EXAMENES RESUELTOS DE PROGRAMACION I

Comentarios por Jose Antonio Vaqué

EXAMEN DE Septiembre de 1999

SOLUCIONES OFICIALES

1. Dado el siguiente fragmento de código:

```
CASE j OF
  enero..junio:INC(j);|
  diciembre..julio:DEC(j);
  ELSE
END;
```

A.- la variable j tiene que ser de tipo ordinal para ser correcto
B.- es necesaria alguna sentencia tras ELSE
C.- la sentencia ELSE no se puede usar dentro de una instrucción CASE
D.- los rangos establecidos son incorrectos

Texto base, apartado 10.2.1 Sentencia CASE,, página 260 y siguientes..
Texto base, apartado 9.2.2 Uso de tipos enumerados, página 22:.

> "Otras dos operaciones aplicables a los tipos ordinales, y por tanto a los enumerados, corresponden a los procedimientos predefinidos INC y DEC…."

La respuesta A es cierta según lo anterior. La B es falsa, ya que una sentencia vacía es una sentencia correcta, y tras el ELSE hay una sentencia vacía. Como un CASE si admite ELSE, es falsa la C. La D es falsa, ya que se admiten subrangos en los CASE.

2. Para que en Modula-2, la siguiente definición de tipo sea la de una tabla:

```
TYPE TipVect= ARRAY TipInd OF TipElem
```

A.- TipInd debe ser cualquier tipo predefinido y TipElem RECORD
B.- TipInd debe ser un tipo ordinal definido por el usuario y TipElem de cualquier tipo
C.- TipInd debe ser cualquier tipo ordinal y TipElem RECORD
D.- TipInd debe ser RECORD y TipElem de cualquier tipo

Texto base, apartado11.2.1 Declaración de vectores, página 289:

> *"En Modula-2, una estructura de tipo vector se declara de la siguiente forma:*
>
> *TipoVector = ARRAY TipoIndice OF TipoElemento*
>
> *…El TipoIndice… puede ser un escalar enumerable, un rubrango o cualquiera de los tipos predefinidos CHAR o BOOLEAN. Finalmente TipoElemento puede ser de cualquier tipo…"*

La respuestas A y D se descartan por el **TipInd**, que no se puede usar (ni puede ser cualquiera ni un registro). La B se descarta porque debe ser un tipo ordinal, pero no tiene porque ser definido por el usuario.

3. Dado el siguiente procedimiento:

```
PROCEDURE proc(a,b:INTEGER);
VAR aux:INTEGER;
BEGIN
  aux:=a+b+c;
  c:=aux;
END proc;
```

Para que fuese puro:

A.- la variable c se debería pasar por referencia
B.- Es ya un procedimiento puro
C.- bastaría con pasar las variables a y b por referencia
D.- Todas las variables, incluida aux, deben pasarse por referencia

Texto base, apartado 8.1.3. Acciones abstractas. Procedimientos, página 193:
"…procedimientos puros, entendiendo por ello que no produzcan efectos laterales o secundarios."
Texto base, apartado 7.6.2 Efectos secundarios, página 177:
"Cuando un subprograma modifica alguna variable externa, se dice que está produciendo efectos secundarios o laterales…"

Como **c** no está definida en el procedimiento, debe ser global, y esto descarta la respuesta B. Si solo pasamos **a** y **b** por referencia, no evitamos usar **c**, por lo que descartamos la respuesta C. La variable **aux** solo se usa dentro del procedimiento, por lo que es inútil pasarla como parámetro, descartamos D. Y evidentemente la A es la correcta, lo que de todas formas se ve a primera vista.

4. Sabiendo que en Modula-2 existe el tipo predefinido BOOLEAN:

```
TYPE BOOLEAN=(FALSE, TRUE)
```

Se puede afirmar:

A.- ORD(FALSE)=1
B.- El tipo BOOLEAN no es un tipo ordinal
C.- Con los tipos predefinidos no se puede utilizar la ORD
D.- ORD(FALSE)=0

Texto base, apartado 9.2.1 Definición de tipos enumerados, página 224:
"Este orden se define de forma implícita e impone que el primer elemento de la lista ocupa la posición 0, el siguiente la 1, y así sucesivamente…"

El primer elemento es FALSE, por lo que ocupa la posición 0, lo que invalida la respuesta A pero valida la D. Los tipos enumerados son tipos ordinales (que sus elementos tiene un orden preestablecido, como los de tipo INTEGER o los CHAR), y ORD se puede usar con cualquier tipo ordinal. De todas formas, en la página 226 también dice literalmente que ORD(FALSE) = 0.

5. La compilación segura:

A.- Tiene como objetivo comprobar la compatibilidad de tipos
B.- Necesita un modulo de definición
C.- Produce un programa objeto más eficiente
D.- Mejora la reutilización

Texto base, apartado 14.1.2 Compilación separada, página 398:
"COMPILACION SEGURA: Al compilar un fichero fuente el compilador comprueba que el uso de los elementos de otros módulos es consistente con la interfaz. "
Texto base, apartado 14.2.2 Módulos de definición, página 401 y siguientes

La descomposición modular pretende la compilación segura, utilizando el módulo de definición como intermediario entre el programa que lo llama y la propia implementación de las funciones. Si descartamos A, C y D por que no se ajustan a la definición, solo queda la B de todas formas.

6. Dado el siguiente fragmento de código:

```
TYPE t1=RECORD c1, c2:REAL; END;
    t2=POINTER TO REAL;
    t3=POINTER TO t1;
VAR a:t1; b:t2; c:t3;
...
NEW(b); NEW(c);
```

La asignación correcta es:

A.- b:= c^.c1;
B.- c^.c1:=a.c2;
C.- a:=c;
D.- c.c1:=a.c2;

Texto base, apartado 13.3 Variables dinámicas. Página 363 y siguientes.

Es fundamental saber diferencia entre una variable de tipo puntero, y lo que apunta esa variable. Si vemos el programa, **a** es una variable de tipo registro, **b** es un puntero a un numero real, y **c** es un puntero a una variable de tipo registro. A es incorrecta, pues se asigna a un puntero un real, debería usarse **b^:=c^.c1;** La respuesta B es correcta, se asigna a una variable real otra variable real. C es incorrecta, pues se asigna a un registro un puntero, debería usarse **a:=c^;** D es incorrecta, pues no se puede usar **c.c1**, ya que c es un puntero, debería usarse **c^.c1:=a.c2;**

7. Dado la siguiente declaración de variables:

```
    VAR c1:T1; c2:T2;
```

Después de ejecutar

```
    Leer(c1); Leer(c2);
```

las variables c1 y c2 toman los valores “esto” y “aquello” respectivamente.
A la vista del resultado, el procedimiento Leer tendrá como argumento:

A.- (VAR a: ARRAY OF CHAR)
B.- (a:T1,T2)
C.- (VAR a:T1,T2)
D.- (a: ARRAY OF CHAR)

Texto base, apartado 11.5 Vectores de caracteres: Ristras (String). Páginas 308 y siguientes.

Si se obtienen cadenas, deben pasarse cadenas a la función. Las cadenas se pasan como argumentos de tipo vector abierto (descartamos B y C por ser respuestas absurdas). Si tras la llamada a la función, cambia el valor de la variable de llamada, debe llamarse por referencia (respuesta A) no por valor (respuesta D).

8. Después de ejecutar el siguiente fragmento de código:

```
VAR pt1, pt2, pt3: POINTER TO REAL;
...
NEW(pt1); pt1^:=3.0; pt2:=pt1; pt3:=pt2;
DISPOSE(pt2);
```

A.- Sólo se libera pt3
B.- Sólo se libera pt3 y pt2
C.- Se produce un error, y no se libera la memoria
D.- Se liberan los punteros pt1, pt2 y pt3

Texto base, apartado 14.4.2 Tipos opacos. Página 415 y siguientes.

Como se han igualado los tres punteros, los tres apuntan a la misma zona de memoria. Cuando se libera esa zona, todo lo que apunta a esa zona automáticamente se libera.

9. La definición completa de un tipo opaco que se declara en un módulo de definición debe completarse en:

A.- El propio módulo de definición
B.- El módulo de implementación correspondiente
C.- El módulo principal del programa
D.- La declaración también se hace en el módulo de implementación

Texto base, apartado 14.4.2 Tipos opacos. Página 415:

> *“Un tipo opaco se define en un “módulo de definición”…La definición completa del tipo ha de hacerse en el correspondiente “módulo de implementación”.”*

10. Dado el siguiente procedimiento:

```
PROCEDURE
Calcular (VAR A: INTEGER; B: INTEGER) : INTEGER;
BEGIN
    A := B MOD 2; B := A * A + 3 * B - 6; RETURN A+B
END Calcular;
```

El valor de X tras ejecutar X:=4; X:=Calcular(X,X); será:

A.- 6
B.- 0
C.- 4
D.- 12

Hay una mezcla de cosas para intentar despistarnos, ya que normalmente el uso del VAR con variables globales induce a errores, pero en este caso es muy sencillo, pues toma el valor de retorno de la función. Realizaré una traza de programa para aclararlo:

SENTENCIA	COMENTARIO	X	A	B
1) X:=4;	Se da valor a X	4	--	--
2) X:=Calcular(X,X);	Se llama a la función Calcular con (4,4)	4	4	4
3) A := B MOD 2;	Se da el valor 0 a la variable **A**, que al ser VAR es X	0	0	4
4) B := A * A + 3 * B – 6;	Se da el valor 6 a la variable **B**	0	0	6
5) RETURN A+B	Se retorna el valor 6	0	0	6
2bis) X:=6;	Que se asigna a la variable X	6	--	--

EJERCICIO DE PROGRAMACIÓN

Construir un dato encapsulado que sea una tabla de 50 elementos, en la que se almacenan los datos de una persona: nombre, apellido1, apellido2, dirección y teléfono. Las operaciones serán añadir un elemento, eliminar un elemento y un procedimiento de búsqueda selectiva por nombre o por número de teléfono. Si se encuentra la persona en la tabla se devolverá cierto y se mostrará toda la información disponible de esa persona y si no se devolverá falso.

Faltan muchos datos necesarios, como por ejemplo, cuando borras un registro ¿cual borras?, o si das de alta un registro, ¿necesitas saber cual has dado de alta? Como en el examen se trata de demostrar que sabes manejar cadenas y vectores principalmente, solo desarrollo eso.

Como no está muy claro que quieren decir con los de “dato encapsulado”, he creado un dato de tipo opaco para la estructura, de la que solo se conoce como es una persona, no como están organizadas (vector o estructuras enlazadas por punteros por ejemplo).

```
DEFINITION MODULE Pers;
   CONST LMAX = 25;                      (* Longitud máxima de un dato *)
   TYPE TBuscar = (BNombre, BTelefono);  (* Tipos de búsquedas *)
        Persona = RECORD                 (* Para guardar una persona *)
           nom, ap1, ap2, dir, tel : ARRAY [0..LMAX] OF CHAR;
        END;
   PROCEDURE Alta(dato:Persona);
   PROCEDURE Baja(reg:INTEGER);
   PROCEDURE Buscar(tipo:TBuscar;que:ARRAY OF CHAR;VAR reg:INTEGER):BOOLEAN;
END Pers.
```

```
IMPLEMENTATION MODULE Pers;
FROM InOut IMPORT WriteString,WriteInt,WriteLn;
   CONST NMax = 50;                                (* Máximo de personas que guardamos *)
   VAR VPos : INTEGER;                             (* Puntero *)
       VPersona : ARRAY [1 .. NMax] OF Persona;    (* Las personas que manejamos *)
   PROCEDURE Alta(dato:Persona); (* Alta de una persona *)
   BEGIN
      IF (VPos = NMax) THEN
         WriteString("Estructura llena, no puede introducir otro elemento");WriteLn;
      ELSE
         INC(VPos);   VPersona[VPos] := dato;
      END;
   END Alta;
   PROCEDURE Baja(reg:INTEGER); (* BAJA de una persona *)
   VAR i:INTEGER;
   BEGIN
      IF (reg < 1) OR (reg > VPos)  THEN
         WriteString("Elemento fuera de rango, no se puede borrar");WriteLn;
      ELSE
         FOR i:=reg TO VPos-1 DO
            VPersona[i] := VPersona[i+1];
         END;
         DEC(VPos);
      END;
   END Baja;
   PROCEDURE Buscar(tipo:TBuscar;que:ARRAY OF CHAR):BOOLEAN; (* Buscar una persona *)
   VAR i:INTEGER;  e:BOOLEAN;
   BEGIN
      i:=0;  e:=FALSE;
      REPEAT
         INC(i);
         CASE tipo OF
            BNombre   : e:= Comparar(que,VPersona[i].nom); |
            BTelefono : e:= Comparar(que,VPersona[i].tel);
         END;
      UNTIL (e = TRUE) OR (i = VPos);
      IF (e = TRUE) THEN
         WriteString("Encontrado en posición: ");WriteInt(i,2); WriteLn;
         WriteString("   Nombre....: ");WriteString(VPersona[i].nom);WriteLn;
         WriteString("   Apellido 1: ");WriteString(VPersona[i].ap1);WriteLn;
         WriteString("   Apellido 2: ");WriteString(VPersona[i].ap2);WriteLn;
         WriteString("   Dirección.: ");WriteString(VPersona[i].dir);WriteLn;
         WriteString("   Teléfono..: ");WriteString(VPersona[i].tel);WriteLn;
      END;
      RETURN e;
   END Buscar;
   PROCEDURE Comparar(uno,dos:ARRAY OF CHAR):BOOLEAN; (*Comparar dos cadenas *)
   VAR i,max1,max2:INTEGER;  e:BOOLEAN;
   BEGIN
      max1 := HIGH(uno);   max2 := HIGH(dos);   e := TRUE;   i:= 0;
      LOOP
         IF (uno[i] <> dos[i]) THEN   e := FALSE;   EXIT;   END;
         INC(i);
         IF (i > max1) OR (i > max2) THEN   EXIT;   END;
      END;
      RETURN e;
   END Comparar;
BEGIN
   VPos := 0; (* Inicializamos el puntero a cero *)
END Pers.
```