210: Compiler Design

Spring 2018

Homework 2

Due: March 29th, 10 a.m.

30 Points

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 **up-to-date** Javali language specification is available on the course web page.

You may notice that the Javali syntax allows for semantically invalid programs 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.

To construct the lexer and parser for Javali, you need to extend the ANTLR file `Javali.g4` and the Java file `JavaliAstVisitor.java`. `Javali.g4` defines an ANTLR *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`, which implement the visitor pattern over the intermediate AST generated from parsing a Javali program.

Lexer and Parser Rules

In the first part of this homework, you 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, from which the parser will produce an intermediate AST. Both lexer and parser rules ought to be specified in the file `Javali.g4`.

Javali AST Generation

Once the lexer and parser are generated, the resulting intermediate AST needs to be converted to a corresponding Javali AST. In the file `JavaliAstVisitor.java`, you implement a visitor that traverses the intermediate AST and constructs the Javali AST using the classes from `cd.ir.Ast`.

To integrate your generated parser in the framework, we provide appropriate targets in the Ant `build.xml` file. The target `generate-parser` generates the lexer, parser, and visitors from your parser grammar. In Eclipse, do not forget to refresh the view of the directory (press `F5`) after executing `generate-parser`, so that new and changed files are recognised and recompiled.

As always, we provide a few simple test programs, but you are required to add some of your own. When in doubt about how you should handle a case, creating a test case and observing what the reference solution produces (`.parser.ref`) can often provide the answer.

Hand-in

Please ensure that your final submission is checked into Subversion by the deadline. 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.