EXAMENES RESUELTOS DE PROGRAMACION I

Comentarios por Jose Antonio Vaqué

EXAMEN DE Febrero de 1996, segunda semana

Soluciones no oficiales

1.- Dado el siguiente fragmento de código:

```
IMPLEMENTATION MODULE maquina
...
PROCEDURE Arrancar ( útil : tipoherramienta );
...
```

Si queremos exportar este procedimiento la declaración de "tipoherramienta" debe estar en:

A. El módulo de definición correspondiente.
B. El módulo de implementación.
C. El módulo donde lo usemos.
D. Usar el procedimiento sin más.

Texto base, Tema 14.2.2 Módulos de definición, Página 401 y siguientes

Solo es posible exportar procedimientos que estén declarados en los módulos de definición.

2.- Si disponemos de un espacio limitado de memoria y pretendemos manejar temporalmente datos muy extensos lo resolveremos mediante...

A. punteros.
B. índices.
C. conjuntos.
D. módulos.

Texto base, Tema 13.3. Variables dinámicas, Página 363:

> "Una manera de realizar estructuras de datos ilimitadas en Modula-2 es mediante el empleo de variables dinámicas. Una variable dinámica no se declara como tal, sino que se crea en el momento necesario, y se destruye cuando ya no se necesita. Las variables dinámicas no tienen nombre, sino que se designan mediante otras variables llamadas punteros o referencias."

Descartamos la opción D por trivial, solo es una forma de dividir el programa no los datos. Descartamos B, ya que un índice solo es una variable, Descartamos la C, ya que un conjunto no sirve realmente para datos extensos, al deber estar definidos de antemano. Solo queda la A, que además es la mejor forma de tratar con datos de volumen variable.

3.- Dada la siguiente gramática BNF:

```
Y := xA
A := {z} | pA
```

¿Qué cadenas son generadas por la misma?

A. xppzz y xxppzz
B. xppx y xzzx
C. xpz y xzp
D. xppzz y xpppz

Texto base, Tema 2.1. Notación BNF, Página 27 y siguientes

Estas definiciones son recursivas, quieren decir que Y es una letra 'x', seguida de un elemento de tipo A. Un elemento de tipo A se define como una de estas dos opciones, o bien es la letra 'z' cero o mas veces, o bien es la letra 'p' seguida de otro elemento de tipo A. De esta definición vemos que si usamos una letra 'p', podemos seguir usando tantas como queramos, pero si unamos una o varias 'z', terminamos.

Por lo tanto una cadena Y será una letra 'x', seguida de una cadena A, que será o nada, o una cadena de una o varias 'z', o una cadena de una o varias 'p' seguida de nada o de una o varias 'z'. Con esta definición, vemos que:

a) 'xppzz' es posible, pero 'xxppzz' no al comenzar por varias 'x'.
b) 'xppx' y 'xzzx' no son posibles, la 'x' solo puede estar una vez al comienzo.
c) 'xpz' es posible, pero 'xzp' no, las 'z' deben ir detrás de las 'p' siempre
d) 'xppzz' y 'xpppz' son ambas posible, comienzan por x, seguidas de varias 'p', y terminan en una o varias 'z'

4.- Los tipos opacos:

A. <u>Sólo se definen por un nombre.</u>
B. Son tipos predefinidos de Modula-2.
C. Son siempre tipos estructurados.
D. Sólo se pueden realizar mediante punteros.

Texto base, Tema 14.4.2 Tipos opacos, Página 415:

> *"Un tipo opaco se define en un "módulo de definición" escribiendo solo la parte inicial de una definición de tipo, en la forma:*
>
> *TYPE nombre_de_tipo;*

La respuesta A se corresponde con esta definición, las otras no tienen nada que ver, ya que no son predefinidos (INTEGER, REAL…), no tiene por que ser estructurados, y no tienen por que ser punteros (como se explica en el libro mas adelante en el mismo apartado, solo es así en algunas versiones del Modula-2).

5.- Los vectores abiertos:

A. <u>Sólo se pueden utilizar como argumentos de subprogramas.</u>
B. Simplifican las condiciones de contorno.
C. Son idénticos a las ristras o string.
D. Utilizan siempre un centinela.

Texto base, Tema 11.6 Argumentos de tipo vector abierto, Página 31 y siguientes

La respuesta A es siempre cierta, no se pueden usar en otro lugar. La B no tiene nada que ver, la C no es cierta, ya que los string son solo un tipo de vector, y la D tampoco, no usan centinela sino la función HIGH.

6.- Los efectos secundarios se producen:

A. <u>Por las reglas de visibilidad de bloques.</u>
B. Con el paso de argumentos por valor.
C. Con paso de argumentos por referencia.
D. Cuando crece el número de argumentos.

Texto base, Tema 7.6.2. Efectos secundarios, Páginas 177 y siguientes:

> "Cuando un programa modifica alguna variable externa, se dice que está produciendo efectos secundarios o laterales."

Este apartado esta mal explicado en el libro, ya que solo se refiere al uso de variables externas a una función dentro de la misma, y se pueden producir también por el paso indebido de variables por referencia en lugar de por valor, pero como esto no está así en el libro, solo queda como posible la respuesta A.

7.- De los datos encapsulados y los datos persistentes podemos afirmar:

A. <u>Son completamente diferentes.</u>
B. Resuelven el mismo problema.
C. Los datos persistentes son también datos encapsulados.
D. Los datos encapsulados son siempre datos persistentes.

Texto base, Tema 13.4 Datos persistentes, página 373 y siguientes:

> "Se denominan datos persistentes aquellos que conservan su valor entre ejecuciones sucesivas de los programas que operan con ellos."

Texto base, Ttema 14.4.3 Datos encapsulados, página 416

> "Cuando en un programa sólo hay que manejar una única variable de este tipo, se puede conseguir fácilmente la ocultación total sin necesidad de recurrir a tipos opacos, si dicha variable única es interna al módulo en el que se desarrolla el tipo abstracto. "

Son dos tipos de datos completamente diferentes, los encapsulados son para mejorar el diseño del programa, y los persistentes para conservar su valor entre ejecuciones separadas del programa.

8.- Dada la siguiente declaración de procedimiento:

```
PROCEDURE Proceso( VAR x : INTEGER; y : INTEGER );
```

y la declaración de variable:

```
VAR a : INTEGER;
```

¿Cuántas de las llamadas siguientes son correctas?:

```
Proceso( a, a );          Proceso( a, a MOD 2 );
Proceso( a MOD 2, a );    Proceso( a MOD 2 , a MOD 2 );
```

A. 2.
B. 1.
C. 3.
D. 4.

Texto base, Tema 7.4.1 Paso de argumentos por valor, página 170

"Los argumentos reales en la llamada al subprograma pueden darse en general en forma de expresiones, cuyos tipos de valor deben ser compatibles en asignación con los tipos de los argumentos formales. "

Texto base, Tema 7.4.2 Paso de argumentos por referencia, página 171

"Si un argumento se pasa por referencia, ya no será válido usar como argumento real una expresión. El argumento real usado en la llamada debe ser necesariamente una variable del mismo tipo. Esta variable será utilizada en el subprograma como si fuera suya."

La función usa dos argumentos, el primero se pasa por referencia y el segundo por valor. Las llamadas al argumento por valor todas son válidas, pero por referencia solo se pueden pasar variables simples, por ello las dos primeras con válidas, y las dos siguientes no son válidas.

9.- Dado el siguiente fragmento de código:

```
TYPE PTR = POINTER TO Dato;
Dato = RECORD
  x : INTEGER;
  sgte : PTR
END;
VAR a, b, c : PTR;
...
NEW( a ); NEW( a^.sgte ); b := a^.sgte; NEW( b^.sgte ); c := b;
a^.x := 1; b^.x := 3; c^.x := 2;
...
```

El valor de a^.sgte.x es:

A. 2.
B. 1.
C. 3.
D. Indeterminado.

Texto base, Tema 13.3 Variables dinámicas, página 363 y siguientes

Sigamos el programa para ver el resultado. Primero se define una estructura de registros enlazados mediante punteros, y se crean tres variables puntero para su manejo. Luego se ejecutan estas operaciones (asumo que las posiciones de memoria son la M1, M2, M3…, sin considerar su tamaño real)

	a	b	c	M1		M2		M3	
				a.x	a.sgte	b.x	b.sgte	c.x	c.sgte
	???	???	???	???	???	???	???	???	???
NEW(a)	M1	???	???	???	???	???	???	???	???
NEW(a^.sgte)	M1	???	???	???	M2	???	???	???	???
b := a^.sgte	M1	M2	???	???	M2	???	???	???	???
NEW(b^.sgte)	M1	M2	???	???	M2	???	M3	???	???
c := b	M1	M2	M2	???	M2	???	M3	???	???
a^.x := 1	M1	M2	M2	1	M2	???	M3	???	???
b^.x := 3	M1	M2	M2	1	M2	3	M3	3	???
c^.x := 2	M1	M2	M2	1	M2	2	M3	2	???

El punto clave es ver que se asigna **c** directamente a **b**, no a **b^.sgte**, por lo que c y b pasan a apuntar a la misma zona de memoria (para entendernos, ahora son la misma variables, pero con dos nombres diferentes).

10.- Tras la ejecución del siguiente fragmento de código:

```
...
TYPE mueble = (silla, mesa, cama, sofa, galan);
VAR a : mueble; b : ARRAY [1..10] OF mueble;
BEGIN
a := silla;
b[ORD( a )] := a;
...
```

La variable b pasa a tomar el valor:

A. **Error, acceso fuera del rango del vector.**
B. ORD(silla).
C. Error, no se puede usar la función ORD para acceder al elemento.
D. galan.

Texto base, Tema 11.2 Vectores, página 288 y siguientes, y tema 9.2. Tipos enumerados, página 224:

> *"Este orden se define de forma implicita e impone que el primer elemento de la lista ocupe la posición 0, el siguiente la 1…"*

Primero se define un tipo enumerado, de las que se crean una variable y un vector. Luego a la variable **a** se le asigna el primer elemento de la lista, por lo tanto ORD(a) valdrá CERO. Luego se asigna al elemento ORD(a) del vector (ORD(a) proporciona un número, que es posible usar como índice sin problemas, lo que invalida la respuesta C), el contenido de **a**, que es **silla** (no ORD(silla) lo que invalida la respuesta B). Al ser un vector de elementos del mimo tipo que la variable **a** es correcto, pero en la definición del vector se le dice que sus elementos van del 1 al 10, por lo que el elemento CERO no es un valor correcto para el índice. La respuesta D es absurda.