Prof. Jingke Li (FAB120-06, lij@pdx.edu), Mon 8:15-10:05 @FAB 171, Lab: Wed 8:15-10:05 @FAB 88-10

Assignment 3: Parser with AST Generation

(Due 2/25/15 @ 11:59pm)

In this assignment, you are going to implement a complete parser in JavaCC. "Complete" means that the parser will not only check the syntax of the input program, but also generate an AST for the program. The source language to this parser is the miniJava's AST language itself. The parser reads an AST in its external form from the input source, and converts it to its internal form. It then prints it out to the output. This parser behaves like an "echo" program, except that it works on structured input.

This assignment carries a total of 75 points.

Preparation

Download the zip file "`hw3.zip`" from the D2L website. After unzipping, you should see an `hw3` directory with the following content:

- `hw3.pdf` — this document
- `AstGrammar.txt` — the AST language grammar
- `AstParser0.jj` — a starter version of the parser; it contains only the main routine
- `ast` — a directory containing the AST definition program file, `Ast.java`
- `tst` — a directory containing some test programs
- `Makefile` — for building the parser
- `run` — a script for running tests

The AST Language

The miniJava's AST representation is defined internally as a collection of Java classes (see the program file, `Ast.java`). However, it also has an external form, which is used by the miniJava compiler to dump an AST out for viewing.

This external form is itself a well-defined language. It has its own token and grammar definitions, and people can program directly in it. We call the external form the `AST` language. It's token and grammar definitions are given in the file `AstGrammar.txt`.

Note that the internal and external forms of the miniJava AST are one-to-one corresponding — each internal AST tree has a unique external form, and vise versa.

Your Task

Your task is to implement a parser for the AST language in JavaCC. Name your program `AstParser.jj`.

There are three parts to this implementation:

1. *Transfer the token definitions into JavaCC code.* You may look into last homework's starter program to get some general ideas.

2. *Transfer the grammar into JavaCC code.* The given grammar is in an LL(2) form — many `Exp` productions share a common prefix `"("`. You need to factor this prefix out to make the grammar LL(1). After this simple transformation, the conversion of the grammar into JavaCC code should be straightforward, as we have practiced this process in labs and in the last homework.

3. *Insert semantic actions into the parsing routines to construct internal AST nodes.* This is the new part. For each parsing routine, you need to decide which AST node to create — it should obvious since the external form and the internal form of the AST are one-to-one corresponding. You also need to define variables to receive values from recursive calls to other parsing routines.

   One footnote on this part: The external form has a `void` type, while the internal form does not. The internal form represents `void` simply by leaving a type field empty (*i.e.* set to `null`).

Running and Testing

You can use the `Makefile` to compile your parser:

```
linux> make
```

To run a batch of tests from the `/tst` directory, use the `run` script:

```
linux> ./run AstParser tst/*.ast.ref
```

Note that your program should read from the `.ref` copy of the AST program. The `run` script will place your parser's output in `*.ast` files and use `diff` to compare them one-by-one with the reference version in `.ast.ref` files. For each test program, the output AST should match exactly with the input AST.

Requirements and Grading

This assignment will be graded mostly on your program's correctness. We may use additional miniJava AST programs to test.

The minimum requirement for receiving a non-F grade on this assignment is that your parser program compiles with JavaCC and Java, and are free of any JavaCC and Java compiler warnings. Furthermore, it correctly reconstructs at least one AST.

What to Turn in

Submit a single file, `AstParser.jj`, through the "Dropbox" on the D2L class website.