CS335 (Compiler Design) : Project Milestone 1

Deepak Sangle
200860
Department of Computer Science
sangleds20@iitk.ac.in

Shreyasi Mandal
200956
Department of Computer Science
shreyansi20@iitk.ac.in

Vartika
201089
Department of Computer Science
vartikag20@iitk.ac.in

JAVA17 Parser

We created a JAVA 17 (combined) lexer and parser called **ASTGenerator**. ASTGenerator is a combined scanner and parser for the JAVA 17 language, which generates an abstract syntax tree (AST) for any given input java file.

It takes a java file as an input, extracts the tokens from it and generates a dot file containing the abstract syntax tree for the input file.

The grammar rules used for creating the parser is available at - https://docs.oracle.com/javase/specs/jls/se17/html/jls-19.html

Compilation and Execution Instructions

The compilation instructions are as follows -

```
make
./ASTGenerator --input=<input_file_name>.java --output=<
    output_file_name>.dot
dot -Tsvg <output_file_name>.dot -o <svg_file_name>.svg
```

Note: We are generating **.svg** file to show the visualization of the **Abstract Syntax Tree** (AST). This was because we had no application to support viewing of PostScript (**.ps**) files.

We support the following command-line options -

- input : Takes the input java file (<input_file_name.java>)
- output : Denotes the output dot file (<output_file_name.dot>)
- verbose : This command prints logs for the compilation as shown below -

```
...
Entering state 411
Reducing stack by rule 115 (line 240):
   $1 = nterm unary_expression (1.1-1.1: )
-> $$ = nterm multiplicative_expression (1.1-1.1: )
Stack now 0 25 63 30 32 73 87 107 130 174 257 458 217 288 229
    336 331 527 713 879 405 600
Entering state 410
Next token is token ')' (1.1-1.1: )
```

Preprint. Under review.

```
Reducing stack by rule 112 (line 235):
   $1 = nterm multiplicative_expression (1.1-1.1: )
-> $$ = nterm additive_expression (1.1-1.1: )
Stack now 0 25 63 30 32 73 87 107 130 174 257 458 217 288 229
    336 331 527 713 879 405 600
...
```

- help : This command generates a brief description of AST Generator.

Tools Used

The following tools are used for creating **AST Generator**.

- Flex - This is used for the lexer implementation of the java program
- Bison - This is used to generate the parser
- Graphviz - Graph visualisation tool to visualize the AST

Language Features Supported

- Interfaces
- Operators
- Control Flow - If-else, For, While, Do-While, Switch
- Type Casting
- Static polymorphism
- Dynamic polymorphism
- Generics
- Import statements
- Primitive data types
- Multidimensional arrays
- Basic Operators
- Methods and method calls
- Recursion
- Library function println()
- Classes and objects - Provides support for **nested/inner classes** with default and protected access modifiers
- Enums and Records
- Strings - Supports concatenation

Assumption

We have assumed the following assumptions for our Java compiler.

- Many different modifiers were present in the original Java grammar like package modifiers, constant modifiers, class modifiers, etc. We have clubbed all the modifiers into one common terminal "modifiers".
- We have ignored only the **Annotation** feature present in Java grammar.
- According to original Java grammar, some terminals like TypeIdentifier and UnqualifiedMethodIdentifier do not include keywords like yield, permits, etc. We have assumed that these keywords can be present in such terminals. This complication will be dealt with in later phases of compiler.

Example

```
public class test {
    int i = 4 + 5;
}
```

This code generates the following AST -

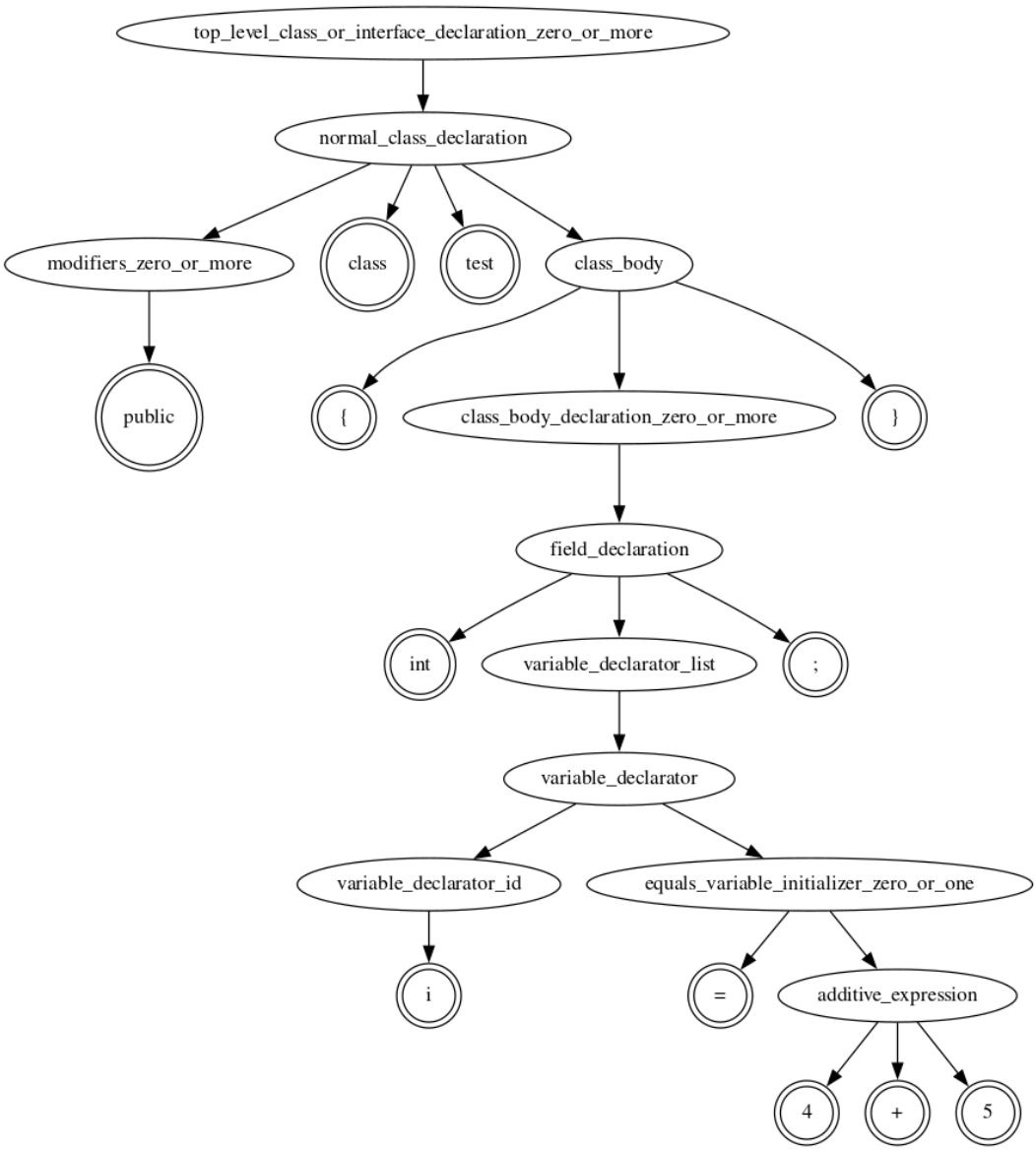

Figure 1: Abstract Syntax Tree

Conflicts

Even though our grammar has 113 shift/reduce and 5 reduce/reduce conflicts, none of them yield any problem in parsing the grammar mainly because of the following reasons

- During shift/reduce conflicts, bison always choose to shift. This is also the choice which our grammar need to do in almost every case. Thus, it yields no problem during parsing.
- During reduce/reduce conflicts, bison chooses to reduce a rule which is placed higher in the code structure. We have placed our rules in such a manner that the reduce/reduce conflicts always choose correct rule to reduce and thus also yields no problem during parsing.

Note

If the makefile does not compile properly, use the following command,

```
    g++ lex.yy.c parser.tab.c -o ASTGenerator
```

instead of

```
    g++ -std=c++11 lex.yy.c parser.tab.c -o ASTGenerator
```

in the Makefile