Language Processors Lab 6 - The Source Programming Langue

In this lab you will work with the grammar, pretty-printer and interpreter of a subset of the Source Programming Language. The pretty-printer and interpreter are implemented as visitors. This will be another opportunity to practice the use and modification of AST visitors.

Downloading the files

Start a Unix shell window and move to your LanguageProcessors directory. Download the file `sourcepl.tar.gz` from CitySpace (week8) or from `http://www.soi.city.ac.uk/~sbbc287/sourcepl.tar.gz`. Unzip and untar the file with the following commands:

```
gunzip sourcepl.tar.gz       // this will genenerate sourcepl.tar
tar -xvf sourcepl.tar
```

The last command will generate the `sourcepl` directory. It is possible that, while downloading, the file system automatically unzips, and even expands (untars), the file. If that's the case, you may have to skip one or both of the above commands, though make sure to copy the new directory. Load java and javacc by executing the command `module add java javacc`.

The Program

The contents of the `sourcepl` directory are:

- `sourcepl.jj`: Defines the grammar of the Source Programming Language (SPL), as introduced in week8's lecture. It contains Java code that builds the abstract syntax tree (AST) of expressionsan SPL program.
- `Main.java`: Entry point of the program. It calls the parser, stores the AST (of class Program) and instantiates and calls the pretty-printer and interpreter vistor. At each run, the program displays the SPL program text and the result of its evaluation.
- The `ast` directory: It contains the classes that implement the abstract syntax tree for SPL programs. The files `Stm.java` and `Exp.java` define the abstract classes `Stm` and `Exp`.
- The `visitor` directory:
  - `Visitor.java`: Defines the `Visitor` interface.
  - `PrinterVisitor.java`: Implements the `Visitor` interface. Traverses and prints the contents of a parsed program.
  - `Interpreter.java`: Implements the `Visitor` interface. Traverses and executes a SML program.
  - `VisitorException.java`: Exception thrown when there's an error during traversal.

Execute JavaCC, compile the generated java files and execute the program by typing the usual commands:

```
javacc sourcepl.jj          javac Main.java          java Main test.spl
```

where `test.spl` is a text file that conatins a sample SPL program. Modify the file and run the program several times to see how your changes affect the final result. Try to introduce runtime errors.

Adding the WHILE statement

The while statement, defined in your class notes, is missing from SPL's implementation. For example, we want to type statement of the form:

```
whie (x > 5) { y := y*2; x := x-1; }
```

Perform the following changes and additions:

- Modify the grammar and add a new nonterminal for the while statement. Open sourcepl.jj and **modify** the code for `stm()` so it looks like:

```
Stm stm() :
{ Token t; Exp e=null; Stm s; }
{
   t=<ID> ":=" e=exp() ";" { return new AssignStm(t.image,e); }
|  "print(" e=exp() ")" ";" { return new PrintStm(e); }
| s = ifstm() { return s; }
| s = whilestm() { return s; }
}
```

  and add the following non-terminal for whilestm:

```
Stm whilestm() :
{ Exp e=null; List<Stm> ls=null; }
{
 <WHILE> "(" e=exp() ")"  ls = block()
 { return new WhileStm(e,ls);  }
}
```

  Do not forget to add the token definition `< WHILE: "while" >` together with the other reserved words.

  Note that we are making a call to the constructor of the `WhileStm` class, which does not exist yet. That's our next step.

- Create a new file `WhileStm.java`, in directory `ast`, with the following contents:

```
package ast;
import visitor.*;
import java.util.*;

public class WhileStm extends Stm {
    public Exp exp;
    public List<Stm> body;
    public WhileStm(Exp e, List<Stm> b) { exp=e; body=b; }
    public int accept(Visitor v) throws VisitorException
    { return v.visit(this); }
}
```

- Add a declaration of the `visit` method for the new `WhileStm` class inside the `Visitor` interface:

```
public interface Visitor {
...
public int visit(WhileStm n) throws VisitorException;  // new line in Visitor.java

}
```

- And, finally, write the implementation of `visit` inside PrinterVisitor and Interpreter for the new class. Insert the following method definition in `PrinterVisitor.java`:

```
public int visit(WhileStm s) throws VisitorException {
   System.out.print(indent+"while (");
   s.exp.accept(this);
   System.out.println(")  {");
   incIndent();
   Iterator<Stm> iterator = s.body.iterator();
   while (iterator.hasNext()){
    (iterator.next()).accept(this);
   }
   decIndent();
   System.out.println(indent+"}");
   return 0;
}
```

and the follwing in `Interpreter.java`:

```
public int visit(WhileStm s) throws VisitorException {
  int v = s.exp.accept(this);
  checkExpType(s.exp,s.exp,valType,Type.BOOL);
  Iterator<Stm> iterator;
  while (v!=0) {
    iterator = s.body.iterator();
    while (iterator.hasNext()){
     iterator.next().accept(this);
    }
    v = s.exp.accept(this);
  }
  valType=Type.NOTYPE;
  return 0;
}
```

Compile again and test with a SPL program that contains a while statement.