[redacted]

1. Describe the advantages that Prolog has over the traditional imperative programming languages such as C and Java. Also, give an account of Prolog's shortcommings.

2. Apply the method of project 5 (path/3 applied to a graph) to solve the "Monkey and tha Banana with two Boxes" problem (project 2). Demonstarte your solution by displaying the monkey's moves.

3. Use DCG to define Prolog's syntax and demonstrate it on sample programs. Suggestion: Start by defining a simple subset of Prolog, e.g., without predicates and functions (i.e., propositional Prolog), and then gradualy keep adding the extra features.

4. Modify the following meta-interpreter for Prolog so that it will "prove" predicates which are defined using a three-valued logic (true, false and maybe). Define and use a binary operator to keep track of the calculated truth values (see the "kitchen" Expert system bellow). Write a Prolog program which prompts the user to enter a three-valued definition of predicates and calls the modified meta-interpreter to process the questions in the three-valued logic (see the definition of the predicate myprolog bellow).

```
/* prove/1 is a tracing meta-interpreter */

prove( true) :- !.

prove( ( Goal1, Goal2)) :- !,
  prove( Goal1),
  prove( Goal2).

prove( Goal) :-
  write( 'Call: '), write( Goal), nl,
  clause( Goal, Body),
  prove( Body),
  write( 'Exit: '), write( Goal), nl.

/* myprolog/0 is a shell for calling prove/1 */
myprolog :- write('STARTING ...   '),

    consult('/Users/petergabrovsky/Desktop/prove.pl'),
    write('enter the file name of a program:   '), read(X), consult(X),
    /* example: 'family.pl'  */
    nl, write('myprolog – ask a question:     '), read(Y), prove(Y).

/* Kitchen Expert system using many-valued logic */

:-dynamic(fact/2).
:-op(100, xfx, :).
:-op(150, xfx, is).
:-op(200, xfy, and).
:-op(300, xfy, or).
:-op(700, xfx, then).
:-op(800, fx, if).
```