CS352s2014 Project 2: Parsing

January 31, 2014

1 Objective

The objective of this project is to write a parser for MiniJava in the JavaCC parser generator, both as a recognizer and targeting the AST supplied in the classes `edu.purdue.cs352.minijava.ast.*`.

2 Code

The project's code environment is available on the course web page. We are providing Java class files for a complete compiler, including the parser, but are not providing accompanying Java source. Through the course of these projects, it will be your job to rewrite the Java source for each component, creating a complete compiler of your own design. You are advised to keep a copy of the Java class files so that you can use them to generate correct output and test against your own.

As Project 1 is still ongoing, we will not provide the lexer component at this time. It will be provided once project 1 is graded.

2.1 Testing

The compiler provides a number of tools for testing each stage. They are available in the `bin` directory. Two are of value for testing the parser.

- `bin/mjparse`: Runs the syntactic recognizer (parser without AST generation).
  `bin/mjparse examples/Factorial.java` will output nothing, indicating success. When run with an invalid file, it will throw an exception.
- `bin/mjparse-ast`: Runs the parser with AST generation.
  `bin/mjparse-ast examples/Factorial.java` will output the AST for Factorial.java.

Your versions of these programs should have identical output to those provided.

3 JavaCC

JavaCC is a parser generator ("compiler compiler") for Java. Given a definition of a language's grammar, it generates a parser for that language. In this project, you will use JavaCC's full parsing capabilities.

A template of the parser is available in `edu/purdue/cs352/minijava/parser/Parser.jj-template`. Use it as a basis to create both `edu/purdue/cs352/minijava/parser/Parser.jj` and `edu/purdue/cs352/minijava/parser/ParserAST.jj`. This template includes a main function and small examples; you must fill in the remainder.

`.jj` files are compiled into `.java` files using `javacc`. A copy of `javacc` is included in the project code directory. To compile `Parser.jj`, use the following command:

```
javacc/bin/javacc -OUTPUT_DIRECTORY=edu/purdue/cs352/minijava/parser \
edu/purdue/cs352/minijava/parser/Parser.jj
```

Please note that the included `Makefile` will also rebuild the parser if `Parser.jj` or `ParserAST.jj` change.

The AST classes are provided with source, in `edu/purdue/cs352/minijava/ast`. You are expected to produce an identical AST to the provided parser. The output formatting for the AST is "S-expressions", which will be discussed in class.

The full documentation for JavaCC is available online at `https://javacc.java.net/doc/docindex.html`

4 Goals

For this assignment, you must implement two Java classes: `edu.purdue.cs352.minijava.parser.Parser` and `edu.purdue.cs352.minijava.parser.ParserAST`. Both are to be generated by JavaCC. The former requires only grammar statements. i.e., it will not produce an abstract syntax tree. The latter should have an identical grammar, but additionally produce an AST.

When either parser is run, it expects a filename as the first argument.

The grammar for our variation of MiniJava is provided in BNF format in `docs/minijava-bnf.txt`. Please note that our version of MiniJava is not identical to others. Use the grammar provided.

A number of examples are provided in the `examples` directory. These examples will help in testing, but are *not* the examples that will be used to grade your submission. In particular, none of the provided examples are invalid MiniJava. You should create your own examples to test for errors or corner cases in your own parser.

For this project, you *may not* place a `LOOKAHEAD` directive in the `STATIC` section of the `.jj` file. That is, you may not set a global lookahead greater than the default 1. Depending on how you write your parser, `LOOKAHEAD` directives may be necessary within the grammar itself, but these *must* be specified where required, and not globally. You may be deducted points for unnecessary uses of `LOOKAHEAD`, but you are not expected to rewrite the grammar to avoid necessary uses of `LOOKAHEAD`.

4.1 Example

Given the following file in `examples/test.java`:

```
class Simple {
    public static void main(String[] a) {
        System.out.println(42);
    }
}
```

ParserAST should output the following:
`(Program (Main (PrintStatement (int 42))))`

5 Submission

You will submit your project code via the `turnin` command. Please include the entire project code directory, including all the class files provided by us, the `bin` and `examples` directories, etc.

To turn in a project directory: `turnin -c cs352 -p proj2 <project directory>`

This project is due by Friday, February 28, 2014 at 12:30PM Purdue time. Late submissions will not be accepted. Please note that this project is not expected to take an entire month to complete; its assigned and due times are an anomaly of NP-hard scheduling.

6 Grading

Your code will be tested on both correct and incorrect source code examples. We will test your code on machines similar to those in the Linux labs used by this course's PSOs. Your grade will be based on correctness in the following criteria for both the generated and hand-written lexers:

- Correct parsing of main class and method.
- Correct parsing of class syntax.
- Correct parsing of method syntax.
- Correct parsing of statement syntax.
- Correct parsing of expression syntax.
- Correct parsing of variable declaration syntax.
- Correct AST generation of class syntax.
- Correct AST generation of method syntax.
- Correct AST generation of statement syntax.

- Correct AST generation of expression syntax.
- Correct AST generation of variable declaration syntax.
- Rejection of syntactically incorrect program source.
- Correct and unlimited handling of any files of any length.