Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-3641 - Lenguajes de Programación I
Septiembre-Diciembre 2010

Carnet:_____

Nombre:_____

Examen I

(25 puntos)

Pregunta 0	Pregunta 1	Pregunta 2	Total
12 puntos	6 puntos	7 puntos	25 puntos

Pregunta 0 - 12 puntos

Esta pregunta consta de seis (6) subpreguntas de selección, numeradas de 0.0 a 0.5. Cada una de las subpreguntas viene acompañada de cuatro posibles respuestas (a, b, c, d), entre las cuales sólo **una** es correcta. Ud. deberá marcar completamente y sin posibilidad de confusión la opción que considere correcta o, si desea no contestar la subpregunta, marcar completamente y sin posibilidad de confusión la última opción (e) que indica que Ud. prefiere omitir la respuesta.
Cada una de las subpreguntas tiene un valor de dos (2) puntos. Tres subpreguntas incorrectas eliminan una correcta. Las subpreguntas omitidas (marcadas en la opción e) no suman ni restan puntos.

0.0. En un lenguaje con alcance estático, en el registro de activación se almacenan apuntadores que permitan acceder a las cadenas estática y dinámica. En relación a estas cadenas siempre es cierto:

a) La cadena estática debe ser más larga que la dinámica.

b) Se puede omitir almacenar la cadena dinámica, ya que con alcance estático esta información no es necesaria.

c) **Los miembros de la cadena estática de un registro de activación son un subconjunto de los miembros de la cadena dinámica de ese registro.**

d) Ninguna de las anteriores.

e) No sabe / No contesta.

0.1. Se quiere obtener un compilador para un lenguaje de alto nivel L para una máquina específica M, que se ejecute en M y genere código para M. Actualmente sólo cuenta con un interpretador de L ejecutable en M, y un compilador para L, escrito en L, cuya salida es para M. ¿Cuál de las siguientes afirmaciones es cierta?

a) La única manera de obtener el compilador deseado es haciendo uso de la técnica conocida como *bootstrapping*.

b) Para obtener el compilador deseado basta con reescribir el interpretador en L y pasarlo como entrada al compilador que se tiene.

c) **Para obtener el compilador deseado basta con interpretar el compilador que se tiene usando como entrada el compilador escrito en L.**

d) Ninguna de las anteriores.

e) No sabe / No contesta.

0.2. Considere el siguiente programa escrito en pseudocódigo:

```
var n := -1 : int;

procedure A ( x : int; p : procedure)

    var m := -1 : int;

    procedure porX ( y : int )
        m := x * y;

    if (x <= 0)
        A (x + 1, porX);
    else
        var x := 2 : int;
        p(x-1);

    escribir(m);

procedure mostrar (z : int)
    escribir(z);

A(n, mostrar);
```

en el cual `escribir` es una subrutina que escribe en pantalla el valor pasado como argumento.

Considerando alcance estático y asociación profunda (*deep binding*), ¿cuál es el primer valor que se muestra en pantalla?

a) **-1**

b) 0

c) 1

d) Ninguno de las anteriores.

e) No sabe / No contesta.

0.3. Continúe con el programa dado en la subpregunta (0.2), pero ahora considerando alcance estático y asociación superficial (*shallow binding*), ¿cuál es el primer valor que se muestra en pantalla?

a) -1

b) 0

c) **1**

d) Ninguna de las anteriores.

e) No sabe / No contesta.

0.4. Continúe con el programa dado en la subpregunta (0.2), pero ahora considerando alcance dinámico y asociación profunda (*deep binding*), ¿cuál es el primer valor que se muestra en pantalla?

a) **-1**

b) 0

c) 1

d) Ninguna de las anteriores.

e) No sabe / No contesta.

0.5. Continúe con el programa dado en la subpregunta (0.2), pero ahora considerando alcance dinámico y asociación superficial (*shallow binding*), ¿cuál es el primer valor que se muestra en pantalla?

a) -1

b) 0

c) 1

d) **Ninguna de las anteriores.**

e) No sabe / No contesta.

Pregunta 1 - 6 puntos

El tiempo de vida de un objeto y el tiempo de vida de las asociaciones a éste no necesariamente coinciden. Más aún, el objeto al que está asociado un nombre puede variar según el alcance sea estático o dinámico.
Escriba **un** programa tal que al usar un tipo de alcance se presente un caso en el cual el tiempo de vida del objeto es mayor que el tiempo de vida de la asociación, pero que al usar el otro tipo de alcance se presente un caso en el cual el tiempo de vida de la asociación es mayor que el tiempo de vida del objeto. Explique *brevemente* por qué su programa ilustra los dos casos planteados.
Nota: escriba su programa en pseudocódigo.

Pregunta 2 - 7 puntos

En los lenguajes donde el alcance es dinámico el interpretador debe contar en todo momento con la información de los objetos cuyas asociaciones están activas. El libro de texto (Michael Scott – "Programming Language Pragmatics") comenta dos implantaciones habituales para mantener el ambiente de referencia: listas de asociación (*A-lists*) y tablas centrales de referencia.
Las tablas centrales de referencia podrían ser implantadas en Haskell fácilmente usando la siguiente estructura de datos:

```
type TCR = [ (String,[Info]) ]
```

En esta estructura el tipo `String` es utilizado, por supuesto, para los identificadores, y el tipo `Info` corresponde a la información que, según el lenguaje, interese asociar a cada identificador.
Implante la función

```
enterScope :: TCR -> [(String,Info)] -> TCR
```

de entrada a una nueva región local. Esta función recibe una tabla de símbolos y una lista de nuevas asociaciones ("bindings") locales en la región, para devolver, por supuesto, la tabla de símbolos resultante. Suponga que el lenguaje **no** permite sobrecarga de nombres.

Como el lenguaje no permite sobrecarga de nombres, se concluye que durante el análisis sintáctico y de contexto se *garantiza* que no hay más de un símbolo definido por alcance, y en consecuencia la lista de nuevas asociaciones a tiempo de **ejecución** solamente contiene símbolos diferentes. Así, para escribir la función solamente es necesario considerar dos casos:

1. No se agrega ninguna asociación en la región local, por lo tanto la Tabla Central de Referencias se mantiene idéntica.

2. Se agregan una o más asociaciones en la región local. En ese caso, cada nueva asociación puede ser totalmente nueva pues no existe en la TCR, o se trata de un nuevo valor para una asociación existente, que debe agregarse como el valor más reciente en la entrada correspondiente dentro de la TCR. Entonces, la operación es simple: para cada asociación a incorporar se recorre la TCR; si el símbolo ya está en la TCR, su nuevo valor se agrega al **principio** de la lista de valores actuales, pero si no está en la TCR, se agregará la nueva asociación al final.

```
enterScope t  [] = t
enterScope t  bs = foldl addOrReplace t bs
  where addOrReplace []          (s,i)  = [(s,[i])]
        addOrReplace ((s,ob):rs) (ns,i) =
          if s == ns then (s,i:ob) : rs
                     else (s,ob)   : addOrReplace rs (ns,i)
```

Note que en esta solución se hace uso de `foldl` para no tener que escribir la recursión explícita, que también es válida como solución. Incluso debería ser `foldl'` si uno quiere apuntar hacia la máxima eficiencia.