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

Carnet: _____

Nombre: _____

Examen II

(40 puntos)

Antes de empezar, revise bien el examen.

Nota
40 puntos

(Espacio adicional)

Instrucciones:

A continuación se le presentan veinte (20) preguntas de selección, numeradas de 0 a 19. Cada pregunta viene acompañada de cuatro posibles respuestas (a,b,c,d) de las cuales *sólo una* es correcta. Ud. deberá marcar la opción que considere correcta o, si desea no contestar la pregunta, marcar la opción (e) que indica que Ud. prefiere omitir la respuesta.

Cada una de las veinte (20) preguntas tiene un valor de dos (2) puntos. Tres preguntas malas eliminan una buena. Las preguntas omitidas (marcadas en la opción (e)) no suman ni restan puntos.

0. Considere la siguiente declaración de un arreglo bi-dimensional, en la cual se utiliza sintaxis del estilo de la familia de los Algol:

```
m:  array [inf0..sup0][inf1..sup1] of T
```

donde `inf0`, `sup0`, `inf1` y `sup1` son constantes de tipo entero, y `T` es un tipo cualquiera.

Suponga que el arreglo ha sido almacenado por filas, esto es, que su *memory layout* utiliza *row-major order*.

Se desea que Ud. señale la fórmula de cálculo que debe ser utilizada para determinar la dirección de `m[i][j]`, suponiendo que `i` y `j` se encuentran en los rangos adecuados, y teniendo que:

- la variable `m` ha sido asociada a la dirección de memoria α,
- `i` y `j` son variables enteras almacenadas en las direcciones de memoria β y γ,
- `inf0`, `sup0`, `inf1` y `sup1` son constantes conocidas estáticamente, y
- $TamT$ es la cantidad de bytes utilizada por cada elemento del tipo base `T`.

Utilizaremos $*X$ para referirnos al contenido de cualquier dirección de memoria X.

(a) $\alpha + (*\beta \times (\texttt{sup1} - \texttt{inf1} + 1) + *\gamma) \times TamT$.

(b) $\alpha + ((*\beta - \texttt{inf0}) \times \texttt{sup1} + (*\gamma - \texttt{inf1})) \times TamT$.

(c) $\alpha + ((*\beta - \texttt{inf0} + 1) \times (\texttt{sup1} - \texttt{inf1} + 1) + (*\gamma - \texttt{inf1} + 1)) \times TamT$.

(d) $\alpha + ((*\beta - \texttt{inf0}) \times (\texttt{sup1} - \texttt{inf1} + 1) + (*\gamma - \texttt{inf1})) \times TamT$.

(e) No sabe / No contesta.

1. Considere ahora la misma información de la pregunta anterior 0, excepto que se utiliza para `m` el modelo de almacenamiento por medio de referencias del estilo de Java. Esto es, cambiamos a *memory layout* de *row-pointer*, y usamos modelo de referencia tanto para el manejo de `m` como para el manejo de los elementos de tipo `T` almacenados en éste.

Su nueva fórmula de cálculo para la dirección de `m[i][j]`, suponiendo que cada apuntador ocupa 4 bytes de memoria, sería:

(a) $*(*\alpha + (*\beta - \texttt{inf0} + 1) \times 4) + (*\gamma - \texttt{inf1} + 1) \times 4$.

(b) $*(*\alpha + (*\beta - \texttt{inf0}) \times 4) + (*\gamma - \texttt{inf1}) \times 4$.

(c) $*\alpha + (*\beta - \texttt{inf0}) \times 4 + (*\gamma - \texttt{inf1}) \times 4$.

(d) $*(*\alpha + (*\beta - \texttt{inf0}) \times (\texttt{sup1} - \texttt{inf1} + 1)) + (*\gamma - \texttt{inf1}) \times 4$.

(e) No sabe / No contesta.

2. De nuevo regrese a la información de la pregunta 0, retomando el almacenamiento por filas (*memory layout* en *row-major order*) considerado originalmente.

   Suponga ahora que `inf0`, `sup0`, `inf1` y `sup1` no son constantes estáticas, sino constantes que sólo se conocen dinámicamente al momento de su nacimiento (*elaboration-time constants*). El arreglo `m` sería una variable local de alguna subrutina, con tiempo de vida local convencional, y su tamaño sería fijo pero determinado al momento de su nacimiento.

   En la dirección α lo que almacenaríamos entonces sería un descriptor, o *dope vector*, para `m` en el cual tendríamos: primero, empezando en α, los cuatro límites `inf0`, `sup0`, `inf1` y `sup1`, en ese orden, y luego la referencia al contenido del arreglo.

   Señale qué habría que cambiarle a su fórmula original de la pregunta 0, suponiendo que cada entero y cada apuntador ocupa 4 bytes de memoria:

   (a) Sólo se cambiaría α por $*\alpha$.

   (b) Sólo se cambiaría cada límite (`inf0`, `inf1`, etcétera) por su $*(\alpha + \text{desplazamiento})$ correspondiente.

   (c) Se cambiaría cada límite (`inf0`, `inf1`, etcétera) por su $*(\alpha + \text{desplazamiento})$ correspondiente, y se cambiaría α por $*(\alpha + 16)$.

   (d) No se cambiaría nada.

   (e) No sabe / No contesta.

3. El sistema de tipos de Pascal es considerado inseguro. Esto se debe a que la verificación del buen uso de variantes es demasiado costosa bajo las reglas de Pascal, por lo cual las implementaciones del lenguaje generalmente omiten tal verificación.

   Por ejemplo, usualmente se permite la siguiente violación del sistema de tipos:

```
var r: record
         case b: boolean of
           false: (x: real);
           true:  (y: integer)
       end
...
r.b := false;
r.x := 7.2;
r.b := true;
write(r.y)
...
```

   Considere las siguientes características de los lenguajes Algol 68 y Ada, a efectos de analizar por qué en ellos, a diferencia de Pascal, el sistema de tipos sí es seguro en relación con el manejo de variantes/uniones:

   I. En Algol 68 las asignaciones al campo discriminante, esto es, el *tag field*, (`b` en nuestro ejemplo) son implícitamente manejadas por el lenguaje en lugar de explícitamente manejadas por el programador.

   II. En Algol 68 la sintaxis no permite acceder a la información de cada una de las variantes sin preguntar previamente por el valor del campo discriminante implícito.

III. En Ada el campo discriminante puede ser declarado constante/inmutable para algunos registros.

IV. En Ada el campo discriminante no puede ser cambiado sin asignar simultáneamente valores a todos los otros campos asociados a la variante correspondiente.

¿Cuáles de estas características garantizan la seguridad de las variantes/uniones?

(a) Sólo I en Algol 68, y sólo III en Ada.

(b) La combinación de I y II en Algol 68, y la combinación de III y IV en Ada.

(c) La combinación de I y II en Algol 68, y sólo III en Ada.

(d) La combinación de I y II en Algol 68, y sólo IV en Ada.

(e) No sabe / No contesta.

4. Pascal no permite que el campo discriminante, esto es, el *tag field*, de un registro variante sea pasado a una subrutina como parámetro por referencia. ¿Por qué tiene sentido esto?

(a) Para evitar que la subrutina altere el discriminante, en caso de que la implementación esté verificando la seguridad de las variantes.

(b) Para evitar que la subrutina altere el discriminante, pues un campo discriminante nunca debería ser cambiado.

(c) Porque un campo discriminante debería ser pasado como parámetro por nombre, y Pascal no permite tal tipo de pasaje de parámetros.

(d) Porque no tiene sentido pasar como parámetro un campo discriminante, si la mayoría de las implementaciones no verifica la seguridad de las variantes.

(e) No sabe / No contesta.

5. Considere dos variables de tipo apuntador en un lenguaje tipo Pascal:

```
p,q:  ^T
```

donde `T` es un tipo cualquiera.

Suponga que se está implementando verificación de referencias colgadas (*dangling references*) mediante claves de acceso (*locks and keys*). Por ello, cada apuntador está implementado como un registro con la clave de acceso y el apuntador propiamente dicho, en ese orden; y cada objeto del *heap* está implementado también como un registro con la clave de acceso y luego el valor del objeto en sí, de nuevo en ese orden.

Se desea que Ud. señale qué acciones deben ser ejecutadas a bajo nivel para la instrucción `p := q`, considerando que:

- `p` y `q` están almacenados respectivamente en las direcciones de memoria α y β,
- cada clave de acceso y cada dirección ocupa 4 bytes,
- *null* es una dirección inválida correspondiente a apuntadores nulos,
- $*X$ se refiere al contenido de la dirección de memoria X, y
- $X \leftarrow Y$ significar almacenar el <u>valor</u> Y en la <u>dirección</u> X.

(a) Si $*(\beta + 4) \neq null$ y $\beta \neq *(\beta + 4)$, error de referencia colgada;
en caso contrario, $\alpha \leftarrow *\beta$ y $\alpha + 4 \leftarrow *(\beta + 4)$.

(b) Si $*(\beta + 4) \neq null$ y $*\beta \neq **(\beta + 4)$, error de referencia colgada;
en caso contrario, $\alpha \leftarrow *\beta$ y $\alpha + 4 \leftarrow *(\beta + 4)$.

(c) Si $*\beta \neq **(\beta + 4)$, error de referencia colgada;
en caso contrario, $\alpha \leftarrow **\beta$ y $\alpha + 4 \leftarrow **(\beta + 4)$.

(d) Si $*(\beta + 4) \neq \mathit{null}$ y $*\beta \neq **(\beta + 4)$, error de referencia colgada;
en caso contrario, $\alpha \leftarrow **\beta$ y $\alpha + 4 \leftarrow **(\beta + 4)$.

(e) No sabe / No contesta.

6. Continúe considerando toda la información de la pregunta inmediatamente anterior 5.

   Se desea ahora que señale las acciones que deben ser ejecutadas a bajo nivel para la instrucción `p^ := q^`, entendiendo que "`^`" es el operador de indirección y asumiendo que ya se verificó que ninguna de las dos referencias es nula ni está colgada.

   Note además que, en el estilo de Pascal, la asignación corresponde a modelo de valor, y asuma además que esta asignación es permitida aunque el tipo base `T` sea un tipo compuesto.

   (a) $\alpha \leftarrow *\beta$ y $\alpha + 4 \leftarrow *(\beta + 4)$.

   (b) $\alpha \leftarrow **\beta$ y $\alpha + 4 \leftarrow **(\beta + 4)$.

   (c) $*(\alpha + 4) + 4 + k \leftarrow *(*(\beta + 4) + 4 + k)$
   para k igual a 0, 4, 8, etcétera, según el tamaño de `T`.

   (d) $*(\alpha + 4) + k \leftarrow *(*(\beta + 4) + k)$
   para k igual a 0, 4, 8, etcétera, según el tamaño de `T`.

   (e) No sabe / No contesta.

7. Continúe considerando la información de las dos preguntas anteriores 5 y 6.

   Suponga ahora que, además de las claves de acceso (*locks and keys*) para detección de referencias colgadas (*dangling references*), agregamos el uso de contadores de referencias (*reference counts*) para recolección de basura (*garbage collection*). Debido a esto, ahora cada objeto del *heap* estaría implementado como un registro con tres campos: la clave de acceso, luego el contador de referencias (también de 4 bytes), y por último el valor del objeto en sí, en ese orden.

   En la asignación `p := q`, ¿qué acciones de bajo nivel deben ser ejecutadas para mantener la consistencia de los contadores de referencias? Suponga que ya se determinó que ninguna de las dos referencias es nula ni está colgada.

   (a) Incrementar el contenido de la dirección $*(\alpha + 4) + 4$,
   decrementar el contenido de la dirección $*(\beta + 4) + 4$,
   liberar memoria de *heap* si hace falta.

   (b) Incrementar el contenido de la dirección $*(\beta + 4) + 4$,
   decrementar el contenido de la dirección $*(\alpha + 4) + 4$,
   liberar memoria de *heap* si hace falta.

   (c) Incrementar el contenido de la dirección $*(\alpha + 4) + 4$,
   incrementar el contenido de la dirección $*(\beta + 4) + 4$.

   (d) Decrementar el contenido de la dirección $*(\alpha + 4) + 4$,
   liberar memoria de *heap* si hace falta,
   decrementar el contenido de la dirección $*(\beta + 4) + 4$,
   liberar memoria de *heap* si hace falta.

   (e) No sabe / No contesta.

8. Considere un lenguaje con alcance estático en el que se permita además anidamiento de subrutinas. Al momento de llamar a una subrutina, ¿quién debe encargarse, entre la subrutina llamadora (*caller*) y la subrutina llamada (*callee*), de la creación del enlace de la cadena estática, y por qué?

   (a) La subrutina llamada, pues, pudiendo cualquiera de las dos determinar tal enlace estático, en la subrutina llamada tal código se tendría una sola vez.

   (b) La subrutina llamadora, pues, pudiendo cualquiera de las dos determinar tal enlace estático, se ahorra tiempo en la subrutina llamadora.

   (c) La subrutina llamada, pues es ella quien puede determinar su nivel de anidamiento estático en relación con la subrutina llamadora.

   (d) La subrutina llamadora, pues es ella quien puede determinar su nivel de anidamiento estático en relación con la subrutina llamada.

   (e) No sabe / No contesta.

9. Se tiene el siguiente programa escrito en pseudocódigo:

```
procedure P (x,y: int)
    x := x + y
    y := x + y
endp

/* programa principal */
begin
    var a: int := 5

    P (a,a)
    write (a)
end.
```

   Considere para el procedimiento `P` las siguientes combinaciones de modos de pasaje de sus parámetros: primero, que `x` sea pasado por valor y `y` por valor/resultado; segundo, que tanto `x` como `y` sean pasados por referencia.

   ¿Cuál sería la salida del programa en estos dos casos, respectivamente?

   (a) 15 y 20.

   (b) 10 y 20.

   (c) 20 y 20.

   (d) 15 y 10.

   (e) No sabe / No contesta.

10. En lenguaje C, la declaración de un parámetro formal de tipo arreglo "`T a[]`" puede omitir el tamaño del arreglo, pues éste será determinado por el parámetro real. Sin embargo, el tipo base `T` es imprescindible para poder determinar en compilación como manejar la aritmética de apuntadores `a+k` y de indexación `a[k]`.

    De acuerdo con esto, en el caso de un arreglo bi-dimensional de memoria contigua almacenado por filas (esto es, cuyo *memory layout* usa *row-major order* continuo y no *row-pointer*), ¿qué se puede omitir en la declaración de un parámetro formal?

(a) Sólo el tamaño de la primera dimensión, más no de la segunda: "`T a[][N]`".

(b) Sólo el tamaño de la segunda dimensión, más no de la primera: "`T a[N][]`".

(c) Los tamaños de ambas dimensiones, al igual que en el caso uni-dimensional: "`T a[][]`".

(d) Ni el tamaño de la primera dimensión, ni el de la segunda: "`T a[N][M]`".

(e) No sabe / No contesta.

11. El primer mecanismo de implementación de excepciones considerado por M.L.Scott en su texto es el de mantener una pila de manejadores de excepciones durante la ejecución.

    Con esta alternativa, ¿cuáles de las siguientes características de la implementación son postuladas por Scott?

    I. Todo `throw` debe empilar un manejador.

    II. Todo `throw` debe desempilar un manejador.

    III. Sólo los `throw` que ocurren en bloques protegidos `try` deben desempilar un manejador.

    IV. Se debe empilar un manejador cada vez que se entra a un bloque protegido `try`.

    V. Se debe empilar un manejador cada vez que se entra a un subrutina no protegida.

    (a) Sólo II, IV y V.

    (b) Sólo III y IV.

    (c) Sólo III, IV y V.

    (d) Sólo I y IV.

    (e) No sabe / No contesta.

12. En relación con la implementación de corrutinas e iteradores, ¿cuáles de las siguientes afirmaciones son ciertas?

    I. Cada corrutina debe contar con su propia pila de ejecución.

    II. Permitiendo la declaración de corrutinas sólo en el nivel más externo de anidamiento estático, se evita la necesidad de pilas del estilo "cactus".

    III. Los iteradores del estilo del lenguaje Clu pueden ser implementados mediante corrutinas.

    IV. En ausencia de corrutinas, los iteradores del estilo del lenguaje Clu pueden ser implementados sobre una sola pila de ejecución.

    (a) Sólo II y III.

    (b) Sólo I y IV.

    (c) Sólo I, II y IV.

    (d) Todas las cuatro afirmaciones I, II, III y IV.

    (e) No sabe / No contesta.

13. En los encabezados de módulo de Modula-2, los cuales señalan la información pública exportada por el módulo, se permite la llamada "exportación opaca" de tipos:

```
TYPE T;
```

mediante la cual se exporta el nombre de un tipo sin exportar su estructura.

Tales encabezados de módulo proporcionan la única información que es usada por los compiladores a la hora de procesar otros módulos que importan elementos externos.

La definición de Modula-2 exige que un tipo `T` exportado en el encabezado de manera opaca sea implementado en el cuerpo del módulo como un apuntador hacia cualquier otro tipo base.

¿Por qué Modula-2 impone esta restricción?

(a) Porque en un lenguaje que maneja las variables mediante modelo de referencia, todo tipo debe ser declarado como una referencia.

(b) Porque en un lenguaje que maneja las variables mediante modelo de valor, esto permite al compilador calcular cuánto espacio ocupa cada variable de un tipo importado opaco.

(c) Porque en un lenguaje que maneja las variables mediante modelo de valor, tanto los tipos opacos como los no-opacos deben ser declarados como referencias.

(d) Porque en un lenguaje que maneja las variables mediante modelo de referencia, no es posible hacer exportación no-opaca de tipos.

(e) No sabe / No contesta.

14. Teniendo una clase abstracta `A` en el lenguaje C++, se permite la declaración de variables de tipo `A*` mas no de tipo `A`. ¿A qué se debe esto?

(a) Al modelo de referencia que maneja el lenguaje, el cual favorece el tipo referencia `A*` sobre el tipo básico `A`.

(b) Al modelo de valor que maneja el lenguaje, y a no poder manejar objetos incompletos.

(c) A que en C++, a diferencia de Java, la herencia no corresponde a la noción de subtipo.

(d) Al modelo de valor que maneja el lenguaje, bajo el cual los tipos básicos deben ser manejados como clases abstractas.

(e) No sabe / No contesta.

15. Considere un lenguaje orientado por objetos en el que la herencia corresponde a la noción de subtipo y las variables son manejadas con modelo de valor (como ocurriría en Java bajo modelo de valor, o como ocurriría en C++ permitiendo sólo la herencia pública).

Suponga que en este lenguaje se tiene una cadena de tres clases `A`, `B` y `C`, en la que `A` es subclase de `B` y `B` es subclase de `C`. Si se declara un procedimiento `P` de la siguiente forma:

```
procedure P (in b0:  B; out b1:  B)
```

para el que, de acuerdo con el modelo general de valor bajo el que se manejan las variables, el parámetro de entrada `b0` es pasado por valor y el parámetro de salida `b1` es pasado por resultado.

¿En qué casos puede permitir un compilador una llamada de la forma `P(x,y)`?

(a) `x` es de tipo `A` y `y` es de tipo `C`.

(b) `x` es de tipo `C` y `y` es de tipo `A`.

(c) `x` y `y` son ambos de tipo `A`.

(d) `x` y `y` son ambos de tipo `C`.

(e) No sabe / No contesta.

16. Sea la siguiente definición de función escrita en el lenguaje Haskell:

```
f x y g = 3 + g (x + 1, 2 * y)
```

Suponiendo que el único tipo númerico del lenguaje es `Int`, ¿cuáles de las siguientes afirmaciones son correctas?

I. El tipo de `f` es `Int -> Int -> ((Int,Int) -> Int) -> Int`.

II. El tipo de `f` es `( ( Int -> Int ) -> ((Int,Int) -> Int) ) -> Int`.

III. El tipo de `f` es `Int -> ( Int -> ( ((Int,Int) -> Int) -> Int ) )`.

IV. `f` no es una función currificada, pero su último argumento `g` sí lo es.

V. `f` es una función currificada, pero su último argumento `g` no lo es.

VI. Ni `f` ni su último argumento `g` son funciones currificadas, a pesar de que Haskell favorece la currificación.

(a) Sólo II y IV.

(b) Sólo I, II y VI.

(c) Sólo III y V.

(d) Sólo I, III y V.

(e) No sabe / No contesta.

17. El lenguaje funcional Scheme ofrece las primitivas `delay` y `force` debido a que:

(a) El orden de evaluación por defecto utilizado en el lenguaje es el aplicativo.

(b) El orden de evaluación por defecto utilizado en el lenguaje es el normal.

(c) El orden de evaluación por defecto utilizado en el lenguaje es el perezoso.

(d) No hay un orden de evaluación por defecto utilizado en el lenguaje, y `delay` y `force` son la única vía para escoger un orden de evaluación.

(e) No sabe / No contesta.

18. Se tiene el siguiente programa escrito en el lenguaje Prolog:

```
pertenece(X, [Y|L]) :- pertenece(X, L).
pertenece(X, [X|L]).
```

Considere el orden de búsqueda de soluciones utilizado por Prolog, y un segundo orden de búsqueda en el que se consideran primero las últimas reglas y se va avanzando hacia las primeras, para la evaluación de la pregunta (*query*) siguiente:

```
?- pertenece(A, [10,20]).
```

¿Qué soluciones para la variable `A`, y en qué orden, serán arrojadas para esta pregunta bajo los dos órdenes de búsqueda antes señalados, respectivamente?

(a) Ninguno y 10,20.

(b) 10,20 y 20,10.

(c) 20,10 y 10,20.

(d) Sólo 20 y 10,20.

(e) No sabe / No contesta.

19. Considere de nuevo el cambio del orden de búsqueda de soluciones de Prolog de la manera señalada en la pregunta anterior 18, esto es, utilizar primero las últimas reglas e ir avanzando hacia las primeras. Se mantiene intacto, sin embargo, el orden en que se evalúan los lados derechos de las reglas.

    ¿A qué cambio corresponde esto en el árbol de búsqueda "And/Or" que utiliza Prolog?

    (a) A ningún cambio: el árbol se recorre de la misma manera que antes.

    (b) A invertir el orden de recorrido de los hijos de todos los nodos, tanto los nodos "And" como los nodos "Or".

    (c) A invertir el orden de recorrido de los hijos de los nodos "Or", manteniendo el orden de recorrido de los hijos de los nodos "And".

    (d) A invertir el orden de recorrido de los hijos de los nodos "And", manteniendo el orden de recorrido de los hijos de los nodos "Or".

    (e) No sabe / No contesta.
