Name: **PID:**

Midterm Exam

- This is a closed-book exam; you may not use any tools besides a pen.
- You have 75 minutes to answer all questions.
- There are a total of 75 points available.
- Please write legibly; answers that I cannot decipher will receive no credit.
- Raise your hand if you have a question.

Problem 1 [7 points]

Indicate for each of the following statements whether it is true or false.
(Correct answer: 1 point; incorrect answer: -0.5 points; in total no less than 0 points.)

A. Functional programming is focused on message passing. True or false:_____
B. It is generally easier to compile declarative languages into efficient machine code than to do so with imperative languages. True or false:_____
C. Virtual machines are a popular implementation strategy because they do not require compilation. True or false:_____
D. Writing code in a programming language should be easier than reading it since programmers spend a lot of time writing code. True or false:_____
E. Assembly languages are procedural in nature. True or false:_____
F. Java is a procedural language. True or false:_____
G. It is always correct for an optimizer to remove arithmetic expressions that include divisions if the computed result is not used. True or false:_____

E is correct; the rest is incorrect.

Problem 2 [2 points]

Name an advantage and disadvantage of separate compilation.

Pro: enables code reuse/libraries, parallel development and collaboration.
Con: difficult to maintain, version mismatch bugs, some optimizations not possible.

Problem 3 [2 points]

Briefly describe the purpose of the lexical and syntax analysis phases in a compiler.

Lexical analysis: discover tokens, i.e., group individual input characters that together have atomic meaning; reject programs with invalid/unexpected characters.

Syntax analysis: discover the structure of the program; reject programs with invalid/unexpected structure.

Problem 4 [5 points]

Indicate for each of the following statements whether it is true or false.
(Correct answer: 1 point; incorrect answer: -0.5 points; in total no less than 0 points.)

A. There exist regular grammars that no DFA can recognize. True or false:_____
B. The Kleene Star is "notational sugar" and can be expressed in terms of concatenation and alternation. True or false:_____
C. NFAs and DFAs are equivalent, i.e., there exists a corresponding DFA for every NFA. True or false:_____
D. One can construct a DFA that ensures that some string is enclosed by exactly three pairs of matching parenthesis (e.g., "`(((abc)))`"). True or false:_____
E. There exist context-free grammars that no NFA can recognize. True or false:_____

C, D, and E are correct.

Problem 5 [3 points]

Based on the below grammar, perform a left-most derivation of a program fragment using *assignment* as the start symbol. Your derivation should involve at least five intermediate steps. Show all intermediate steps.

assignment → *qname* := *expr* ;
qname → `id`
qname → `id` . *qname*
expr → *qname* | `number` | *expr op expr*
op → + | - | * | /

(Terminal symbols are underlined.)

assignment ⇒ *qname* := *expr* ;
⇒ `id` . *qname* := *expr* ;
⇒ `id` . `id` := *expr* ;
⇒ `id` . `id` := *expr op expr* ;
⇒ `id` . `id` := `number` *op expr* ;
⇒ `id` . `id` := `number` + *expr* ;
⇒ `id` . `id` := `number` + `number` ;

Problem 6 [2 points]

Give two reasons why the above grammar is not a LL(1) grammar.

The *qname* productions have a common prefix.
One of the *expr* productions is left-recursive.

Problem 7 [12 points]

Create regular expressions that specify the following (simplified) Prolog token types (adapted from the Siemens IF/Prolog manual):

- *Integers* are sequences of digits, possibly preceded by the minus sign (-).
- *Floating-point numbers* are sequences of digits containing a decimal point, possibly preceded by the minus sign.
- *Octal numbers* consist of a prefix `0o` (Zero-o), and a sequence of octal digits (from 0 to 7), the whole possibly preceded by the minus sign.
- *Hexadecimal numbers* consist of a prefix `0x` (Zero-x), and a sequence of hexadecimal digits (from 0 to 9, from a to f and from A to F), the whole possibly preceded by the minus sign.
- *Atoms* are character sequences (i) beginning with a lowercase letter and consisting only of letters, digits and underscores, or (ii) zero or more letters, digits and underscores enclosed in single quotes.
- *Variables* are also sequences of letters, digits and underscores that begin with an uppercase letter or with an underscore to distinguish them from atoms.

You may use the Kleene Star (*), alternation (|), concatenation, and grouping with parentheses. Further, for conciseness, you may use character ranges, e.g., you may use '`[X0-9x-z]`' to represent '`(X|0|1|2|3|4|5|6|7|8|9|x|y|z)`'.
Do not use any other constructs.

int: `(-|)[0-9][0-9]*`

float: `(-|)([0-9]*.[0-9][0-9]*|[0-9][0-9]*.[0-9]*)`

oct: `(-|)0o[0-7][0-7]*`

hex: `(-|)0x[0-9a-fA-F][0-9a-fA-F]*`

atom: `[a-z][a-zA-Z0-9_]*|'[a-zA-Z0-9_]*'`

var: `[A-Z_][a-zA-Z0-9_]*`

This problem pertains to Prolog structures based on the the following (simplified) description: "a structure is an atom followed by an opening parenthesis, followed by one or more comma-separated terms, followed by a closing parenthesis." Recall that a Prolog term is either an atom, a variable, a numeric literal, or a structure.

Problem 8 [8 points]

Create a context-free grammar that defines Prolog terms.

Problem 9 [4 points]

The grammar for Problem 8 is an LL(1) grammar.

(Use the token types from Problem 7 and *left-paren*, *right-paren*, and *comma* as the terminal symbols in your grammar. Only one correct answer is required to get full credit for both Problems 8 and 9.)

term → int | float | hex | oct | var
term → atom *tuple*
tuple → ε
tuple → left-paren *term term-list* right-paren
term-list → ε
term-list → comma *term term-list*

Recall that we used two clauses, `mother/2` and `father/2`, to describe a royal family tree in class. For this problem, assume that a couple has kids if and only if they are married (since this is, after all, a very honorable and royal family).

Problem 10 [4 points]

Write a Prolog rule that expresses the "mother in law" relationship.

Problem 11 [4 points]

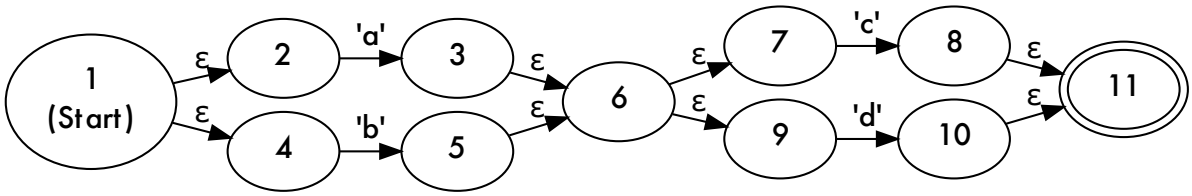
Write a Prolog rule "generations" that counts how many generations two persons are apart. E.g., if Sam is the father of Bill, and Bill is the father of Sue, then Sam is two generations apart from Sue.

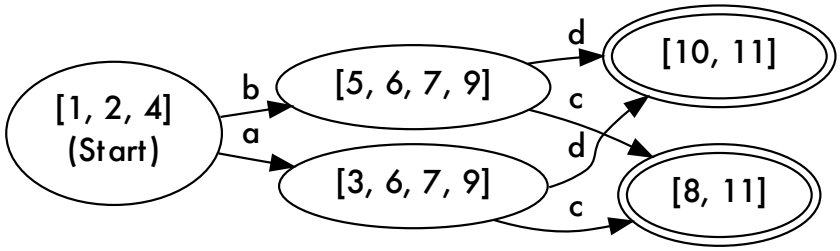
```
mother_in_law(X, Y) :-
  mother(X, A), father(A, B), mother(Y, B).
mother_in_law(X, Y) :-
  mother(X, A), mother(A, B), father(Y, B).

parent(X, Y) :- father(X, Y).
parent(X, Y) :- mother(X, Y).

generations(X, Y, Num) :-
  parent(X, Y),
  Num = 1,
  !.
generations(X, Y, Num) :-
  parent(X, Z),
  generations(Z, Y, Gen),
  Num is Gen + 1.
```

Note: the generations solution only takes direct inheritance into account, and only works from top to bottom. I gave full credit for anything that remotely approached this simplified solution.

Problem 12 [5 points]

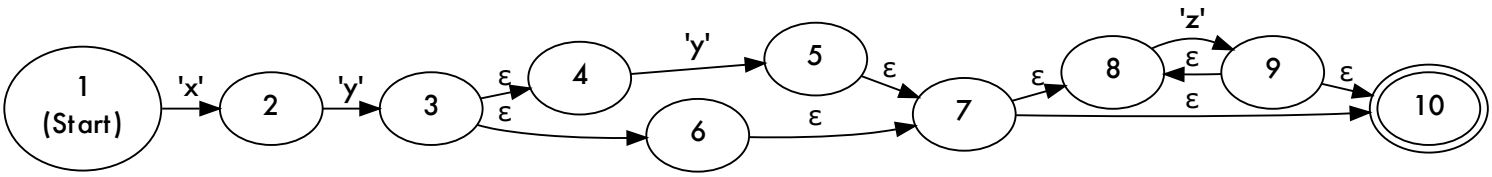
Convert the below NFA to an equivalent DFA using the "sets of states" construction as discussed in class. Indicate the set of states that each DFA state corresponds to.





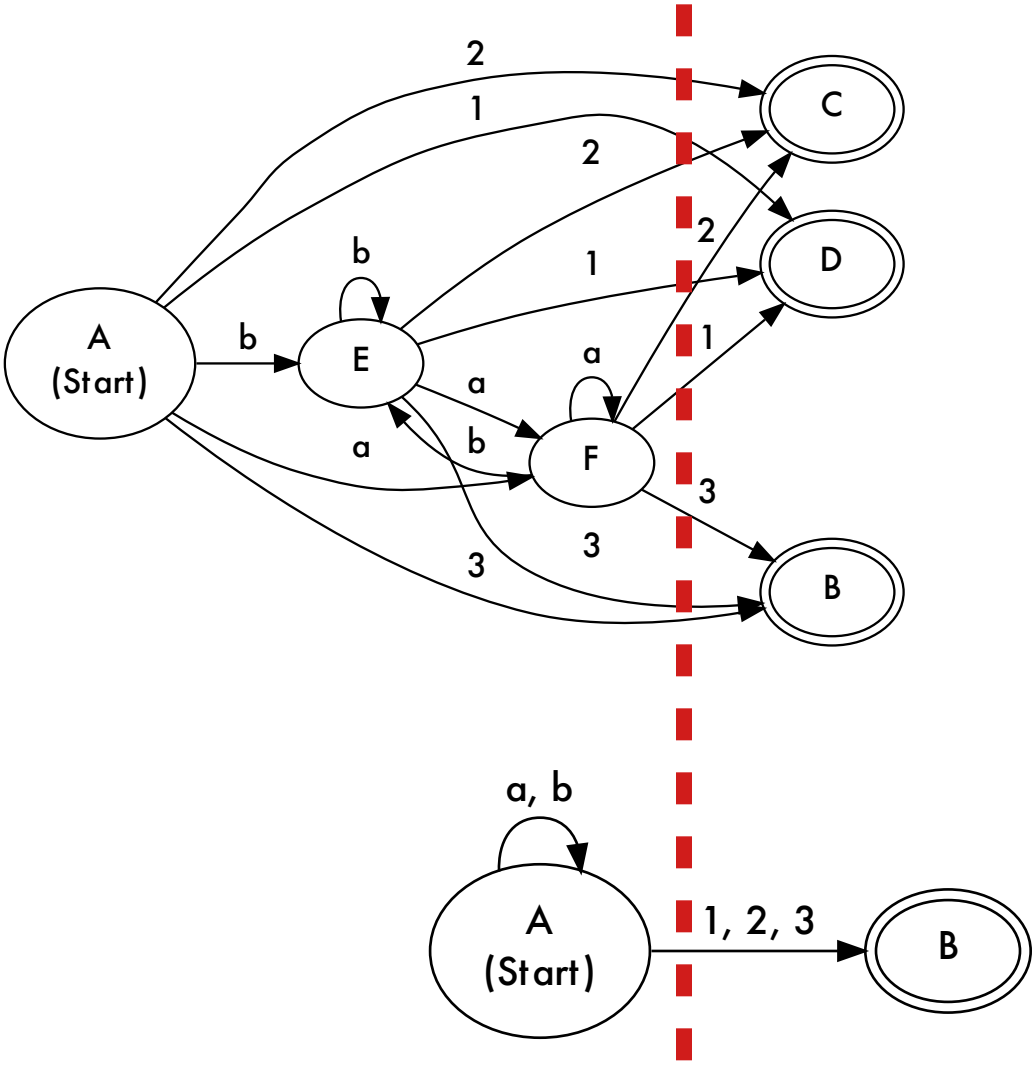
Problem 13 [5 points]

Create an NFA that is equivalent to the regular expression "`xy(y|)z*`".



Problem 14 [5 points]

Find a DFA with a minimal number of states that is equivalent to the below DFA by partitioning equivalent states into equivalence classes.
Indicate all partitions by drawing one or more lines (as we did in class).



Problem 15 [3 points]

Name the three principle memory allocation classes and briefly describe the corresponding object lifetimes.

Static: objects exist throughout program runtime; corresponding memory is allocated by OS before program execution starts.

Runtime stack: objects exist for the duration of one subroutine execution.

Heap: objects can have arbitrary lifetimes; allocation and deallocation is possible at any point in time.

Problem 16 [4 points]

What is the "output" of the below pseudo code assuming bindings are resolved using the closest nested scope rule and (respectively)

A. dynamic scoping; and

B. lexical (i.e., static) scoping?

```
VAR count: INTEGER;

PROCEDURE procX IS
BEGIN
  VAR count: INTEGER;
  SET count TO 100;
  CALL report
END

PROCEDURE procY IS
BEGIN
  SET count TO 200;
  CALL report
END

PROCEDURE report IS
BEGIN
  PRINT "count = " + count
END

MAIN PROGRAM IS
BEGIN
  SET count TO 300;
  CALL procX;
  CALL procY
END
```

A)
count = 100
count = 200

B)
count = 300
count = 200