CSE 341 Final Exam

(100 points total)

Autumn 1999

Name: Solutions

1. (3 points) A friend of yours is tired of always having to cast objects when using Java collections. Your friend proposes to rewrite a version of Java that is only dynamically typed. What is a disadvantage of this proposal?

   A disadvantage would be that many program errors what would have been caught by the compiler at compile time would now not be detected until runtime when that piece of code is executed. Debugging would me more difficult.

2. (12 points) Scheme vs. Miranda:

   (a) In Scheme there is a function `for-each` which works like `map` but does not accumulate a list. `for-each` is quite useful in Scheme. Why does Miranda not have a builtin equivalent of `for-each`?

   Miranda has no side-effects. Thus, performing operations that do not yield a result is pointless.

   (b) In Scheme we defined an `overload` function. Can we do the same in Miranda? Why or why not?

   Miranda is staticly typed. An object passed to a function can only be of one type. Thus, you can not define a function that chooses which of two other functions to call based on the type of the argument.

   (c) In Scheme, a function can take a variable number of arguments. Give some reasons for why Miranda might disallow this.

   Miranda is statically typed and the type of a function includes how many arguments it takes. What would the type of `+` be if `+` took 2 or 3 arguments?

   Also, Miranda is automatically curried. If `+` took 2 or 3 arguments and we passed it 2, how would we know that we didn't still want to pass it one more?

   (d) Miranda and Scheme are both functional programming languages. What is it about Miranda that allows us to operate (on parts of) infinite lists?

   Miranda is lazy.

3. (15 points) Fill out the following table, if there are multiple answers give all of them (do not fill in the entry labeled *skip*):

	C++	Scheme	Miranda	Java	CLP($\mathcal{R}$)
parameter passing	value , reference	value	lazy	value	constraint
side-effects	yes	yes	no	yes	skip
type-safe	no	yes	yes	yes	yes
garbage collected	no	yes	yes	yes	yes
typed	static	dynamic	static	static , dynamic	dynamic
scoping	static	static	static	static	none

4. (4 points) One of your research colleagues in the UW Physics department is implementing a pure version of Scheme to be used in the the real-time controller of a little publicized nuclear reactor that is under red square. This version of Scheme disallows side-effects. What are the advantages and disadvantages of using reference-counting as the only means of garbage collection?

   Advantage: Reference counting is incremental (it does not stop the program to collect garbage).

   Disadvantage: Reference counting adds a constant factor to the runtime of memory operations (Note: circular data structures are not a problem because circular data structures can not be created in a purely functional language).

5. (2 points) Is CLP($\mathcal{R}$) complete? Sound?

   CLP($\mathcal{R}$) is sound but not complete.

6. (4 points) A *palindrome* is a string that is the same forwards and backwards. For example, "rotator" and "deed". You have been provided you with the definition of `reverse(Xs,Ys)` which is true if `Xs` and `Ys` are the reverse of each other. Use this to define the unary `palindrome` predicate which is true only if the list it is passed is a palindrome.

```
palindrome(List) :- reverse(List,List).
```

7. (6 points) Side-effects are useful for all sorts of programming tasks. Lazy evaluation is also very useful. Combined, they are a bit problematic.

   (a) Why?

   In a lazy evaluated language, expressions are evaluated when their results are needed. Typically, side-effect expressions do not produce results, but even if they did, orchestrating the *need* of the result of a side-effect operation to get the output in the correct order is difficult. There is also the issue of memoization. If expression values are memoized (which is typically done in lazy languages) then a its output would only happen once even if the function was called twice.

   (b) Give some Scheme code that if lazily evaluated might not have the desired results.

```
(define x 10)
(define y (+ x 10))     ;; (+ x 10) not evaluated until 'y' needed.
(set! x 100)
y                       ;; now x + 10 evaluates to 110.
```

8. (9 points) In CLP($\mathcal{R}$) we define the following predicates:

```
anyone(1,_,_).
anyone(_,1,_).
anyone(_,_,1).
```

What are the results of evaluating (include retry behavior):

   (a) `anyone(0,0,0).`

```
1 ?- anyone(0,0,0).

*** No
```

   (b) `anyone(1,1,0).`

```
2 ?- anyone(1,1,0).

*** Retry? y

*** Retry? y

*** No
```

   (c) `anyone(X,Y,0), X + Y = 0, Y < 0.`

```
3 ?- anyone(X,Y,0), X + Y = 0, Y < 0.

Y = -1
X = 1

*** Retry? y

*** No
```

9. (5 points) And now, a little Miranda puzzle:

   (a) What is `mystery`?

   ```
   mystery = 0 : 1 : map2 (+) mystery (tl mystery)
   ```

   The infinite list of Fibonacci numbers: `[0,1,1,2,3,5,8,...]`.

   (b) How many times is `+` evaluated when evaluating `mystery ! 1`?

   None.

10. (4 points) Many languages include primitive arrays for storing data. Arrays are characterized by having $O(1)$ cost for *lookup* and *set* operations (`x = array[i]` and `array[i] = x`). Miranda does not have an array type. Why might this be the case?

    You can't implement an array with a side-effect free *set* operation that runs in $O(1)$ time (Note that it is possible to implement a *persistent array* with slower *set* operation).

11. (6 points) In Java:

    (a) When defining class `XXX` that implements interface `III`, if function `fff` in the interface are not defined then the compiler gives the error message: *Class* `XXX` *must be declared abstract. It does not define* `fff` *from interface* `III`. When we take this advice and make `XXX` abstract, this error message goes away and our code compiles. Why does Java require that you make `XXX` abstract?

    An abstract class cannot be instantiated. No class implementing `III` that does not implement the `fff` method should be able to be instantiated. Java forces this by requiring that `XXX` be declared abstract.

    (b) Why is it ok for a Java class to implement two interfaces that contain the same method?

    It is ok because no code is inherited by implementing an interface. The class will implement the particular method itself and this one implementation will fulfill both interface requirements.

12. (20 points) Given the following Miranda expressions,

```
append3 a b c = a ++ b ++ c
bar a b c = a:b:c
my_const y x = x

my_map f [] = []
my_map f (a:as) = f a : my_map f as
```

What are the results of the following. If there are errors say so.

(a) `(append3 [0..] [0..] [0..]) ! 10` => 10

(b) `append3 ::` => `[*]->[*]->[*]->[*]`

(c) `append3 [] ::` => `[*]->[*]->[*]`

(d) `append3 [1] [2]::` => `[num]->[num]`

(e) `bar ::` => `*->*->[*]->[*]`

(f) `bar "ab" "bc" ::` => `[[char]]->[[char]]`

(g) `my_const (1/0) 10` => 10

(h) `my_const 10 (1/0) ::` => num

(i) `my_map ::` => `(*->**)->[*]->[**]`

(j) `my_map bar ::` => `[*]->[*->[*]->[*]]`

13. (9 points) Consider the following program written in an Algol like language.

```
begin
   integer i;
   array a[1..2] of integer;
   procedure fly(j,k: integer);
   begin
      print(i,j);
      j = 0;
      k = 2;
      print(i,j);
   end

   a[1] := 10;
   a[2] := 20;
   i := 1;
   fly(a[i],i);
   print(a);
end;
```

What is the result when arguments are passed

(a) by value?

```
1   10
1   0
10  20
```

(b) by name?

```
1  10
2  20
0  20
```

(c) by reference?

```
1  10
2  0
0  20
```

14. (1 point) CLP($\mathcal{R}$) has a builtin zero argument predicate `clpr` which the manual describes as:

```
clpr
     Check if we're in heaven.
```

What do you believe CLP($\mathcal{R}$) does on the goal `clpr`?

```
1 ?- clpr.

*** Yes
```