02332 Compiler Construction
Assignment 1, Syntax and Parsing

Hand-out: 17. September 2019
Due: 11. October 2019 23:55
Hand-in: on DTU Inside Course Page
Group hand-in is allowed, but maximum 4 people per group
To hand in:

- All relevant source files (grammars and java)
- Your test examples.
- A small (1-2 page) report in PDF format that documents what you did for each task (including answers to questions of the task), possibly with code snippets. Clearly state the name of **all group members** (names and student number) with the task on the title page of the report.

Week 3: Parser Generators

The initial task – if you have not yet done it already – is to download and install ANTLR for Java. When working in groups, we expect every group member to have a running installation on their own machine.

- You can download and find brief installation instructions for Windows, Linux and Mac on `http://www.antlr.org`. There are more detailed instructions on `https://github.com/antlr/antlr4/blob/master/doc/getting-started.md`.
- For effective group work we also highly recommend a revision system like `subversion` or `git`. For instance on `https://repos.gbar.dtu.dk` you can create your own repositories.
- Optional: You are welcome to use IDEs like Eclipse (on your own administration). See `http://www.antlr.org/tools.html` for ANTLR plugins for Eclipse.
- Optional: There is also a graphical visualization/editor for ANTLR that many people like: `http://tunnelvisionlabs.com/products/demo/antlrworks`
- On DTU inside, we provide a standard example discussed in the lecture: a simple calculator. It consists of the following files:
  - `Makefile` for people who like to work with editor and command line. It contains all necessary compilation commands (you may need to adapt the path of ANTLR in that Makefile): you should be able to compile everything with `make` and `make test` to run a sample input to the calculator. Whenever you change anything, another `make test` causes the re-compilation of only those files that are affected by your changes.
  - `simpleCalc.g4` the ANTLR grammar. It must be run with ANTLR with the option `-visitor` and this generates the files:
    - `simpleCalcLexer.java` – the lexer for the calculator,
    - `simpleCalcParser.java` – the parser for the calculator,
    - `simpleCalcVisitor.java` – this file defines an interface `simpleCalcVisitor` which is the key to actually do something useful with a parse tree that the parser returns.
  - `main.java` The actual main program that reads an input file given as command line argument, feeds it to the lexer and then to the parser. Finally, it implements the `simpleCalcVisitor` interface in a class called `Interpreter`. This visits the parse tree to recursively compute a result.
  - `simpleCalc_input.txt`. A sample input to the program.

Please try to compile and run the calculator, and play a bit with it, trying out different inputs. For all the above installation, there is nothing to hand in.

Task 1: The given calculator supports only addition and multiplication. Please extend it with subtraction and division by modifying the given example. You MUST NOT modify any of the ANTLR-generated files, but only the grammar and the main java file.

Test the calculator on several inputs – does it always compute the desired result, especially when we mix several operators without parentheses?

Task 2: This task is about designing the grammar for a language first on "paper", i.e., without trying to get it into ANTLR (we do that in another task below). Also we do not worry about operator precedence and associativity at this point.

The task is to extend the calculator to a small programming language with

- assignment like `x=2*x+17;`
- sequences of commands like `{x=x+1; z=2x;}`
- conditional branching like `if (x==0 || y>z) then z=1 else z=0;`
  where conditions can contain comparison of expressions (equal, smaller,...), and Boolean connectives (and, or, not)
- and loops like `while (x>0) x=x-1;`

You are welcome to design the concrete syntax to your personal taste, as long as all above concepts are supported (see `https://en.wikipedia.org/wiki/LOLCODE` for a rather bizarre example).

Specify this as a context-free grammar, and give for each construct an example of correctly used syntax.

Week 4: Lab Week

On our course's file sharing, we upload this week an extended version of the simple calculator that allows for a sequence of assignments of the form $x = e$; where x is a variable and e is an (arithmetic) expression. Each expression may use variables that the previous assignments have introduced. After this sequence of assignments follows a single expression e and the interpreter outputs the value of e as a result.

Task 3: Extend this calculator to your programming language from task 2, i.e., replace the sequence of assignments with your more complex programming language. Please keep the final expression e that is evaluated and returned as a result. This should be directly implemented in the ANTLR grammar. Check that all examples of **Task 2** can indeed be accepted by the parser.

Note that in order to compile the `main.java` file, you will have to extend the class `Interpreter` with a visitor method for each new syntax element you have introduced–see the interface definition in `simpleCalcVisitor.java` that is generated from your grammar. (The type parameter `<T>` becomes `double`.) For starters you may leave the implementation of all new visitor methods for next week's task; since Java requires these methods to return something, take some double value of your choice.

Check in particular that the operators in arithmetic expressions and Boolean conditions have the right precedence:

- multiplication and division bind stronger than addition and subtraction
- "not" binds stronger than "and", "and" binds stronger than "or".
- all the binary operators associate to the left.

For checking this, you should develop small test cases and document them in your report.

Week 5: Abstract Syntax

Task 4: Complete the implementation of the visitor methods for your language from the previous tasks.