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

Examen II

(30 puntos)

Nombre:

Carnet:

1. **(21 puntos)** Considere cuidadosamente las siguientes cuestiones y seleccione exactamente una respuesta de las alternativas que se presentan. Cada cuestión contestada correctamente **suma tres (3) puntos** pero una cuestión contestada incorrectamente **resta un (1) punto**. Si Ud. selecciona más de una opción, la pregunta se considera contestada **incorrectamente.**

   (a) Considere la siguiente declaración de dos variables de tipo apuntador en un lenguaje tipo Pascal

   ```
   foo, bar : ^T
   ```

   donde T es un tipo cualquiera. Suponga que se está implantando detección de referencias colgadas (*dangling references*) mediante llaves y cerraduras (*locks and keys*). En ese caso, cada apuntador en realidad es un registro que contiene el apuntador propiamente dicho y la llave de acceso, en ese orden; y cada objeto del *heap* es un registro con la llave de acceso y el valor del objeto en sí, en ese orden. Considerando que

   - `bar` y `foo` están almacenadas en las direcciones de memoria α y β.
   - Cada llave de acceso ocupa 4 bytes y cada apuntador ocupa 4 bytes.
   - `nil` es una dirección inválida correspondiente a un apuntador nulo.
   - `*X` se refiere al *contenido* de la dirección de memoria `X`.
   - `X:=Y` significa almacenar el *valor* `Y` en la *dirección* `X`.

   ¿cuáles acciones deben ser ejecutadas a bajo nivel para la instrucción `foo:=bar`?

   i. Si $*\alpha \neq nil \wedge *\alpha \neq *(\alpha + 4)$, error de referencia colgada;
   en caso contrario, $\beta := *\alpha$ y $\beta + 4 := *(\alpha + 4)$.

   ii. Si $*\beta \neq nil \wedge *\beta \neq *(\beta + 4)$, error de referencia colgada;
   en caso contrario, $\beta := *\alpha$ y $*(\beta + 4) := *(\alpha + 4)$.

   iii. **Si** $*\alpha \neq nil \wedge **\alpha \neq *(\alpha + 4)$**, error de referencia colgada;**
   en caso contrario, $\beta := *\alpha$ **y** $\beta + 4 := *(\alpha + 4)$**.**

   iv. Si $*\beta \neq nil \wedge **\beta \neq *(\beta + 4)$, error de referencia colgada;
   en caso contrario, $\alpha := *\beta$ y $\alpha + 4 := *(\beta + 4)$.

(b) Continúe considerando la información de la pregunta inmediatamente anterior. Se desea ahora que señale las acciones que deben ser ejecutadas a bajo nivel para la instrucción

```
foo^ := bar^
```

entendiendo que "^" es el operador de indirección en el lenguaje y asumiendo que ya se verificó que ninguna de las dos referencias es nula ni está colgada. Así mismo, asuma que el lenguaje emplea *modelo de valor* con *copia profunda*.

i. $*(\beta + 4) + k := *(*\alpha + 4 + k)$ con k igual a $0, 4, 8, \ldots$ hasta alcanzar el tamaño de T.

ii. $*\beta + 4 + k := *(*\alpha + 4 + k)$ **con k igual a $0, 4, 8, \ldots$ hasta alcanzar el tamaño de T.**

iii. $*(\beta + 4) + k := **\alpha + 4 + k$ con k igual a $0, 4, 8, \ldots$ hasta alcanzar el tamaño de T.

iv. $*\beta + 4 + k := **\alpha + 4 + k$ con k igual a $0, 4, 8, \ldots$ hasta alcanzar el tamaño de T.

(c) Continúe considerando la información de las dos preguntas anteriores. Suponga ahora que, además de las llaves y cerraduras (*locks and keys*) para detección de referencias colgadas (*dangling references*), agregamos el uso de contadores de referencias (*reference counts*) para facilitar la recolección de basura (*garbage collection*). Ahora, cada objeto en el *heap* sería un registro con la clave de acceso, el contador de referencias (también de 4 bytes) y por último el valor del objeto en sí, en ese orden.
En la asignación

```
foo:=bar
```

¿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.

i. Decrementar $*\beta + 4$,
liberar memoria del *heap* si hace falta,
incrementar $*\alpha + 4$.

ii. Decrementar $*\alpha + 4$,
liberar memoria del *heap* si hace falta,
incrementar $*\beta + 4$.

iii. Incrementar $*(*\beta + 4)$,
decrementar $*(*\alpha + 4)$,
liberar memoria del *heap* si hace falta.

iv. **Incrementar $*(*\alpha + 4)$,**
decrementar $*(*\beta + 4)$,
liberar memoria del *heap* si hace falta.

(d) Considere la siguiente declaración de un arreglo bi-dimensional:

```
m : array [i0..s0] of array [i1..s1] of T
```

donde i_0, s_0, i_1 y s_1 son constantes enteras de valor conocido estáticamente y `T` es un tipo cualquiera. Suponga que las variables enteras `i` y `j` están almacenadas en las direcciones α y β y la base de `m` ha sido almacenada en la dirección de memoria γ. Si el lenguaje de programación almacena los arreglos usando listas de apuntadores a filas (*row-pointer layout*), cada apuntador ocupa 4 bytes, y cada posición de una fila contiene un valor concreto de tipo T ¿cuál es la fórmula para determinar el valor de `m[i][j]`?

i. $*(*\gamma + (*\alpha - i_0) \times 4 + (*\beta - i_1) \times sizeof(T))$

ii. $*(*(*\gamma + (*\alpha - i_0) \times 4) + (*\beta - i_1) \times 4)$

iii. $*(*(*\gamma + (*\alpha - i_0) \times 4) + (*\beta - i_1) \times sizeof(T))$

iv. $*(*\gamma + (*\alpha - i_0) \times sizeof(T) + (*\beta - i_1) \times 4)$

(e) Considere las siguientes declaraciones de estructuras de datos en C en una arquitectura donde los enteros y los apuntadores ocupan 4 bytes.

```
int *foo[n];
int (*bar)[n];
int (*(*baz[n])())[n];
int (*(*qux)())[n];
```

¿Cuál de las siguientes afirmaciones es **falsa**?

i. `foo` ocupa *exactamente* $4 \times n$ bytes.

ii. `bar` **ocupa *exactamente*** $4 \times (n + 1)$ **bytes.**

iii. `foo` y `baz` ocupan exactamente el mismo espacio.

iv. `baz[i] = qux` es válido siempre que $0 \leq i < n$.

(f) El recolector de basura de Java opera con el método de *marcado y barrido*, selección generacional con copia y compactado de páginas. Entonces, ¿cuál de las siguientes afirmaciones es **cierta**?

i. **Necesita poder diferenciar entre números enteros y apuntadores para hacer su trabajo.**

ii. Es incapaz de recuperar la memoria asociada a estructuras con referencias circulares que son inaccesibles desde el conjunto raíz de referencias de búsqueda.

iii. Es posible que tome más tiempo y consuma recursos de procesador, pero siempre será capaz de recuperar toda la basura en el *heap*.

iv. Los programas sacan máximo provecho del *caché* del procesador pues se garantiza que los accesos a las páginas de memoria son contiguos.

(g) Considere la siguiente declaración en un lenguaje con soporte nativo para conjuntos, en el cual los operadores $+$ y $*$ corresponden a la unión e intersección de conjuntos respectivamente, adoptando las precedencias habituales

```
var foo : set of 'g'..'m';
    bar : set of 'e'..'i';
    baz : set of 'a'..'k';
    i   : 'j'..'l';
    j   : 'a'..'d';
```

Ahora considere la instrucción

```
baz := foo + (bar * ['m'..'p', i, j ])
```

¿Qué tipo de verificaciones y cuándo deben hacerse?

i. A tiempo de compilación se detecta que la instrucción es inconsistente en el uso de los tipos.

ii. A tiempo de compilación se detecta que siempre puede hacerse la asignación.

iii. A tiempo de ejecución debe generarse un error siempre y cuando $'g' \leq foo \leq' k'$.

iv. **A tiempo de ejecución se permite la asignación siempre y cuando** $'g' \leq foo \leq' k'$**.**

2. Considere la siguiente declaración de algún lenguaje de programación que implanta registros variantes con discriminante al estilo de Pascal

```
foo : array [-11..-2] of array [-5..n] of array [3..9] of
      record
        a : char
        b : short
        c : float
        d : integer
        case e of
          true)
            record
              g : integer
              h : short
              i : boolean
              j : char[3]
              k : short
            end record
          false)
            record
              l : *integer
              m : float
            end record
        end case
      end record
```

En el lenguaje los tipos de datos se definen

Tipo de Datos	Representación
`char`	1 byte
`boolean`	1 byte
`short`	2 bytes
`integer`	4 bytes
`float`	8 bytes
apuntador	8 bytes

y por restricciones de la arquitectura de hardware subyacente, cada objeto del tipo básico T debe alinearse en una **dirección par múltiplo de la representación del tipo T** (note que ésto implica que la alineación de cada tipo de datos fundamental es diferente). Así mismo, la dimensión de cualquier registro debe ser múltiplo de 8 bytes.

Justifique respuestas para las siguientes preguntas, presentando la disposición en memoria de las estructuras y todos los cálculos pertinentes de manera ordenada, escribiendo en el espacio **solamente** la información que forma parte de la respuesta.

(a) **(2 puntos)** Proponga una organización para el registro que, preservando el orden de los campos, minimice el tiempo de acceso a los campos sin importar el espacio ocupado.

(b) **(2 puntos)** Proponga una organización para el registro, posiblemente reordenando los campos, que minimice el espacio ocupado pero maximice la eficiencia de acceso a los campos.

(c) **(2 puntos)** Proponga una organización para el registro, posiblemente reordenando los campos, que minimice el espacio ocupado en desmedro de la eficiencia de acceso a los campos.

(d) **(3 puntos)** ¿Cuál es la fórmula para calcular la ubicación `foo[i,j,4]` con el *mínimo* número de operaciones a tiempo de ejecución, considerando que `i`, `j` y `n` son variables cuyo valor se conocerá a tiempo de ejecución? Justifique su respuesta indicando la fórmula definitiva y cuáles de sus operaciones son efectuadas necesariamente a tiempo de ejecución.