Tuesday, September 22, 2009
3:51 PM

Asg. 3

- implement postfix expression evaluator first
- then work on getting infix to postfix conversion
- use stringstream objects to compose postfix expression & then parse/evaluate it
  (like you would have done with cin >>)
- examples of stringstreams { std::istringstream
  std::ostringstream

```
std::ostringstream os;
```

- use this just like std::cout, e.g.,

```
os << operand << " ";
```

- use `os.str("");` to clear os (ostringstream)
- to print os, do this:

```
std::cout << os.str().c_str();
```

`str()` → C++ string obj; `c_str()` → the char * contained within

- using std::istringstream:

```
std::istringstream is(expression);
```

`expression` → const char *

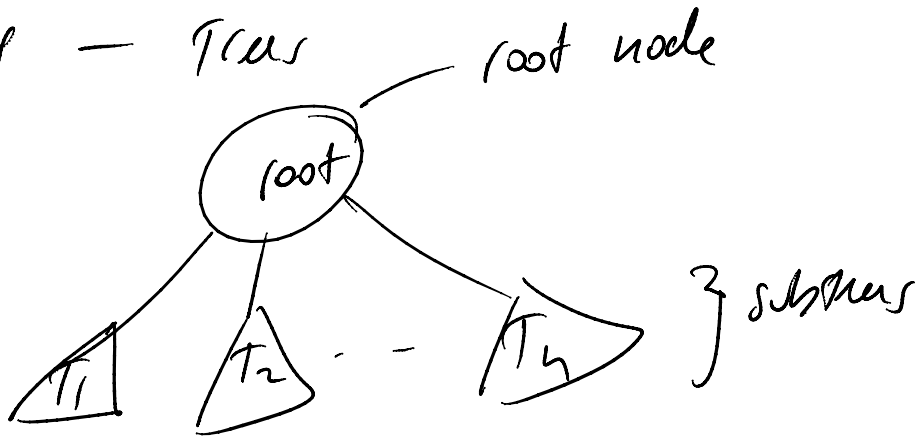
then use is just like std::cin, e.g.,

std::cin, e.g.,

```
while( c = is.get() ) {
    if ( isdigit(c) ) { is.putback(c); is >> num; }
    else {
        : // process punct
    }
```

Tuesday, September 22, 2009
4:05 PM

Chapter 4 — Trees



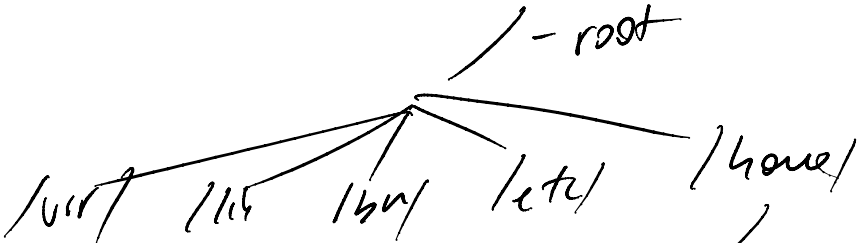
- in general, each node can have any number of children

e.g. struct Node {

```
    std::vector<Node*> child;
};
```

each node maintains a list (linked list) of children

- e.g., Unix directory is a large n-ary tree
  (each dir can have n subdirs)



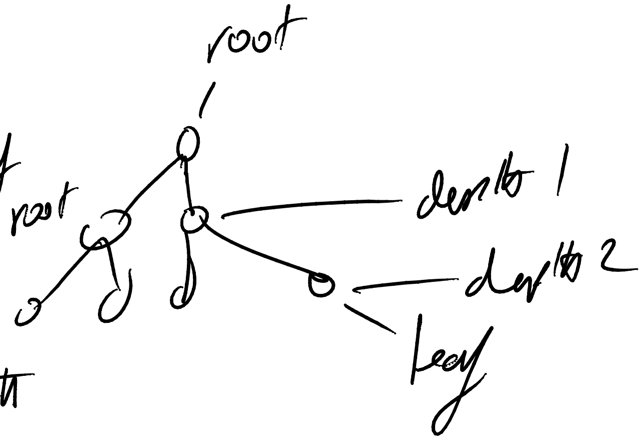
/local/ /include/

/bin/

andrew/

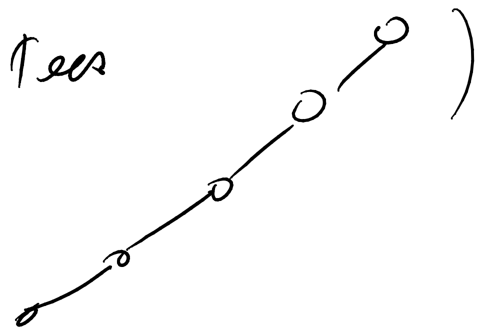
- some tree terms:

depth of a node: length of path from root



height of a tree

↳ longest path to a leaf

(be wary of degenerate trees )

parent, child

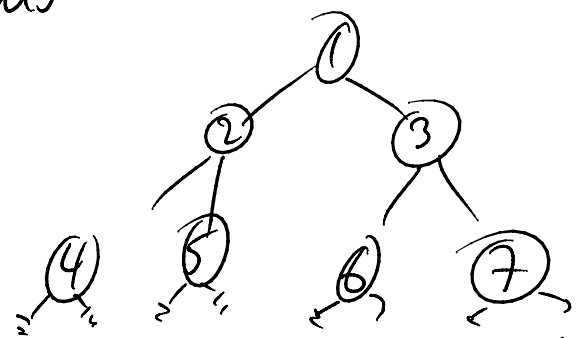
- implementation: (binary search tree)

↑ specialized case, good for sorting & searching

```
struct Node
{
    T data;
    Node *left;
    Node *right;
}
```

Tuesday, September 22, 2009
4:17 PM

Traversals



pre-order: root, left, right (e.g. in printing)

1, 2, 4, 5, 3, 6, 7

post-order: left, right, root

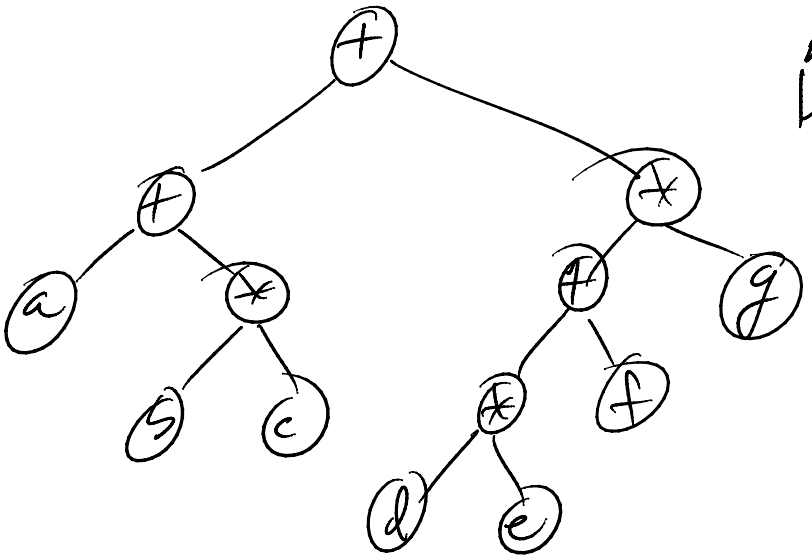
4, 5, 2, 6, 7, 3, 1

in-order: left, root, right

4, 2, 5, 1, 6, 3, 7

Tuesday, September 22, 2009
4:22 PM

Application of inorder traversal — expressions



nodes: operators
leaves: operands

- print in in-order:

$$(a + (b * c)) + (((d * e) + f) * g)$$

- post-order:

a b c * + d e * f + g * +

- Note: note precedence operators

( ) * / + −

special case in asg 3

when popping (in asg 3), check predence order
— use ASCII table
(on Linux box, type 'man ascii' to get ascii table)

Tuesday, September 22, 2009
4:29 PM

- Building an expression tree:

given postfix expression,

if symbol is an operand,

create a leaf & push onto stack

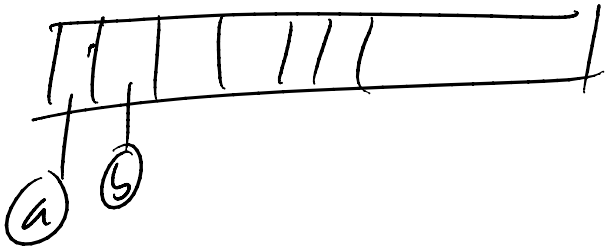
if symbol is an operator,

~~pop~~ pop two tree pointers;

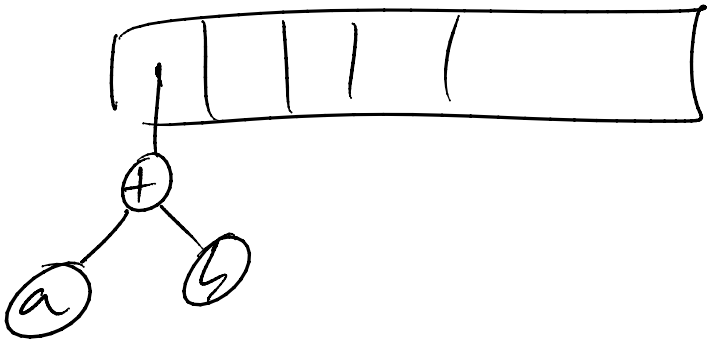
form a new subtree

with operator as root

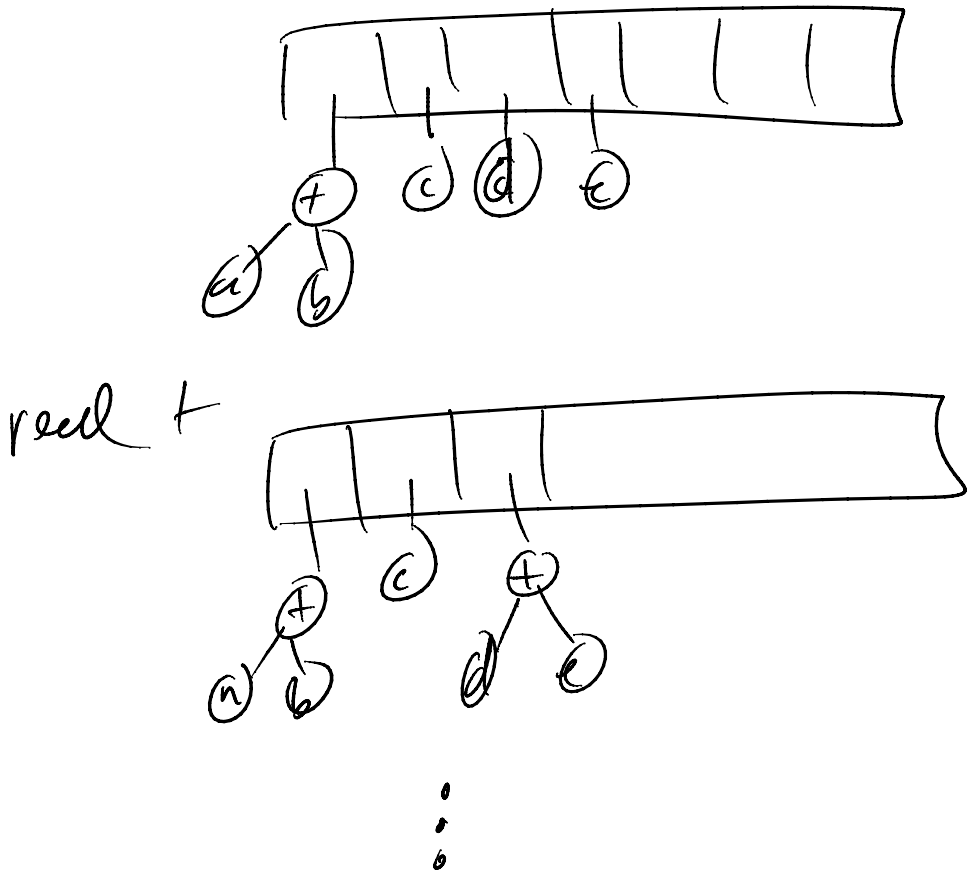
a b + c d e + * *



+



read c, d, e


read +
+
c
d
e
a
b
+
c
+
+
a
b
d
e

Tuesday, September 22, 2009
4:33 PM

4.3.1 Binary Search Trees:

- one of the more important ADTs
- idea is for find() to take O(lg n) instead of O(n)
- key property: all nodes in left subtree are smaller than in right
- we can place any kind of object into tree so long as you define operator< defined

- tree operations:
  contains(): the find operation
  ↓
  key observation:
  "on average" (balanced tree)
  you discard
  searching ½ nodes at
  top level

  min(): return left-most child
  max(): " right-most child
  insert(): like find, then attach as child
  remove(): lot more difficult