EXAMENES RESUELTOS DE PROGRAMACION I

Comentarios por Jose Antonio Vaqué

EXAMEN DE septiembre 1995

Soluciones no oficiales

1.- Dado el siguiente fragmento de código
VAR d1:INTEGER;d2:REAL;
PROCEDURE Contar(VAR c:REAL;h:INTEGER):INTEGER;
La invocación correcta será:

A. d1:=Contar(Contar(d2,d1),d1);
B. d1:=Contar(d2,Contar(d1,d2));
C. d1:=Contar(34,Contar(d2,34));
D. d1:=Contar(d2,Contar(d2,34));

Al procedimiento Contar se le pasan dos parámetros, un REAL y un INTEGER, y devuelve un INTEGER. Analizamos los casos:

a) Se le pasa un INTEGER (devuelto por Contar) y un INTEGER.
b) Se le pasa un REAL, y un INTEGER (devuelto por Contar), pero en esa segunda llamada se le llama con INTEGER y REAL
c) Se le pasa un INTEGER y un INTEGER (devuelto por Contar)
d) Se le pasa un REAL y un INTEGER (devuelto por Contar), y en esa segunda llamada se le llama con REAL e INTEGER.

2.- Los tipos enumerados y las formaciones:

A. Son tipos acotados
B. Sirven para lo mismo
C. Son tipos estructurados
D. Utilizan índices

Los tipos acotados son lo que tienen un límite inferior y uno superior. Claramente B no es correcto ya que no sirven para lo mismo. C no es correcto, ya que los tipos estructurados son los registros. D no es correcto ya que solo las formaciones usan índices.

3.- ¿Qué carácter imprimirá por pantalla la siguiente sentencia?
Write(CHR(ORD("C")+ORD(CHR(ORD("d")-ORD("a")))));

A. D (mayúscula)
B. e (minúscula)
C. F (mayúscula)
D. c (minúscula)

Vamos resolviendo por los paréntesis mas interiores hacia afuera:

CHR(ORD("C")+ORD(CHR(**ORD("d")-ORD("a")**)))	ORD("d")-ORD("a")	Entre "a" y "d" hay dos letras, por lo que será igual a 3.
CHR(ORD("C")+**ORD(CHR(3))**)	ORD(CHR(3))	Se queda igual, 3
CHR(**ORD("C")+3)**)	ORD("C")+3)	Serán 3 mas que el código de la letra C
CHR(Código de la C mas 3)		Es la F, que está tres posiciones tras la C.

4.- Las reglas de visibilidad en Modula-2 permiten:

A. Consultar y modificar las variables de los bloques más internos.
B. Consultar y modificar las variables de los bloques más externos.
C. Consultar las variables de los bloques más internos.
D. Consultar las variables de los bloques más externos.

Texto base, apartado 7.5. Visibilidad. Estructura de bloques, páginas 175:

"Cualquier bloque tiene visibilidad hacia los bloques externos, pero nunca hacia los interiores a él mismo. "

5.- Analizar el número de instrucciones que se ejecutarán en el peor caso en este fragmento de programa.

```
A:=7;
IF A<C THEN      A:=C;B:=C;C:=3
ELSE             C:=2*A;B:=C-5;
END
```

A. 4
B. 5
C. 6
D. 7

Texto base, Tema 6.4.2. Análisis de programas. Páginas 146 y siguientes

Esta muy mal explicado en el libro, pero si habéis estudiado ETC I, pensar en que eso se traduce a instrucciones en código máquina. Para verlo mejor, vamos a escribirlo en esta forma, colocando el número de instrucciones que se cuentan en cada parte

```
A:=7;              1
IF A<C THEN        2 (una por el propio IF y otra por la comparación)
   A:=C;           1
   B:=C;           1
   C:=3;           1
ELSE               0 (va incluida en el IF)
   C:=2*A;         2 (una por el 2 y otra por el A)
   B:=C-5;         2 (ídem)
END;               0 (va contada en el propio IF)
```

Se aprecia que el primer bloque del IF tiene 3 instrucciones, de complejidad 3, mientras que el segundo solo tiene 2, pero de complejidad 4, por ello en el pero caso se ejecutarían las siguientes líneas:

```
A:=7;              1
IF A<C THEN        2
ELSE
   C:=2*A;         2
   B:=C-5;         2
END;
```

6.- La técnica de refinamientos sucesivos es una estrategia de desarrollo de programas:

A. Descendente
B. Abstracta
C. Ascendente
D. Horizontal

Texto base, Tema 8.2.1. Desarrollo descendente. Página 194Análisis de programas. Páginas 146 y siguientes:

> *La estrategia de desarrollo descendente (en inglés "Top-Down"), es simplemente el desarrollo por refinamientos sucesivos, teniendo en cuenta además la posibilidad de definir operaciones abstractas.*

Si se entiende el tema de los desarrollos descendente y ascendente, esta pregunta es sencilla, aunque toda la parte de metodología de desarrollo no está muy bien explicada bajo mi punto de vista.

7.- Con qué objetivos se definen los subprogramas:

A. Aumentar la claridad.
B. Aumentar la potencia de cálculo.
C. Aumentar la eficiencia.
D. Aumentar la robustez.

Texto base, Tema 8.2.1. Desarrollo descendente, página 194

> *"El beneficio obtenido es, como cabría esperar, una mejora en la claridad del programa. Hay que decir que esto implica un costo ligeramente mayor en términos de eficiencia...."*

O nos sabemos esto de memoria, o vamos a ir por descartes, ya que no tiene mas sentido:

- Mediante funciones no podemos aumentar la potencia de cálculo, aunque si no hay una forma estándar de calcular algo, la podemos implementar, realmente esto no aumenta la potencia.
- No aumentamos la eficiencia, ya que es justo al contrario, se produce una ligera ineficiencia, al tener que ir llamando a las funciones.
- Es más dudoso el apartado de la robustez, que depende de que el programa funcione bien o mal, no de cómo se escriba. Pero usando funciones, podemos aumentar la facilidad de modificación del programa, lo que indirectamente incrementa su robustez. Esta es una de las características que hacen más robustos los programas orientados a objeto.

8.- El desarrollo ascendente:

A. Mejora la robustez
B. Evita la verificación.
C. <u>Facilita la reutilización.</u>
D. Disminuye la complejidad.

Texto base, Tema 8.2.7. Desarrollo ascendente, página 210:

> *"La metodología de desarrollo ascendente (en inglés "Botton-Up") … La técnica tiene cierta analogía con el desarrollo de subprogramas pensando en su reutilización posterior."*

O nos sabemos esto de memoria, o vamos a ir por descartes, ya que no tiene mas sentido:

- La robustez depende de que el programa funcione bien o mal, no de cómo se escriba. En este caso, no interviene para nada
- La verificación es necesaria siempre que añadamos algo al programa. Un error muy común es realizar una ligera modificación, y no probarla, asumiendo que funciona bien. Esto es una fuente segura de errores.
- La complejidad depende mas del enunciado que del programa, un problema complejo requiere un programa complejo (aunque muchos principiantes hacen programas complejos para problemas sencillos),

9.- La transparencia referencial se da cuando:

A. Un subprograma produce siempre los mismos resultados independientemente de los argumentos con que se le invoque.
B. Se produce doble referencia en un subprograma.
C. <u>Un subprograma produce los mismos resultados siempre que se le invoque con los mismos argumentos.</u>
D. A un programa se le pasan solamente argumentos por referencia.

Texto base, Tema 8.1.2.Funciones. Argumentos, Página 191:

> *"…la transparencia referencial significa que la función devolverá siempre el mismo resultado cada vez que se la invoque con los mismos argumentos."*

Otro trozo mas a sabemos de memoria.

EJERCICIO

Codificar un procedimiento en Modula-2 que calcule la suma de dos matrices de 4x4 elementos. Este procedimiento se realizara en un módulo diferente del Principal y se deberá usar este.

```
DEFINITION MODULE Matrices;
   TYPE Matriz = ARRAY [1..4],[1..4] OF INTEGER;
   PROCEDURE Sumar(VAR M1,M2,M3:Matriz);
END Matrices.
```

```
IMPLEMENTATION MODULE Matrices;
   (* Suma M1 y M2, poniendo el resultado en M3 *)
   PROCEDURE Sumar(VAR M1,M2,M3:Matriz);
   VAR i,j : INTEGER;
   BEGIN
     FOR i:=1 TO 4 DO
         FOR j:=1 TO 4 DO
             M3[i,j] := M1[i,j] + M2[i,j];
         END;
     END;
   END Sumar;
END Matrices.
```

```
MODULE Examen;
   FROM InOut IMPORT WriteLn,WriteInt;
   IMPORT Matrices;

VAR
   M1, M2, M3 : Matrices.Matriz;
   i,j : INTEGER;

BEGIN
     FOR i:=1 TO 4 DO     (* Carga unos valores, para ver que funciona *)
         FOR j:=1 TO 4 DO
             M1[i,j] := 1; M2[i,j] := 2;
         END;
     END;

     Matrices.Sumar(M1,M2,M3);

     FOR i:=1 TO 4 DO     (* Imprime el resultado *)
         FOR j:=1 TO 4 DO
             WriteInt(M3[i,j],3);
         END;
         WriteLn;
     END;

     WriteLn;

END Examen.
```

Este ejercicio es sencillo, se trata de plantear los módulos de definición y de implementación, y luego llamarlos en el programa principal. Se podría haber evitado los dos bucles FOR anidados del programa principal, ya que no nos piden esto en el planteamiento, pero si no no hay forma de verificar que funciona.

Como ejercicio, podremos plantearnos lo siguiente:

- Habría que realizar mas operaciones con las matrices, como el producto por un escalar, el producto de matrices, el calculo del determinante, etc.
- Mas complejo es buscar la forma de definir las matrices de tamaño variable, en lugar de solo tratar de 4x4.
- Mucho más complejo, permitir cualquier tipo de elemento, no solo enteros, incluso numero complejos.