Prof. Jingke Li (FAB120-06, lij@pdx.edu), Tue 10:00-11:50 @SRTC 155, Labs: Tue 14:00-15:50 & Wed 12:00-13:50 @FAB 88-10

Assignment 3: Parser with AST Generation

(Due 11/18/14 @ 11:59pm)

This assignment is a follow up to Assignment 2. In this assignment, you are going to insert semantic actions to parsing routines to generate an abstract syntax tree (AST) for the input program.

This assignment requires a correct LL1 or LL2 miniJava grammar. You are encouraged to use your own `mjParser0.jj` program as the base for this assignment. However, an LL2 grammar for miniJava will be made available to you as well, after all assignment 1 programs are turned in. In either case, add a comment line at the top of your parser program indicating which version of the grammar it is based on, *e.g.* "`This parser is built with my own grammar.`"

Name the parser program for this assignment `mjParser.jj`.

Preparation

Download the zip file "`hw3.zip`" from the D2L website. After unzipping, you should see an `assignment3` directory with the following items:

- `hw3.pdf` — this document

  `mjManual14.pdf` — the miniJava language manual

  `mjParser0.jj` — a starting version of miniJava parser; it contains only the lexer portion

  `ast` — a directory containing the AST definition program file, `Ast.java`

  `tst` — a directory containing sample miniJava programs and their corresponding AST dump outputs

  `Makefile` — for building the parser

  `runp` — a script for running tests

* `mjLL2Grammar.txt` — an LL(2) miniJava grammar (This file will be released separately, after all assignment 2 programs are turned in.)

Details

The following are suggested steps to complete this assignment.

1. *Complete Lab 6.* This week's lab serves as a warm-up for this assignment. It's important that you finish the lab first.
2. *Familiarize yourself with the AST class definitions, especially the constructors.* In your parser program, you'll insert semantic actions to create AST nodes. Knowing what your choices are and what exact forms you need to follow will be very useful. Look inside the program, `ast/Ast.java`, to see the AST nodes' details.
3. *Decide a return type for each parsing routine.* In Assignment 2's parser program, parsing routines do not return anything. In the new version for this assignment, all parsing routines participate in the construction of an AST; each routine contributes its share by returning an AST object. Here is a rough guidance for return types (Details depend on your exact grammar.):

Parsing Routine	Return Type
Program()	`Ast.Program`
ClassDecl()	`Ast.ClassDecl`
MethodDecl()	`Ast.MethodDecl`
Param()	`Ast.Param`
VarDecl()	`Ast.VarDecl`
Any Type routine	`Ast.Type`
Any Stmt routine	`Ast.Stmt`
Any Expr routine	`Ast.Exp`
Id()	`String`
IntLit()	`int`
BoolLit()	`boolean`

Note that three of the return types, `Ast.Type`, `Ast.Stmt`, and `Ast.Exp`, are abstract classes. Any routine returning these types need to return an object of a concrete subclass of them (*e.g.* `Ast.IntType`).

4. *Insert semantic actions into the parsing routines.* You may want to start with the simpler ones first. Most statement routines have direct corresponding AST nodes, so they are easy to handle. Binary and unary operation routines are also easy to handle. The more difficult routines are those involving `ExtId`, since they cover multiple miniJava constructs, *e.g.* method calls, assignments, array elements, and object fields. For these routines, you need to analyze each case carefully to decide what type of AST node to return. In some cases, a single routine may need to return different types of AST nodes for different sub cases.

5. *Compile and run tests.* You can use the `Makefile` to compile your parser:

   ```
   linux> make
   ```

   Your program should be free of JavaCC and Java compiler warnings, and it should still detect all miniJava syntax errors as your Assignment 2 program did.

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

   ```
   linux> ./runp mjParser tst/*.java
   ```

   It 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, its AST should be unique; therefore, your parser's output should match the reference copy exactly.

   You could also manually run a single test:

   ```
   linux> java mjParser tst/test01.java
   ```

Requirements and Grading

This assignment will be graded mostly on your parser's correctness. We may use additional miniJava programs to test. The minimum requirement is that your parser runs and generates at least one valid AST output.

What to Turn in

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