Sample Questions for Final Exam

Please consult sample questions for the midterm for representative questions for the early part of the course. As before we will have “knowledge questions” testing understanding of concepts or processes, “analysis questions” asking you to explain or analyze e.g. some code, and “programming questions” asking you to write a small program in C++, ML, or Prolog.

1. Explain what are row-major, column major, and row-pointer memory layouts of arrays. Why is it important that the programmer knows whether allocation is row or column major?

2. Explain the idea behind the locks and keys method for solving the problem of dangling references.

3. What does garbage collection mean. List two advantages or disadvantages of using garbage collection. Do implementations of C++ support garbage collection? Do implementations of ML support it?

4. Explain what are heap fragmentation and heap compaction. Mention an implementation or algorithm that can perform heap compaction.

5. Give pseudo-code examples for the exceptions mechanism including raising and handling exceptions. What parts/operations are needed? You may base your example on C++ or ML exceptions. Briefly explain what the various parts do.

6. Consider the following Prolog database maintained by a garage

```prolog
% owns(person,car_id).
owns(joe,car1).
owns(jim,car2).
owns(jeff,car3).
owns(joane,car4).
owns(jenny,car5).
owns(joe,car6).

% car(car_id,make,year).
car(car1,toyota,1966).
car(car2,toyota,1998).
car(car3,ford,2000).
car(car4,jeep,1999).
car(car5,mazda,1998).
car(car6,bmw,2000).

% there are two yearly times for car checkup
checktype(midyear).
checktype(earlyyear).

YOUR TASK I: Consider the rules

friendly(X,Y) :- owns(X,Z1),owns(Y,Z2), X=Y,
car(Z1,Model1,Year1), car(Z2,Model,Year2).

friendly(X,Y) :- owns(X,Z1),owns(Y,Z2), X=Y,
car(Z1,Model1,Year), car(Z2,Model2,Year).
```

1
List the first 4 answers produced by the system on query `friendly(A,B)` using the database above. For each answer give the binding for A and B.

YOUR TASK II: Checkups for cars follow the following rules: Every car needs two checkups per year (midyear and endyear) except that (1) toyotas can skip the midyear checkup in their first 5 years (2) all car makes can skip both checkups for their first 2 years. For example a ford made in 1999 can skip checkups in 1999 and 2000 but not in 2001.

Write a predicate `avoid(Car,CheckUp,Year)` which succeeds if CAR can skip CheckUp given that the current year is Year. For your predicate you may assume that Year is always an input (a number) but Car and CheckUp may be input or output. Here are some examples:

?- `avoid(Car,CheckUp,2001).`
Car = car2
CheckUp = midyear

?- `avoid(car6, endyear, 2001).`
Yes

?- `avoid(car4, midyear, 2001).`
No

Note: The order in which different outputs are produced is not important. So if your code is correct but outputs something different as the first answer to the first question above that’s fine.

7. Consult the following Prolog procedure

```prolog
% myS(+L,-N)
myS([], 0).
myS([H|T], N) :- H > 0, myS(T, NT), N is NT + H.
myS([H|T], N) :- H =< 0, myS(T, N).
```

What does the procedure compute? Draw the search tree (as we did in class) for running the query `myS([3,-5,7], N)` until the first output is produced. Make sure to annotate the tree with binding for variables. In addition, mark potential backtracking points in the tree.

8. The following ML function computes the dot-product of two lists of numbers of the same length. The dot product is defined as follows

\[ \text{dotProd}(x_1, \ldots, x_n, y_1, \ldots, y_n) = \sum_i x_i \times y_i \]

```ml
fun dotProd([], []) = 0
| dotProd([h1::t1,h2::t2]) = h1*h2 + dotProd(t1,t2);
```

val a = dotProd([1,2,3],[1,2,3]); (* returns 14=1+4+9 *)
val b = dotProd([1,2,3],[1,2,1]); (* returns 8=1+4+3 *)
val c = dotProd([1,2,3,~2],[1,2,1,1]); (* returns 6=1+4+3-2 *)

The function above is not tail-recursive. Modify the implementation to be tail-recursive by introducing an accumulator. Hide the implementation from the user by using a let expression that gives the user the same interface as above.

9. Study the following code and (1) give the type of the function f and briefly explain how you inferred the type, (2) draw a picture illustrating the computation of foldl in the computation of a and give the final result. (The function foldl is included for your convenience - it is identical to the one covered in class).
fun foldl f e [] = e
| foldl f e (h::t) =
  foldl f (f(h,e)) t;

fun f L = foldl (op -) 100 L;

val a = f [10, 3, 1];