Introduction to Prefix, Infix and Postfix.

Recursive Tree traversals: Applications of each of them, connection Project 4.

Building the expression tree while executing the 2-stacks algorithm.

Why the 2-stacks algorithm works.

Folder/Document (Directory/File) viewer.

Place of humans in NIH's taxonomy tree: Microsoft Windows Explorer.

CSI 310: Lecture 21
then; and pay attention. (NOT a good computer algorithm)!

The "by hand" process: Write values of subexpressions over, underneath, or near

DSO 7.2) WITH the "by hand" process.

Compare the operation of the 2-stacks algorithm (see the two stacks example

Why the 2-stacks algorithm works:
(fail if checks fail)
check tok matches LEXTK
  numbers.push(result);
operand operand operand;
  calculate the numeric result of
  check LEXTK is +,-,* or /; check OPERAT is (], or >
  operand = numbers.pop();
  operand = operators.pop();
  LEXTK = operators.pop();
  }
else if (is-right tok)
  operators.push(tok);
else if (is-left tok)
  operators.push(tok);
else if (is-operator tok)
  numbers.push(tok);
  if (is-num(tok)
    ( ( )
  while tok = nexttoken())

Each subexpression corresponds to a number stack entry. When two numbers are popped, combined, and pushed, the new stack entry corresponds to the combination of the two old stack entries.
First, some basic structures to implement a binary tree...

(a) Build the expression tree.

(b) Traverse the expression tree 3 ways.

Part 2:
Binary Tree Node with Internal Data

```c
struct Node {  
  struct Data {  
    // perhaps additional stuff  
  };  
  char dch;  
  Node *pL;  
  Node *pR;  
};
```

Binary Tree Node with External Data

```c
struct NodeE {  
  // perhaps constructors & accessors;  
  char mch;  
} NodeE *pL;  
NodeE *pR;  
struct Data *pD;  
```

Binary Tree Node

```c
struct Node {  
  // perhaps constructors & accessors;  
  Node *pr;  
  Node *pl;  
  Data *pd;  
} struct Node
```

Binary Tree Node

```c
struct NodeE {  
  // perhaps constructors & accessors;  
  char dch;  
} struct NodeE
```

```
with External Data
Binary Tree Node
```
Node *tL = new Node;
tL->pL=NULL; tL->pR=NULL;
tL->dch = 'B';

Node *tR = new Node;
tR->pL=NULL; tR->pR=NULL;
tR->dch = 'C';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'A';

tL = tT;
tR = tT;

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'A';

tL = tT;
tR = tT;

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'B';

tL = tT;
tR = tT;

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'C';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'A';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'A';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'B';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'C';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'A';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'A';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'B';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'C';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'A';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'A';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'B';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'C';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'A';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'A';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'B';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'C';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'A';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'A';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'B';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'C';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'A';

Node *tT = new Node;
tT->pL=NULL; tT->pR=NULL;
tT->dch = 'A';
1. Finish building each subtree as soon as the corresponding number is pushed in the number stack.

2. Store in each number stack structure a pointer to root of the subtree corresponding to that number.

IDEAS:

Part 2 of Project 4 Goal: Build the expression tree.

Toolkit C++-Object Oriented Style. (Prevent C/SI333/402 complaints.)
There are TWO separate expression trees. (Tree for "(3+4)"")

The red numbers are the Steps In Time when tree nodes are constructed.

Step 4: (Tree for "2")

After processing digit '2', the next Step is Step 2: (Tree for "(3+4)"")

The red numbers are the Steps In Time when tree nodes are constructed.
Reduction of $*$ step 5.

$((3+4)*2)$
(print specified results of each step).

equal the expression tree value evaluated by post-order traversal.

Finally program a TEST whether the SAVED value (from your part algorithm)

test and print that tree in post-order form.

4. Evaluate and print that tree in in-order form.

3. Print that tree in in-order form.

2. Print that tree in pre-order form.

1. Count the nodes in the expression tree.

Printed AND SAVED for the TEST, do 4 tree traversals:

Once the expression tree is built, and the value from the 2-stacks algorithm is
process PN->data (C) 
{
    ... 
    traverse(pn->right) ; 
    ... 
    } (B) 
    ( if (pn->right = NULL) 
    {
    ... 
    traverse(pn->left) ; 
    ... 
    } 
    ( if (pn->left = NULL) 
    Code these in any order. 
    ... 
    ) (A) 
    ( if (pn) 
    ( if (PN->left) 
    ( if (PN->right) 
    recursive tree traversal algorithms... Really Easy! 

University at Albany Computer Science Dept.
{ 
return;

if ( pn->pnext != NULL ) 
leaf ( pn->pnext )

cout >> pn->p);

if ( pn->pnext != NULL ) 
leaf ( pn->pnext )

assert ( pn != NULL )
}

void Inorder ( BinaryNode *pn )

(the usual) infix notation:
IN-ORDER binary tree traversal—prints the expression expressed by the tree in


```c
{
    // etc...
    //
    if (pn < d == 0)
        return
    static_cast<
        double
        (pn->d - 0.0)
        (?

        if (isDigit(pn->d))
            return
        last step: compute and return the EVALUATION.
        if (pn->pn->pright == NULL) (value = postord (pn->pright));
        if (pn->pn->pleft == NULL) (value = postord (pn->pleft));
        double value, result;
        assert (*Either *pn is a leaf OR *pn has two children*/);
        assert (pn != NULL);
    
    double postord (BinaryNode *pn);
}

using namespace std;

#include <ctype>

#include "PostOrderBinaryTreeTraversal EVALUATE the expression tree";
```

University at Albany Computer Science Dept.
preOrder (root, 0 ); //call with depth=0 to print whole tree.
{
  return;
}
}

etc. traverse all the subtrees if not binary

if ( pn->left != NULL ) preOrder ( pn->left, depth+1 );
if ( pn->right != NULL ) preOrder ( pn->right, depth+1 );

cout << pn->data;
/* indent depth spaces */
assert( pn != NULL );
}

void preOrder ( BTreeNode *pn, int depth )
         // DEPTH-prints the "DOM" tree view of the tree.