CS 313
- Kangmei Yang
Expression Tree

- A full (proper) binary tree to represent a mathematic expression
  - eg: 3 + 4

- Each internal node represents an operator which has exact two children

- Two children are the operands, first operand is the left subtree, second operand is the right subtree

- Leaf nodes are the values
Sample Expression Tree

- What expression does the above expression tree represent?

- operator *, first operand 3, second operand + (the sum of its first operand 4 and second operand 5)
  \[3 \times (4 + 5)\]
- the root is the LAST evaluated operator
Extract Expression to Tree

How to construct the tree from an expression?
- Find the root (last evaluated operation)
- Identify its first operand (left subtree)
- Set up its left subtree the same way
- Then the right subtree

1 + 2 - 3 * (4 + 5) / 6
- Order: left operand, operator, right operand in-order traversal?
  3 * 4 + 5
- Does these two evaluate the same?
  3 * (4 + 5)
- In Order alone doesn’t represent the correct order of operation
- Fully parenthesized expression
  (3 * (4 + 5))
Print Expression Tree

- How does following equation look like?
  $3 \times 4 + 5$

![Expression Tree Diagram]
- Prefix expression: Pre-Order traversal
  * 3 + 4 5
- Postfix expression: Post-Order traversal
  3 4 5 + *
- We may also construct the tree from either prefix expression or postfix expression
Prefix Expression to Tree

- First operator is the root
- Next value starts its left subtree
  - If its a numerical value, then it’s a leaf
  - Otherwise, another operator
- Once left subtree is complete, start its right subtree

- \(+ 1 2 / + 4 5 6\)
Postfix Expression to tree

Use stack to help store the operand to construct the tree
Read in the character
- If it’s a value, push on to the tree
- If it’s an operator
  - pop two operand from stack
  - construct the subtree tree
    - first popped operand is its right subtree
    - second one is its right subtree
  - operator is the root
  - push the root of this subtree to the stack

45 + 3 *