PL Final Exam

1 **Briefly** answer the following questions. (24%)

a) What are the pros and cons of typed languages?

b) What are the two conditions for a language to be implemented without a runtime stack?

c) Consider the following two ML expressions

`let val x=`*exp* **`in`** *body* **`end;`**

`(fn x=>` *body***`)`** *exp***`;`**

Are they always equivalent? Why or why not?

d) What are the two kinds of overloading?

Besides overloading, what are the remaining two kinds of polymorphisms?

e) The word ***thunk*** has two meanings mentioned in class. What are the two meanings?

f) What is the result of executing the following Perl program? Why?

```
sub A { z: return z; }   # return the label z
sub B { goto (A); }      # goto the label returned by the call to A
B;                       # N.B. Not  goto A; since A isn't a label
z: print 2;
```

2 Fill in the following blanks. (10%)

If __a)__, then call-by-name and call-by-need yield the same result.

If __b)__, then call-by-value and call-by-reference yield the same result.

If __c)__, then call-by-name and call-by-reference yield the same result.

If __d)__ and __e)__, then call-by-value-result and call-by-reference yield the same result.

3 Consider the following C++ program

```
int gcd(int a,int b)
{
    if (b==0) return a;
    else return gcd(b,a%b);
}
int main() { cout << gcd(3,2); }
```

a) Draw the contents of the runtime stack at the point when the recursion reaches its end, i.e. when the boundary condition **`b==0`** becomes true, assuming that the C++ compiler does NOT perform tail-recursive optimization. (4%)

Be sure to indicate the values of parameters, instruction pointers, and dynamic pointers.

b) Repeat a), but this time assumes that the program is compiled by a C++ compiler that does tail-recursive optimization. (4%)

c) Tail-recursively optimize the function **`gcd`** by yourself. (A goto version suffices.) (4%)

4 Infer the type of the following lambda expression (6%)

λf.λa.λb.λc.c (f a) (f b)

5 Given the following Prolog relation

```
append([],Ys,Ys).
append([X|Xs],Ys,[X|Zs]) :- append(Xs,Ys,Zs).
```

a) Draw the search tree for the goal

```
?- append(Xs,[6|Ys],[Z,6]).
```

Write down the solution of the goal alongside each success node of the tree. (6%)

b) Draw the underlying term structures at the point when the solution **Xs=[]**, **Ys=[6]**, **Z=6** is found. (4%)

6 Consider the infinite sequence of even integers [0, 2, 4, 6, 8, 10, 12, …]

a) Let's define **evens** by

```
evens = [2*x|x<-[0..]]
```

or, equivalently,

```
evens = map (2*) (enumFrom 0)
```

Explain why this definition is inefficient. (2%)

b) Give an efficient definition that defines **evens** by *generation* using list comprehension. Draw the data structure that represents **evens** after the following evaluation. (4%)

```
Hugs> take 4 evens
[0,2,4,6] :: [Integer]
```

c) Redo b), but this time defines **evens** as a cyclic data structure using list comprehension. (4%)

7 Given the following Haskell functions

```
($!) f x = seq x (f x)
```

a) Draw the graph for **$!**. (3%)

b) Draw the graphs step-by-step during the reduction of (7%)

```
(\x->x+x) $! (2*3)
```

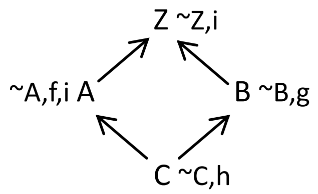
You may ignore the portion of the graphs that becomes garbage during the reduction.

Note: There are 7 steps.

8 Draw a diagram showing the contents of the Scheme runtime stack during the evaluation of the following two expressions, assuming that the Scheme compiler does NOT perform last-call optimization, including tail-recursive optimization. (8%)

```
(define f
   (letrec ((g (lambda (x) (if (= x 3) 3 (g (+ x 1))))))
      (lambda (y) (g y))))
(f (let ((h (lambda (z) z))) (h 2)))
```

9 Consider the following class lattice in which all functions are public virtual and all inheritances are public.



a) Draw a picture showing the structure of an A object. (4%)

b) Draw a picture showing the structure of a C object. (6%)