Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3641 – Lenguajes de Programación I
Septiembre-Diciembre 2006

Carnet: _____

Nombre: _____

Examen I

(40 puntos)

Antes de empezar, revise bien el examen, el cual consta de 4 (CUATRO) preguntas.

Pregunta 0	Pregunta 1	Pregunta 2	Pregunta 3	Total
8 puntos	10 puntos	10 puntos	12 puntos	40 puntos

Pregunta 0 — 8 puntos

Considere el siguiente programa escrito en pseudocódigo:

```
var x,y: int := 800,7     /* variables globales */

procedure sumar ()
    x := x + y
endp

procedure tercero (P: procedure)
    var y: int := 40
    P ()
endp

procedure segundo ()
    var y: int := 31
    tercero (sumar)
endp

procedure primero (n: int; Q: procedure)

    procedure asignar ()
        y := n
        segundo ()
    endp

    /* inicio cuerpo de subprograma primero */
    if (n < 50)
        primero (3 * n, asignar)
    else
        Q ()
    endif
endp

/* inicio programa principal */
begin
    primero (25, sumar)
    write (x)
end
```

Indique cuáles son las cuatro posibles salidas de este programa, según las reglas definidas para el pseudolenguaje de programación utilizado en cuanto a alcance —estático/dinámico— y en cuanto a asociación (*binding*) de ambientes no locales de subprogramas pasados como parámetro —profundo/superficial (*deep*/*shallow*)—.

Nota: Considere que cada instancia recursiva de un subprograma debe ser independiente, con su propia copia de la información local (de la manera usual manejada en lenguajes de programación convencionales).

Otra nota: Si desea ahorrar tiempo, es factible analizar una sola configuración de pila de activaciones de subprogramas y, sobre ella, calcular las cuatro salidas requeridas.

Última nota: Revise muy bien sus resultados.

Pregunta 1 — 10 puntos

Considere la siguiente versión recursiva del conocido algoritmo *Quicksort*, escrita en el lenguaje Java como un método estático público de alguna clase:

```
public static void quicksort (int[] a, int inf, int sup)
{
    if (sup - inf <= 18)
        insertionsort (a, inf, sup);
    else {
        int med = partition (a, inf, sup);
        quicksort (a, inf, med);
        quicksort (a, med + 1, sup);
    }
}
```

El cuerpo hace uso de otros dos métodos estáticos, `insertionsort` (para manejar los casos base) y `partition` (el típico subprograma auxiliar de *Quicksort*), cuyas implementaciones detalladas no son relevantes.

Una de las dos llamadas recursivas de `quicksort` corresponde a recursión de cola. Suponiendo que estamos utilizando un compilador que no hace un manejo inteligente de la recursión de cola, se desea que Ud. como programador subsane esta deficiencia del compilador. Para ello, reprograme el método `quicksort` transformando la recursión de cola en una iteración. Para la llamada recursiva no correspondiente a recursión de cola, mantenga la recursión.

Pregunta 2 — 10 puntos

Considere los iteradores (*iterators*) del lenguaje Clu.

A continuación presentamos un iterador `misterio` construido en cuasi-Clu, esto es, pseudocódigo similar al del lenguaje Clu en el que, para evitar las complicaciones de las estructuras de datos provistas por Clu, utilizaremos la sencilla versión de tipos recursivos provistos por el lenguaje Haskell:

```
data arbolE = vacio | nodo (int,arbolE,arbolE)

misterio = iter (a,b: int) yields (arbolE)

    if (a >= b) then
        yield (vacio)
    else
        for k: int in rangoE (a,b)
            for i: arbolE in misterio (a,k)
                for d: arbolE in misterio (k+1,b)
                    yield (nodo (k,i,d))
                end
            end
        end
    end

end misterio
```

En el cuerpo de `misterio`, se hace uso de otro iterador `rangoE`, el cual genera la secuencia ascendente de enteros consecutivos desde su primer parámetro hasta el segundo, incluido el primero y excluido el segundo. Esto es, la activación `rangoE(m,n)` del iterador generaría en forma ascendente los enteros del conjunto $\{\, x \in \mathbb{Z} \mid \mathtt{m} \leqslant x < \mathtt{n} \,\}$.

Responda ahora las siguientes preguntas:

(a) Indique cuáles son los 3 (tres) primeros árboles de la salida generada por la iteración

```
for w: arbolE in misterio (26,29) do
    dibujar (w)
end
```

suponiendo que el subprograma `dibujar` imprime una representación gráfica razonable de un objeto del tipo `arbolE`.

Nota: Por favor, señale claramente los 3 (tres) árboles requeridos, separándolos de cualquier cálculo intermedio que Ud. haya necesitado realizar.

(b) Indique cuáles son los objetos generados por una activación `misterio(X,Y)` de nuestro iterador en el caso general.

Puede ignorar el orden de generación y limitarse a describir cuáles son los objetos generados.

Pregunta 3 — 12 puntos

Michael Scott, en su texto "*Programming Language Pragmatics*", al reseñar coerciones como un "*controversial subject in language design*", presenta los siguientes ejemplos en lenguaje C (comentarios incluidos):

```
short int s;
unsigned long int l;
...
s = l; /* l's low-order bits are interpreted as a signed number. */
l = s; /* s is sign-extended to the longer length, then
            its bits are interpreted as an unsigned number. */
```

En relación con estos dos ejemplos de coerción y los comentarios de Scott, responda las siguientes preguntas:

(a) Explique por qué, al realizar los procesadores de lenguaje C estas conversiones de la manera indicada en los comentarios, en ambos casos los resultados pueden no corresponder a la semántica aritmética entera convencional. Esto es, explique por qué en ambos casos los resultados pueden no tener sentido desde el punto de vista matemático de los números enteros.

Dé además, para cada una de las dos coerciones, un ejemplo de valor que sea convertido adecuadamente (esto es, correctamente según la semántica entera convencional) y un ejemplo de valor que no lo sea. Construya estos ejemplos suponiendo que un `short int` ocupa 4 bits y que un `long int` ocupa 6 bits.

Nota: Los tamaños señalados de 4 y 6 bits son claramente irreales en relación con implementaciones actuales de lenguaje C. Se propone estos tamaños para facilitarle a Ud. la construcción de los ejemplos; sin embargo, si Ud. prefiere utilizar tamaños mayores más cercanos a la realidad, tiene libertad de hacerlo.

(b) Sabemos que entre los criterios de diseño prioritarios de lenguaje C se tiene el favorecer acceso a representaciones de bajo nivel y el lograr buena eficiencia de ejecución de los programas escritos en el lenguaje. Suponga que abandonamos tales criterios de diseño y decidimos, por una parte, favorecer representaciones conceptuales de alto nivel y, por otra, sacrificar eficiencia para permitir la detección de conversiones inadecuadas.

¿Cómo definiría Ud., según estos nuevos criterios de diseño, el comportamiento de las dos coerciones de nuestro ejemplo?