Individual Exercises-Lecture 4

1. Browse the language specifications listed above
   - Java: The Java Language Specification, Third Edition - TOC
     http://docs.oracle.com/javase/specs/
   - C#:
     http://www.ecma-international.org/publications/standards/Ecma-334.htm
   - JavaScript (ECMAScript):
     http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf
   - Standard ML:
     http://www.lfcs.inf.ed.ac.uk/reports/88/ECS-LFCS-88-62/ECS-LFCS-88-62.pdf

2. (Optional) Do Sebesta Review questions 1, 2, 3, 4, 6, 7, 25 on pages 180-181

   1. Define lexeme and token.

   A lexeme is a sequence of characters in the source program that matches the pattern for a token and is identified by the lexical analyzer as an instance of that token.

   2. How are programming languages formally defined?

   Programming languages are formally defined using a context free grammar in combination with structural and operational semantics.

   3. In which form is the programming language syntax commonly described?

   The syntax of programming languages is described using a context free grammar either in Backus-Naur-Form (BNF) or some variant of Extended-Backus-Naur-Form (EBNF).

   4. What is a metalanguage? A language used to describe another language, for example Backus-Naur-Form.

   5. What is a derivation in the context of grammar? A derivation is the process of replacing a non-terminal token with one of that nonterminal's definitions.

   6. What is an ambiguous grammar? A grammar is ambiguous, if there exists an input from which two (or more) different parse trees can be derived.

   7. What is a left-recursive grammar? Non-terminals appear recursively on the left side (Makes LL(1) parsers impossible)

   25. What is the problem with using a software pure interpreter for operational semantics?

   The detailed characteristics of the particular computer would make actions difficult to understand. Such a semantic definition would be machine-dependent.

3. Do Sebesta Problem Set 2a, 2b on page 181 – check your result against the definition

   2a: Write a EBNF for a Java class definition header statement

   http://docs.oracle.com/javase/specs/jls/se7/html/jls-8.html

```
ClassDeclaration :==  NormalClassDeclaration | EnumDeclaration
NormalClassDeclaration:== ClassModifierspt class Identifier
TypeParametersopt Superopt Interfacesopt ClassBody
```

   2b: Write a EBNF for a Java method call statement

   http://docs.oracle.com/javase/specs/jls/se7/html/jls-15.html#jls-15.12

```
MethodInvocation :== MethodName ( ArgumentListopt )
    | Primary . NonWildTypeArgumentsopt Identifier ( ArgumentListopt )
    | super . NonWildTypeArgumentsopt Identifier ( ArgumentListopt )
    | ClassName . super . NonWildTypeArgumentsopt Identifier (
ArgumentListopt )
    | TypeName . NonWildTypeArguments Identifier ( ArgumentListopt )
```

Primary = on objects and “this”

Super = super class method

ClassName = static metoder

TypeName = interface

4. Do Sebesta Problem Set 4 on page 181

   4. *Rewrite the BNF of Example 3.4 to add the ++ and -- unary operators of Java*

Old:

```
<assign> = <id> "=" <expr>
<id> = A | B | C
<expr> = <expr> "+" <term>
       | <term>
<term> = <term> "*" <factor>
       | <factor>
<factor> = "("<expr>")"
       | <id>
```

New: Added ++ and -- unary operators

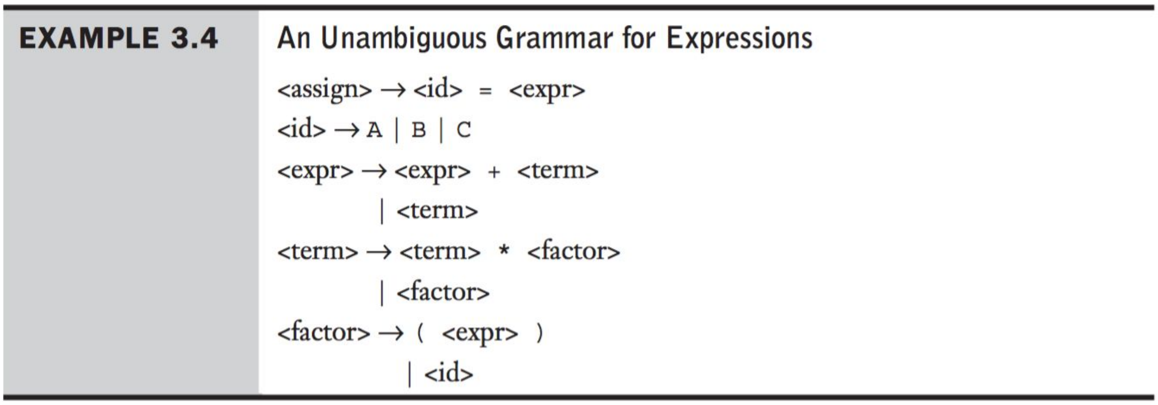
```
<assign> = <id> "=" <expr>
<id> = A | B | C
<expr> = <expr> "+" <term>
       | <term>
<term> = <term> "*" <factor>
       | <factor>
<factor> = "("<expr>")"
       | <id>
       | <id> ++
       | <id> --
       | ++ <id>
       | -- <id>
```

5. Go through the following material
    - Skim the paper “Status Report: Specifying JavaScript with ML”
    - Skim the Web article A brief history of ECMAScript versions
    - Also browse the website http://www.jscert.org/index.html

Group Exercises - Lecture 4

1. Discuss the outcome of the individual exercises
   Did you all agree on the results?

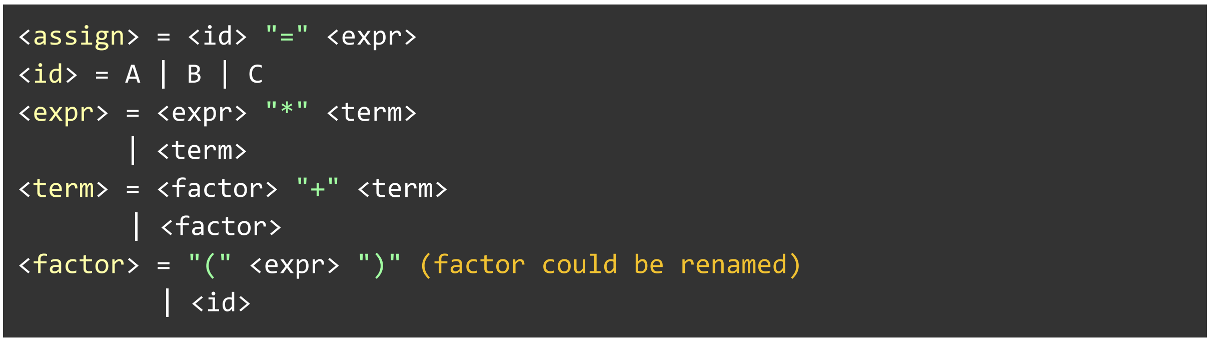
2. Do Sebesta exercise 3 on page 183
   3. *Rewrite the BNF of Example 3.4 to to give + precedence over * and force + to be right associative*



EXAMPLE 3.4 An Unambiguous Grammar for Expressions

```
<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <expr> + <term>
       | <term>
<term> → <term> * <factor>
       | <factor>
<factor> → ( <expr> )
         | <id>
```

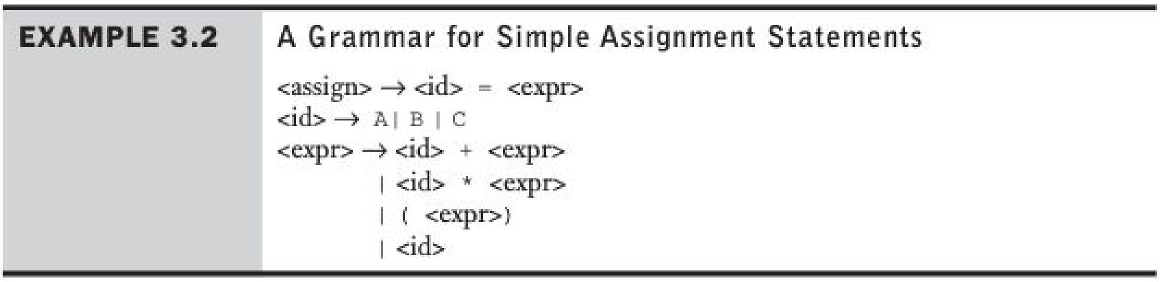
Precedence: switch + and * in grammar
Make + right associative: Switch the order of factor and term
Result:



```
<assign> = <id> "=" <expr>
<id> = A | B | C
<expr> = <expr> "*" <term>
       | <term>
<term> = <factor> "+" <term>
       | <factor>
<factor> = "(" <expr> ")" (factor could be renamed)
         | <id>
```

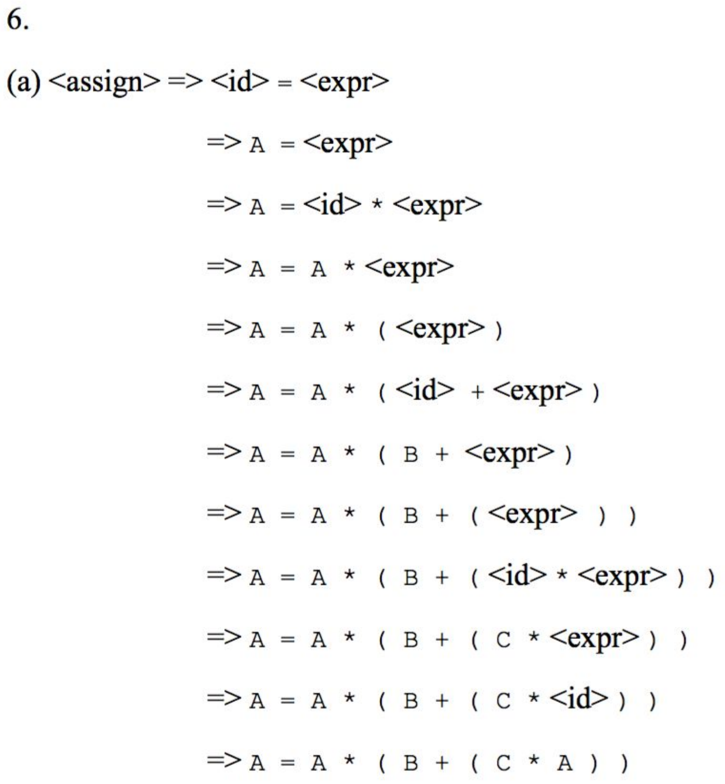
3. Do Sebesta exercise 6a on page 183
   6a. Using the grammar in Example 3.2, show a parse tree and a leftmost derivation for each of the following statements:

   (a) A = A * (B + (C * A))



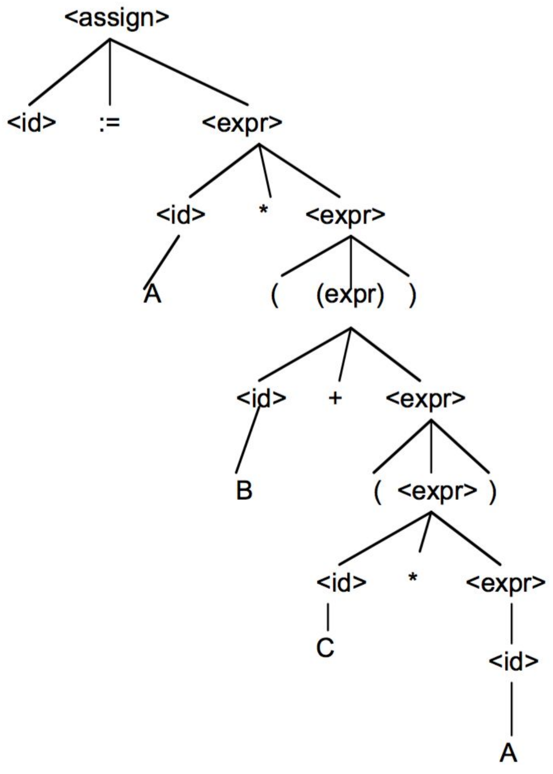
EXAMPLE 3.2 A Grammar for Simple Assignment Statements

```
<assign> → <id> = <expr>
<id> → A| B | C
<expr> → <id> + <expr>
         | <id> * <expr>
         | ( <expr>)
         | <id>
```

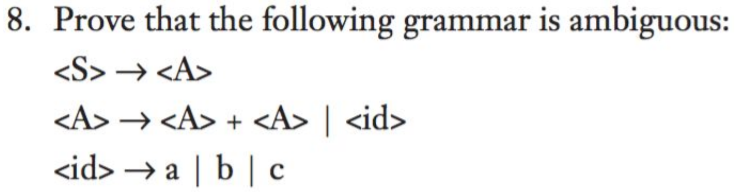


6.

```
(a) <assign> => <id> = <expr>
              => A = <expr>
              => A = <id> * <expr>
              => A = A * <expr>
              => A = A * ( <expr> )
              => A = A * ( <id> + <expr> )
              => A = A * ( B + <expr> )
              => A = A * ( B + ( <expr> ) )
              => A = A * ( B + ( <id> * <expr> ) )
              => A = A * ( B + ( C * <expr> ) )
              => A = A * ( B + ( C * <id> ) )
              => A = A * ( B + ( C * A ) )
```



4. Do Sebesta exercise 8 on page 184

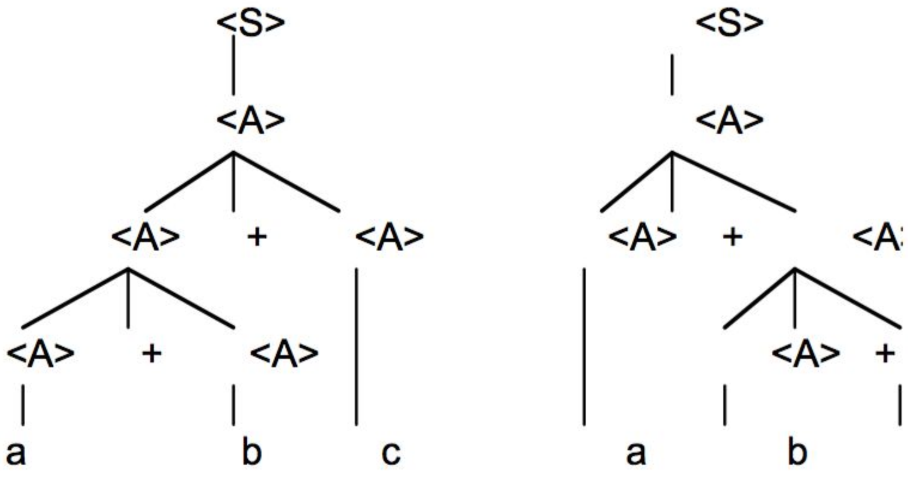


8. Prove that the following grammar is ambiguous:

```
<S> → <A>
<A> → <A> + <A> | <id>
<id> → a | b | c
```

The international edition the book uses * instead of +, x, y, z instead of a, b, c

Use string : a + b + c which has two parse trees (a + (b + c) vs. (a + b) c)



5. Discuss why the specification of ECMAScript version 4 was abandoned

   They could not agree on which features to include, so they decided to split the feature set into multiple versions (ES6, 7 8 and so on). ES4 was supposed to be a radical addition to JavaScript, but they just split it into multiple versions. This made it easier to agree on which features to use in ES5.

   See examples for ES5 here: https://www.w3schools.com/js/js_es5.asp

6. Discuss why the specification of ECMAScript version 5 is now being formalized and mechanized

   Most browsers (major vendors) have supported ES5 since 2012-2013. Most browsers support ES6 (2016-2017). The amount of support warrants a formalisation of ES5.