**Parsing Concepts**

**Document Id 25**

**Well Formed Expressions**

A well-formed expression must conform to the grammar for an expression

Example XML documents are well formed <tag></tag>

Example HTML documents are not well formed

Is possible to build xml parsers but not possible to build html parsers

A regular language uses well-formed expressions and hence can be parsed by regular expressions

Parsers depend on regular expressions and context-free grammars

**switch** (c) {

**case '+'**:

rslt = leftVal + rightVal;

**break**;

**case '-'**:

rslt = leftVal - rightVal;

**break**;

**case '\*'**:

rslt = leftVal \* rightVal;

**break**;

**case '/'**:

rslt = leftVal / rightVal;

**break**;

**default**:

**throw new** IllegalArgumentException(**"invalid postfix expression"**);

}

**Abstract Syntax Tree (AST)**

The most basic AST will use stacks and a queues

Loop through all your expressions

End up with a stack of tokens

Add operators to the operator stack and generally constants / values to a queue

The stack will hold operators and constants / values will be on the queue

Then you will rebuild the queue with a parallel stack and end up with operator value expressions based on priorities(grammer)

**Semantic validation**

A language is a set of valid sentences

validity ⇐> correct syntax and semantics

Queues are important as expressions can be semantically validated after expression syntax checks

function call semantics

Call stacks are only necessary in languages that allow recursion or mutual recursion

<https://en.wikipedia.org/wiki/Call_stack>

**Implement a simple AST parser**

**public class A {**

**int i = 9;**

**int j;**

**ArrayList<Integer> al = new ArrayList<Integer>();**

**j=1000;**

**}**