CSE 341 - Programming Languages
Midterm - Autumn 2005 - Answer Key

75 points total

1. (9 points) Evaluate each of the following. If the evaluation raises an exception, say so.

   (a) `(fn x => 42) (hd [])`
   `uncaught exception Empty`

   (b) `(fn x => 42) (fn () => hd [])`
   `42`

   (c)
```
let val x=0;
    val y=x+1
in
    let val x=5
    in
        x+y
    end
end

6
```

2. Consider a function `last` that takes a list as its argument and that returns the last element in the list. If the argument is the empty list, it raises an exception `EmptyList`.

   (a) (12 points) Write a definition of `last`.

```
exception EmptyList;
fun last [] = raise EmptyList
| last [x] = x
| last (x::xs) = last xs;
```

   (b) (3 points) What is the type of `last`?

```
'a list -> 'a
```

   (c) (3 points) Is your function `last` tail-recursive? (It doesn't need to be — you just need to be able to say whether your definition is or is not tail-recursive.)

   Yes. The recursive call to `last` is in the tail position.

3. (27 points) Suppose the following functions and exception have been defined:

```
exception UnequalLengthLists;

fun map2 (f, [], []) = []
| map2 (f, (x::xs), (y::ys)) = f(x,y) :: map2 (f,xs,ys)
| map2 _ = raise UnequalLengthLists;

fun map2curried f [] [] = []
| map2curried f (x::xs) (y::ys) = f x y :: map2curried f xs ys
| map2curried _ _ _ = raise UnequalLengthLists;

fun average x y = (x+y)/2.0;
```

What is the *value* of each of the following expressions? (If evaluating it raises an exception, say so.)

(a) `map2( (op +), [1,2,3], [10,11,12] )`
`[11,13,15]`

(b) `map2( (op +), [1,2,3], [10] )`
`exception UnequalLengthLists`

(c) `map2curried average [] []`
`[]`

What is the *type* of each of the following expressions? Some of them may give type errors — if so, say that.

(a) `average`
`real -> real -> real`

(b) `map2`
`('a * 'b -> 'c) * 'a list * 'b list -> 'c list`

(c) `map2curried`
`('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list`

(d) `map2 (average, [], [])`
`type error`

(e) `map2curried average [] []`
`real list`

(f) `map2curried average`
`real list -> real list -> real list`

4. (12 points) Consider the following code in an Algol-like language.

```
begin
integer m;
procedure clam(k: integer);
    begin
    print(k,m);
    m := m+3;
    print(k,m);
    end;
m := 5;
clam(2*m);
end;
```

Assume that the statement `print(k,m)` prints two numbers on the same line, separated by a space. (But we're not worried about the details of spacing for this question!) What is the output when m is passed using:

(a) call by value

```
10 5
10 8
```

(b) call by name

```
10 5
16 8
```

5. (9 points) In ML, a program can be ill-typed (so that it doesn't compile at all), it can execute and terminate normally, or it can fail at runtime (either by halting with an exception, going into an infinite loop, or running out of memory).

Consider a new language, DynML, that is like ML except that it is dynamically typed rather than statically typed. There are no type declarations in DynML – all type checking is done at run time when the particular expression involved is being evaluated. For example, the expression `3+"clam"` would give a runtime error if you try to evaluate it; but until it's evaluated, there is no indication that something might be wrong.

Are there any programs that have different behavior with ML and DynML? If so, give an example and describe the behavior in both ML and DynML. If not, explain why not.

Answer: Yes, there are some programs with different behavior — specifically, there are some programs that won't compile in ML due to a type error, but that will execute in DynML. (When they execute, they could give a correct answer, or fail somehow at runtime.)

Here is an example:

```
(if true then 42 else 3+"clam")
```

In ML, this doesn't compile due to a type error. In DynML, it executes successfully, and evaluates to 42.