

In this assignment you will review or learn how to build scanners and parsers using the tool SableCC. SableCC is also capable of generating an abstract syntax tree with corresponding visitor classes. The goal for this assignment is to write the token specifications, concrete syntax grammar, abstract syntax grammar, and the conversion from the concrete syntax to abstract syntax for the straight-line programming language specified on page 7 in the book. You will do this assignment individually.

1 Motivation

The MiniJava compiler that we will be working with for Projects 2 through 4 uses SableCC. The SableCC tool automates a significant amount of the work needed in the front-end of a compiler. This project will help you review lexical analysis and parsing concepts that are typically covered in an undergraduate compilers course. You will also learn about abstract syntax trees, visitors over abstract syntax trees, and how to visualize graphs with the dot tool that is part of Graphviz.

2 The Assignment

- Starting with the SampleParser provided in Project 0, create a parser for the straightline programming language specified on page 7 in the Appel book. You will need to create two .scc files.
 - straightline.scc
 - straightline-ast.scc

The first file should just have the concrete syntax grammar. The second file should have the same concrete syntax grammar, but also include an abstract syntax grammar and a translation from the concrete syntax to the abstract syntax. Keep in mind that abstract syntax grammars can be ambiguous, therefore you can just copy the grammar from page 7. Notice in the sample file that I append all of the non-terminals in the concrete syntax grammar with `cst_`. This helps keep the concrete syntax and the abstract syntax grammars organized.

- Use ParserDot.java to create dot files to visualize the difference between the concrete and abstract syntax. You will describe the illustrate and describe the difference in your report.
- Create at least two test input programs. Each concrete and abstract syntax node should occur in at least one test file.

3 Getting Started

1. Java, jar files, SableCC, and dot are all covered in Project 0. Information about converting from concrete syntax to abstract syntax in SableCC can be found at the following websites:
 - <http://nat.truemesh.com/archives/000531.html>
 - <http://sablecc.org/documentation/cst-to-ast.html>
2. Review lexing and parsing, especially read the section on ambiguous grammars in Chapter 3 of the Appel book. Keep in mind that SableCC generates LR parsers. The online Bergman Book for Java (<http://elvis.rowan.edu/~bergmann/books.html>) includes more examples, see Chapters 3 and 5.
3. Get a copy of the SampleParser discussed in the Project 0 setup notes. Notice that the SampleParser provides a significant head start.

There is a main in ParserTest.java and ParserDot.java. You will need ParserDot.java to generate dot files.

4 Your Report

The report is an essential part of your completed assignment. Use it to describe your solution, assumptions, difficulties, insights, and results. Organize and present your document as if it were the only basis for your assignment's grade. The format of your writeup is up to you, but it should minimally answer the following questions:

- How did you resolve the ambiguities in the straight-line programming language grammar? Provide references.
- Illustrate the difference between the concrete syntax of the straight-line programming language and the abstract syntax using dot.
- Describe the differences between the concrete and abstract syntax. What are the benefits of converting to an abstract syntax?
- What problems did you encounter while developing your program? If you knew someone who was just about to start work on this assignment, what advice would you give them?
- What, if any, outside sources did you use (e.g., articles, books, other students)? This is particularly important. It's OK to look at books and articles and speak with your professor and fellow students (although sharing code and working together is strictly forbidden), but as with any scientific document, you should always cite your references and collaborations. You can either cite collaborations in footnotes or in a separate Acknowledgment section.

- How did you test your program? Does your program work on the examples provided?
- What other examples have you tried your program on? For this assignment, you should give at least two non-trivial examples not in this document that show off your program's functionality.

There's no exact number of pages you should write, but if you've got between two and four then you're in the right ballpark.

5 Hints for doing the assignment

I can not emphasize the importance of your report enough. It is entirely possible that I will grade your whole project based solely on your report.

You should start this assignment early; it is due at the end of next week. I will be available during regular office hours and by email. I can look at your code and help point you in the right direction, but the amount of help I can give may be inversely proportional to the amount of time until the due date. By no means should you spend several hours trying to figure out a weird bug; consult me for help. When e-mailing me about the project, send a copy of the relevant section of your code (not as an attachment; send it as text appended to your message). Give a good description of the problem including information about the stack in a debugger.

6 What to turn in

Turn in a hard copy of the report and email a copy in pdf format to mstrout@cs.colostate.edu.

Create a jar file that includes all of the bytecode files, the source files, and your test cases. The jar file should also include a README that gives specific command-lines for running the jar file on your provided test cases.

7 Due date

This assignment is due Friday September 1st, **at 2:10pm**. Late assignments will be penalized 10% per day.