Binary Expression Trees

- The leaves are operands and the other nodes are operators.
- The left and right subtrees of an operator node represent subexpressions that must be evaluated before applying the operator at the root of the subtree.

**Binary Expression Trees: Examples**

- $a + b$
- $-a$
- $(a + b) * (c - d) / (e + f)$

**Merits of Binary Tree Form**

- Left and right operands are easy to visualize
- Code optimization algorithms work with the binary tree form of an expression
- Simple recursive evaluation of expression

**Levels Indicate Precedence**

- The levels of the nodes in the tree indicate their relative precedence of evaluation (we do not need parentheses to indicate precedence).
- Operations at lower levels of the tree are evaluated later than those at higher levels.
- The operation at the root is always the last operation performed.

**A Binary Expression Tree**

What value does it have? $(4 + 2) * 3 = 18$
Traversals and Expressions

- Inorder traversal produces the infix representation of the expression.
- Postorder traversal produces the postfix representation of the expression.
- Preorder traversal produces a representation that is the same as the way that the programming language Lisp processes arithmetic expressions!
class ExprTree {
    public:
        ExprTree();    // Constructor
        ~ExprTree();   // Destructor
        void build();  // build tree from prefix expression
        void expression() const;  // output expression in fully parenthesized infix form
        void clear(); // clear tree
        void showStructure() const; // display tree
    private:
        void showSub();    // recursive partners
        struct TreeNode *root;
};

Each node contains two pointers
struct TreeNode {
    InfoNode info;     // Data member
    TreeNode* left;    // Pointer to left child
    TreeNode* right;   // Pointer to right child
};

InfoNode has 2 forms
enum OpType { OPERATOR, OPERAND };
struct InfoNode {
    OpType whichType;
    union {  // ANONYMOUS union
        char operation;
        int operand;
    };
};

Expression Tree Algorithm
Read the postfix expression one symbol at at time:
- If the symbol is an operand, create a one-node tree and push a pointer to it onto the stack.
- If the symbol is an operator, pop two tree pointers T1 and T2 from the stack, and form a new tree whose root is the operator, and whose children are T1 and T2.
- Push the new tree pointer on the stack.
Example

a b + : a b c d e + : a b c d e + * : a b c d e * : a b c d e

Note: These stacks are depicted horizontally.