Compiler Construction

Homework Series 1

Submission deadline: 24 May, 23:59 (Blackboard)

The homework exercises are to be done individually. You may discuss and exchange ideas freely, but not share code — except insofar this was developed during the lab sessions — or copy solutions.
In packaging and submitting your solutions, please stick to the following guidelines (which are meant to lighten the task of assessment). Failure to do so may reduce your grade.

- Submit your solution on BLACKBOARD (also if you are late). You may submit multiple times, but *only the last submission will be rated* (and its submission time will be used to deduct any points if you are late!)
- Put all your files into a single archive (zipped, tarred or however combined). Loose files will not be graded.
- Combine your textual answers into *one* pdf file, clearly recognizable from its file name, and with your name and student number included. No other formats! If you must use Word, convert to pdf! If you are up for LaTeX, submit only the pdf even so! If you want to include drawings or hand-written text, scan them in and combine them into a single pdf!
- Put all your classes, grammars and other electronic resources into a JAVA package `pp.s1234567.q1_X`, *replacing the `1234567` by your own student number and the `X` by the question number*. You may reuse classes from the lab sessions, provided they are completely unchanged; otherwise, copy them into your package. Put a list of your electronic resources into the pdf file of the previous item.

In case of late submission, the grading rules described on BLACKBOARD apply.

Question 1 *(10 points)* XTEXT is a parser generator in ECLIPSE that also includes many other goodies. In this exercise you are asked to compare XTEXT with ANTLR. Do so by carrying out one or more of the following steps, for a total of at most 10 points:

1. *(Max. 10 points)* Write a grammar for a small language of your own choice in both XTEXT and ANTLR, and compare the two using your own criteria;
2. *(Max. 5 points)* List some advantages of XTEXT above ANTLR;
3. *(Max. 5 points)* List some advantages of ANTLR above XTEXT.

Grading: Your grade for this question will be the sum of the grades for the subquestions, with a maximum of 10 points. □

Question 2 *(25 points)* The method `PrintWriter#printf` (most commonly used for `System.out.printf`) takes as its first parameter a string with *format specifiers*. The syntax of format specifiers is documented in (for instance) `https://docs.oracle.com/javase/8/docs/api/java/util/Formatter.html`. An example usage is:

```
System.out.printf("Price␣of␣%2$s:␣%1$,+8.2f%n", 25., "bananas");
```

which yields as output

```
Price of bananas:   +25.00
```

1. *(10 points)* Give an ANTLR lexer grammar defining legal format specifiers of the categories *general*, *character*, *integral* and *floating point*. (In other words, each of these four categories should be specified as a token type.) In doing so, you may ignore the exceptions 1–5 listed below the "Flags" table. Use **`fragment`**s to keep your grammar readable.

   Give a (comprehensive) JUNIT test showing that your solution recognizes and rejects (sequences of) tokens correctly.
2. *(10 points)* Give (in a drawing) a *single* DFA that combines these four token types; distinguish between the token types by tagging the final state(s) with the type that was just recognized.
3. *(5 points)* In the Java 7 version of the documentation (`https://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html`), there is a small difference when it comes to the explanation of the optional *width* fragment: in the Java 8 version, this is required to be a *positive* decimal integer whereas in the Java 7 version it was required to be a *non-negative* decimal integer. What error in the Java 7 syntax has been repaired by this change? Explain your answer.

□

Question 3 *(35 points)* Consider the following expression grammar G_0, with start symbol C, which specifies the first line of a **`class`** declaration:

$$
\begin{array}{lrll}
1 & C & \rightarrow & P \text{ `class'} \; ID \; X \; Y \\
2 & P & \rightarrow & \text{`public'} \\
3 & & | & \varepsilon \\
4 & X & \rightarrow & \text{`extends'} \; ID \\
5 & & | & \varepsilon \\
6 & Y & \rightarrow & \text{`implements'} \; I \; ID \\
7 & & | & \varepsilon \\
8 & I & \rightarrow & ID \text{ `,'} \; I \\
9 & & | & \varepsilon
\end{array}
$$

Here, *ID* is a token type described by the regular expression `[a-z]+`.

1. *(5 points)* Is G_0 ambiguous or not? If it is, give a sentence that can give rise to more than one parse tree; if it is not, argue why ambiguity can never arise.
2. *(10 points)* Compute the FIRST, FOLLOWand FIRST^+-tables of G_0, and using those, show that the grammar does not satisfy the LL(1)-criterion.
3. *(5 points)* Give a sentence that can be generated by G_0 for which the problem with LL(1)-parsing occurs, give the corresponding parse tree, and explain in words what the problem is.
4. *(5 points)* Construct an ANTLR grammar that precisely corresponds to G_0. Show that the grammar produces the correct parse tree for your problematic sentence of the subquestion 3.
5. *(5 points)* Refactor G_0 to an equivalent grammar G_1 (i.e., which accepts the same sentences, though not necessarily generating the same parse trees) that is LL(1).
6. *(5 points)* Show the parse tree of the sentence of subquestion 3 under G_1. Do you have any preference for one of the parse trees (under G_0 or G_1)? Explain why, or why not.

□

Question 4 *(15 points)* Consider the following ANTLR grammar `Tree.g4` (also given in the predefined source files), which specifies a format for binary trees consisting of natural numbers.

```
grammar Tree;
top : node ;

node
  : NUM
  | LPAR node LEFT NUM  RPAR
  | LPAR NUM RIGHT node RPAR
  | LPAR node LEFT NUM RIGHT node RPAR
  ;

LPAR  : '(';
RPAR  : ')';
LEFT  : '<';
RIGHT : '>';

NUM : [0-9]+;
```

A tree is said to be *ordered* if all number occurring to the left of a node are smaller than the number in that node itself, and all number to the right of a node are larger than that number. An example of an ordered tree is `((1 < 3) < 5 > ((6 < 8) < 10 > 15))` and an example of an unordered tree is `((1 < 5) < 3 > 2)` (where `5` is to the left of `3` and `2` is to the right of `3`).

1. *(8 points*) Formulate a combination of synthesised and inherited attribute rules that determine whether a `top` or `node` instance is ordered. (*Hint:* Define lower and upper bounds as inherited attributes and the orderedness of subtrees as a synthesised attribute.) Give the rules in the style of Chapter 4 of EC (e.g., as in Fig. 4.11).
2. (*7 points*) Write a *TreeListener* that implements the attribute rules, using `ParseTreeProperty` maps to store the attribute values. Also write a JUNIT test to demonstrate the correctness of your solution.

□

Question 5 *(15 points)* The grammar below (also given in the predefined source files) defines the syntax of a domain-specific language for family trees.

```
grammar Birth;
/** Branch of a family tree */
branch : ( enter | beget )+ ;
/** Figures may enter with unknown parenthood. */
enter : ENTER sex ID (COMMA ID)* ;
/** Figures may be born from mother and father. */
beget : ID COMMA ID BEGET sex ID (COMMA ID)* | LEFT branch RIGHT ;
/** Persons are always male or female. */
sex : MALE | FEMALE ;

COMMA  :  ',';
LEFT   :  '[';
RIGHT  :  ']';

BEGET  :  'beget';
ENTER  :  'enter';
FEMALE :  'female';
MALE   :  'male';

ID : [A-Z][a-z]*;
```

There are two types of statements in this language, illustrated in the following example program:

```
enter male Adam
enter female Eve
[ Eve, Adam beget male Kain, Abel ]
```

With **enter**, one or more new (male or female) persons are introduced without specification of parenthood, whereas with **beget**, a couple consisting of female and male (in that order) is declared to have had one or more (male or female) children.

There are associated typing and scoping rules:

- A person can only beget a child if he or she has been introduced through **enter** or **beget**;
- The sex of a person is determined by the **male** or **female** prefix at introduction;
- The brackets `[` and `]` are used for scoping. A person cannot be redefined in the same or a deeper scope, but can be redefined in an independent scope, also with a different sex.
- A **beget** statement must be preceded by two known persons, separated by commas, of which the first has to be female and the second male.

For instance, the following program contains several errors:

```
1 enter male Peter
2 Betty, Peter beget female Andrea
3 Peter, Peter beget male Selfie
4 [ enter female Betty
5   Betty, Peter beget male Andrea
6 ]
7 enter female Betty
8 Betty, Peter beget Betty
```

- Line 2: `Betty` has not been introduced;
- Line 3: The first person in front of the **beget** is male;
- Line 5: `Andrea` has already been introduced;
- Line 8: `Betty` has already been introduced (in the previous line).

Write a class `BirthChecker`, extending `BirthBaseListener`, that detects errors in family tree specifications. For this purpose, you will have to extend the `SymbolTable` of block 3 so that it does not just remember the fact that a person was declared, but also its sex. Provide at least one more correct and one more erroneous program to test the correctness of your solution. □