Instructions: Quiz #2, on 09/30, will consist of questions taken or inspired from Part I of this homework ("Short Questions") and will also serve to assess your understanding of Part II ("Longer Problems").

1 Short Questions

Disclaimer: some questions were inspired by Compilers: Principles, Techniques, and Tools *by A. V. Aho, R. Sethi & J. D. Ullman.*

1. Translate each of these regular expressions into a context-free grammar.
   (a) `((xy+x)|(yx+y))?`
   (b) `((0|1)+"."(0|1)*) | ((0|1)*"."(0|1)+)`
2. Write an unambigous grammar that characterizes the language of palindromes over the alphabet `a, b`. You can run your grammar through bison to make sure it is unambigous.
3. Write an unambigous grammar that characterizes the strings over the alphabet `(, ), [, ]` that have balanced parentheses and square brackets. An example of such string could be `([ [] (() [ () ] []) ])`.
4. Construct the directed acyclic graph for the expression `x = 12 + y - 8 + (3 - x) * (3 - y) / 12`, assuming usual association and precedence.
5. Convert the following code to three-address code, and then represent it as quadruples: `a = b * - d`.
6. Convert the following code to three-address code:
   (a)
   ```
   if (x < y) then 1 else 0;
   ```
   (b)
   ```
   if (y < x) {if (z > j) then t = 1;} else t = 0;
   ```
   (c)
   ```
   for(i = 1; i<=10; i++)
   {
       x = x * 5;
   }
   ```

2 Longer Problems

Problem 1 Have a look at the "Deterministic context-free languages" Section of the Wikipedia page comparing parser generators, and then answer the following:

- List five different parsing algorithms.
- Which parsing algorithms seem to be the most used? Make an educated guess explaining that popularity.
- Why is the "Lexer" field either "included", "none", "generated", or "external"? Look up e.g. Coco/R to confirm what "generated" means.
- Observe that some parsers takes EBNF grammar as input. Using e.g. this example, make sure you understand the meaning of `[ ... ]` and `( ... )`.

Problem 2

1. As a warm-up, have a second look at the bison example file that was shared. Make sure you remember how to compile and execute the lexer and parser files using flex and bison.
2. Download the "Example: Calculator II" files: lexer.l and parser.y.
3. Compile and execute the lexer and parser[1], execute the resulting binary, and input some examples, as "12 - 2", "29*12", "12 - 2", "12/20-(12*5)",,"1-3*4+5" and "-2/3". Make a clear statement on the limitations of the parser you observed.
4. Observe the lexer file, and answer the following:

   (a) When is the `DONE` token returned?
   (b) Make sure you understand the rule `[ \t]{}`. Try removing it, and find *two* new ways of making the parser return an error statement.
   (c) Upgrade the lexer so that it also accepts unicode signs × (for "times"), and ÷ (for "division").

[1] If you have an error regarding `%error-verbose` being deprecated, simply replace it with `%define parse.error verbose` as indicated.

5. Observe the parser, and answer the following:

   (a) Consult the documentation to understand what `union` does. Try changing the datatype of `value` to `int`, and observe the result: how is the value displayed? What operation is now performed?
   (b) Consult the documentation to understand what `type` does.
   Comment the line that starts with `%type` and observe the error message that is now returned.
   (c) What is the start symbol in the parser? Use the syntax we used in class to explicitely specify the start symbol, and make sure it works as expected.

☛ **Problem 1** Read the abstract, introduction and Section 5 ("*Experimentation on a C99 Parser*") of *Validating LR(1) Parsers* (whose pre-print is freely available at `https://hal.inria.fr/hal-01077321/document`), and answer the following questions:

- What does "end-to-end verification" means?
- What are the limits of the approach consisting in making sure that parse trees produced by a parser conforms to the grammar?
- Had parser *generators* ever been formaly verified prior to this work?
- Briefly remind what LALR and SLR are, and which is included in the other. Refer to 4.7.4 in this homework's solution for an example of grammar in one class but not the other.
- What is "compile-compile time"?
- What are the three sources of ambiguity in the original ISO C99 grammar?
- Is the verified parser slower or faster than the original one ("OCamlYacc's execution engine")?
- Read the paragraph starting with "As shown in Figure 1.1, only phase 3…" at `https://compcert.org/man/manual001.html#sec10` and answer the following two questions:
  1. Is the CompCert compiler enforcing "end-to-end verification"?
  2. Is CompCert's parser formally verified?
- Have a brief look at the source code presented in this paper. How many "versions" of the C syntax are supported?