Language Processors Lab 4

Note: Read this through *before* logging in.

The goal for this lab is to see some JavaCC parsers working that have embedded actions to do calculations and build syntax trees, and how to traverse the trees.

Expression evaluator

Move to the directory in which you want to do your work and copy my directory and set up the shell to use JavaCC etc with the commands:

```
cp -R /soi/sw/courses/daveb/IN2009/ebnfcalc .
cd ebnfcalc
module add java soi javacc/3.2
```

The directory contains a JavaCC specification (tokens and grammar) for a simple expression language in file `Exp.jj`. It also contains actions in the grammar that turns the parser into an evaluator for expressions. The file `Main.java` contains a class Main with the `main()` method that sets everything up and calls the parser. You can create the evaluator with:

```
javacc Exp.jj
javac Main.java
```

and then run it with:

```
java Main
```

If you type in a legal expression (eg `3+4*5+9`), the expression will be evaluated and the result printed out. Read the `Exp.jj` and `Main.java` files and make sure you understand how it works.

There is a BNF version of the expression evaluator at `/soi/sw/courses/daveb/IN2009/bnfcalc`. Copy it and try it in the same way as above. It is a little different to the EBNF version, since the non-terminal rules have to take parameters as well as deliver values in order to perform the calculations.

Abstract syntax tree

A version of the expression evaluator that builds abstract syntax trees and traverses them to perform the evaluation is at `/soi/sw/courses/daveb/IN2009/ebnfastcalc`. The `Exp.jj` file now contains actions in the grammar that build syntax trees rather than perform evaluations. The syntax tree classes are in the subdirectory `syntaxtree`, and each class has an `eval()` method that performs the evaluations and recursively calls `eval()` on any subtrees. Copy it and try it, and then read and understand the various files.

Visitors

A version of the expression evaluator that builds abstract syntax trees and then traverses them using the visitor pattern is at `/soi/sw/courses/daveb/IN2009/ebnfviscalc`. The visitor classes are in subdirectory `visitor`; each syntax tree class now contains an `accept()` method for a visitor. Copy it and try it, and then read and understand the various files.

This example is extended with a simple pretty printer for the syntax trees in `/soi/sw/courses/daveb/IN2009/ebnfvisprint`. Before you look at how I have done it (see file `visitor/AstPrint.java`), work out how you would do it.

MiniJava abstract syntax trees and visitors

The MiniJava parser that constructs abstract syntax trees is at `/soi/sw/courses/daveb/IN2009/minijava/chap4`. The `README` file explains how to compile and run it.