Assignment 2

Programming Languages

1. [30 pts, grammars] Consider the following unambiguous grammar for expressions,

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

   (a.) Modify the above BNF to give + precedence over * and force + to be right associative

   (b.) Modify the above BNF to add the ++ and the -- unary operators of Java

   (c.) Modify the above BNF to add a unary minus operator that has higher precedence than either + or *

2. [30 pts, grammars] Prove that the following grammar is ambiguous

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

3. [10 pts, ply.lex and ply.yacc to build a lexer/tokenizer]
   Use the Python ply library to build a calculator using the resource from the Ply web page (http://www.dabeaz.com/ply/ply.html)

   Your lex and yacc scripts must be able to parse the following expression:

```
-1 + (2*3 + 4) * -5
```

   A few pitfalls to avoid during the implementation:

   (i) Follow the lex and yacc examples on the Ply web page

   (ii) The yacc needs your lexer Python file in the same directory, *e.g.* named as `mylexer.py`, then `myyacc.py` would have the line: `from mylexer import tokens`

   (iii) Anaconda 3 has ply library (in case, you can also install the downloaded ply-3.11.tar.gz after unzip with `python setup.py install`

   (iv) Python 3 must have `input()` instead of `raw_input()` in the yacc example

4. [30 pts, semantics] Consider the following grammar:

   1. Syntax rule: <assign> → <var> = <expr>
      Semantic rule: <expr>.expected_type ← <var>.actual_type
   2. Syntax rule: <expr> → <var>[2] + <var>[3]
      Semantic rule: <expr>.actual_type ← if (<var>[2].actual_type = int) and (<var>[3].actual_type = int) then int else real end if
      Predicate: <expr>.actual_type == <expr>.expected_type
   3. Syntax rule: <expr> → <var>
      Semantic rule: <expr>.actual_type ← <var>.actual_type
      Predicate: <expr>.actual_type == <expr>.expected_type
   4. Syntax rule: <var> → A | B | C
      Semantic rule: <var>.actual_type ← look-up(<var>.string)

   Modify the attribute grammar in the above BNF above to have: data types cannot be mixed in expressions, but assignment statements need not have the same types on both sides of the assignment operator.

   Note that Attribute Grammars have:

   actual_type - synthesized for <var> and <expr>
   expected_type - inherited for <expr>
   env - inherited for <expr> and <var>