EXAMENES RESUELTOS DE PROGRAMACION I

Comentarios por Jose Antonio Vaqué

EXAMEN DE Febrero 1994 - 1ª Semana - Tipo D

Soluciones no oficiales

1.- El siguiente fragmento de programa:

```
i := 0;
WHILE (i < 13) DO
  LOOP
    WriteString("HOLA");
    WriteLn;
    INC( i );
  END;
END;
```

a) **Da un error de ejecución.**
b) Imprime la palabra HOLA de forma ininterrumpida.
c) Imprime 15 veces la palabra HOLA.
d) Imprime 14 veces la palabra HOLA.

Texto base, apartado 2.4.1, página 35:

> *"Cuando se realiza una operación con enteros se debe tener en cuenta su rango disponible en el computador que se está utilizando. Si se produce un resultado fuera de rango del computador se producirá un error. En algunos compiladores este tipo de error no se indican y puede ser difícil su detección."*

El programa contiene un bucle WHILE, dentro del cual se encuentra un bucle LOOP, que no tiene salida mediante un EXIT, por lo que se ejecutará indefinidamente. Dentro de ese bucle LOOP, se imprime la palabra "HOLA" ininterrumpidamente, pero también se incrementa una variable. Cuando esta variable alcance el máximo permitido para su tipo de dato, al intentar incrementar la variable en uno provocará un error de ejecución por desbordamiento del valor de la variable. Lamentablemente, este comportamiento depende mas del compilador que de otra cosa, hay compiladores que no producen error al no verificar este máximo, y otros que si. Por ejemplo ni el compilador FST proporcionado por la escuela, ni la versión gratuita del Excelsior lo verifican.

2.- Dado el procedimiento:

```
PROCEDURE Uno(VAR x, y, z : INTEGER);
BEGIN
  IF x < y THEN
    z := y - x;
  END;
  IF x < z THEN
    y := z - x;
  END;
END Uno;
```

el resultado después de la llamada Uno(3, 4, 5) será:

a) x = 3; y = 2; z = 1
b) **x = 3; y = 4; z = 1**
c) Hay errores de compilación.
d) Ninguna respuesta anterior es correcta.

Texto base, apartado 5.3.2., página 108.

Al procedimiento se pasan los valores por referencia, por lo que se mantienen a la salida. Cuando se entra en el procedimiento, se compara x menor que y lo que es cierto (IF 3 < 4 THEN), asignando a z su diferencia (z = 4 – 3). Luego se compara si x es menor de z (IF 3 < 1), lo que ya no es cierto, y no se cambia el valor de y.

3.- En general, los objetivos de la programación por orden de importancia son:

a) 1º Eficiencia 2º Claridad 3º Corrección
b) 1º Corrección 2º Eficiencia 3º Claridad
c) 1º Eficiencia 2º Corrección 3º Claridad
d) 1º Corrección 2º Claridad 3º Eficiencia

Texto base, apartado 1.2.2., página 9:

> *"Si se trata de establecer una importancia relativa entre los distintos objetivos, habría que considerar como prioritaria la corrección. Piénsese, por ejemplo, que un programa de contabilidad no es aceptable si no calcula correctamente los saldos de las cuentas.*
>
> *A continuación debe perseguirse la claridad, que como ya se ha indicado es necesaria para poder realizar modificaciones, o simplemente para poder certificar que el programa es correcto. En realidad el objetivo de claridad va ligado al de corrección. Es prácticamente imposible asegurar que un programa es correcto si no puede ser entendido claramente por la persona que lo examina.*
>
> *Tal como se ha dicho antes, la claridad facilita la tarea de realizar modificaciones cuando las necesidades así lo exijan. Puede afirmarse que esto ocurre siempre con todos los programas que tienen un cierto interés. Finalmente ha de atenderse a la eficiencia. Este objetivo, aunque importante, sólo suele ser decisivo en determinados casos. En muchas situaciones el aumento de capacidad de los computadores a medida que avanza la tecnología va permitiendo utilizar de manera aceptable, desde el punto de vista económico, programas relativamente menos eficientes. "*

Esta es la típica pregunta para la que hay que saberse estos temitas auxiliares de memoria, ya que la lógica no sirve para nada. Está claro que un programa debe ser ante todo CORRECTO (que haga lo que debe de hacer, aunque esto se conoce como programa ROBUSTO). Un programa CLARO es el que está escrito de forma clara, pero eso depende no solo del que lo escribe, sino del que lo lee. En general, es difícil modificar un programa de otro, por muy claro que esté escrito. Un programa EFICIENTE es el que aprovecha mejor los recursos de la máquina, lo que es difícil en lenguajes de alto nivel, ya que luego el compilador hará lo que quiera. No hay que confundirlo con EFICACIA, que indica la velocidad a la que se ejecuta el programa.

4.- Dada la declaración:

```
VAR
  letra : CHAR;
  peso, talla : INTEGER;
```

y la expresión:

```
(talla < 5) AND NOT (peso > 100) AND (letra < "H") AND (letra > "L")
```

a) El resultado depende de los valores de las variables.
b) El resultado es TRUE.
c) Hay errores en la expresión.
d) El resultado es FALSE.

Texto base, apartado 5.2., página 104.

En una cadena de AND, si uno es falso, todo es falso. Al contrario, en una cadena de OR, si uno es cierto todo es cierto. El programa comienza por (talla < 5) AND NOT (peso > 100), lo que claramente depende de los valores de las variables, pero continua con (letra < "H") AND (letra > "L"), y ninguna letra puede ser a la vez menor de H y mayor de L, por lo que será falso, y toda la expresión será falsa.

5.- El empleo de constantes con nombre permite…

a) Reducir el tamaño del programa.
b) Parametrizar el programa.
c) Aumentar la eficacia del programa.
d) Ninguna respuesta anterior es correcta.

*Texto base, apartado 4.4.5., página 91: "…Este tipo de valores se denominan a veces **parámetros** del programa…"*

Otra vez la típica pregunta para la que hay que saberse estos temitas auxiliares de memoria, ya que la lógica no sirve para nada. Es la primera vez que veo llamar parámetros a las constantes con nombre, pero es cierto que en rigurosa aplicación del concepto, las constantes **PUEDEN** servir para parametrizar el programa, aquí es donde está mi discrepancia.

6.- El resultado: 32,357 se produce con la ejecución de las siguientes sentencias:

a) WriteInt(32, 10); Write(“,”); WriteInt(715 DIV 2, 4);
b) WriteReal(3235.7E-2, 15);
c) WriteInt(1000 DIV 265, 7); WriteString(“2,”); Write(‘3’); WriteInt(32 + 25, 2);
d) Ninguna respuesta anterior es correcta.

En esta pregunta hay dos posibles respuestas, dependiendo de como se plantee cada uno la pregunta, si de forma estricta o con un poco de manga ancha. Vemos lo que imprime cada uno, simbolizando el espacio en blanco mediante “_” :

	Sentencia	**Imprime**	**Resultado final**
a)	WriteInt(32, 10); Write(“,”) WriteInt(715 DIV 2, 4)	“_____32” “,” “_357”	“_____32,_357”
b)	WriteReal(3235.7E-2, 15)	“3.23500E+01”	“3.23500E+01”
c)	WriteInt(1000 DIV 265, 7) WriteString(“2,”) Write(‘3’) WriteInt(32 + 25, 2)	“_____3” “2,” “3” “57”	“_____32,357”

Por lo tanto, o la respuesta es la c) si no consideramos los espacios iniciales, o es la d) si los consideramos. Ante esto, hay que esperar la respuesta oficial, y actuar en consecuencia.

7.- Indicar que operación es incorrecta con las declaraciones siguientes:
TYPE TipoNotas = (Algebra, Calculo, Dibujo, Fisica, Quimica);
VAR Nota : TipoNotas;

a) FOR Nota := Calculo TO Fisica DO …
b) Nota := Nota + 1;
c) DEC(Nota);
d) IF Nota >= Dibujo THEN …

Texto base, apartado 9.2. Tipos enumerados

Aunque expresamente existen ejemplos de los puntos a, c y d, no se dice que el b no sea posible, pero a un tipo enumerado solo se accede por sus enumeraciones, y en este caso “1” no es uno de sus elementos. No es posible aplicar el “+”, hay que usar siempre INC.

8.- La eficiencia de un programa…

a) Depende solamente del tiempo que tarde en ejecutarse.
b) Depende de la calidad del interfaz con el usuario del programa.
c) Depende de los recursos que consume durante su ejecución.
d) Depende de si alcanza o no los objetivos de cálculo prefijados.

Texto base, apartado 1.2.2., página 8:

> “EFICIENCIA: Una tarea de tratamiento de información puede ser programada de muy diferentes maneras sobre un computador determinado, es decir, habrá muchos programas distintos que producirán los resultados adecuados. Algunos de estos programas serán más eficientes que otros. Los programas eficientes aprovecharán mejor los recursos disponibles y, por tanto, su empleo será más económico en algún sentido.”

Esta vez la típica pregunta para la que hay que saberse los temitas auxiliares de memoria, es mas simple, ya que es una de las definiciones clásicas de EFICIENCIA.

9.- La declaración:
PROCEDURE Sencillo(a : ARRAY OF INTEGER);

a) Debería ser: PROCEDURE Sencillo(VAR a : ARRAY OF INTEGER);
b) Es correcta.
c) Debería ser: PROCEDURE Sencillo(a : ARRAY[1..10] OF INTEGER);
d) Debería ser: PROCEDURE Sencillo(a : ARRAY[1..10] OF INTEGER) : BOOLEAN;

Texto base, apartado 11.6., Argumentos de tipo vector abierto

Aunque sería mas eficiente pasar el vector por referencia, ya que por valor hay que copiar el vector en otra posición de memoria, con el aumento del uso de memoria y el tiempo de copiar, por lo que es preferible usando la sintaxis del punto a), esa no es la pregunta.

10.- Dado el siguiente fragmento de programa:

```
n := 1;
WHILE n < 100 DO
  IF (n MOD 10) = 0 THEN
    n := 3 * n;
  ELSIF (n MOD 5) = 0 THEN
    n := 2 * n;
  ELSE
  n := n + 1;
  END;
END;
```

a) El resultado es n = 100.
b) <u>El resultado es n = 270.</u>
c) El resultado es n = 101.
d) Ninguna respuesta anterior es correcta.

Efectuamos la traza del programa para ver el resultado:

	Valor de n
n:=1	1
Entramos en WHILE ya que n < 100	1
Como n no es divisible entre 10 ni entre 5, aumentamos en uno	2
Seguimos en WHILE ya que n < 100	
Como n no es divisible entre 10 ni entre 5, aumentamos en uno	3
Seguimos en WHILE ya que n < 100	
Como n no es divisible entre 10 ni entre 5, aumentamos en uno	4
Seguimos en WHILE ya que n < 100	
Como n no es divisible entre 10 ni entre 5, aumentamos en uno	5
Seguimos en WHILE ya que n < 100	
Como n no es divisible entre 10 pero si entre 5, multiplicamos por dos	10
Seguimos en WHILE ya que n < 100	
Como n es divisible entre 10, multiplicamos por tres	30
Seguimos en WHILE ya que n < 100	
Como n es divisible entre 10, multiplicamos por tres	90
Seguimos en WHILE ya que n < 100	
Como n es divisible entre 10, multiplicamos por tres	270
Como n es mayor de 100, salimos del WHILE	

11.- Dado el siguiente fragmento de programa para N = 5:

```
FOR k := 2 TO N - 1 DO
  FOR j := 1 TO N - k DO
    WriteInt(j, 1);
  END;
  Write('*');
  FOR j := 1 TO 2 * k - 3 DO
    WriteInt(j, 1);
  END;
END;
```

el resultado es:

a) <u>123*112*1231*12345</u>
b) 123*12*123*12345
c) Hay errores de compilación.
d) Ninguna respuesta anterior es correcta.

Efectuamos la traza del programa para ver el resultado:

		Resultado
FOR k:=2 TO 4 DO		-
Con k == 2	FOR j:=1 TO 3 DO WriteInt(j,1); END;	"123"
	Write("*");	"123*"
	FOR j:=1 TO 1 DO WriteInt(j,1); END;	"123*1"
Con k == 3	FOR j:=1 TO 2 DO WriteInt(j,1); END;	"123*112"
	Write("*");	"123*112*"
	FOR j:=1 TO 3 DO WriteInt(j,1); END;	"123*112*123"
Con k == 4	FOR j:=1 TO 1 DO WriteInt(j,1); END;	"123*112*1231"
	Write("*");	"123*112*1231*"
	FOR j:=1 TO 5 DO WriteInt(j,1); END;	"123*112*1231*12345"

12.- ¿Es correcta la siguiente declaración de variables?

```
VAR
  MAX : INTEGER;
  MIN : LONGREAL;
```

a) No. Los distintos tipos de variables se separan por coma.
b) Sí.
c) No. Se utiliza alguna palabra clave como nombre de variable.
d) No. Se debe utilizar un REAL en lugar de un LONGREAL.

Texto base, apartado 3.2, página 54, tras la lista de identificadores predefinidos en donde aparecen MAX y MIN:

> *"Los identificadores predefinidos no son palabras reservadas, sino que el programador puede, si lo desea, definirlos por su cuenta con otro significado diferente del habitual. Al hacerlo así pierde automáticamente la posibilidad de usarlos con el significado predefinido. Además, esta redefinición causaría cierta confusión en otros programadores que lean el programa, ya que no tendrán su significado habitual. Por esta razón, no es aconsejable usar esta facilidad de redefinición, a menos que resulte realmente ventajosa en determinadas aplicaciones. "*

Hay que saber distinguir entre palabras reservadas e identificadores predefinidos, lo que es sencillo, las palabras reservadas son sentencias del lenguaje, y los identificadores son funciones..

13.- Dado el siguiente fragmento de programa:

```
IF Edad < 6 THEN
  tarifa := 0.0;
ELSIF Edad < 18 THEN
  tarifa := 0.5;
ELSIF Edad < 65 THEN
  tarifa := 1.0;
ELSE
  tarifa := 0.25;
END;
```

si la variable Edad toma el valor 5, ¿cuántas condiciones se comprueban?

a) 3
b) 1
c) 2
d) Ninguna

Texto base, apartado 5.3.2, Sentencia IF páginas 108 y siguientes.

En un conjunto de IF-ELSE-ELSIF, si se cumple una condición, el resto ya no se evalúa, lo que es el primer caso. Para aclararnos mejor, lo podemos escribir en forma de IF anidados para verlo mas claro:

```
IF Edad < 6 THEN
        tarifa := 0.0;
ELSE
        IF Edad < 18 THEN
                tarifa := 0.5;
        ELSE
                IF Edad < 65 THEN
                        tarifa := 1.0;
                ELSE
                        tarifa := 0.25;
                END
        END
END;
```

14.- Dado el siguiente programa:

```
MODULE Simple;
VAR DatoA, DatoB, DatoC : INTEGER;
BEGIN
  DatoA := 5;
  DatoB := DatoA + DatoC;
  DatoC := DatoB * DatoA;
END Simple.
```

a) DatoA = 5, DatoB = 5 y DatoC = 25.
b) Errores de ejecución.
c) Errores de compilación.
d) <u>Ninguna respuesta anterior es correcta.</u>

Texto base, apartado 3.4.3. Uso de variables. Inicialización., páginas 60 y siguientes.

Cuando no se asignan valores a las variables, estas no valen cero o espacios, sino un valor aleatorio, dependiendo del contenido anterior de la memoria del ordenador. Por ello la segunda instrucción da un resultado indefinido, ya que no podemos saber si DatoC valdrá cero, doscientos o menos dieciocho.

15.- La definición de las funciones y procedimientos en Modula-2 se sitúan:

a) Al final del bloque del programa.
b) En la parte ejecutiva de un bloque de programa.
c) <u>En la parte declarativa de un bloque de programa.</u>
d) Ninguna respuesta anterior es correcta.

Texto base, apartado 7.5. Visibilidad. Estructura de bloques, páginas172 y siguientes.

Aunque no se expresa directamente así en el texto, las funciones y procedimientos se ubican siempre tras las variables, y antes del bloque principal, lo que constituye la parte declarativa. Si pensamos que las funciones y procedimientos son solo declaración de una forma de operar con las variables, a lo mejor está un poco mas claro.

EJERCICIO

Realizar un programa en Modula-2 que indique al pagador de una empresa como tiene que pedir al banco el dinero para pagar una nómina. El programa tendrá las siguientes características:

- Se introducirán los datos del personal. Estos datos serán: número del empleado y sueldo neto a cobrar. Para indicar el último empleado se introducirá como número de empleado un 0 y sueldo igual a 0.
- El programa tiene como objetivo emplear el menor número posible de monedas y billetes para el pago de la nómina, dándole a cada empleado su sueldo. Para ello hay que determinar la cantidad total de billetes de 10.000, 5.000, 2.000 y 1.000 pts. y de monedas de 500 y 100 pts. que hay que pedir al banco.
- El resultado del programa será un listado de la petición del dinero.

```
MODULE Examen;
FROM InOut IMPORT WriteString, WriteLn, ReadCard, WriteCard;
FROM RealInOut IMPORT ReadReal,WriteReal;

VAR
Codigo : CARDINAL; Salario : REAL;  (* Variables de un empleado *)
B10, B5, B2, B1, M5, M1 : CARDINAL; (* Cantidades de billetes y monedas *)
Resto : REAL;                       (* Por si sobran cantidades *)

BEGIN
     B10:=0; B5:=0; B2:=0; B1:=0; M5:=0; M1:=0;
     LOOP
         WriteString("Código  del Empleado (0=FIN): "); ReadCard(Codigo);
         WriteString("  Salario del Empleado (0=FIN): "); ReadReal(Salario);WriteLn;
         IF (Codigo = 0) AND (Salario = 0.0) THEN
            EXIT;
         END;
         WHILE (Salario >= 10000.0) DO INC(B10); Salario:=Salario - 10000.0; END;
         WHILE (Salario >=  5000.0) DO INC(B5) ; Salario:=Salario -  5000.0; END;
         WHILE (Salario >=  2000.0) DO INC(B2) ; Salario:=Salario -  2000.0; END;
         WHILE (Salario >=  1000.0) DO INC(B1) ; Salario:=Salario -  1000.0; END;
         WHILE (Salario >=   500.0) DO INC(M5) ; Salario:=Salario -   500.0; END;
         WHILE (Salario >=   100.0) DO INC(M1) ; Salario:=Salario -   100.0; END;
         Resto := Resto + Salario;
     END;
     WriteLn;WriteLn;
     WriteString("La relación de billetes y monedas necesaria es");WriteLn;
     WriteString("Billetes de 10.000: ");WriteCard(B10,10);WriteLn;
     WriteString("Billetes de  5.000: ");WriteCard(B5 ,10);WriteLn;
     WriteString("Billetes de  2.000: ");WriteCard(B2 ,10);WriteLn;
     WriteString("Billetes de  1.000: ");WriteCard(B1 ,10);WriteLn;
     WriteString("Monedas  de    500: ");WriteCard(M5 ,10);WriteLn;
     WriteString("Monedas  de    100: ");WriteCard(M1 ,10);WriteLn;
     IF (Resto <> 0.0) THEN
        WriteString("Otros             : ");WriteReal(Resto,10);WriteLn;
     END;
END Examen.
```

No es un programa muy elegante, pero hace justo lo que se pide, y cabe en muy poco espacio. Ya que no nos piden guardar los datos de los empleados, no nos guardamos nada, pero aunque no está en el enunciado, hay una posibilidad de desbordamiento, si el importe no es múltiplo de 100, por lo que se añade una cantidad adicional, denominada Resto, para tener en cuenta estas posibles discrepancias.

Como posibilidades adicionales, podríamos desarrollar lo siguiente:

a) Sumar la cantidad total de los salarios, y presentarla en pantalla.
b) Una rutina de impresión de números reales, para que tanto el total como el Resto se impriman de forma mas legible
c) Una estructura de punteros enlazados que guarde los datos de todos los empleados, imprimiéndolos al final por orden de código o de salario.
d) Un pequeño menú, que permita seleccionar entre introducir datos, presentar los totales, presentar los empleados por códigos, o presentar los empleados por salario.