Introduction to Language Theory and Compilation
Exercises
Semantic Analysis

Ex. 1. Consider the following grammar:

$$
\begin{array}{llll}
(1) & Y & \rightarrow & Y+Y \\
(2) & Y & \rightarrow & Y-Y \\
(3) & Y & \rightarrow & Y*Y \\
(4) & Y & \rightarrow & Y/Y \\
(5) & Y & \rightarrow & (Y) \\
(6) & Y & \rightarrow & \texttt{valInt} \\
(7) & Y & \rightarrow & \texttt{valFloat} \\
(8) & Y & \rightarrow & \texttt{valStr}
\end{array}
$$

We assume that `valInt`, `valFloat`, `valStr` are provided with a value from the scanner. This is done by using the presence of a `.` in numbers for `valFloat`, and the fact that a `valStr` is enclosed in quotation marks.

1. Transform the grammar to:
   (a) Introduce priorities (classical arithmetic priorities).
   (b) Eliminate left-recursion.
   (c) Left-factorize the grammar
2. Check that the obtained grammar is LL(1) and build the LL(1) action table.
3. Execute the run of the scanner on $3+4+\text{”.”}+(1+2*3.0)$.
4. Execute the run on the LL(1) parser on the result of the previous question. Build the parse tree at the same time.
5. Annotate the parse tree with the type of each node.
6. Based on that, what does the following code produce?

```java
public class Exercise3{
    public static void main(String[] args){
        System.out.println(3+4+"."+(1+2*3.0));
    }
}
```

Ex. 2. Assuming that the Java grammar contains rules of Figure 1, explain why these methods are (not) semantically correct.

```java
public class Exercise1{
   public static void exerciseA(String[] args){
      if(!args) throw new IllegalArgumentException();
      System.out.println("All right!");
   }
   public static void exerciseB(String[] args){
      if(new Boolean(true)) throw new IllegalArgumentException();
      System.out.println("All right!");
   }
}
```

<IF>	$\rightarrow$	if (<EXPRESSION>) {<IF_BODY>}
<EXPRESSION>	$\rightarrow$	<BOOLEAN_EXPRESSION>
<EXPRESSION>	$\rightarrow$	<NUMERIC_EXPRESSION>
<EXPRESSION>	$\rightarrow$	<CHAR_EXPRESSION>
<EXPRESSION>	$\rightarrow$	<OBJECT_EXPRESSION>

Figure 1: Some speculative rules about conditions of the Java grammar

<EXP>	$\rightarrow$	<EXP> + <EXP>
<EXP>	$\rightarrow$	<EXP> * <EXP>
<EXP>	$\rightarrow$	<EXP> - <EXP>
<EXP>	$\rightarrow$	<EXP> / <EXP>
<EXP>	$\rightarrow$	*(int)* <EXP>
<EXP>	$\rightarrow$	(<EXP>)
<EXP>	$\rightarrow$	ID
<EXP>	$\rightarrow$	Bool
<EXP>	$\rightarrow$	Real
<EXP>	$\rightarrow$	Int
<EXP>	$\rightarrow$	String

Figure 2: Some speculative rules about expressions of the Java grammar

Ex. 3. Assuming that the Java grammar contains rules of Figure 2, identify the scope of all variables

- Give the table of symbols (ToS)
- Give the parse tree of each numerical expression
- Annotate the parse trees with changes of the table of symbols

Report any semantic error.

```java
public class Exercise3{
   public static final double PI = 3.141592653589793;
   public static double diameter;
   public static void main(String[] args){
      double perimeter = 50;
      diameter = perimeter / PI;
      final int diameter = Exercise3.diameter;  //15.915494309189533
      System.out.println(((int)(diameter*100))/100.0);
   }
}
```