Troy Cope

Concepts of Programming Languages

CS4308, Section 03

Deliverable 2, Parser

March 27th, 2022

Sharon Perry

Project Deliverable is 100% completed and working as designed.

As with part one, the Scanner, Java was the language of choice for the second deliverable, the Parser. My implementation of a parser for the language Julia was structured after a tree using multiple classes to invoke other classes. This then needed to have children attached to each segment, which is shown slightly, but needs to have the interpreter set up to fully test. Therefore, the node-like tree structure is shown through print statements and console logs rather than a full implementation. I plan to pull this implementation out as I work on the interpreter anyway, that way the files look much neater.

The full implementation of the parser works as follows:

1. Build an ArrayList (I call it allTokens) with the output from the Scanner
2. Generation of the Parser object which functions as the function call giving it allTokens
   1. Purges the end statement and the unimportant function name from allTokens
3. Build blocks based on whether allTokens has arguments in it still
4. Set those blocks to become Assignments
   1. Determine what type of Statement the Assignment is
   2. This is done via the buildChild(ArrayList<String> allTokens) function
5. Follow the case statement to find the Assignment type, then use the formal definition to be implemented as described in the parse tree
6. Finally break those down into Arithmetic Statements and Boolean Statements
   1. Finish the logic with arguments being stored in these, this is what the compiler will run or pass up into the Assignment types

Using this basic outline, there is high flexibility within the program because it is in the process of implementing two or three different solutions. The side effect of this, however, is that the code is very messy and cluttered, having some semi-complete solutions (some of which can be seen through the use of passing up information as well as the use of children similar to a node). Of course, the program does work as shown in the input/output section, but I wanted it to be noted that this is nowhere near a completed program for the Parser alone. I will finish the implementation of the Parser when the final build of the Interpreter is in place simply due to a conflict of what will be the most interesting for me.

The classes themselves will be further refined as well, possibly through using interfaces and abstract classes to give more structure. I was considering using threading as well but found that this implementation had no a problem with speed.

There will be section in my program that deal with String manipulation, and that is because I chose to use the formatted outputs from the Scanner. These outputs followed the form: “word -> token” and therefore were split around the arrow as needed. The program also uses plenty of counters to keep track of the program. There are also lingering System.out.println() functions, once again for testing because this is a complete project for the Parser, but incomplete project for the whole Interpreter/Parser/Scanner project implementation. For that reason, I decided it would be a good idea to show my thought process as I do not want it to seem like my project is perfected. Thus, as designed it works perfectly, but in the future, it will be much more refined.

Input and Output Capture

Test1, input:

Graphical user interface, text

Description automatically generated

Test 1, output:

**Program Struct -> Block -> Statement -> Assign -> Arithmetic -> Literal Integer -> Block -> Statement -> Print -> Arithmetic -> Id ->**



Test2, input:

Text

Description automatically generated

Test2, output:

**Program Struct -> Block -> Statement -> Assign -> Arithmetic -> Literal Integer -> Block -> Statement -> While -> Boolean -> Arithmetic -> Id -> Arithmetic -> Literal Integer -> Block -> Statement -> Assign -> Arithmetic -> Arithmetic -> Id -> Arithmetic -> Literal Integer -> Block -> Statement -> Print -> Arithmetic -> Id ->**







Test3, input:

Text

Description automatically generated

Test3, output:

**Program Struct -> Block -> Statement -> Assign -> Arithmetic -> Literal Integer -> Block -> Statement -> If -> Boolean -> Arithmetic -> Id -> Arithmetic -> Literal Integer -> Block -> Statement -> Print -> Arithmetic -> Literal Integer -> Block -> Statement -> Print -> Arithmetic -> Literal Integer ->**





