This exercise is based on the paper[1] describing *yacc*.

1. List the members of your group below. Underline your name.

2. What is the smallest yacc program?

3. Consider a simple list-based calculator that operates as follows: Input tokens are separated by whitespace. The input consists of integers in the conventional notation mixed in with parentheses (‘`(`’ and ‘`)`’) and the arithmetic operators `+`, `-`, `*`, and `/`, with their usual meanings. Lists are represented using parentheses, so that `(3 10 4)` denotes a list with elements 3, 10, and 4. Each arithmetic operator is applied to the two item immediately preceding it. Arithmetic operators are applied to scalars with the usual semantics. They are applied to lists in an element-wise manner; for instance, `(1 3 5) (2 1 1) +` yields `(3 4 6)`. If the lists are of unequal lengths, the shorter one is extended using the appropriate identity element for the operator. A scalar is promoted to a singleton list when needed.

   List the output of the calculator on the following input:

   ```
   3 (5 9) + (22) 2 - / (99 12) *
   ```

[1] Stephen C. Johnson, Yacc: Yet Another Compiler-Compiler, Unix Programmer's Manual, Volume 2b. AT&T Bell Laboratories. Murray Hill, New Jersey, 1978.

4. Provide yacc code for the calculator of Question 3. The resulting program should consume standard input and write to standard output the result of the application of each operator in the input, as soon as possible.

5. Provide yacc code for a program that reads from standard input a collection of dates, one per line, and writes those dates to standard output after sorting and duplicate-elimination. The program should accept dates in three formats: the two described in the paper, as well as the format `yyyy-mm-dd`. Different representations of the same date are considered duplicates (e.g., '2010-02-16' and 'February 16, 2010'). The format of dates from input to output should be preserved; in case of duplicates with differing formats, the one appearing first in the input is preserved.

6. Describe an algorithm that computes an Eulerian path in a given graph, or determines that no such path exists. Explain your algorithm with a suitable example.