

# CS 335 Semester 2019–2020-II: Project Description

**Due** Your submission is due by Feb 16 2020 11:59 PM IST.

## General Policies

- Do not plagiarize or turn in solutions from other sources. We WILL check your submission(s) with plagiarism checkers. You will be PENALIZED if caught.
- Each group will get a total of FOUR FREE DAYS to help with your project submission deadlines. You can use them as and when you want throughout the duration of the project. For example, you can submit Milestone 1 two days late without penalty, and you will have two free days remaining to potentially utilize for the rest of the semester.

## Description

The goal of this project is to implement a compilation toolchain, where the input is in Java language and the output is x86 code.

## Milestone 1

In this milestone, you have to construct a scanner and a parser for the Java language. The output of your compiler should be the abstract syntax tree (AST) in a graphical form. The leaves of the AST should be labeled by the token names. Also include the lexemes within parentheses in the labels, for e.g., ID (myVarName).

The following is a *conceptual* outline of steps that you can follow for your implementation.

1. You can extend your Assignment 1 (Lexer) implementation. Do check that you have included all necessary lexical details and grammar rules for Java.
2. Use Java grammar<sup>1</sup> to implement the parser using any automated parser generator like Yacc. For example, you will need to convert the Java grammar to an input script that is understood by Yacc.
3. Integrate the lexer and parser interfaces so that the (1) parser can drive the lexer, and (2) the lexer returns tokens to parser.
4. Add actions to the grammar script so that the output of the parser is a DOT script representing the abstract syntax tree of the input program. Study the documentation of your parser generator (for e.g., Yacc or ANTLR) to understand how actions are specified.
5. The DOT script, when processed by the Graphviz tool called `dot`, should produce a postscript file with the diagram of the parse tree.
6. Submit TEN nontrivial Java programs that can be compiled with your implementation.
7. Handle errors in the input Java programs. Print suitable error messages for the errors encountered. Your compiler can terminate compilation and exit after the first error is encountered.

---

<sup>1</sup><https://docs.oracle.com/javase/specs/jls/se7/html/jls-18.html>

**Other details.** Your implementation must accept input file(s) and other options as command line parameters. Use options like `-input`, `-out`, `-help`, and `-verbose`.

**Graphviz and DOT.** You can use a free graph visualization tool called Graphviz<sup>2</sup> to visualize the AST. There are two components of Graphviz that you will have to use: (1) the language DOT for describing the AST, and (2) a tool called `dot`. For example, if `graph1.dot` is a DOT script describing the AST, then the following generates a postscript program containing the visualization of the AST.

```
$ dot -Tps graph1.dot -o graph1.ps
```

The `dot` tool takes care of the tree layout. You only need to specify the tree structure (i.e., nodes, labels, and edges) with the DOT language. You can read more about the DOT language from <http://www.graphviz.org/documentation/>.

### Submission.

- Submission will be through Canvas.
- Create a zip file named “`cs335_project_<roll>.zip`”. The zipped file should contain a folder `milestone1` with the following contents:
  - All your source files must be in `milestone1/src` directory. You are free to choose your implementation language.
  - Create a directory `milestone1/tests` for your test cases. Name the test files as “`test_<serial number>.java`”.
  - Use Makefile (or equivalent build tools like `ant`) for your implementation. You are free to use wrapper scripts to automate building *and* executing your compiler.
  - Your submission must include a `milestone1/doc` directory and a PDF file. The PDF file should describe any tools that you used, and should include compilation and execution instructions. Document all command line options.

You SHOULD USE L<sup>A</sup>T<sub>E</sub>X typesetting system for generating the PDF file.

### Evaluation.

- Your implementation should follow the prescribed steps so that the expected output format is respected.
- We will evaluate your implementations on a Unix-like system, for example, a recent Debian-based distribution.
- We will evaluate the implementations with our OWN inputs and test cases, so remember to test thoroughly.
- The TAs will meet with each group for evaluation. Make sure that your implementation builds and runs correctly. A typical evaluation session could be as follows.

```
$ cd <milestone1>/src
$ make
$ ./myASTGenerator ../tests/test5.java --out=test5.dot
```

---

<sup>2</sup><http://www.graphviz.org/>