Linnaeus University
Code Transformation and Interpretation, 4DV505/4DT902
Dr Jonas Lundberg
jonas.lundberg@lnu.se

ANTLR Exercise

First of all: The exercise described here is for you to try out ANTLR. It is not to be submitted and will therefore not be graded. You can use it as a playground to try out different approaches to parse tree construction etc., before you get started with the first assignment. In this exercise you should write a new parser `SimpleParser` to be able to parse programs like

```
program Compute {
    // Declarations
     int n = 100;
     int count = 2;
     int fib;
     int prev = 1;
     int prevPrev = 1;

    // Compute fibonacci(n)
     if (n > 2) {
          while (count < n) {
               fib = prev + prevPrev;
               prevPrev = prev;
               prev = fib;
               count = count + 1;
          }
     } else {
          fib = 1;
     }

     if (fib > 1) {
          print(fib);
     }
}
```

We suggest that you use Expressions.g4 as a base and then perform the following steps.

1. Program declaration and print(): Add features to the grammar saying that each program must start with keyword program followed by a name for the program and that the code should be enclosed in curly brackets. Add also support for a predefined print statement named print(...). See the example above.

2. Comments: Add comments to the lexer rules. If you're unsure on how to do this, just steal (yes, this one time it's OK to copy!) from a complete Java grammar [1]

[1] https://github.com/antlr/grammars-v4/blob/master/java/java8/Java8Lexer.g4

3. Variable Declaration: Change the grammar so that all variables can be explicitly declared (int) before the first statement begins. They may be initialized; initialization may look like this:

```
int first = 1;
int second = first + 2;
int third;
```

4. Boolean Variables and Expressions: Introduce a new type boolean, the boolean literals true and false, and the operations $<, >$ and `&&` (logical AND). Make sure that correct operator priorities are explicit in the resulting syntax tree. The following type of program should be accepted:

```
 // Declarations
int a = 1;
boolean v1;
boolean v2 = true;

// Statements
v1 = 8 < a + 1; // RHS Priority: 8 < (a+1)
v2 = 4 < a && 5 < a; // RHS Priority: (4<a) && (5<a)
```

5. While- and If-Statements: Add support for if and while statements that look like this:

```
while ( ... ) {
    ...
}
```

```
if ( ... ) {
    ...
}
```

```
if ( ... ) {
    ...
 }
 else {
    ....
 }
```

That is, the else part in if-statements should be optional. It should be possible to have nested statements (statements within statements).

You are done! You should now be able to parse the entire program presented earlier (page 3). Once again: **this exercise is for your training only; this is not Assignment 1!**