Universidad Simón Bolívar
Dpto. de Computación y Tecnología de la Información
CI4645 – Lenguajes de Programación I
Enero-Marzo 2004

Carnet: _____

Nombre: _____

Examen Parcial I

(35 puntos)

Antes de empezar, revise bien el examen, el cual consta de 5 (CINCO) preguntas.

Pregunta 1	Pregunta 2	Pregunta 3	Pregunta 4	Pregunta 5	Total
7 puntos	7 puntos	7 puntos	7 puntos	7 puntos	35 puntos

Pregunta 1 — 7 puntos

Suponga que estamos trabajando con uno de los típicos dialectos de Fortran, en el que no se permite subrutinas recursivas (ni directa ni indirectamente recursivas). Sabemos que esto permite que la asignación de espacio en memoria ("*storage allocation*") para las variables locales de las subrutinas se realice estáticamente, en un lugar fijo dentro del espacio del programa.

En relación con este lenguaje, responda las dos siguientes preguntas:

(a) (3 puntos) El grafo de llamadas potenciales de un programa tiene como nodos a las subrutinas de éste, incluyendo al programa principal, y como aristas dirigidas (P, Q) aquéllas en que el texto de la subrutina origen P contiene una llamada a la subrutina destino Q. Decoremos, además, los nodos de este grafo con la cantidad de espacio local requerida por cada subrutina.

Indique qué característica de este grafo puede ser utilizada para determinar la cantidad mínima de espacio total requerida por el programa para sus variables.

(Basta especificar el problema de análisis de grafos necesario; no se requiere un algoritmo que resuelva éste.)

(b) (4 puntos) Un programador usuario de este lenguaje decide monitorear la cantidad de veces que una de sus subrutinas es invocada. Para esto, sabiendo que el dialecto de Fortran en cuestión inicializa automáticamente las variables enteras en cero, coloca una variable entera *local* contador en la subrutina y, como últimas instrucciones del cuerpo, incrementa el contador y lo imprime.

Indique por qué el programador está cometiendo un error que, sin embargo, podría funcionar bien. Para esto, mencione al menos 3 posibles estrategias de asignación de espacio local, de las cuales una haga que el contador funcione bien, y una haga que definitivamente no se logre la intención del programador con el contador.

(Sea *muy breve*, pero indique para cada estrategia si falla el contador o no.)

(Espacio adicional para su respuesta a la pregunta 1)

Pregunta 2 — 7 puntos

En muchos lenguajes de programación, la definición del lenguaje establece que el orden de evaluación de los operandos de un operador binario es arbitrario.

En relación con esto, responda las dos siguientes preguntas:

(a) **(4 puntos)** Dé un *pequeño* programa en Java en el que, asumiendo tal orden de evaluación como arbitrario, los resultados computados por el programa puedan variar según el orden utilizado.

Indique, además, el resultado que se computaría según la verdadera definición de Java.

(b) **(3 puntos)** En muchos casos, la definición del lenguaje también establece que la mayoría de los operadores binarios asocian a la izquierda.

¿Entraría esto en contradicción con lo antes indicado sobre orden de evaluación? ¿Por qué?

(Espacio adicional para su respuesta a la pregunta 2)

Pregunta 3 — 7 puntos

En el lenguaje Ada, los límites del tipo índice de un arreglo no varían durante el tiempo de vida del arreglo, y pueden ser determinables estáticamente o, en caso contrario, determinables al momento del nacimiento del arreglo.

Considere el siguiente arreglo uni-dimensional declarado *localmente* en una subrutina:

```
A: array [Inf..Sup] of T   ,
```

donde `Inf` y `Sup` son enteros, y `T` es cualquier otro tipo (excepto arreglo nuevamente, pues, si no, `A` no sería uni-dimensional).

Como en Ada los arreglos locales nacen y mueren en cada activación y desactivación de la subrutina, éstos son almacenados en la pila (a diferencia de, por ejemplo, los arreglos de Java, que son almacenados en el "*heap*").

Responda ahora las dos siguientes preguntas:

(a) **(3 puntos)** Suponga que `Inf` y `Sup` son determinables estáticamente. Siendo *da* el desplazamiento ("*offset*") asociado a `A` estáticamente, para ser usado con respecto al registro `FP` ("*frame pointer*"), dé la fórmula de cálculo de la dirección del elemento `A[i]` que debe ser usada durante la ejecución.

(b) **(4 puntos)** Suponga ahora que `Inf` y `Sup` no son determinables estáticamente, sino al momento del nacimiento del arreglo. Indique cómo variaría su respuesta a **(a)** en este caso.

(Espacio adicional para su respuesta a la pregunta 3)

Pregunta 4 — 7 puntos

En relación con la implementación de tablas de símbolos para lenguajes con alcance dinámico, el libro de texto (Michael Scott – "*Programming Language Pragmatics*") comenta las siguientes dos posibilidades: listas de asociación, y tablas centrales de referencia.

Éstas podrían ser implementadas en Haskell fácilmente con las estructuras de datos siguientes:

`[ (String , [Info]) ]` *y* `[ (String , Info) ]` ,

donde 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.

Ahora responda:

(a) **(2 puntos)** ¿Cuál de las dos estructuras de datos corresponde a listas de asociación, y cuál a tablas centrales de referencia?

(b) **(5 puntos)** Para ambas estructuras de datos, implemente la función `enterScope` de entrada a una nueva región local. Esta función recibe una tabla de símbolos y la lista `[ (String , Info) ]` de nuevos "*bindings*" locales de la región, y devuelve, por supuesto, la nueva tabla de símbolos resultante.

Puede asumir que en la lista de nuevos "*bindings*" locales no hay repetición de identificadores.

(Espacio adicional para su respuesta a la pregunta 4)

Pregunta 5 — 7 puntos

Suponga que Java es extendido con iteradores/generadores del estilo del lenguaje Clu.

Éstos son declarados como una subrutina cualquiera, excepto que se agrega la palabra clave `generator` antes del nombre de la subrutina. Al igual que en Clu, la generación de valores se indica mediante la instrucción `yield`.

Se desea que Ud. utilice esta nueva facilidad de Java para programar un generador iterativo de elementos de un árbol binario en pre-orden. Suponga que el esqueleto de la clase árbol es como sigue:

```
class ArbBin
{
    private Nodo raiz;
    ...
}
```

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

```
class Nodo
{
    Object contenido;
    ArbBin izq, der;
}
```

Programe entonces el generador deseado de elementos en pre-orden, como subrutina local a la clase árbol, con la restricción de que *no* puede utilizar recursión, sino iteración.

Nota: Puede asumir la existencia de una clase pila con sus típicas operaciones.

(Espacio adicional para su respuesta a la pregunta 5)

(Espacio adicional)