0.5

3. (Prolog). (2 pts) Consider the syntactically correct Prolog database intended to implement the factorial function:

```
/* a simple fact (factorial(1)=1) */

factorial(1,1).

/* a rule to recursively define factorial  */

factorial(N,Result) :- Imin1 is N-1,
factorial(Imin1,Fmin1),
Result is N*Fmin1.
```

This implementation is incomplete, however. For example, consider:

```
?- factorial(0,What).
ERROR: Out of local stack
?- factorial(-2,Who).
ERROR: Out of local stack
```

Specifically, the problem with this Prolog formulation concerns zero or negative values of the first argument. *In the space below*, provide a revised version of the Prolog formulation which fixes this problem. Sample desired behavior of the fixed version is shown below:

```
?- factorial(0,What).
What = 1.

?- factorial(-1,What).

Do not use factorial with a negative argument
true.
```

~~if (N=0),~~ factorial(0,1). -0.5
~~Result is 1.~~

~~if (N<0),~~ factorial(N,_):- N<0, nl, -0.5
write('Do not use factorial with a negative argument true.').

-0.5

4. Prolog. (4 pts.) Consider a modified Prolog database used to replace the Dean:

```
gets_tenure(Faculty) :- publishes(Faculty), gets_research(Faculty),
                        teaches_well(Faculty).

publishes(Professor) :- does_research(Professor), documents_research(Professor).

gets_research(Researcher) :- writes_proposals(Researcher), gets_funded(Researcher).

teaches_well(Educator) :- prepares_lectures(Educator), lectures_well(Educator),
                          gets_good_evaluations(Educator).

prepares_lectures(rjs).
prepares_lectures(mab).
prepares_lectures(pw).

does_research(mab).
does_research(pw).
does_research(rg).
does_research(mp).

gets_funded(mab).
gets_funded(rc).
gets_funded(rjs).
gets_funded(rmg).
gets_funded(mp).

teaches_well(mab).
teaches_well(mp).
teaches_well(bd).

documents_research(_).
documents_research(rjs).
documents_research(pw).
documents_research(rmg).


writes_proposals(pw).
writes_proposals(rjs).
writes_proposals(rmg).
```

```
writes_proposals(_).

gets_good_evaluations(rc).
gets_good_evaluations(pw).
gets_good_evaluations(rjs).
gets_good_evaluations(mab).
gets_good_evaluations(mp).

lectures_well(pw).
lectures_well(mab).
lectures_well(rjs).
lectures_well(_).
```

(a) (3pts) Suppose the above database is consulted and Prolog is then given the goal:

```
?- gets_tenure(Anyone).
```

Show the response of the Prolog system to this goal, i.e., all possible bindings for variable Anyone.

Anyone = pw x -1

Anyone = mab

mp -1.5

Yes

(b) (1pt) Using this Prolog database, which of the following queries succeed? CIRCLE THEM.

?- gets_tenure(pw). x -0.3

?- gets_tenure(mab).

?- gets_tenure(xp).

1. (Prolog). (20 points) Consider a modified Prolog database used to replace the Dean:

```
gets_tenure(Faculty) :- publishes(Faculty),
                        gets_research(Faculty),
                        teaches_well(Faculty).

publishes(Professor) :- does_research(Professor),
                        documents_research(Professor).

gets_research(Researcher) :- writes_proposals(Researcher),
                             gets_funded(Researcher).

teaches_well(Educator) :- prepares_lectures(Educator),
                          lectures_well(Educator),
                          gets_good_evaluations(Educator).
```

```
prepares_lectures(_).
writes_proposals(rja).
prepares_lectures(rja).
gets_good_evaluations(rc).
does_research(mab).
gets_funded(mab).
documents_research(rja).
documents_research(pw).
writes_proposals(pw).
gets_good_evaluations(pw).
gets_good_evaluations(rja).
lectures_well(pw).
lectures_well(mab).
lectures_well(rja).
gets_funded(rc).
gets_funded(rja).
does_research(pw).
```

Suppose the above database is consulted and Prolog is then given the goal:

```
?- gets_tenure(Anyone).
```

Show below the response of the Prolog system to this goal, i.e., all possible bindings for variable Anyone.

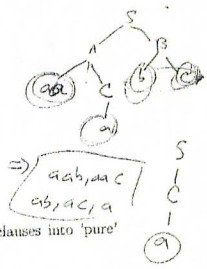
FALSE

2. (Prolog-LGN) (20 pts)
Consider the following file, containing Prolog LGN clauses for a grammar, $G_{inprolog}$.

```
/* Quiz 2, Problem 2 */
/* t* denotes a terminal to avoid confusion */

s --> a,b.
s --> c.
a --> c.
a --> [ta,ta].
b --> [tb].
b --> [tc].
c --> [ta].
```

S → AB
S → C
A → C,
A → aa
B → b
B → c
C → a



a.) Show the translation of each of the following LGN clauses into 'pure' Prolog.

S → AB
a --> a,b.

```
s(A,C) :- a(A,B),
          b(B,C).
```

comma list

S → C
a --> c

```
s(A,B) :- c(A,B).
```

B → c
b --> [tc]. terminal c

```
b([tc|A],A).
```

b.) Circle any of the following strings (represented as Prolog lists) which are elements of the Language of $G_{inprolog}$.

[ta]

[tt]

[ta,ta]

[ta,ta,tc]

[tb,tc]

5. Simple Prolog and Lists (20 pts)
The following Prolog database is syntactically correct and is consulted.

```
/* some_data2.pro for SSII2016 */
some_data2([[0.0, 0.0, 0.0, 1.0], [-1.00]]).

predicate1([H|_],H).

predicate2([_|[_|T]],T).
```

For the Prolog goals entered below, if the goal succeeds, show the resulting variable bindings where indicated. If the goal fails, simply write 'FAIL'.

```
?- some_data2([H|T]),predicate1(H,What).
```

H = [0.0, 0.0, 0.0, 1.0],

T = [-1.00],

What = 0.0 .

```
?- some_data2(L),predicate2(L,What).
```

L = [[0.0, 0.0, 0.0, 1.0], [-1.0]],

What = [].

Show, in the space indicated, all solutions and variable bindings for each of the following goals. If the goal fails, simply write 'FAIL'.

```
?- some_data2(Data), member(-1.00,Data).
<ans. here>
```

FALSE

not [-2.0]

```
?- some_data2(Data), member([0.0, 0.0, -1.0],Data).
<ans. here>
```

FALSE

```
?- some_data2(Data), member(What,Data).
<ans. here>
```

Data = [[0.0, 0.0, 0.0, 2.0], [-2.0]],
What = [0.0, 0.0, 0.0, 2.0];

Data = [[0.0, 0.0, 0.0, 2.0], [-2.0]],
What = [-2.0].

20

3. (Prolog cut, not, fail) 20 points total.
We have studied the use of the cut, fail and not in Prolog. The following syntactically correct database contains modification of class and book file smpl-unify1.pro using these constructs, and is consulted.

goal1(who, what)
what=4, 6

goal2(who, what)

goal3(who, what)
6
a

```
/* smpl_unify1_mod3.pro
   for quiz 1
*/

goal1(_,Y) :- second(Y), first(Y).

goal2(X,Y) :- first(X), second(Y), fail.

goal3(X,Y) :- first(X), !, second(Y).

goal4(X) :- second(X), not(second(X)).

goal5(X,Y) :- first(X), first(Y), not(second(X)),not(second(Y)).
```

(2,5) (5,2)

```
first(2).
first(4).
first(5).
first(8).

second(4).
second(6).
second(8).
second(10).

third(3).
third(4).
third(5).
third(6).
```

For each Prolog goal on the next page, show all Prolog solutions for each goal (including variable bindings, if any), directly under the goal.

* ?- goal1(Who,What).

What = 4

What = 8

* ?- goal2(Who,What).

false

?- goal3(Who,What).

Who = 2, What = 4

Who = 2, What = 6

Who = 2, What = 8

Who = 2, What = 10

?- goal4(Who).

false

?- goal5(Who,What).

Who = 2, What = 2

Who = 2, What = 5

Who = 5, What = 2

Who = 5, What = 5

16.

4. (Prolog Lists) 20 Points total
Given the following Prolog database:

```
/* Problem 4 (problem4.pro) */
/* Note: The Prolog manual indicates:
length(?List, ?Int)
True  if  Int represents  the number  of  elements in  List.
*/

pred1(A,R) :- length(A,L), pred2(A,R,L,0).

pred2(_,[],L,L) :-!.

pred2(A,[A|T],L,S) :- S1 is S+1, pred2(A,T,L,S1).
```

Show the Prolog response (including variable bindings, if any) to each of the following goals:

```
?- pred2(_,[],10,10).
```

true

```
?- pred2(What,[],6,5).
```

[What | [ ]]

false

(-4)

```
?- pred1([],What).
```

What = [ ]

```
?- pred1([4,3,2],What).
```

What = [[4,3,2], [4,3,2], [4,3,2]]

```
?- pred1([1,2,3,4],What).
```

What = [[1,2,3,4], [1,2,3,4], [1,2,3,4], [1,2,3,4]] 5