**Background**

In this assignment you will implement versions of LFP with

(i) call-by-value evaluation and dynamic scoping,
(ii) call-by-name evaluation and lexical scoping, and
(iii) call-by-name evaluation and dynamic scoping.

The version of LFP we’ll be working with has two new commands:

1. The `return` command works pretty much as it does in C.

2. The `print` command does what you expect — sort of.

\[
\begin{align*}
\text{Print:} & \quad \{ \rho \vdash (e_1, s_1) \downarrow (v_1, s_{i+1}) \} = 1, \ldots, n \\
& \quad \{ \langle \text{print} (e_1, \ldots, e_n), s_1 \rangle \downarrow (\text{skip}, s_{n+1}) \} \quad (\text{the values of } e_1, \ldots, e_n \text{ are printed})
\end{align*}
\]

The `print` command is implemented on the cheap (i.e., via Haskell’s `trace` function) and so `print`’s can show up in the output in very odd places and in very odd orders because of Haskell’s laziness. (Try evaluating `et0a` and `et0b` to see what I mean.)

The big-step operational semantics rules for (i) Call-by-value, lexical-scoping, (ii) Call-by-value, dynamic-scoping, (iii) Call-by-name, lexical-scoping, and (iv) Call-by-name, dynamic-scoping are given below. I’ve put closures in boxes. Also, the hat in \( \hat{\rho} \) is just a distinctive decoration.
Problem 4

For each of the following LFP programs, figure out by hand what the program prints or returns.

(a) Consider:

\[ \text{let } x = 1 \text{ in let } f = (\lambda z. (x + z)) \text{ in let } x = 100 \text{ in } (f 10) \]

(i) (4 points) What is returned under call-by-value/lexical scoping?

(ii) (4 points) What is returned under call-by-value/dynamic scoping?

(b) Consider:

\[ \text{let } x = 100 \text{ in let } f = (\lambda u. (u * x)) \text{ in let } g = (\lambda x. (f 2)) \text{ in print } ((f 10), (g 6)) \]

(i) (4 points) What is printed under call-by-value/lexical scoping?

(ii) (4 points) What is printed under call-by-value/dynamic scoping?

An answer for 4(b.i)

<table>
<thead>
<tr>
<th>Part i</th>
<th>ENVIRONMENT</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rho_0 ):</td>
<td>( \emptyset )</td>
<td>let ( x = 100 ) in ...</td>
</tr>
<tr>
<td>( \rho_1 ):</td>
<td>( x \mapsto 100 )</td>
<td>let ( f = ) ...</td>
</tr>
<tr>
<td>( \rho_2 ):</td>
<td>( f \mapsto \lambda z. (x + z) ) ( \rho_1 )</td>
<td>let ( x = 100 ) in ...</td>
</tr>
<tr>
<td>( \rho_3 ):</td>
<td>( x \mapsto 100 )</td>
<td>( (f 10) )</td>
</tr>
<tr>
<td>( \rho_4 ):</td>
<td>( z \mapsto 10 ) ( \rho_1 )</td>
<td>( x + z = \rho_4(x) + \rho_4(z) = 100 + 10 = 110 )</td>
</tr>
</tbody>
</table>

An answer for 4(b.ii)

<table>
<thead>
<tr>
<th>Part ii</th>
<th>ENVIRONMENT</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rho_0 ):</td>
<td>( \emptyset )</td>
<td>let ( x = 1 ) in ...</td>
</tr>
<tr>
<td>( \rho_1 ):</td>
<td>( x \mapsto 1 )</td>
<td>let ( f = ) ...</td>
</tr>
<tr>
<td>( \rho_2 ):</td>
<td>( f \mapsto \lambda z. (x + z) ) ( \rho_1 )</td>
<td>let ( x = 100 ) in ...</td>
</tr>
<tr>
<td>( \rho_3 ):</td>
<td>( x \mapsto 100 )</td>
<td>( (f 10) )</td>
</tr>
<tr>
<td>( \rho_4 ):</td>
<td>( z \mapsto 10 )</td>
<td>( x + z = \rho_4(x) + \rho_4(z) = 100 + 10 = 110 )</td>
</tr>
</tbody>
</table>

So \((110, 110)\) is printed.
### An answer for 4(b.ii)

#### Part i  
**Environment** | **Expression**
--- | ---
$\rho_0$: $\emptyset$ | let $x = 100$ in ...

$\rho_1$: $x \mapsto 100$ | let $f = \ldots$

$\rho_2$: $f \mapsto \lambda z. (x + z)$ | let $g = \ldots$

$\rho_3$: $g \mapsto \lambda x. (f 2)$ | print ((f 10), (g 6)) | $\rho_0$: $\emptyset$ | let $f = \ldots$

$\rho_1$: | let $y = (f 40)\cdots$

$\rho_2$: | $y \mapsto 40$ | $\rho_0$: $\emptyset$ | let $f = \ldots$

$\rho_3$: | $y \mapsto (f 40)\rho_1$ | let $y = (f 40)\cdots$

So $\{(110,8)\}$ is printed.

#### Part ii  
**Environment** | **Expression**
--- | ---
$\rho_0$: $\emptyset$ | let $f = \ldots$

$\rho_1$: | let $y = (f 40)\cdots$

$\rho_2$: | $y \mapsto 40$ | $\rho_0$: $\emptyset$ | let $f = \ldots$

$\rho_3$: | $y \mapsto (f 40)\rho_1$ | let $y = (f 40)\cdots$

So 130 is printed.

### An answer for 4(c)

Note that in either case, every time we call $f$ on an argument $X_1$ is increased by 10.

In part (i), we call $f$ once in the let $y = (f 40)\cdots$.

In part (ii), the let $y = (f 40)\cdots$ sets $y$’s value to be the thunk $(f 40)\rho_1$ and $f$ is not call then. However, in the two used of $y$ in $X_2 := y + 2 * y$ force two separate evaluation of $(f 40)\rho_1$; each time 40 is returned, but $X_1$ is increased by 10 twice.

#### Part i  
**Environment** | **Expression**
--- | ---
$\rho_0$: $\emptyset$ | let $f = \ldots$

$\rho_1$: | let $y = (f 40)\cdots$

$\rho_2$: | $y \mapsto 40$ | $\rho_0$: $\emptyset$ | let $f = \ldots$

$\rho_3$: | $y \mapsto (f 40)\rho_1$ | let $y = (f 40)\cdots$

So 130 is printed.

#### Part ii  
**Environment** | **Expression**
--- | ---
$\rho_0$: $\emptyset$ | let $f = \ldots$

$\rho_1$: | let $y = (f 40)\cdots$

$\rho_3$: | $y \mapsto (f 40)\rho_1$ | let $y = (f 40)\cdots$

So 140 is printed; because every time we look up the value of $y$, $(f 40)$ is evaluated and $X_{10}$ is increased by 10.

(c) Assume the location $X_1$ (or $\ell_1$ if you prefer) starts out with contents 0. Consider:

```plaintext
let $f = \lambda y. \{ X_1 := !X_1 + 10; \ return \ y \} $

in let $y = (f 40)$

in \{ $X_2 := y + 2 * y; \ return (val (X2)+ val (X1)) \} 
```

(i) **(4 points)** What is returned under call-by-value/lexical scoping?

(ii) **(4 points)** What is returned under call-by-name/lexical scoping?
(d) Assume \( X_1 \) starts out with contents 0. Consider:

\[
\text{let } f = \lambda y. \{ \ X_1 :=! X_1 + 100; \text{return } y \} \\
\text{in let } g = \lambda z. 10 \\
\text{in let } w = (g (f \, 999)) \text{ in } (w + \text{val} \, (X_1))
\]

(i) (4 points) What is returned under call-by-value/lexical scoping?
(ii) (4 points) What is returned under call-by-name/lexical scoping?

An answer for 4(d)

Note that in both cases, every time we call \( f \, k \), for some argument \( k \), argument \( X_1 \) is increased by 100 and the value of \( k \) is returned. Also in both cases, every time we call \( g \) on some value, 10 is returned; so \( w \) will end up with the value 10 in both cases.

In part (i): the call \( g (f \, 999) \) will force the argument \( f \, 999 \) to be evaluated. So, \( X_1 \) will go from 0 to 100 and 999 will be returned and not used in the body of \( g \). Hence, \( 10 + 100 = 110 \) is returned.

In part (ii): the call \( g (f \, 999) \) build the environment where

\[
z \mapsto (f \, 999)_{\rho_3}
\]

But since \( z \) is not used in the body of \( g \), \( X_1 \) stays at 0 and \( 10 + 0 = 10 \) is returned.

(e) Assume \( X_1 \) starts out with contents 0. Consider:

\[
\text{let } \text{tick} = 1 \\
\text{in let } \text{tock} = \lambda u. \{ \ X_1 :=! X_1 + \text{tick}; \text{return } !X_1 \} \\
\text{in let } \text{tick} = 100 \text{ in } (\text{tock} \, \text{tock} \, \text{tock} \, 0))
\]

(i) (2 points) What is returned under call-by-value/lexical scoping?
(ii) (2 points) What is returned under call-by-value/dynamic scoping?
(iii) (2 points) What is returned under call-by-name/lexical scoping?
(iv) (2 points) What is returned under call-by-name/dynamic scoping?

An answer for 4(e)

For all cases, whenever we call \( \text{tock} \), the value of \( X_1 \) is increased by whatever the value of \( \text{tick} \) is and the current value of \( X_1 \) is then returned.

For the lexical scoping cases, the value of \( \text{tick} \) will be 1.
For the dynamic scoping cases, the value of \( \text{tick} \) will be 100.
For the call-by-value cases, \( \text{tock} \) is called three times.
For the call-by-name cases, since \( u \) is never used in the body of \( \text{tock} \), in \( (\text{tock} \, \text{tock} \, \text{tock} \, 0)) \) the argument \( (\text{tock} \, \text{tock} \, 0) \) is never evaluated, i.e., \( \text{tock} \) is called only once.

So . . .

For part (i): The value 3 is returned (three calls to \( \text{tock} \) with \( \text{tick}=1 \)).
For part (ii): The value 1 is returned (one call to \( \text{tock} \) with \( \text{tick}=1 \)).
For part (iii): The value 300 is returned (three calls to \( \text{tock} \) with \( \text{tick}=100 \)).
For part (iv): The value 100 is returned (one call to \( \text{tock} \) with \( \text{tick}=100 \)).