CIS 425 : Final Exam – Winter 2016

Your Name : _____________________________
[10 points] Calculate the ML type for the following function. Show how the type is derived from the set of constraints on types using the parse tree of the expression.

fun ff f x y = if (f x y) then (f 3 y) else (f x "zero")
[10 points] Calculate the ML type for the following function. Show how the type is derived from the set of constraints on types using the parse tree of the expression.

```ml
fun gg f x y = if (f x y) then (f 3 y) else (f y "zero")
```
[10 points] This question asks about memory management in the evaluation of the following statically-scoped pseudo-ML expression.

```ml
let x = 2
  fun f y = if y > x then 5 else 0
  fun g h = let x = 1 in h 3 end
in
  let x = 0 in g f end
end;
```

1. Fill in the missing information in the following depiction of the runtime stack after the call to \( h(3) \) inside the body of \( g \).

```
<table>
<thead>
<tr>
<th>Activation Records</th>
<th>Closures</th>
<th>Compiled Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) static link ( )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>(2) static link ( )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3) g(f) static link ( )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>h</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4) h(3) static link ( )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

2. What is the value of this expression? Explain briefly how you get that value.
[10 points] Draw the run-time structure for the activation records in place after the call \( g(5) \) for the following expression:

```haskell
let
  fun f(x) = let val y = [1,2,3] in fn z => x+hd y + z end
  val g = f(3)
in g 5
end
```
[5 points] Write a program which would loop if evaluated in call-by-value but would terminate under call-by-name (or lazy evaluation).
[10 points] Suppose we have a simple language:

\[ e ::= x \mid \text{fn} \ x \Rightarrow e \mid e \ e \]

whose semantics is given in terms of the following rules:

\[
\begin{align*}
\text{env} \vdash \text{lookup}(x, \text{env}) \Rightarrow v \\
\text{---------------------------------} \\
\text{env} \vdash x \Rightarrow v \\
\text{env} \vdash (\text{fn} \ x \Rightarrow e) \Rightarrow (\text{fn} \ x \Rightarrow e) \\
\text{env} \vdash e_1 \Rightarrow (\text{fn} \ x \Rightarrow e) \quad \text{env} \vdash e_2 \Rightarrow v_1 \quad \text{env}, (x,v_1) \vdash e \Rightarrow v \\
\text{-------------------------------------------------} \\
\text{env} \vdash e_1 \ e_2 \Rightarrow v
\end{align*}
\]

1) Explain briefly why this semantics implements call-by-value and dynamic scope. 2) Modify the above rules to implements call-by-value and static scope.
[10 points] Suppose we have a simple language:

\[ e ::= x \mid \text{fn } x \Rightarrow e \mid e \ e \]

whose semantics is given in terms of the following rules:

\[
\begin{align*}
\text{env} \vdash \text{lookup}(x, \text{env}) \Rightarrow v \\
\text{env} \vdash x \Rightarrow v \\
\text{env} \vdash (\text{fn } x \Rightarrow e) \Rightarrow (\text{fn } x \Rightarrow e) \\
\text{env} \vdash e_1 \Rightarrow (\text{fn } x \Rightarrow e) \text{ env} \vdash e_2 \Rightarrow v_1 \text{ env}, (x, v_1) \vdash e \Rightarrow v \\
\text{env} \vdash e_1 e_2 \Rightarrow v
\end{align*}
\]

This semantics implements call-by-value and dynamic scope. Modify the above rules to implement call-by-name and static scope.
[10 points] Write a ML function in CPS (Continuation Passing Style) which multiplies all the elements of a list. You can return 1 if the list is empty.
Write an efficient ML function which multiplies all the elements of a list. You can return 1 if the list is empty. The efficiency should come from the fact that it returns 0 as soon as one of the elements of the list is 0.
[10 points] Consider the following function $f$ written in pseudo-ML:

$$f \ [] \ y = true$$
$$f \ (x::xs) \ y = if \ x < y \ then \ true \ else \ f \ xs \ y \ ;$$

1) What is the type of function $f$ in ML? 2) Is function $f$ polymorphic in ML?

Would the call

$$f \ [true,false] \ true$$

give you an error in ML? If yes, would the error be raised at compile-time or run-time? If not, what is the result of the function call?
[10 points] Consider now the same function in pseudo-Haskell:

\[
\begin{align*}
  f \; [] & \quad y = \text{true} \\
  f \; (x::xs) & \quad y = \text{if } x < y \text{ then } \text{true} \text{ else } f \; xs \; y \\
\end{align*}
\]

What is the type of function \( f \) in Haskell? Is function \( f \) polymorphic in Haskell? If not, how would you classify function \( f \) in Haskell?

Would the call

\[
f \; [\text{True},\text{False}] \; \text{True}
\]

give you an error in Haskell?
[5 points] Summarize briefly the main difference between ML and Haskell in typing function $f$. 
[5 points] Write a Haskell function `forever` which takes an IO action as parameter and repeats that action forever.

```haskell
forever :: IO () -> IO ()
forever a = ?
```

Example use:

```
Main> forever (putChar 'a')
```


[5 points] Write a Haskell function `each` which evaluates each action in a list and collects the results in a list.

```haskell
each :: [IO a] -> IO [a]
each [] = ?
each (a:as) = ?
```

Example use:

```
Main> each [getChar,getChar]
```

will return a list of the next two characters in standard input.
Suppose you have the following ML function definition:

```ml
fun f x y = if x = 0 then 0 else y + 1;
```

What is the result of the following invocation in ML:

```ml
fun loop x = loop x ;

f 0 (loop 9);
```

Change function \( f \) and its invocation so to return 0 when the first actual parameter is 0.