Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3641 - Lenguajes de Programación I
Enero-Marzo 2007

Examen I

(30 puntos)

Nombre:

Carnet:

1. **(5 puntos)** Considere cuidadosamente las siguientes cuestiones y seleccione exactamente una respuesta de las alternativas que se presentan. Cada cuestión contestada correctamente vale **un (1) punto** pero una cuestión contestada incorrectamente **resta medio punto**.

   (a) En un lenguaje con alcance dinámico

   i. Sólo se usa la pila de ejecución para almacenar variables.
   No, pues es posible utilizar el *heap* para almacenar objetos e incluso no existir la pila de ejecución sino utilizar una lista de asociación.

   ii. Todas las decisiones de asociación se toman a tiempo de ejecución.
   No, pues las asociaciones de palabras claves y de representación de datos se toman a tiempo de diseño de lenguaje, y las asociaciones entre nombres de rutinas y su código asociado a tiempo de escritura del programa.

   iii. **La cadena estática es inútil.**

   iv. No puede haber variables globales.
   No, pues puede declararse una variable en el ambiente global, y si no se vuelve a utilizar ese mismo nombre en ninguna otra parte, la variable se comporta, en efecto, como si fuese global.

   (b) Si un objeto es almacenado estáticamente entonces

   i. No puede ser una variable local de una subrutina.
   No, pues cuando dentro de una subrutina se declara una variable estática (con `static` en C o Java) ésta sólo es alcanzable desde la subrutina, pero es almacenada en el espacio estático.

   ii. **Tiene un tamaño fijo durante la ejecución del programa.**

   iii. Sólo es accesible si el lenguaje tiene alcance estático.
   No, pues si el lenguaje tuviese alcance dinámico, también sería alcanzable si en todo el programa no se declara ninguna otra variable con el mismo nombre.

   iv. Es accesible durante toda la ejecución del programa.
   No, pues puede ser cubierta por una redeclaración del mismo nombre dentro de una subrutina.

   (c) Considere la expresión `((b++ + 41) || 3/b)` de algún lenguaje de programación en el cual || es la disyunción booleana. ¿Cuál es el resultado de evaluar esa expresión cuando $b = 0$?

   i. La expresión vale 41 y la variable b se incrementa.

   ii. La expresión vale 42 y la variable b se incrementa.

   iii. La subexpresión $3/b$ ocasiona una división entre cero.

   iv. **No hay suficiente información.**
   Para poder decidir el valor de la expresión sería necesario saber si el lenguaje evalúa expresiones de izquierda a derecha, de derecha a izquierda o si queda indefinido el orden; sería necesario saber si los operadores booleanos usan o no evaluación con cortocircuito; y sería necesario saber si hay diferencia o no entre operaciones booleanas y aritméticas.

(d) Las variables locales utilizadas en el cuerpo de un iterador verdadero se almacenan

i. En el *heap*.

ii. En una estructura similar a aquellas utilizadas para las clausuras.

iii. En el área estática.

iv. **En una pila de ejecución separada.**
Libro de Texto, página 279. Cuando se habla de un hilo de ejecución diferente, se implica una pila de ejecución separada.

(e) En un lenguaje que usa el modelo de referencias

i. Las variables pueden ser *l-values* o *r-values*.
No, pues en un lenguaje que usa el modelo de referencias *todas* las variables son *l-values*.

ii. **Según el contexto en que ocurre una variable, debe usarse la referencia o seguir la referencia.**
Libro de Texto, página 240.

iii. Los programas son menos eficientes que en lenguajes que usan el modelo de valores.
No, pues si se provee de-referenciación implícita detectada a tiempo de compilación, el rendimiento final es equivalente.

iv. Es imposible copiar valores porque se consideran inmutables.
No, pues lo único inmutable son los números o constantes, pero cualquier otro valor es susceptible de copia profunda.

2. **(5 puntos)** Considere la siguiente función en Haskell que recibe una lista de números y retorna una tupla cuyos elementos indican respectivamente cuántos números pares y cuántos números impares contiene la lista

```
paridad [] = (0,0)
paridad (a:x) = if ((a `mod` 2) == 0) then (p+1,i)
                                      else (p,i+1)
                where (p,i) = paridad x
```

Esto es, si se evalúa

```
paridad [1,2,3,4,5] -> (2,3)
paridad [0,2,3,5,7,7,7,2] -> (3,5)
```

Reescriba la función `paridad` para que siga teniendo un argumento único, pero utilice recursión de cola.

Respuesta: la función presentada no utiliza recursión de cola pues después de la llamada recursiva a `paridad x`, se utilizan los valores retornados `p` e `i` en operaciones aritméticas para producir el resultado final de la invocación. De manera que es necesario producir una función que pueda producir su resultado final **sin** realizar ningún cómputo después de la llamada recursiva.

La solución más simple sería

```
paridad lista = ayuda lista 0 0
ayuda [] p i = (p,i)
ayuda (a:x) p i = if ((a `mod` 2) == 0)
                  then (ayuda x (p+1) i)
                  else (ayuda x p (i+1))
```

Nótese que la estructura de la solución es idéntica al ejemplo hecho en la clase del 2007-01-29, lámina 11. El elemento clave para tener la respuesta correcta es que no haya **ninguna** operación después de hacer la llamada recursiva.

3. **(9 puntos)** Suponga que Java se ha extendido con iteradores del estilo de Python o Ruby. Estos son declarados como una rutina cualquiera salvo que debe agregarse la palabra clave `iterator` antes del nombre de la subrutina. Al igual que Python o Ruby, la producción de valores se indica mediante la instrucción `yield`.

   Se desea que utilice esa nueva facilidad para programar un conjunto de iteradores para los elementos de un árbol binario. Suponga que el esqueleto de la clase árbol es como sigue:

```
class BinTree {
   private Node root;
   ...
}
```

   donde la clase para los nodos del árbol, que debe ser considerada local y privada a la clase árbol, solamente contiene los atributos de información de un nodo

```
class Node {
  Object contenido;
  BinTree left, right;
}
```

   Programe entonces los generadores de elementos en pre, post e in-orden, como sendas subrutinas locales a la clase árbol.

   La clase para los nodos del árbol (`Node`) es local y privada a la clase del árbol (`BinTree`), por tanto los métodos dentro de ésta última tienen acceso a los atributos de la primera, i.e. no necesito asumir la existencia de métodos especiales para acceder a ellos. Los iteradores operan sobre una instancia de árbol, de modo que los tres iteradores serían

```
Object iterator preorder() {
  if (this.root) {
    Object c;
    yield this.root.contenido;
    while (c = this.root.left.preorder())  { yield c; }
    while (c = this.root.right.preorder()) { yield c; }
  }
}
Object iterator postorder() {
  if (this.root) {
    Object c;
    while (c = this.root.left.postorder())  { yield c; }
    while (c = this.root.right.postorder()) { yield c; }
    yield this.root.contenido;
  }
}
Object iterator inorder() {
  if (this.root) {
    Object c;
    while (c = this.root.left.inorder())  { yield c; }
    yield this.root.contenido;
    while (c = this.root.right.inorder()) { yield c; }
  }
}
```

   Nótese que la estructura de la solución es idéntica al ejemplo hecho en la clase del 2007-01-29, lámina 14. El elemento clave para tener la solución correcta es notar que los iteradores retornan un `Object` y que, tal como se explicó en clase, no es natural (ni eficiente) hacerlos directamente recursivos pues la instrucción `yield` no es exactamente equivalente a un `return` así que no puede retornar una invocación recursiva directamente.

4. Considere el siguiente programa escrito en pseudocódigo

```
int x = 42;
int y = 69;
int b = 17;
proc add
  x := x + y
proc bar( X : proc)
  int x := b;
  X()
proc foo( a : int, b : int)
  int y := a;
  bar(add)
main
  foo(x,9)
  print(x)
end;
```

   (a) **(4 puntos)** Asumiendo que el lenguaje utiliza alcance **estático**, ¿qué imprime el programa si el lenguaje utiliza alcance estático?

   (b) **(7 puntos)** Asumiendo que el lenguaje utiliza alcance **dinámico** y ***deep binding***, ¿qué imprime el programa?

   En **ambos** casos asuma que las clausuras se construyen en el momento en que las funciones son **pasadas como parámetros.**

   Nota: si solamente muestra los resultados (aunque sean correctos) no obtendrá puntos; es **imprescindible** exhibir los contenidos de la pila de ejecución hasta el momento en que se invoca la función `add` incluyendo las cadenas dinámica y estática así como las clausuras construidas.