EXAMENES RESUELTOS DE PROGRAMACION I

Comentarios por Jose Antonio Vaqué

EXAMEN DE Febrero de 1997, segunda semana

Soluciones no oficiales

1.- Dado el siguiente esquema de selección

```
IF C1 THEN..
ELSIF C2 THEN ...
..
ELSIF Ci THEN..
ELSE..
END
```

Donde C1..Ci son expresiones condicionales.

Cuantas expresiones se evalúan en el peor caso

a) Sólo la condición que se cumpla.
b) i
c) i+1
d) 1

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

En un conjunto de IF-ELSIF-ELSE, el pero caso es que no se cumpla ninguno, y se acabe por el ELSE final. En este caso se han realizado **i** comparaciones (IF C1, IF C2 … IF Ci), por lo que son evaluadas i condiciones, el ELSE no se evalúa en sí mismo, simplemente si no se cumple la última condición, se pasa a ejecutar.

2.- A la representación de un programa en lenguaje simbólico se le llama:

a) Programa fuente
b) Programa objeto
c) Programa ejecutable
d) Programa compilador

Texto base, apartado 1.4. Compiladores e intérpretes, página 12:

> "A la representación del programa en lenguaje simbólico se le llama programa fuente, y su representación en código máquina se le llama programa objeto."

3.- Después de la ejecución del siguiente bucle:

```
p := 2; q := 5;
REPEAT
  q := (p + q) DIV 2;
  p := (p + q) DIV 2;
  q := q + 1;
UNTIL q > 4;
```

a) p = 2 y q = 5
b) p = 2 y q = 6
c) <u>Es un bucle infinito</u>
d) El bucle solo se ejecuta 1 vez

Vamos a hacer la traza del programa, para ver su comportamiento, analizando los valores de p y q:

SENTENCIA	COMENTARIO	P	Q
p:=2; q:=5;	Se da un valor inicial	2	5
REPEAT	Comienza el bucle		
q:=(p+q) DIV 2;	Q == (2 + 5) DIV 2 == 7 DIV 2 == 3	2	3
p:=(p+q) DIV 2;	P == (2 + 3) DIV 2 == 5 DIV 2 == 2	2	3
q:= q+1;	Q == 3 + 1 == 4	2	4
UNTIL q > 4;	Q no es mayor de 4, continuamos el bucle		
REPEAT	Comienza un nuevo bucle		
q:=(p+q) DIV 2;	Q == (2 + 4) DIV 2 == 6 DIV 2 == 3	2	3
p:=(p+q) DIV 2;	P == (2 + 3) DIV 2 == 5 DIV 2 == 2	2	3
q:= q+1;	Q == 3 + 1 == 4	2	4
UNTIL q > 4;	Q no es mayor de 4, continuamos el bucle	2	4
REPEAT	Comienza otro nuevo bucle		
q:=(p+q) DIV 2;	Q == (2 + 4) DIV 2 == 6 DIV 2 == 3	2	3
p:=(p+q) DIV 2;	P == (2 + 3) DIV 2 == 5 DIV 2 == 2	2	3
q:= q+1;	Q == 3 + 1 == 4	2	4
UNTIL q > 4;	Q no es mayor de 4, continuamos el bucle		

Está claro que a partir de la segunda ejecución, el bucle siempre se repide igual, por lo que nunca conseguirá salir del mismo.

4.- Dado el siguiente fragmento de código:

```
TYPE anhos=(1900, 1990, 1995, 2000);
VAR n:anhos;
...

n:=1900;
```

a) Es correcto.
b) La definición del tipo es incorrecta.
c) La definición de la variable debe ser previa a la del tipo.
d) Incompatibilidad de tipos en la asignación, ya que 1900 es un valor entero.

Texto base, apartado 9.2.1 Definición de tipos enumerados, página 224:
"La sintaxis exacta de la declaración de los tipos enumerados es la siguiente:
Tipo_enumerado ::= (Lista_de_identificadores)
Lista_de_identificadores ::= Identificador { , Identificador }

En una enumeración se usan identificadores, y los identificadores DEBEN COMENZAR CON UNA LETRA, por eso, auque en el libro no está muy claro, la definición no es correcta..

5.- Teniendo en cuenta como funciona el procedimiento predefinido EXCL la cabecera podría ser

a) EXCL (v1:tipoconjunto;elem:tiporeferencial);
b) EXCL (VAR v1,v2:tipoconjunto);
c) EXCL (v1,v2:tiporeferencial)
d) <u>EXCL (VAR v1:tipoconjunto;elem:tiporeferencial);</u>

Texto base, apartado 9.6.3. Operaciones entre conjuntos, páginas 237:
"EXCL(S,X) Excluye el elemento X del conjunto S."

Se excluye de un conjunto un elemento, por ello el primer parámetro debe pasarse por referencia, y debe ser un tipo conjunto, mientras que el segundo se pasa por valor, y es del tipo de un elemento. La única respuesta con VAR y los dos tipos adecuados es la D.

6.- Dado el siguiente fragmento de código:

```
VAR a : CHAR;
....
FOR a := 'a' TO 'h' BY 2 DO Write ( a ) END; WriteLn;
```

Después de su ejecución aparecerá por pantalla

a) aaaa
b) aceg
c) acegi
d) Hay un error en el código.

Texto base, apartado 5.3.4 Sentencia FOR, páginas 112 y siguientes

Aunque parezca un poco extraño, las variables de tipo CHAR se pueden usar como variables numéricas, por su valor ASCII. Sería equivalente a las sentencias:

```
VAR n : INTEGER;

FOR n := ORD('a') TO ORD('h') BY 2 DO
        Write ( CHR(n) )
END;
WriteLn;
```

7.- Para crear un programa con un elemento definido en otro módulo debemos tener.

a) El módulo de definición e importarlo.
b) No se pueden definir elementos en otros módulos.
c) El módulo de definición, el de implementación e importarlo
d) Sólo hace falta importarlo.

Texto base, apartado 14.2.4 Uso de módulos, páginas 405 y siguientes.

Para poder usar el elemento en nuestro módulo solo hay que importarlo (respuesta D), pero para poder compilar nuestro módulo necesitamos también el módulo de definición (respuesta A). Pero para poder linkar el programa necesitamos o bien el módulo de implementación ya compilado, o bien compilar nosotros mismos el módulo de implementación (respuesta C). Como no se plantea que poseamos el módulo ya compilado, asumimos que la respuesta C es la correcta, aunque creo que sería mas normal poner la A, y sería igual de correcta, ya que la expresión "crear un programa" puede referirse solo a crear el código fuente.

8.- Después de ejecutar las siguientes sentencias

```
siguiente^.valora:=7.0;
siguiente^.valorb:=4;
a:=siguiente^.valora;
```

El valor de a es indeterminado. Podemos decir que

a) La primera asignación es errónea.
b) siguiente apunta a un registro con dos campos fijos.
c) siguiente apunta a un registro con dos variantes.
d) valora es un registro con variantes.

Texto base, apartado 12.5 Registros con variantes, páginas 343 y siguientes.

Vamos por eliminación, ya que la respuesta está planteada para despistar por completo. Vemos que siguiente^ se usa con un punto y dos valores, por lo que deducimos que apunta a un registro. Descartamos A por tonta, descartamos D ya que no se podría usar directamente, si apuntara a un registro de campos fijos, no podría ser indeterminado, por tanto solo queda la C. El hecho es que el único caso en que se produce indefinición es cuando intervienen registros con variantes.

9.- En una parte declarativa el orden de definición será:

a) El orden depende de su posterior uso.
b) Constantes, variables, tipos, subprogramas.
c) Constantes,tipos, variables , subprogramas.
d) Subprogramas, tipos,variables, constantes.

El orden de los elementos se define a gusto del programador, y se pueden mezclar, puedo usar variables, luego constantes, luego tipos, luego variables, más tipos, una constante, etc. Lo único importante es que si uso algo, debo tenerlo definido antes, por lo que la respuesta es la A.

10.- Queremos que un bucle WHILE se ejecute mientras no introduzcamos por teclado ni 'f' ni 'F'. El teclado lo leemos con READ(c);

La condición que se debe evaluar será:

a) **(c <> 'f ') AND (c <> ' F ')**
b) (c <> 'f ') OR (c <> ' F ')
c) (c # 'f ') OR (c <> ' F ')
d) (c <> (' f ' AND ' F ')

Texto base, apartado 5.2 Expresiones condicionales, páginas 104 y siguientes.

No necesita más explicaciones, esta pregunta es un regalo de un punto, pues es demasiado sencilla.

EJERCICIO DE PROGRAMACIÓN

Realizar los módulos de DEFINICIÓN y de IMPLEMENTACIÓN para un nuevo tipo de dato abstracto: FRACCION formado por numerador y denominador. Las operaciones básicas de este nuevo tipo serán: SUMAR, RESTAR, MULTIPLICAR y DIVIDIR con el significado habitual de estas operaciones en las fracciones. Utilizar en un módulo PRINCIPAL el nuevo tipo abstracto creado para sumar las fracciones: 3/4 + 5/7.

```
DEFINITION MODULE Fraccion;
TYPE TFraccion = RECORD
     Num,Den :INTEGER;
END;
PROCEDURE SumF(VAR Res:TFraccion; Num1,Num2:TFraccion);
PROCEDURE ResF(VAR Res:TFraccion; Num1,Num2:TFraccion);
PROCEDURE MulF(VAR Res:TFraccion; Num1,Num2:TFraccion);
PROCEDURE DivF(VAR Res:TFraccion; Num1,Num2:TFraccion);
END Fraccion.
```

```
IMPLEMENTATION MODULE Fraccion;
PROCEDURE SumF(VAR Res:TFraccion; Num1,Num2:TFraccion);
BEGIN
   IF (Num1.Den = Num2.Den) THEN
      Res.Num := Num1.Num + Num2.Num;
      Res.Den := Num1.Den;
   ELSE
      Res.Num := (Num1.Num * Num2.Den) + (Num2.Num * Num1.Den);
      Res.Den :=  Num1.Den * Num2.Den;
   END;
END SumF;
PROCEDURE ResF(VAR Res:TFraccion; Num1,Num2:TFraccion);
BEGIN
   IF (Num1.Den = Num2.Den) THEN
      Res.Num := Num1.Num - Num2.Num;
      Res.Den := Num1.Den;
   ELSE
      Res.Num := (Num1.Num * Num2.Den) - (Num2.Num * Num1.Den);
      Res.Den :=  Num1.Den * Num2.Den;
   END;
END ResF;
PROCEDURE MulF(VAR Res:TFraccion; Num1,Num2:TFraccion);
BEGIN
   Res.Num := Num1.Num * Num2.Num;
   Res.Den := Num1.Den * Num2.Den;
END MulF;
PROCEDURE DivF(VAR Res:TFraccion; Num1,Num2:TFraccion);
BEGIN
   Res.Num := Num1.Num * Num2.Den;
   Res.Den := Num1.Den * Num2.Num;
END DivF;
END Fraccion.
```

```
MODULE Examen;
FROM Fraccion IMPORT TFraccion,SumF;
FROM InOut IMPORT WriteString,Write,WriteInt,WriteLn,Read;
VAR Frc1,Frc2,Tot:TFraccion;
BEGIN
Frc1.Num := 3;Frc1.Den := 4;   Frc2.Num := 5;Frc2.Den := 7;
SumF(Tot,Frc1,Frc2);
WriteString("Resultado: ");
WriteInt(Tot.Num,3);Write('/');WriteInt(Tot.Den,3);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.

Como ejercicio, podremos plantearnos lo siguiente:

- Habría que realizar una operación fundamental con fracciones, la simplificación de las mismas, que se pudiera llamar de forma independiente, y se llamara tras cualquier operación con fracciones..
- También habría que hacer algo para imprimir con mejor estilo las fracciones resultantes..