Prof. Jingke Li (FAB120-06, lij@pdx.edu), Tue&Thu 12:00-13:15 @SH 212, Lab: Fri 10:00-11:15/11:30-12:45 @EB 103

Homework Assignment 3: AST Generation

(Due 11/19/15 @ 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 AST for the input program. This assignment carries a total of 10 points.

This assignment requires a correct LL(1) or LL(2) miniJava grammar. You are encouraged to use your own `LLGrammar.jj` program from Assignment 2 as the base for this assignment. However, a reference LL(2) grammar for miniJava will be made available to you as well, after all Assignment 2 programs are turned in.

Name the parser program for this assignment `Parser.jj`. Add a comment line at the top indicating which version of the grammar you are using, *e.g.* "`This parser is based on my own grammar.`"

Preparation

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

- `hw3.pdf` — this document
- `ast` — a directory containing the AST definition program file, `Ast.java`
- `Parser0.jj` — a starter version; it contains the lexer portion and a couple of parsing routines
- `Makefile` — for building the parser
- `run` — a script for running tests
- `tst` — a directory containing sample miniJava programs and their corresponding AST dump outputs
- `RefGrammar.txt` — a reference LL(2) miniJava grammar (Will be released next week.)

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.
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`
XXXLit()	`Ast.XXXLit`
`Id()`	`Ast.Id` or `String`

Notes: (1) The three 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`). (2) For the `Id()` routine, you have a choice to either return an `Ast.Id` object or a `String`. The `ID` tokens in a miniJava program can represent either a string name (as in method name) or an ID (as in an expression).

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 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.

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

   ```
   linux> ./run Parser 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 Parser tst/test01.java
   ```

Minimum Requirement for Passing The minimum requirement for receiving a non-F grade on this assignment is that your parser compiles without error and generates at least one valid AST output.

What to Turn in

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