CS122 Algorithms and Data Structures

MW 11:00 am - 12:15 pm, MSEC 101

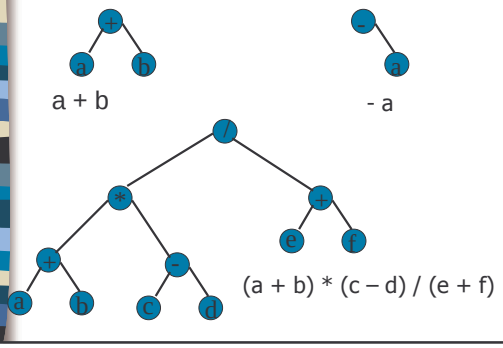
Instructor: Xiao Qin

Lecture 11: Binary Expression Trees

Uses for Binary Trees…
-- Binary Expression Trees

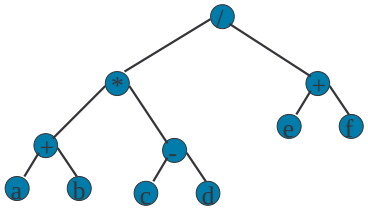
- Binary trees are a good way to express arithmetic expressions.
  - The leaves are operands and the other nodes are operators.
  - The left and right subtrees of an operator node represent subexpressions that must be evaluated before applying the operator at the root of the subtree.

Binary Expression Trees: Examples



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

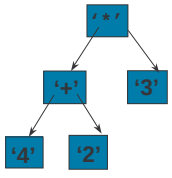
The levels of the nodes in the tree indicate their relative precedence of evaluation (we do not need parentheses to indicate precedence).

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.

5

A Binary Expression Tree

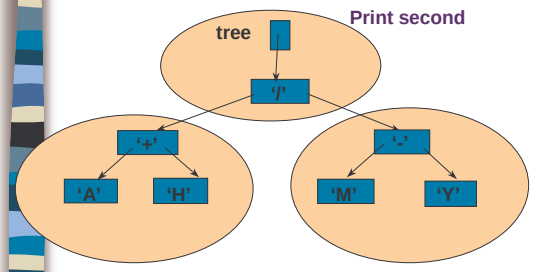


What value does it have?

(4 + 2) * 3 = 18

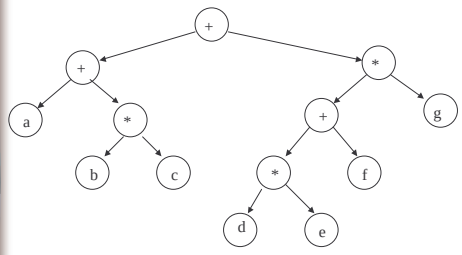
6

Inorder Traversal: (A + H) / (M - Y)



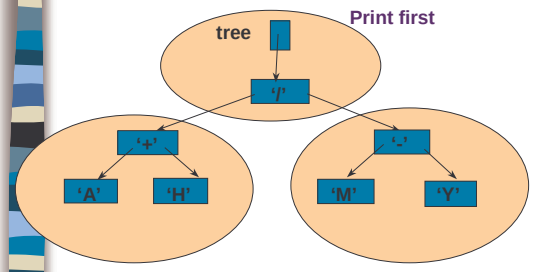
7

Inorder Traversal (cont.)



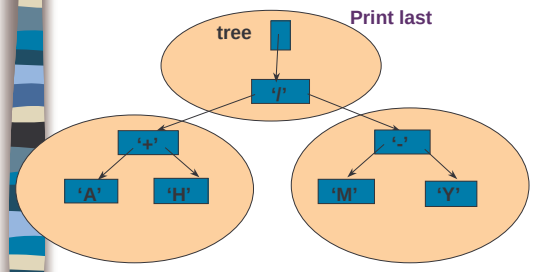
Inorder traversal yields: (a + (b * c)) + (((d * e) + f) * g)

Preorder Traversal: / + A H - M Y



9

Postorder Traversal: A H + M Y - /

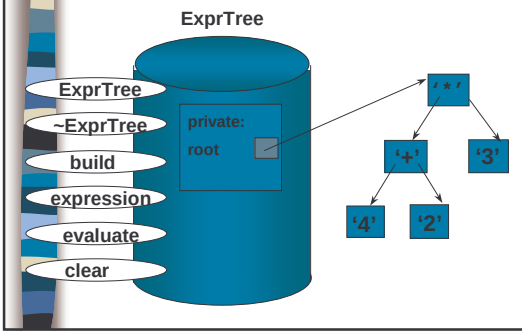


10

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



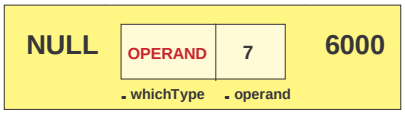
12

```
class   ExprTree  {
public:
   ExprTree ( );          // Constructor
   ~ExprTree ( );         // Destructor
   void   build ( );      // build tree from prefix expression
   void   expression ( ) const;
   // output expression in fully parenthesized infix form
   float  evaluate ( ) const;          // evaluate expression
   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
};
```



.left .info .right

14

InfoNode has 2 forms

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

. whichType . operation . whichType . operand

15

```
int   Eval ( TreeNode*  ptr )

{    switch  ( ptr->info.whichType )
     {
        case OPERAND :  return  ptr->info.operand ;
        case OPERATOR :
           switch ( tree->info.operation )
           {
              case  '+'  :  return  ( Eval ( ptr->left ) + Eval ( ptr->right ) ) ;

              case  '-'  :  return  ( Eval ( ptr->left )  -  Eval ( ptr->right ) ) ;

              case  '*'  :  return  ( Eval ( ptr->left )  *  Eval ( ptr->right ) ) ;

              case  '/'  :  return  ( Eval ( ptr->left )  /  Eval ( ptr->right ) ) ;
           }
     }
}
```

16

Constructing an Expression Tree

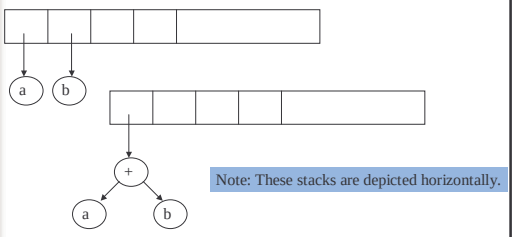
- There is a simple O(*N*) stack-based algorithm to convert a postfix expression into an expression tree.
- Recall we also have an algorithm to convert an infix expression into postfix, so we can also convert an infix expression into an expression tree without difficulty (in O(*N*) time).

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

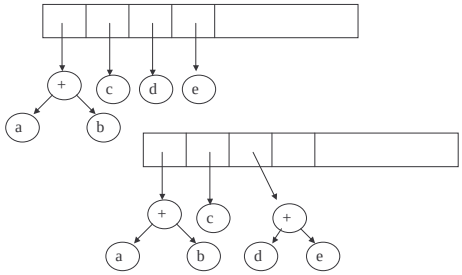
a b + :



Note: These stacks are depicted horizontally.

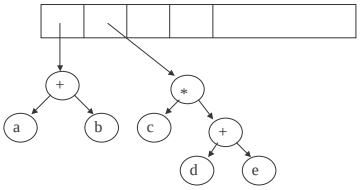
Example

a b + c d e + :



Example

a b + c d e + * :



Example

a b + c d e + * * :