# CSE 341 Final Exam

(75 points total)
Winter 1999

**Name:** ______________________________

1. (5 points) Write a Scheme function `clone` that creates a complete copy of the tree structure that is its argument. Subsequent changes to either the original tree or the clone should not affect the other. Example:

```
(clone '((a b) c (d e) ()))          => ((a b) c (d e) ())
```

```
(define (clone tree)
   (if (pair? tree)
       (cons (clone (car tree)) (clone (cdr tree)))
       tree
   )
)
```

2. (4 points) Suppose we try to define binary trees and linked lists in Miranda using the following definitions.

```
tree * ::= Empty | Node * (tree *) (tree *)
list * ::= Empty | Node * (list *)
```

This pair of definitions is not legal in Miranda. Why does Miranda reject this and why is it necessary that Miranda reject this?

- Miranda regects it because the constructors `Empty` and `Node` must have unique names.
- Miranda *must* reject this because it must be able to determine the type of every object. If it allowed two constructors with the same name, then it would be unclear which one was being refered to. It must be able to answer the question, "What type is `Empty`?"

3. (4 points) In the homework we implemented an evaluator for a subset of Scheme using the normal call-by-value rules for function calls. Suppose we wanted to change the evaluator to implement lazy evaluation instead of call-by-value. What changes would be needed? You don't need to describe the code changes needed in detail. Instead, identify the parts of the evaluator that would need to be modified and briefly describe how they would have to be changed.

   - Instead of evaluating a function's arguments first and then applying the function (in `my_eval`), we must instead package the arguments up into an object that can be evaluated on demand later. We then pass this function a list of these wrapped arguments.
   - We must also add to `my_eval` the ability to get the value out of one of these wrapped arguments. Thus, `my_eval <wrapped-arg>` should return the value of the argument.

4. (6 points) Given the following predicate definitions (`length` and `sum` are straight from the homework):

```
length([],0).
length([_|Xs], L+1) :- L >= 0, length(Xs,L).

sum([],0).
sum([X|Xs],X + S) :- sum(Xs,S).

foo([X|Xs]) :- sum([X|Xs],S), length([X|Xs],S).
```

   What are the results of evaluating the following goals?

| | |
|---|---|
| `foo([1,2,X]).` | `X = 0`<br>`*** Yes` |
| `foo([-2,10,0,X,Y]).` | `X + Y = -3`<br>`*** Yes` |
| `foo([]).` | `*** No` |

5. (5 points) Write rules for a CLP(R) predicate `same` that is satisfied if all of the elements of its list argument are identical. i.e., the following are satisfied

```
same([]).
same([1]).
same([cat,cat,cat]).
```

and the following are not

```
same([dog,cat]).
same([1,1,1,1,0,1]).
```

```
same([]).
same([X]).
same([X,Y|Xs]) :- X=Y, same([Y|Xs]).
```

6. (5 points) Recall from lecture and homework the predicates `member` and `pairwise_leq`:

```
member(X,[X|Xs]).
member(X,[Y|Ys]) :- member(X,Ys).

pairwise_leq([],[]).
pairwise_leq([X|Xs],[Y|Ys]) :- X <= Y, pairwise_leq(Xs,Ys).
```

Now consider the following definition

```
squid(List,X) :- pairwise_leq(List,[X|Xs]), same([X|Xs]),
                 member(X,List).
```

(a) What is the result of evaluating the constraint

```
squid([12,42,17],Answer).
```

```
Answer = 42

*** Yes
```

(b) Briefly describe the values for `List` and `X` that satisfy the predicate `squid` (i.e., what does it do?). What would be a better name for this predicate?

The predicate `squid(List,X)` is satisfied when `X` is the maximum element of list `List`. It would be better named `listmax`.

7. (14 points) Given the following Miranda function definitions

```
muddle x (x:xs) = True
muddle y z = False

odd x = x mod 2 = 1

remove x [] = []
remove x (x:as) = as
remove x (a:as) = a : remove x as

my_filter f [] = []
my_filter f (x:xs) = x : my_filter f xs, if f x
                   = my_filter f xs, otherwise
```

What is the result of evaluating the following Miranda expressions? (If there is a compile-time type error, or a runtime error, or a non-terminating computation, say so). If the expression is followed by `::`, then give the type of the result instead of the value.

| | |
|---|---|
| `muddle ::` | `* -> [*] -> bool` |
| `remove ::` | `* -> [*] -> [*]` |
| `muddle 17 ::` | `[num] -> bool` |
| `muddle 12 [10..15]` | `False` |
| `my_filter odd [1..8]` | `[1,3,5,7]` |
| `my_filter (muddle 17) ::` | `[[num]] -> [[num]]` |
| `my_filter (> 0) ::` | `[num] -> [num]` |

8. (8 points) The following are Miranda definitions of functions `flatten` and `sumtree` that work on a binary tree:

```
tree * ::= Empty | Node * (tree *) (tree *)

flatten Empty = []
flatten (Node n left right) = n : flatten left ++ flatten right

sumtree Empty = 0
sumtree (Node n left right) = n + sumtree left + sumtree right
```

(a) The functions `flatten` and `sumtree` have the same basic structure, but use different functions and values to calculate their result. Define a higher-order function `treefold` that abstracts this structure and can be used to implement both `sumtree` and `flatten`.

```
treefold d_op c_op id Empty = id
treefold d_op c_op id (Node n left right)
    = d_op n (c_op (treefold d_op c_op id left)
                   (treefold d_op c_op id right))
```

(b) Use currying and your `treefold` function to give new definitions for `flatten` and `sumtree`.

```
flatten = treefold (:) (++) []
sumtree = treefold (+) (+) 0
```

9. (6 points) Given the following Java classes:

```java
public class Bird
{
    public String tweet()
    {
       return "Tweet";
    }
}

public class Crow extends Bird
{
    public String tweet()
    {
       return "Caw";
    }
}

public class Main
{
    public static void birdChatter(Bird b)
    {
       System.out.println("A Bird says: " + b.tweet());
    }

    public static void birdChatter(Crow c)
    {
       System.out.println("A Crow says: " + c.tweet());
    }

    public static void main(String [] args)
    {
       Bird b1 = new Bird();
       Bird b2 = new Crow();

       Crow c1 = new Crow();

       birdChatter(b1);
       birdChatter(b2);
       birdChatter(c1);
    }
}
```

What is printed out when `main` is run?

```
A Bird says: Tweet
A Bird says: Caw
A Crow says: Caw
```

10. (4 points) In an interactive Scheme session we can define functions on the fly, i.e., type in `(define (f x) ...)` to the top-level Scheme prompt. In an interactive Miranda session we can't do this. Why not?

    Defining a new function with `(define (f x) ...)` adds a new function to the symbol table in Scheme. If we were to do this in Miranda, this would be a side-effect. Since Miranda does not allow side-effects it cannot allow the user to define new functions in the global environment on the fly.

11. (4 points) The Miranda type system allows polymorphic types that contain type variables such as `*`, `[*]`, `*->**`, `*->bool`, and so forth. The claim was made that this type system can't be used to express the typing rules in object-oriented languages like Java. What rule(s) of the Java type system can't be written using Miranda-style polymorphism?

    Miranda does not have the ability to express the *subclass* type rule. The subclass type relationship allows Java to have variables or parameters to functions that can contain an instance of any subclass of a given class. In Miranda, either a function takes an argument of a specific type (like `num`) or it takes in anything (`*`).