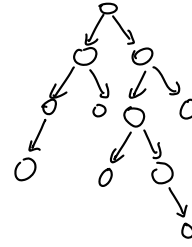


lecture 19

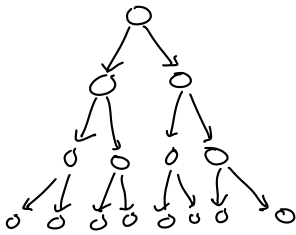
binary trees

- definitions
- traversals
- expression trees
- pre/in/post fix expressions

Binary Tree
 ≡ each node has at most 2 children

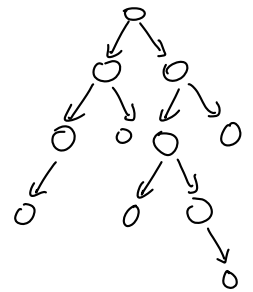
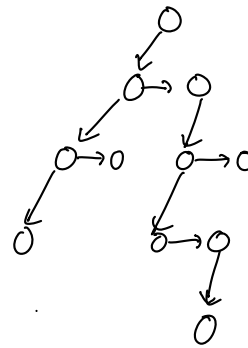


[n-ary tree -
 each node has at most n children]



What is the maximum number of nodes in a binary tree of height h?

$$n \leq \sum_{i=0}^{h-1} 2^i = 2^h - 1$$



```
class TreeNode<E> {
    E element
    TreeNode<E> firstChild
    TreeNode<E> nextSibling
}
```

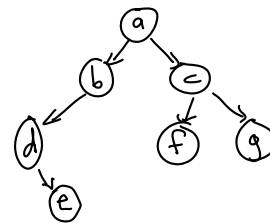
```
class BTreeNode<E> {
    E element
    BTreeNode<E> left
    BTreeNode<E> right
}
```

```
pre orderTraversal (root) {
    if root != null {
        visit root
        pre orderTraversal (left child)
        pre orderTraversal (right child)
    }
}
```

```
post orderTraversal (root) {
    if root != null {
        post orderTraversal (left child)
        post orderTraversal (right child)
        visit root
    }
}
```

```
in orderTraversal (root) {
    if root != null {
        in orderTraversal (left child)
        visit root
        in orderTraversal (right child)
    }
}
```

Example



preorder : abdecfg

in order : debafcg

post order : edbfgca

level order : abcdfge
 (breadth first)

no algorithm discussed today

Expressions with binary operators

$$a + 4 * y - 6 ^ 7 ^ 8$$

Recursive definition of expressions
(see COMP 330)

base Exp \equiv number | variable

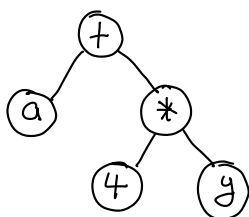
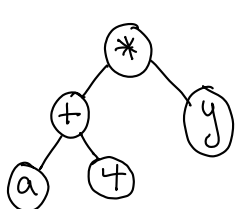
operator \equiv + | - | * | / | ^

exp \equiv baseExp | exp operator exp

where | means "or"

Expression trees

eg. $a + 4 * y$



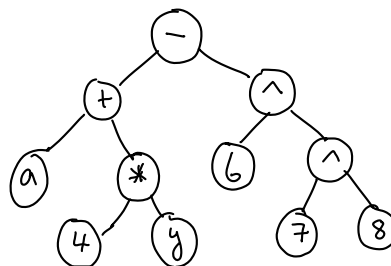
↑
the usual meaning

$$a + 4 * y - 6 ^ 7 ^ 8$$

$$\equiv a + 4 * y - 6 ^ 7 ^ 8$$

$$\equiv (a + (4 * y)) - (6 ^ (7 ^ 8))$$

need a rule



Evaluating an expression tree

```

eval ET (root) {
  if (root is a leaf)
    return root
  else {
    operand1 ← eval ET (left)
    operand2 ← eval ET (right)
    operator ← root.element
    return evaluate (operand1, operator, operand2)
  }
}
    
```



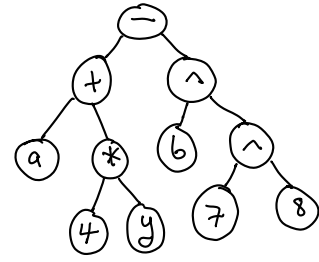
In fix expression $4 * y$

Pre fix expression $* 4 y$

Post fix expression $4 y *$

$\text{base Exp} \equiv \text{digit} \mid \text{letter}$
 $\text{op} \equiv + \mid - \mid * \mid / \mid ^$
 $\text{inExp} \equiv \text{baseExp} \mid \text{inExp op inExp}$
 $\text{preExp} \equiv \text{baseExp} \mid \text{op preExp preExp}$
 $\text{postExp} \equiv \text{baseExp} \mid \text{postExp postExp op}$

$\left\{ \begin{matrix} \text{In} \\ \text{Pre} \\ \text{Post} \end{matrix} \right\}$ order traversals yield $\left\{ \begin{matrix} \text{In} \\ \text{Pre} \\ \text{Post} \end{matrix} \right\}$ expressions



In fix $a + 4 * y - 6 ^ 7 ^ 8$
 Pre fix $- + a * 4 y ^ 6 ^ 7 8$
 Post fix $a 4 y * + 6 7 8 ^ ^ -$

Infix expressions are complicated to evaluate because operator precedence must be respected (e.g. * before +)

For prefix and postfix expressions, ordering of operations is determined by the expression itself.

e.g. postfix

$a 4 y * + 6 7 8 ^ ^ -$

Example : $a 4 y * + 6 7 8 ^ ^ -$

a
 a 4
 a 4 y
 a(4 y *)
 (a(4 y *) +)
 (a(4 y *) +) 6
 (a(4 y *) +) 6 7
 (a(4 y *) +) 6 7 8
 (a(4 y *) +) 6(7 8 ^)
 (a(4 y *) +) (6(7 8 ^) ^)
 ((a(4 y *) +) (6(7 8 ^) ^) -)

Evaluating a postfix expression

$S \leftarrow \text{empty stack}$
 $cur \leftarrow \text{head}$
 while (cur != null) {
 if (cur.element is number or variable)
 S.push(cur.element)
 else {
 operand 2 \leftarrow S.pop()
 operand 1 \leftarrow S.pop()
 result \leftarrow operand 1 cur.element operand 2
 S.push(result)
 }
 cur \leftarrow cur.next
 }