1 Project 2 A Pretty Printer

Due Date: 10/8 by 11:59p

Important Reminder: As per the course Academic Honesty Statement, cheating of any kind will minimally result in receiving an F letter grade for the entire course.

1.1 Aims of This Project

The aims of this project are as follows:

- To familiarize you with ANTLR.
- To expose you to some of the issues involved in pretty-printing.

1.2 Project Specification

Create a `submit/prj2-sol` folder in your gitlab project such that typing `make` within that directory builds a `build/libs/prj2-sol.jar` in that directory. Running that jar using a command like:

```
java -classpath "$HOME/cs572/lib/*:build/libs/*" \
     Jpp < TEST.decaf
```

will read a program written in the decaf subset of Java from standard input (`TEST.decaf` above) and print on standard error any syntax errors in that program. If there are no syntax errors, it should pretty-print the program on standard output as follows.

- No blank lines; an increasing indentation level is indicated by 2 additional spaces at the start of a line.
- Each class definition starts on a separate line at column 0.
- Class members should appear on a separate line at the next indentation level from their parent class.
- The starting brace `{` of a block or class-body must appear at the end of the line containing its parent. The terminating `}` of the block or class-body must appear on a line by itself at the same indent as its parent.
- Each statement must start on a separate line at the appropriate level.
- Nested statements must appear at the next indentation level from their parent statement.
- The `else` in a *if-then-else* must appear on a separate line at the same indentation as the matching `if`.
- No newlines other than those implied by the above rules.

- All tokens within a single line should be separated by a single space.
- Comments should be retained. Each comment should appear on a separate line. The start of the comment should be indented at the same indent as that of the following code. The body of the comment should be identical to what occurs in the source, including whitespace (even if that messes up the pretty-printing).
- If the source contains a comment within a declaration or statement, then the pretty-printed comment should occur before the declaration or statement.

The grammar must enforce all the syntactic restrictions listed in the decaf-syntax document. It is sufficient to merely use the error handling facilities provided by ANTLR.

1.3 Provided Files

The files directory provides the following:

Decaf Syntax Specification The specification document for decaf syntax.

Ant project A ant project set up for this project. Some notable files:

Makefile A wrapper which simply runs ant. It also contains a `run` target which allows running the `Jpp` program with input taken from standard input.

build.xml The build file for ant. It is currently set up to build a listener for ANTLR. If you want to change to build no listener or to build a visitor, please make changes in the `antlr` target in this file (as per comments).

README You will need to edit this file to provide your id and other information.

Decaf.g4 The grammar from the decaf syntactic specification converted to ANTLR syntax. Note that the changes are minimal.

CountingErrorListener.java A wrapper around ANTLR's error facilities which tracks the number of errors encountered.

Jpp.java A skeleton file which provides a partial `main()` function. You will need to add code to this file.

Decaf Test Files A folder containing some decaf test files used in the log shown below.

1.4 Log

A log of building and running the program is shown below:

```
$ cd prj2-sol
$ make
./gradlew build
:generateGrammarSource
:compileJavaNote: /home/umrigar/...
Note: Recompile with -Xlint:deprecation for details.

:processResources NO-SOURCE
:classes
:jar
:startScripts
:distTar
:distZip
:assemble
:generateTestGrammarSource NO-SOURCE
:compileTestJava NO-SOURCE
:processTestResources NO-SOURCE
:testClasses UP-TO-DATE
:test NO-SOURCE
:check UP-TO-DATE
:build

BUILD SUCCESSFUL in 2s
6 actionable tasks: 6 executed
$ make run <decaf/Test1.decaf
java -cp '/home/umrigar/cs572/lib/*:build/libs/*' Jpp
//Here is a comment
/* additional comment */
class T {
  public static void main ( ) {
    return 0 ;
  }
}
$ make run < decaf/Point.decaf
java -cp '/home/umrigar/cs572/lib/*:build/libs/*' Jpp
class Point {
  //x coordinate
  private int _x ;
  //y coordinate
  private int _y ;
  Point ( int x , int y ) {
    _x = x ;
    _y = y ;
```

```
  }
  int getX ( ) {
    return _x ;
  }
  int getY ( ) {
    return _y ;
  }
  public int mag_square ( ) {
    return _x * _x + _y * _y ;
  }
}
$ make run < decaf/Statements.decaf
java -cp '/home/umrigar/cs572/lib/*:build/libs/*' Jpp
/* some comment */
class Statements {
  int f ( int a , int a [ ] ) {
    while ( x ) {
      x = x + 1 ;
    }
    //another comment
    if ( x ) {
      a = b ;
    }
    /* else comment */
    else
      while ( a )
        /**/
        if ( x )
          f ( ) ;
  }
}
$
```

1.5 Hints

The hints provided below are purely advisory. You may choose to ignore some or all of them after reading them.

0. Review material on grammars, regular expressions and ANTLR.

1. Copy the *prj2-sol folder* over to your working directory. Your working environment should have ant and jdk8 executables available on its path and have the course directory available in your home directory (this is true on the remote.cs machines).

2. The provided Decaf.g4 file is not acceptable to ANTLR (if you try to compile the project as provided, you will get ANTLR errors). Add suitable lexical rules to the grammar and transform the grammar to make it acceptable to ANTLR. Send comments to a special channel as in the labels example in the

slides which uses ANTLR's lexical channels.

3. Transform the grammar so as to make it enforce the specified syntax restrictions. Test your grammar on syntactically invalid programs.

4. Decide on a strategy (listeners, visitors, grammar actions) for pretty-printing and implement that strategy. Note that the provided ant build file build.xml is currently set up to build a listener for ANTLR. If you want to change to build no listener or to build a visitor, please make changes in the `antlr` target in this file (as per comments).

The skeleton Jpp.java file has comments within `main()` giving an approximation of the code needed to get started for each of the three strategies.

Note that you will need to use ParserRuleContext; each specific grammar rule will have its own specific context which inherits from the ParserRuleContext (to distinguish among the different grammar rules for a non-terminal, you will need to label the grammar rules as in the examples covered in class). In this specific context, you will find accessors to get contexts for each of the grammar symbols of the rule RHS. You will find the definitions for the specific contexts within the generated parser file in `build/java`.

5. Add code to print out comments from the special channel where they have been squirrelled away.

6. Iterate until all the project requirements have been met.

1.6 Project Submission

Submit your project via gitlab:

- Submit your project files in a top-level `submit/prj2-sol` directory of your `cs572` project in gitlab.
- You should **not** submit executables, class files or object files.
- The `submit/prj2-sol` should contain the `Makefile` so that typing `make` in that directory builds the `prj2-sol.jar` in the `build/libs` folder.
- Make sure that you edit the `README` to provide your ID information by replacing the `XXXX` entries.

If your project is incomplete on the due date, please add a file called `.LATE` in your directory so that it will not be graded. Once the project has been completed late, please email the TA.

1.7 References

ANTLR web site.

Java 8 API.