Language Processors Lab 5
Modifying the MiniJava parser

The goal of this lab is to modify the MiniJava parser, which includes the creation of a new AST class and modification of the JavaCC file and visitor classes.

The MiniJava Parser

Move to the directory in which you want to do your work and copy the directory containing the MiniJava parser files with the command:

```
cp –R /soi/sw/courses/daveb/IN2009/minijava/chap4 .
cd chap4
module add java soi javacc/3.2
```

Follow the instructions in the README file to compile the MiniJava parser. The program is executing with the command:

java Main *filename*

Where *filename* is the name of a file containing a MiniJava program. You'll have to create such a file. E.g. create the file `test.minijava` with the following contents:

```
class Test {
  public static void main(String[] args)
    {   System.out.println((new A()).m()); }}

class A {
    public int m() {
        int x; int y; int z;
        x =3; y = x * 10; return 5; }}
```

Execute the program. If you get any parse errors, fix the contents of test.minijava. What's the output? Replace `return 5` with `return (x / y).` Execute the program – what happens?

Adding a new type of expression – the DIVISION operator

The current grammar does not consider the division ("/") operator. We can add the division operator by first entering the following production (minijava.jj file) inside the Expression() non-terminal specification:

```
LOOKAHEAD( PrimaryExpression() "/" )
  e=DivisionExpression()
```

and then inserting the code below to the minijava.jj file e.g. right after the TimesExpression() specification.

```
Exp DivisionExpression() :
{ Exp e1,e2; }
{
  e1=PrimaryExpression() "/" e2=PrimaryExpression()
  { return new And(e1,e2); }
}
```

Note that the current action calls to the `And` constructor. This is a temporary fix until we create the Division java class file. Modify test.minijava by replacing `return 5` with `return (x / y).` Execute the program; what's the output?

The AST class for DIVISION

Now we need to create the AST for the new Division expression. Create the file Division.java inside the syntaxtree directory with the following contents:

```
package syntaxtree;
import visitor.Visitor;

public class Division extends Exp {
  public Exp e1,e2;
  public Division(Exp ae1, Exp ae2) {
    e1=ae1; e2=ae2;
  }
  public void accept(Visitor v) {
    v.visit(this);
  }}
```

Modifying the Visitor class and pretty printer

Add the method declaration:

```
public void visit(Division n);
```

to the visitor/Visitor.java interface. Add the method definition to the vistor/DBPrettyVisitor.java file:

```
public void visit(Division n) {
    System.out.print("(");
    n.e1.accept(this);
    System.out.print(" / ");
    n.e2.accept(this);
    System.out.print(")");
  }
```

Putting everything together

Compile the new files and changes with the following commands (from the chap4 directory):

```
javac syntaxtree/Division.java
javac visitor/Visitor.java
javac visitor/DBPrettyVisitor.java
```

Execute the program.