Test 1 (Midterm) CSCE 330-002 (85 points)

Name:_____

1. (2) Prolog is an example of a _declarative_ paradigm language.
2. (2) A mouse is a Von Neumann ___input_____ device.
3. (2) A language is __orthogonal__if its features are built upon a small, mutually independent set of primitive operations.
4. (2) A __compiler__ converts source code into machine language.
5. (2) In Natural Language Processing, a _lexicon___ describes the words we use.
6. (2) What's the main difference between a Harvard and a Von Neumann machine?
   Harvard has separate data and instruction buses/memories
7. (3) Distinguish between a co-routine and a concurrent unit.
   Co-routines will interleave execution, involving checkpoints. Concurrent units may execute in parallel.

Word Bank: imperative, declarative, functional, object-oriented, binding, orthogonal, input, output, control, memory, ALU, Von Neumann, Harvard, compiler, interpreter, static, dynamic, world model, grammar, lexicon, parsing

Consider a knowledgebase defined like the following, for the identification of poisonous fruits, which is also populated with normal ones:

% traits are in name,color,typical longest dimension in mm, source order

```
fruit(cherry,red,20,tree).
fruit(grape_seedless,green, 25, vine).
fruit(grape_wine,red,   25, vine).
fruit(holly,red,8,bush).
fruit(holly,brown,8,bush).

%fruit,woody/smooth,low number seeds, high number seeds.
seed(cherry,woody,1,1).
seed(grape_wine, smooth, 3,8).
seed(holly,woody,3,10).

symptom(holly, vomiting).
symptom(holly, diarrhea).
```

8. (2) Write a query to find all red fruits with a single seed:

   ?- fruit(F, red, _,_),seed(F,_,1,1).

9. (1) What key would you press to list them all (one per press)? semicolon **;**

10. (3) What query would you make to find a brown fruit that caused vomiting?

    ?- fruit(F,brown,_,_),symptom(F, vomiting).

11. (4) What query would you make to find all drupes? (A drupe has a single woody seed).

    ?- seed(F,woody,1,1,). %note, don't need fruit/4

DO ONLY 3 OF 12, 13, 14, or 15 (if you do all 4 I will grade 12, 13,and 14).

12. (8) Acme Company has the following structure:
    - Big Bob is the CEO, the head of the company.
    - Jingles N. Folds is the CFO. The CEO is the CFO's boss. (you may do boss part in following question)
    - Dolph Ett is the COO. The CEO is his boss. (you may do boss part in following Question)
    - The CFO is head of the AR (accounts receivable) and Budget Departments.
    - The COO is the head of the Development, Maintenance, and Customer Service Departments.
    - Each department consists of teams, which always has one team lead. Every team belongs to one department.
    - Marsha is team lead of the Omni Project, a team in the Development Dept. Her immediate supervisor/boss would be Dolph Ett.
    - Billy and Kathy are members of the Omni Project

    Write a knowledgebase, containing all this information:

```
ceo(bigBob).
cfo(jingles).
coo(dolph).

head_of(X,ar):-cfo(X).
head_of(X,budget):-cfo(X).
head_of(X,dev):-coo(X).
head_of(X,maint):-coo(X).
head_of(X,custserv):-coo(X).
team(omni,dev).
team_lead(marsha,omni).
member(billy,omni).
member(kathy,omni).
```

a) (7) Using your above Knowledgebase, define a boss rule -- a "boss" is an employee's *immediate* supervisor -- Big Bob is the boss of Dolph but not Marsha. It is reasonable to define the relationships explicitly for the officers and their departments (CEO/CFO/COO) but **not** for the regular employees.

```
boss(X,Y):-ceo(X),cfo(Y).
boss(X,Y):-ceo(X),coo(Y).
boss(X,Y):-team_lead(Y,T),team(T,D),head_of(X,D).
boss(X,Y):-member(Y,P),team_lead(X,P).
```

b) (5) Using the above, write a predicate that will print the chain of command, starting at an employee and going all the way up to the CEO. It should print in Employee->CEO order. Each "level" should be on its own line.

```
coc(X):-ceo(X). %returns false without this (b/c coc(bigBob) ) fails
coc(X):-boss(B,X),print(B),print(' is the boss of '), print(X),nl,coc(B).
```

13. Lists
    a) (10) Write a rule to find the smallest item in a list of numbers. You may use **member** or **append**. If there are multiple instances of a value in the list that are the smallest, your rule should succeed. Hint: Consider negation, another predicate and/or parentheses.

```
smallest(S,L):-member(S,L),\+ (member(Other,L), Other < S).

%might want a cut
smallest2(S,[S]).
smallest2(S, [S|T]):-smallest2(SSub,T),S<SSub.
smallest2(SSub, [S|T]):-smallest2(SSub,T),S>=SSub.

smallest3(S,L):- member(S,L),\+exists_smaller(S,L).
exists_smaller(S,L):-member(S,L),member(B,L), B < S.
```

    b) (10) Write a rule or rules to tell if the sum of a list of numbers is **odd** (modulus in SWI-Prolog is **mod,** e.g. Y is 4 mod 3, Y is 1 succeeds – use mod in an arithmetic expression (like **is**)).

```
odd(X):-Y is X mod 2, Y= 1.
even(X):- \+odd(X).

sum_odd(L):-sum_list(S,L),odd(S).

sum_list(S,[S]).
sum_list(S,[H|T]):-sum_list(Sub,T),S is Sub+H.


sum_odd2(L):-so_sub(even,L).

so_sub(odd,[]).
so_sub(even,[H|T]):- odd(H),so_sub(odd,T).
so_sub(even,[H|T]):- even(H),so_sub(even,T).
so_sub(odd,[H|T]):- odd(H),so_sub(even,T).
so_sub(odd,[H|T]):- even(H),so_sub(odd,T).
```

14. (20)You're having a wedding and are tired of all of your friends being single, for whatever reason, and want them to meet other interesting, single people so you decide to seat them so that they are all at the same circular, six-seat table.
    a) Jill, Joy, and Tory are all female. Bob, Mike, and John are all male.
    b) All six are heterosexual, so males should be besides female and vice versa.
    c) Mike and Joy previously dated so they should not sit beside each other, since it ended badly.
    d) Tory is a rabid Clemson fan and won't date diehard Gamecock fans so she cannot sit beside John.

    Write a program to solve this problem. Please define a **solution(Bob,Mike,John,Jill,Joy,Tory)** that will give a seat number for each name. You may assume a **unique(X1,X2,X3,X4,X5,X6)** predicate is defined that will guarantee all six inputs are unique. You may, of course define any other rules/facts you find necessary. (Hint 1: Consider a beside relationship, *fully. Hint 2: What domain should you use?*).

```
unique(P1,P2,P3,P4,P5,P6):-
  P1 \=P2, P1 \=P3, P1 \=P4, P1 \=P5, P1 \= P6,
  P2 \=P3, P2 \=P4, P2 \=P5, P2 \=P6,
  P3 \=P4, P3 \=P5, P3 \=P6,
  P4 \=P5, P4 \=P6,
  P5 \=P6.


position(0). position(1). position(2). position(3). position(4). position(5).


beside(0,1).
beside(1,2).
beside(2,3).
beside(3,4).
beside(4,5).
beside(5,0).
beside(2,1).
beside(3,2).
beside(4,3).
beside(5,4).
beside(0,5).
beside(1,0).
```

```
solution(Bob,Mike,John,Jill,Joy,Tory):-
position(Bob), position(Mike), position(John), position(Jill), position(Joy), position(Tory),
unique(Bob,Mike,John,Jill,Joy,Tory),
\+beside(Tory,Jill),
\+beside(Jill,Joy),
\+beside(Joy,Tory),
\+beside(Tory,John),
\+beside(Mike,Joy).
```

15. (20)Robot DosSF1SB, a 1 meter(m) wide and long robot, needs to find a path to the goal region. It is in a straight hallway 7m long and cannot turn around. The goal is a meter(m) wide region against one wall. The robot can move forward 4m, exactly, in one move (due to delay it cannot move less or more). The robot can move backward 3m, exactly, in one move as well. The robot starts 2m from the goal region, facing the goal region, and must end in the goal region *without hitting a wall.* Consider the following primitive layout:
    |G _ R _ _ _ _|
    G=goal, R=robot, _=meter-wide space, and | = wall. Given the above information and the given general planning code, write the problem-specific code so that **bplan(Plan)** will give the desired plan. Make sure to define all necessary facts and rules. (Hint: there aren't a lot of legal_moves…)

```
% This looks for plans, short ones first, using the plan predicate.
% bplan(L) holds if L is a plan.
bplan(L) :- tryplan([],L).

% tryplan(X,L): L is a plan and has X as its final elements.
tryplan(L,L) :- plan(L).
tryplan(X,L) :- tryplan([_|X],L).

% This general planner needs the predicates below to be defined:
%    - legal_move(BeforeState,Move,AfterState)
%    - initial_state(State)
%    - goal_state(State)

% plan(L): L is a list of moves from the initial state to a goal state.
plan(L) :- initial_state(I), goal_state(G), reachable(I,L,G).

% reachable(S1,L,S2): S2 is reachable from S1 using moves L.
reachable(S,[],S).
reachable(S1,[M|L],S3) :- legal_move(S1,M,S2), reachable(S2,L,S3).


initial_state(3).
```

```
goal_state(1).

pos(X):-member(X,[1,2,3,4,5,6,7]).

legal_move(X,f,Y) :- pos(X), pos(Y), Y is X-4.
legal_move(X,b,Y) :- pos(X), pos(Y), Y is X+3.
```

%%or

```
initial_state(3).

goal_state(1).

legal_move(5,f,1).
legal_move(6,f,2).
legal_move(7,f,3).
legal_move(1,b,4).
legal_move(2,b,5).
legal_move(3,b,6).
legal_move(4,b,7).

%or
%everyone who tried this method messed up case G and R, which are variables.
initial_state([g,-,r,-,-,-,-]).
goal_state([r,-,-,-,-,-,-]). %traps Robot in goal (OK).

legal_move([g,r,-,-,-,-,-],b, [g,-,-,-,r,-,-]).
legal_move([g,-,r,-,-,-,-],b, [g,-,-,-,-,r,-]).
legal_move([g,-,-,r,-,-,-],b, [g,-,-,-,-,-,r]).
legal_move([g,-,-,-,r,-,-],f, [r,-,-,-,-,-,-]).
legal_move([g,-,-,-,-,r,-],f, [g,r,-,-,-,-,-]).
legal_move([g,-,-,-,-,-,r],f, [g,-,r,-,-,-,-]).
```