Language Processors Lab 2

Note: Read this through *before* logging in.

The goal for this lab is to see a JavaCC-generated lexical analyser working to simply print out some recognised tokens, and then to write some of your own expressions and implement them.

A trivial lexical analyser

Fire up a Unix shell window. To get the environment ready type the command:

```
module add java soi javacc/3.2
```

The module `javacc/3.2` gives you access to the JavaCC tool that includes a lexical analyser builder.

I have written a simple JavaCC input file which recognises a small number of token types. Move to the directory in which you want to do your IN2009 work and copy the example directory with the command:

```
cp -R /soi/sw/courses/daveb/IN2009/lextest .
cd lextest
```

You can see the JavaCC file with:

```
more LexTest.jj
```

(or you can edit it with your editor of choice, of course). Now run JavaCC on the script file with

```
javacc LexTest.jj
```

This produces a Java program in various files. This program is the lexical analyser and recognizes the tokens specified in the `LexTest.jj` file. Now compile these Java classes with:

```
javac *.java
```

And then run the program:

```
java LexTest
```

Type in some identifier names and integers and see what happens.

Now look at the file `LexTest.jj`. The two forms of token (SKIP and TOKEN) are demonstrated, along with most of the kinds JavaCC regular expression, and also local definitions (prefixed by a '`#`' in a TOKEN definition). See the JavaCC document for full details. The syntax-definitions part of this file simply matches tokens and prints them out. Notice that it is possible to capture the token recognised in a Token object (here `Token t`) and then to access and print its kind (from the table `tokenImage` indexed by `Token` field '`kind`') and the string that was matched (from `Token` field '`image`'). TOKENs which are defined as simple strings (eg `KEYTRUE`) will be printed as the string rather than the name (`KEYTRUE`), whereas for those with more complex definitions the names are printed (eg `<IDENTIFIER>`)

Another example can be found in `/soi/sw/courses/daveb/IN2009/appel2.9` (see the README file in the directory).

Modifying the analyser

(Part of the first coursework will ask a question something like this.)

Replace the regular expression definition of the REAL token in the `LexTest.jj` file to instead match signed real numbers as written in Pascal. Such numbers must contain a decimal point, and at least one digit before and after the decimal point. They may optionally be followed by a signed exponent that begins with the letter '`E`' and is followed by a (possibly signed) integer. In this notation, `39.37`, `-6.336E4`, `0.894E-4` and `0.0` are legal, while `.36`, `4.` and `+.7E6` are illegal.