Lecture 19: Iterators, Expression Trees & Array Representation of Trees

CS 62
Fall 2017
Kim Bruce & Alexandra Papoutsaki

Friday

- Quiz on Stacks, Queues, and Trees
  - through today
- Representing trees with arrays

Look at BinaryTree.java

Notice leaves are nodes w/null values

Iterators

- Pre-order: root, left subtree, right subtree
- Post-order: left subtree, right subtree, root
- In-order: left subtree, root, right subtree.

Pre-order

```
if (!isEmpty( )) {
   doSomething to this.value()
   left.inOrder()
  right.inOrder()
}
```

In-order

```
if (!isEmpty()){
   left.inOrder()
   doSomething to this.value()
   right.inOrder()
}
```

Post-order

```
if (!isEmpty()){
   left.inOrder()
   right.inOrder()
   doSomething to this.value()
}
```

Iterators

- How can you stop recursive program in middle and then restart when need next value?
- Simulate recursion with stack!

Pre-order Iterator for Trees

```
protected Stack<BinaryTree<E>> todo = = new StackList<BinaryTree<E>>();
if (root != null) todo.push(root);

public boolean hasNext() {
   return !todo.isEmpty();
}
public E next()
    {
        BinaryTree<E> old = todo.pop();
        E result = old.value();

        if (!old.right().isEmpty()) todo.push(old.right());
        if (!old.left().isEmpty()) todo.push(old.left());
        return result;
    }
```

Other iterators are more complicated!
See BTPostorderIterator, etc.

Iterators for Lists

- Method iterator easy to implement
  - for (E elt: myList) { doSomething(elt)}
- Alternative :
  - myList.forEach(x -> doSomething)
- Different strategies:
  - In first, let from list is parameter to operation *(active)*
  - In second, operation is parameter to list *(passive)*

Define Methods Using Lambdas

From BinaryTree8 *(see code link)*

```
    public void doPostorder(Consumer<? super E> action) {
         if(!isEmpty()) {
            left.doInorder(action);
            right.doInorder(action);
            action.accept(val);
         }
    }

    full.doPostorder(String s -> {System.out.println(s);});
```

An action is a lambda expression that take an element of type E and returns type void (i.e., does an action)

Calculating Using Lambdas

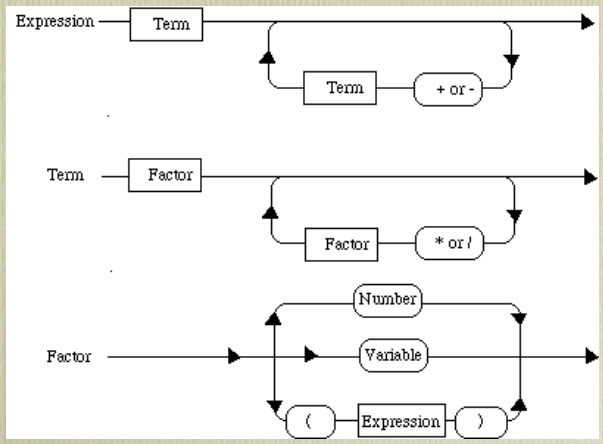
```
public E calcPostorder(TrinaryFunction<E> operation, E id) {
     if(!isEmpty()) {
        return operation.apply(left.calcPostorder(operation,id),
                               val,
                               right.calcPostorder(operation,id));
     }
     return id;
}

System.out.println("The sum is "+
         full.calcPostorder((left,root,right) -> left + root + right, 0));
```

But can't modify variables outside:

```
int sum = 0;
myTree.doPostorder(s -> sum = sum + s);
```

Illegal!

Parsing Expressions



Representing Expressions

- Represent 3 * 7 + 6 / 2 - (3 + 7) as special tree
  - Parser builds tree
  - Send message to tree to print or evaluate
- Mutual recursion in parser
- Different classes for different kinds of nodes.
- See Parser code

Similar to what you did in CS 52

Making Changes

- Easy to add new types of nodes
  - E.g., exponentiation, negation
- Hard to add new operations or modify existing
  - Must go in and change every different kind of node

Visitor Pattern

- Take advantage of lambda expressions to add new operations
- Each node has a process method taking a visitor.

```
public <T> T process(Visitor<T> aVisitor) throws VisitorException
```

- process method asks the visitor to perform appropriate operation on it with internal information
- See Interpreter and PrettyPrinter visitors.