4. Suppose you have a solution to Homework 2. Suppose also that you have a customer who wants a different kind of matcher, which we'll call a "parser". A parser is a function that accepts a complete fragment (i.e., a list of terminal symbols) and returns an optional "iderivation", that is, a list of integers representing a derivation of the fragment, where each integer is an origin-1 index of the grammar rule used for that step of the derivation. If multiple iderivations are possible, the parser should return the first one, in the sense of "first acceptable match" of Homework 2. E.g., with the grammar:

1. Expr → "a" Expr
2. Expr → "b" Expr
3. Expr → "b" Expr
4. Expr → "c"
5. Expr → "d"

matcher → parser

parser → matcher + acceptor

if you have a parser p for this grammar, then (p ["a";"b";"a";"c"]) should return (Some [1;2;1;4]). If the complete fragment is not a valid sentence in the language described by the grammar, the parser should return None.

4a (10 minutes). Write a function 'make_parser' that, given a Homework 2-style grammar, returns a parser for the grammar. Like Homework 2, it is OK if the parser loops for left-recursive grammars. Your function can call Homework 2's parse_prefix function to do its work.

4b (3 minutes). What should be the type of 'make_parser'?

4c (16 minutes). Suppose your customer also wants to iterate through all possible iderivations, not the just the first one. Write a function 'make_backtracking_parser' that, given a grammar, returns a backtracking parser. A backtracking parser is a curried function that accepts two arguments: first, a iderivation acceptor and second, a fragment. A iderivation acceptor is a function that accepts a iderivation and returns an optional value indicating whether the iderivation is acceptable. The caller can therefore iterate through most or all of the possible iderivations by passing a picky iderivation acceptor. For example, if bp is a backtracking parser for the example grammar shown above, the expression

```
let acc x = if List.mem 3 x then Some x else None
in bp acc ["a";"b";"a";"c"]
```

should return (Some [1;3;1;4]). (Recall that List.mem x y returns true if and only if x is equal to a member of y.)

4d (5 minutes). What should be the type of 'make_backtracking_parser'?

5. Professor Millstein has co-invented a Java variant called VBD Java. In VBD Java, variables are volatile by default so the 'volatile' keyword has no effect: if you want a nonvolatile variable, you must declare it with the new-to-VBD keyword 'relaxed'. Experiments by Millstein and his colleagues indicate that their VBD Java implementation runs 28% slower than plain Java on standard generic Java benchmarks, but only 12% slower on standard big-data analytics benchmarks.

5a (5 minutes). Suppose your solution to Homework 3 had been compiled and run with the VBD Java implementation. Describe any problems you would expect to run into in either compilation or execution, and how you would minimally fix any problems. Or, if there would not be any problems, briefly explain why not.

5b (10 minutes). Assuming (a)'s minimal fixes (if any) are applied to get the Homework 3 solution to build and run, how would you expect the performance results to change, compared to standard Java? Briefly justify your answer.

5c (6 minutes). Give a plausible justification for why VBD Java's performance penalty is less on big-data analytics benchmarks than on generic Java benchmarks.

UCLA CS 131 Midterm, Fall 2018
100 minutes total, open book, open notes,
closed computer. Exam is DOUBLE SIDED.

Name: Yuanping Song Student ID _____

1	2	3	4	5	total

1a (10 minutes). Define a curried OCaml function (subbag A B) that returns true if every element of A also appears in B, and in the same order that it appears in A. Duplicates count. For example, (subbag [3;7;3] [3;4;7;3;2]) should return true, and (subbag ["a";"b";"a"] ["b";"c";"a";"a";"b"]) should return false. It's OK to define some auxiliary functions in order to implement 'subbag'.

1b (1 minute). Give the type of 'subbag'.

2 (20 minutes). Would it make sense to write a compiler to translate C++ source code to Java bytecodes? The idea is to run your C++ program on a modern, high-performance Java interpreter; your program would be accompanied by an implementation of the C++ library written in a combination of Java and machine code, just as the traditional C++ library is implemented in a combination of C++ and machine code.

If it would make sense to write the compiler, list any difficulties you'd have in writing it or the associated library, and list practical pros and cons of the resulting system compared to the traditional approach. If it would not make sense to write the compiler, explain why not, and list the features of C++ that you'd need to drop support for, in order to make the job practical. When answering the question, consider all the Java features covered in class.

3. Consider the following grammar for ISO EBNF, modified from the ISO standard. Capitalized identifiers in this grammar stand for tokens that have many different lexemes; for example INTEGER might be 27, META IDENTIFIER might be foobar, and TERMINAL STRING might be 'xyzzy'.

```
syntax = syntax rule, {syntax rule};
syntax rule =
  meta id, '=', symbol, definitions list, ';';
definitions list =
  definition, {'|', definition};
definition = term, {',', term};
term = factor, ['-', exception];
exception = factor;
factor = [INTEGER, '*'], primary;
primary = optional sequence
        | repeated sequence
        | grouped sequence
        | META IDENTIFIER
        | TERMINAL STRING
        | empty;
optional sequence = '[', definitions list, ']';
repeated sequence = '{', definitions list, '}';
grouped sequence = '(', definitions list, ')';
empty = ;
```

3a (5 minutes). Is this grammar ambiguous? If so, give an example of the ambiguity. If not, explain why not.

3b (10 minutes). Give a good syntax diagram for the grammar, with a minimal number of nonterminals.