Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3641 - Lenguajes de Programación I
Enero-Marzo 2008

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**. Si Ud. selecciona más de una opción, la pregunta se considera contestada **incorrectamente**.

   (a) En un lenguaje con alcance dinámico **siempre** es necesario

      i. Un recolector de basura.
      ii. Utilizar la cadena estática cuando se construyen clausuras.
      iii. Diferir verificaciones semánticas hasta la ejecución.
      iv. Operar sin variables globales.

   (b) Una variable local

      i. Tiene una ubicación relativa diferente dentro de cada activación de la subrutina que le contiene.
      ii. Tiene una ubicación fija durante la ejecución del programa.
      iii. Sólo es accesible si el lenguaje tiene alcance estático.
      iv. Es accesible desde subrutinas anidadas a través de la cadena estática.

   (c) En C, la expresión `*p++ = (*q++)+1` es equivalente a

      i. `*p = *(q+=1); p++`
      ii. `*(p+=1) = *(q+=1)+1`
      iii. `*(p+=1)++ = *q; q += 1`
      iv. `*p = *q; ++(*p); p += 1; ++q`

   (d) La diferencia entre un módulo como administrador y un módulo como tipo estriba en que:

      i. El espacio de nombres del primero solamente contiene funciones, mientras que el segundo contiene funciones y tipos de datos.
      ii. El primero sólo puede tener espacio de nombres abierto o cerrado según sea necesario, mientras que el segundo sólo puede tener espacio de nombres cerrado.
      iii. En el primero los datos se pasan a las funciones como argumentos explícitos, mientras que en el segundo se pasan implícitamente.
      iv. Sólo con el segundo es posible crear tipos de datos abstractos.

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

      i. Una asignación de la forma `a := b` siempre produce un *alias*.
      ii. Cualquier expresión puede utilizarse como *l-value*.
      iii. No es posible construir referencias.
      iv. Las variables no son más que contenedores con nombre.

2. Recursión:

(a) **(4 puntos)** Considere la siguiente solución al problema de las Torres de Hanoi escrita en C:

```
void hanoi(int N, int Inicio, int Final, int Auxiliar)
{
    if (N == 1) {
        printf("Mover desde %d hasta %d\n",Inicio,Final);
    } else {
        hanoi(N-1,Inicio,Auxiliar,Final);
        printf("Mover desde %d hasta %d\n",Inicio,Final);
        hanoi(N-1,Auxiliar,Final,Inicio);
    }
}
```

Una de esas llamadas corresponde a recursión de cola. Identifíquela y elimínela.

(b) **(4 puntos)** Considere la siguiente función en Haskell para invertir listas

```
invertir        :: [a] -> [a]
invertir []     =  []
invertir (x:xs) =  invertir xs ++ [x]
```

Reescriba la función para introducir recursión de cola.

3. **(8 puntos)** Suponga un lenguaje *imperativo* con iteradores reales, que son declarados como una rutina cualquiera y donde la producción de valores se indica mediante la instrucción `yield`. Suponga que el lenguaje permite manipular listas de la siguiente manera, i.e.

   - El primer elemento de una lista está en la posición 0.
   - `a.init(i)` retorna una lista con los elementos iniciales de la lista `a` desde la 0-ésima hasta la `i`-ésima posición inclusive.
   - `a.tail(i)` retorna la lista `a` a partir del `i`-ésimo elemento hasta el último elemento.
   - `a.length` retorna la longitud de la lista `a`.
   - `a ++ b` concatena las listas `a` y `b`.
   - `a[i]` retorna el `i`-ésimo elemento de la lista `a`.

   Escriba un iterador `permutaciones(a : lista)` que produzca las permutaciones de la lista, retornándolas una a una con cada invocación. **Pista:** si quiero producir una lista sin el `i`-ésimo elemento de la lista `a`, entonces escribo

   ```
   a_sin_i-esimo := a.init(i-1) ++ a.tail( i+1 )
   ```

4. Considere el siguiente programa escrito en pseudocódigo

```
int u = 42;
int v = 69;
int w = 17;
proc add( z : int )
    u := v + u + z
proc bar( fun : proc)
    int u := w;
    fun(v)
proc foo( x : int, w : int)
  int v := x;
  bar(add)
main
  foo(u,13)
  print(u)
end;
```

¿Qué imprime el programa?

(a) **(3 puntos)** Asumiendo que el lenguaje utiliza alcance **estático**.

(b) **(3 puntos)** Asumiendo que el lenguaje utiliza alcance **dinámico** y ***deep binding***.

(c) **(3 puntos)** Asumiendo que el lenguaje utiliza alcance **dinámico** y ***shallow binding.***

En **todos** los 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.