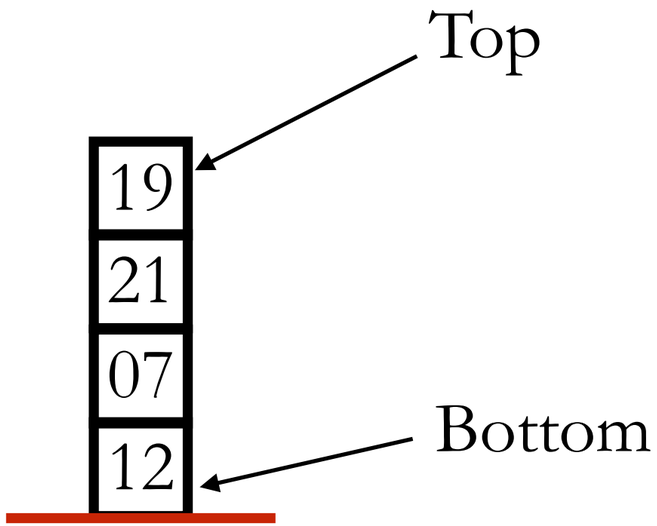
Stacks

- *Push* item on
  - PUSH 11
- *Pop* item off
  - POP



Stacks

- *Push* item on
  - PUSH 11
- *Pop* item off
  - POP
- PUSH (O)
- POP (X)



Examples

- Input string:
  - HELLO

Operations O OOOOX XXXX

Stack H E L L O

O/P O L L E H

Examples

- Input string:
  - HELLO
- Create:
  - HELLO

Stack H

O/P H E L L O

Operations O XOXOX OXOX

Examples

- Input string:
  - HELLO
- Create:
  - OHLEL

Operations O OOOOX

Stack H E L L O

O/P O

Not possible
Stacks are fast and simple
Somewhat restrictive
Dynamic data structure

Back to infix to prefix conversion

1. Reverse the expression
2. Read expression one character at a time:
   - “)”: Push onto stack
   - Operator: Push onto stack
   - Operand: Push on and pop off (straight to output)
   - “(“: Keep popping stack until “)” is encountered
3. Reverse the output

The Stack Method

Example

- Input string:
  - (3 + 1)
- Reverse:
  -) 1 + 3 (

Operations OOXOOXX

Stack) + 3

O/P 13+