Sample Questions for Final Exam

The following questions are taken from a previous exam in comp80. Please consult sample questions for the midterm for additional representative questions for the early part of the course. The exam will include questions on both parts. 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. Consider the following C++ code?

```cpp
class midterm {
    public:
        static int points;
        int questions;
        void setq(int q) {questions=q;}
        void addpoints(int p) {points +=p;}
    }

    int midterm::points=0;

    int main() {
        midterm zeus,hera;
        zeus.setq(1);
        zeus.addpoints(10);
        hera.setq(5);
        hera.addpoints(60);
        cout << zeus.questions << " " << zeus.points << \
             "n;";
        cout << hera.questions << " " << hera.points << "n;"
    }
```

2. Recall that compilers maintain a symbol table for resolving what names in the program refer to. The question is concerned with symbol tables for static scope. Explain (1) what information the compiler needs to keep in the symbol table, (2) what information is needed to resolve what a name reference (e.g. the name x which appears in a particular location in the program) actually refers to.

3. Explain what dangling references are. Give an example program fragment that creates a dangling reference.

4. Explain the idea behind the tombstones method for solving the problem of dangling references (3-4 sentences).

5. Explain what call by name means in the context of passing arguments to functions. What construct in C++ can give a similar effect to call by name?

6. Explain how the exceptions mechanism can be supported by a compiler. How is an appropriate handler found in run time? What is the cost in terms of run time of the program? [It is sufficient to explain and refer to one possible algorithm/implementation]
7. Consult the following Prolog code:

```
tall(a).
wide(a).
tall(b).
tall(b).
tall(c).
narrow(c).

pair(X,Y) :- tall(X), narrow(X), tall(Y).
pair(X,Y) :- wide(X), narrow(Z), tall(Y), \+ X=Y.
```

(1) List in order all the answers produced by the system on query `pair(A,B)`. For each answer give the binding for A and B.
(2) Draw the search tree (as we did in class) for running the query `pair(A,B)` until the second output is produced. Make sure to annotate the tree with binding for variables. In addition, mark potential backtracking points in the tree.

8. Study the following code and (1) give the type of the function `f` (2) briefly explain how you inferred the type (3) give the value of `a` and explain how you computed it. (The function `map` is included for your convenience - it is identical to the one covered in class).

```
fun map f [] = []
  | map f (h::t) = (f h) :: map f t;

fun f g L = map (fn x => g(x)+1) L;

fun h x = 2*x;

val a = f h [1,2,3];
```

9. Write a recursive ML function that computes the number of lower case letters in a list of characters. The implementation should use an accumulator but this should be hidden from the user (at the top level) by using a let or local expression. You can get partial credit for a correct implementation that does not use an accumulator.

You may use the following function as a subroutine.

```
fun isLower x = (x >= #"a") andalso (x <= #"z");
```

The following gives the types of the functions and illustrates the behavior:

- use "...";

```
val isLower = fn : char -> bool
val count = fn : char list -> int

- val a = count ["A", "b", "c"];  
val a = 2 : int

- val b = count ["A", "b", "3", "c", "r", "D", "a", "z"];  
val b = 5 : int
```
10. Write a Prolog procedure delEven(+List1,-List2) that takes a list List1 as input and deletes all elements in even positions in List1, producing a list List2 with the elements from odd positions, as illustrated by the following examples:

?- delEven([a,b,c,d,e],L).
   L = [a, c, e]
   Yes

?- delEven([1,a,2,b,3,c],L).
   L = [1, 2, 3]
   Yes

?- delEven([a,b],L).
   L = [a]
   Yes

?- delEven([a],L).
   L = [a]
   Yes

?- delEven([],L).
   L = []
   Yes