Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3641 - Lenguajes de Programación I
Enero-Marzo 2010

Examen I

(30 puntos)

Nombre:

Carnet:

1. **(24 puntos)** Considere cuidadosamente las siguientes cuestiones y seleccione exactamente una respuesta de las alternativas que se presentan. Cada cuestión contestada correctamente **suma dos (2) puntos** pero una cuestión contestada incorrectamente **resta medio punto**. Si Ud. selecciona más de una opción, la pregunta se considera contestada **incorrectamente**.

   (a) En un lenguaje con alcance estático **siempre** es cierto

      i. Si las funciones son de segunda o primera clase, entonces tiene clausuras.
      ii. ***La cadena estática es indispensable cuando hay rutinas anidadas.***
      iii. El acceso a objetos no locales debe simularse a través de la cadena dinámica.
      iv. La cadena estática es indispensable cuando hay clausuras.

   (b) En un lenguaje que utiliza **exclusivamente** el modelo de referencia

      i. El acceso a los valores concretos siempre es más lento que si usara modelo de valor, pues es necesario un acceso indirecto cada vez.
      ii. Todos los objetos se almacenan en el *heap*.
      iii. ***El problema de* boxing *y* unboxing *desaparece.***
      iv. El acceso indirecto a los objetos es generado automáticamente cuando éstos son empleados como *l-values*.

   (c) La mayoría de los lenguajes de programación no especifica orden de evaluación para los operandos y argumentos. ¿Cuál de las siguientes razones justifica tal decisión?

      i. Si hay efectos de borde en las operaciones, es necesario preservarlas en el orden en que las expresa el programador.
      ii. Si el lenguaje utiliza evaluación ambiciosa, el orden de evaluación vendrá dado de manera natural por la necesidad de obtener los valores.
      iii. Si el lenguaje utiliza evaluación perezosa, el orden de evaluación vendrá dado de manera natural por la necesidad de obtener los valores.
      iv. ***Si el compilador es libre de reordenar las operaciones, puede aplicar técnicas de optimización más agresivas que conduzcan a código ejecutable más eficiente.***

   (d) La ventaja de contar con una instrucción de selección condicional del estilo `switch/case` es que éstas se pueden implementar mediante tablas de salto en lugar de la correspondiente secuencia de `if` anidados. Sobre estas tablas de salto, podemos decir que:

      i. ***Conviene que se implementen como tablas de* hash *cuando el conjunto de valores para las etiquetas de la instrucción es grande y se usan pocos en la instrucción.***
      ii. Conviene que se implementen mediante búsqueda binaria únicamente cuando el conjunto de valores para las etiquetas de la instrucción es grande y se usan pocos en la instrucción.
      iii. Conviene que se implementen como tablas de salto directas cuando el conjunto de valores para las etiquetas es grande.
      iv. Solamente tienen sentido si el procesador cuenta con un mecanismo de direccionamiento indirecto.

(e) Considere el siguiente programa escrito en pseudocódigo

```
int x := 20
int y := 11
procedure sumar ()
  x := x + y
endp
procedure tercero (P : procedure)
  int y := 31
  P()
endp
procedure segundo ()
  int y := 2
  tercero(sumar)
endp
procedure primero(n : int, Q : procedure)
  procedure asignar()
    y := n
    segundo()
  endp
  if (n < 10)
    primero(n+7,asignar)
  else
    Q()
  endif
endp
begin
  primero(5,sumar)
  write(x)
end
```

Indique cuál es la salida de este programa suponiendo que el pseudolenguaje utiliza alcance estático y asociación profunda (*deep binding*).

i. 42
ii. *25*
iii. 32
iv. Ninguna de las anteriores.

(f) Continuando con la pregunta (1.e), indique cuál es la salida del programa suponiendo que el programa utiliza alcance estático y asociación superficial (*shallow binding*).

i. 42
ii. 25
iii. *32*
iv. Ninguna de las anteriores

(g) Continuando con la pregunta (1.e), indique cuál es la salida del programa suponiendo que el programa utiliza alcance dinámico y asociación profunda (*deep binding*) de ambientes no locales para subprogramas pasados como parámetro.

i. 42
ii. 25
iii. 32
iv. ***Ninguna de las anteriores (22).***

(h) Continuando con la pregunta (1.e), indique cuál es la salida del programa suponiendo que el programa utiliza alcance dinámico y asociación superficial (*shallow binding*).

i. `42`
ii. `25`
iii. `32`
iv. ***Ninguna de las anteriores (`51`).***

(i) Considere un lenguaje *imperativo* con iteradores reales similares a los de Clu. Por simplicidad suponga que el lenguaje permite manipular listas como en Haskell, i.e.

- `a.head` retorna el primer elemento de la lista `a`.
- `a.tail` retorna la lista `a` **sin** el primer elemento.
- `a ++ b` concatena las listas `a` y `b`.
- `[]` es la lista vacía.
- `[x]` construye una lista con el elemento `x`.

Considere el iterador

```
P = iterator ( a : [Int] ) yields [Int]
  if (a == []) {
    yield [];
  } else {
    for e : [Int] in P( a.tail ) {
      yield e;
      yield [ a.head ] ++ e;
    }
  }
```

Considere ahora la iteración

```
for i : [Int] in P( [42,69,27] ) do
  ...
endfor
```

La primera lista procesada por ésta iteración es `[]` ¿Cuál es la segunda lista procesada por la iteración?

i. `[27]`
ii. `[69]`
iii. `[42,69]`
iv. ***Ninguna de las anteriores (`[42]`).***

(j) Continuando con la pregunta (1.i), ¿cuál es la tercera lista procesada por la iteración?

i. `[42]`
ii. *`[69]`*
iii. `[69,27]`
iv. Ninguna de las anteriores.

(k) Continuando con la pregunta (1.i), ¿cuál es la cuarta lista procesada por la iteración?

i. `[27]`
ii. *`[42,69]`*
iii. `[42,27]`
iv. Ninguna de las anteriores.

(l) Considere el siguiente procedimiento recursivo P escrito en C

```
void P(int n) {
  if (B0) {
    I0;
    if (B1) { I1;   P(A); }
    else    { P(B); P(C); }
  } else if (B2) {
    P(D); I2;
  } else {
    P(E);
    if (B3) { I3;   }
    else    { P(F); }
  }
}
```

En este procedimiento, `B0`, `B1`, `B2` y `B3` son expresiones booleanas cualesquiera, `I0`, `I1`, `I2` e `I3` son instrucciones cualesquiera, y `A`, `B`, `C`, `D`, `E` y `F` son expresiones enteras cualesquiera, excepto por el hecho de que en ninguna de estas expresiones e instrucciones hay llamadas a `P` o llamadas a alguna otra subrutina que directa o indirectamente pudiese llamar a `P`.
¿Cuáles de las llamadas recursivas de `P` corresponden con seguridad a recursión de cola?

i. `P(A)`, `P(B)`, `P(D)` y `P(E)`.
ii. `P(A)`, `P(C)`, `P(E)` y `P(F)`.
iii. ***`P(A)`, `P(C)` y `P(F)`.***
iv. `P(B)`, `P(D)` y `P(E)`.

2. **(6 puntos)** Continuando con la pregunta 1.l, suponga que estamos utilizando un compilador que no optimiza la recursión de cola, y se desea que Ud. reprograme el procedimiento P transformando la recursión de cola en una iteración. **Nota:** tanto las expresiones booleanas como las expresiones enteras podrían contener efectos secundarios (*side effects*).

```
void P(int n) {
  while (true) {
    if (B0) {
      I0;
      if (B1) { I1;   n = A; continue; }
      else    { P(B); n = C; continue; }
    } else if (B2) {
      P(D); I2; return;
    } else {
      P(E);
      if (B3) { I3; return; }
      else    { n = F; }
    }
  }
}
```