210: Compiler Design

Fall 2015

Homework 2

Due dates
October 22th, 2015, 10:15 a.m.

25 Points

1 Introduction

The objective of this homework is to construct a lexer (lexical analyzer) and a parser (syntax analyzer) for the complete Javali programming language. Please refer to the Javali language specification for the syntax and the rules for identifiers, constants, etc. The updated Javali language specification is available on the course web page.

You may notice that the Javali syntax allows for things that will not be accepted by later compiler phases (at least not by the bare-bones compiler that you are asked to develop). For example, the syntax does not validate types. Further, the Javali syntax accepts certain incorrect constructs, such as `this[0]`, that will be rejected by the semantic analyzer later on. You may reject such incorrect programs in your parser even if the grammar would accept them, but you are not required to. Note that if you accept a different set of programs from the reference solution, it may be necessary to modify the `.ref` files to avoid unit test failures.

The lexer and parser are to be constructed using the *ANTLR* parser generator tool. From the Eclipse Marketplace, you may want to install the *ANTLR 4 IDE*, which provides useful Eclipse integration.

2 Task

To construct the lexer and parser for Javali, you need to complete the ANTLR4 file `cd/frontend/parser/Javali.g4` and the Java file `JavaliAstVisitor.java` in the same folder.

`Javali.g4` defines an ANTLR4 *parser grammar*. A parser grammar specifies both the lexical rules and parser rules for the Javali language. ANTLR generates the files `JavaliLexer.java` and `JavaliParser.java` from this grammar. It further generates the files `JavaliVisitor.java` and `JavaliBaseVisitor.java` that implement the visitor pattern over the intermediate AST generated from parsing a Javali program.

In the file `JavaliAstVisitor.java`, you are then asked to implement a visitor that visits the intermediate AST from the parser and generates the Javali

AST with the classes defined in `cd.ir.AST`.

Part a: lexer and parser rules

In the first part of this homework you are asked to construct the lexer and the parser for Javali. The input to the lexer is a series of characters representing the Javali program, and the output is a sequence of tokens. The input to the parser is the sequence of tokens produced by the lexer, and it will produce an intermediate AST. Both lexer and parser rules ought to be specified in the file `src/cd/frontend/parser/Javali.g4`.

Part b: Javali AST generation

Once you have the lexer and parser generated, you need to visit the produced intermediate AST and generate the corresponding Javali AST. You are asked to complete the visitor `JavaliAstVisitor.java` that will construct the Javali AST using the classes from `cd.ir.AST`.

Note: for the sake of simplicity, you should throw a `ParseFailure` in case you encounter a literal that is outside the range $[-2147483647, 2147483647]$. This restriction cannot be enforced by the grammar, but it is necessary to enforce in order to create the right Javali AST.

3 Framework

We have included the necessary `ANTLR` jar files in the framework. Also, there exist appropriate targets in the Ant `build.xml` file. The target `antlr-parser-source` generates the corresponding Java lexer, parser, and visitors from your parser grammar. Note that in Eclipse you should press `F5` to refresh the view of the directory after executing `antlr-parser-source`, so that Eclipse knows that new files have been generated, and/or that existing files have changed. Otherwise, the generated files will not be (re)compiled.

To complete this homework, you need to edit the files `Javali.g4` and `JavaliAstVisitor.java`, and be sure to make all tests pass. As always, we provide a few simple test programs, but you are required to add some of your own! When in doubt, create a test case and observe what the reference solution produces (`.parser.ref`).

4 Homework submission

Please ensure that your final submission is checked into Subversion by the deadline(s). In addition to the parser grammar file please include any test cases you created. To make it easier for us to locate these test cases, place them in a subdirectory of `javali_tests` named `HW2_team`. In addition, it is very helpful if you include a comment at the top of the test file indicating what you are trying to test. Finally, if there are any comments you should have on your solution, please provide a `README` file in the `HW2` directory of your team's Subversion directory.