Assignment: Parser
Due Saturday, August 25, 2018 at 11:59 PM

1 Introduction

In this assignment you will write a parser for COOL. The assignment makes use of the parser generator (called ANTLR). The output of your parser will be an abstract syntax tree (AST). You will construct this AST using semantic actions of the parser generator.

You certainly will need to refer to the syntactic structure of Cool, found in Figure 1 of The Cool Reference Manual (as well as other parts). The link to ANTLR documentation.

CS3423 registrants must work in a group for this assignment, where a group consists of two students; in special circumstances, three per team may be allowed. A google sheet will be posted where you can mention your group details. You are supposed to work with the same group member/members for all the assignments.

This assignment is divided into two parts: Phase A consisting of parser and Phase B consisting of AST construction.

Phase A was completed in compilers I and this assignment is for phase B.

2 Installation Instructions

You would have setup Antlr-4.7.1 with C++ as the code generation target. But, throughout this course you are supposed to work with **Antlr-4.5** and Java as the code generation target.

2.1 Instructions for Antlr setup:

To setup Antlr, visit `http://www.antlr.org/`, to download and setup **antlr-4.5-complete.jar**.

1. cd */usr/local/lib*
2. curl -O `http://www.antlr.org/download/antlr-4.5-complete.jar`
3. export CLASSPATH=".:/usr/local/lib/antlr-4.5-complete.jar:$CLASSPATH"
4. Verify your installation using *java -jar /usr/local/lib/antlr-4.5-complete.jar*

More details can be found here: `https://github.com/antlr/antlr4/blob/master/doc/getting-started.md`

Please make sure to install the mentioned version as the outputs of the exercise would differ with the new version. The code generation target is Java.

3 Files and Directories

To get started, create a directory where you want to do the assignment and extract **parser.tar.gz** in it. The important files are:

- src/grammar/CoolParser.g4

  This file contains a parser description for Cool. The grammar rules are already given (Phase A). We have also given the AST generation code for one of the rules. You do not need to modify this code to get a working solution, but you are welcome to if you like.

- src/java/cool/AST.java

  This file contains the classes for each type of node in the AST. You will need to study this to generate the AST in the action section of the parser rules. You do not need to modify this file to complete the assignment but can still do so. However, it is **not recommended** to change the class names, their member variables and functions names.

- src/test_cases/*.cl

  Make files that test a few features of the grammar. You should add legal tests to ensure every legal construction of the grammar and bad tests that exercise as many types of parsing errors. Explain your tests in these files and put any overall comments in the README file.

- README.md

  As usual, this file will contain the write-up for your assignment. Explain your design decisions, your test cases, and why you believe your program is correct and robust. It is part of the assignment to explain things in text, as well as to comment your code.

4 Testing the Parser

To test the parser, you will need a working scanner which is already provided.

You will run your parser using **parser**, a shell script (present in **src/grammar/**) that will parse the input file and print the AST if it is syntactically correct.

You can use the following command to test the parser:

```
./parser ⟨test-file⟩.cl
```

You should test this compiler on both good and bad inputs to see if everything is working. Remember, bugs in your parser may manifest themselves anywhere.

5 Parser Output

Your semantic actions should build an AST. The root (and only the root) of the AST should be of type program. For programs that parse successfully, the output of `parser` is a listing of the AST.

For programs that have errors, the output is the error messages of the parser. ANTLR has an inbuilt error reporting routine that prints error messages in a standard format; please do not print custom error messages.

Your parser need only work for programs contained in a single file; dont worry about compiling multiple files.

6 Error Handling

Do not be overly concerned about the the line numbers that appear in the error messages your parser generates. If your parser is working correctly, the line number will generally be the line where the error occurred. For erroneous constructs broken across multiple lines, the line number should be the first line of the construct.

For evaluation we will only check the first error that is printed so ensure that it matches with the output of the cool compiler.

7 What to Turn In

You need to submit your grammar file (src/grammar/CoolParser.g4), a readme file (README.md), and a bunch of test cases (src/test cases/*.cl) as a tar ball. To generate the tar ball go to the starting directory of the assignment directory tree and type the following:

```
./submit <roll-number>
```

This will generate a file by the name `Asn<your-roll-number>.tar.gz`. You need to submit this file in the Google Classroom page for this course.

- Make sure to submit and name the tar ball in the mentioned format. As automated scripts would be used for evaluation, your submission may not be evaluated otherwise.
- Make sure to use `Antlr-4.5` as the submissions would be graded comparing with the mentioned version.
- If you are in a group, only one should submit and the others can just "Turn in" in the classroom page.
- The last submission you do will be the one graded.
- The burden of convincing us that you understand the material is on you. Obtuse code, output, and write-ups will have a negative effect on your grade.
- The late penalty of 10% penalty per day for a maximum of one week will be applied.
- Take time to clearly (and concisely!) explain your results.

8 Remarks

The Cool `let` construct allows multiple definitions of variables. You are to generate the AST in the form of nested lets. For example:

```
let a:Int, b:Int in expr
```

The AST generated should follow:

```
let a:Int in let b:Int in expr.
```

Note that, the above two are semantically equivalent in COOL.