EXAMENES RESUELTOS DE PROGRAMACION I

Comentarios por Jose Antonio Vaqué

EXAMEN DE Febrero de 1999, primera semana

SOLUCIONES OFICIALES

1. Respecto a la sentencia:
Correcto := calculo(matriz) IN grupo;
Se puede decir en cualquier caso que:

A.- La función cálculo devuelve un valor enumerado
B.- calculo es un procedimiento que devuelve como resultado matriz
C.- matriz y grupo son de tipos compatibles
D.- grupo es de tipo BOOLEAN

Texto base, apartado 9.6.1. Definición de tipos conjunto, página 234:
"…en la definición se pueden utilizar tipos ordinales, definidos anteriormente en el programa o tipos predefinidos del lenguaje, o enumeraciones o subrangos de cualquiera de ellos…"

Texto base, apartado 9.6.3. Operaciones entre conjuntos, página 238:
"Otro operador que da lugar a un resultado de tipo BOOLEAN es el operador que permite comprobar si un elemento pertenece o no a un conjunto…:
Elemento IN Conjunto

Para usar el operador IN sabemos que **grupo** debe ser un conjunto, y que **calculo(matriz)** debe ser un elemento de un conjunto. Los elementos de los conjuntos pueden ser valores enumerados, por lo que la respuesta A es cierta. La respuesta B es absurda, calculo debe devolver un elemento de un conjunto. La respuesta C es absurda, un conjunto no es del mismo tipo que sus elementos. Sobre la respuesta D, el resultado de IN es de tipo BOOLEAN, por lo que Correcto debe ser BOOLEAN, pero no grupo que debe ser de tipo conjunto.

2. Los meta símbolos son:

A.- Elementos de la notación BNF
B.- Elementos de la programación lógica
C.- Elementos de la programación funcional
D.- Parte del modelo de flujo de datos

Texto base, Tema 2.1. Notación BNF, Página 27:
"Estas reglas sobre como han de escribirse los elementos del lenguaje en forma de símbolos utilizan a su vez otros símbolos, que se denominan meta símbolos."

Hay que estudiar un poco de vez en cuando, no siempre sirve la lógica.

3. La reutilización se consigue mediante desarrollo:

A.- Ascendente o descendente
B.- Sólo ascendente
C.- Sólo descendente
D.- Robusto

Texto base, Tema 8.2.4 Reutilización, Página 204 y siguientes

La reutilización es no escribir dos veces el mismo código. Si vemos que necesitamos escribir dos veces algo, lo convertimos en una función o procedimiento, y así solo hay que tocar una parte del programa si se cambia algo, o corregir un trozo si hay un error. Esto es independiente de que planteemos el programa con desarrollo ascendente o descendente.

4. Los tipos opacos:

A.- Son tipos abstractos de datos
B.- Son datos persistentes
C.- Son punteros
D.- Son de acceso secuencial

Texto base, Tema 14.4.2 Tipos opacos, Página 415:

> "Para asegurarse de que a los datos de un tipo abstracto se les aplican únicamente las operaciones que se ha definido expresamente para ellos, es posible usar un artificio que permite definir los llamados tipos opacos en Modula-2".

Nuevamente, estudiar un poco es bueno para aprobar.

5. Dado el siguiente fragmento de código:

```
FOR i:=0 TO 1 BY 0.1 DO
WriteString("Esto se escribe 10 veces");
END
```

A.- Se produce un error por incompatibilidad de tipos.
B.- El cuerpo del bucle se ejecuta 10 veces.
C.- La frase se escribe una vez, pero al incrementar el índice se produce un error.
D.- Aunque no hay errores, el cuerpo del bucle nunca se ejecuta.

Texto base, Tema 5.3.4. Sentencia FOR, Página 112 y siguientes.

Las tres partes de una sentencia FOR (valor inicial, valor final e incremento) deben ser del mismo tipo que la variable que controla el bucle. En este caso, valor inicial y valor final son de tipo entero, mientras que el incremento es de tipo real.

6. Dado el siguiente fragmento de código:

```
VAR a:CHAR;
TYPE CHAR=SET OF ['1'..'9'];
VAR b:CHAR;
…
b:=CHAR{'3'..'5'};
a:=b;
```

A.- Se produce un error en la asignación a:=b.
B.- La declaración de las variables es incompatible.
C.- Se produce un error en la asignación b:=CHAR{'3'..'5'}.
D.- Es correcto.

Texto base, Tema 3.2. El vocabulario de Modula-2, Página 54:

> "Los identificadores predefinidos no son palabras reservadas, sino que el programador puede, si lo desea, definirlos por su cuenta con otro significado diferente al 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 estas razones no es aconsejable usar esta facilidad de redefinición, a menos que resulte realmente ventajosa en determinadas aplicaciones."

Para ver lo que pasa, hay que seguir el programa línea a línea:

- Se define una variable **a** de tipo carácter
- Se crea un tipo nuevo de datos, de nombre CHAR (una redefinición legal al no ser una palabra reservada), de tipo conjunto, que admite letras, con valores entre la letra '1' y la letra '9' (no confundir letras con números).
- Se crea una variable **b** del nuevo tipo anteriormente definido, por lo que es un conjunto de letras.
- Se asigna a **b** un subrango del conjunto CHAR, que comprende las letras '3', '4' y '5'
- Se intenta asignar a una variable de tipo carácter **a**, una variable de tipo conjunto **b**.

Hay que estar atentos a las redefiniciones, ver cuando son legales y cuando no, y no dejarse despistar por el nombre que se le da, que siempre trata de confundir (si no el examen sería demasiado fácil, ¿no?).

7. A la vista únicamente de la siguiente declaración:
VAR K:POINTER TO SET OF [1..10];
Sólo se puede decir que:

A.- K es de tipo anónimo.
B.- Es incorrecta.
C.- K es un dato encapsulado.
D.- K es de tipo opaco.

Cuando no se utiliza un tipo definido antes en el programa, la variable es de tipo anónimo. Es muy simple, tan simple que caemos en la trampa. Si la definición fuera:

```
TYPE aux = SET OF [1..10];
VAR K:POINTER TO aux;
```

Entonces la variable K tendría un tipo, sería un puntero a una variable de tipo aux.

8. Dada la siguiente fragmento de código

```
dato=dias{L};
```

podemos decir que

A.- dias{L} es un valor constante de tipo conjunto
B.- L puede ser una variable o una constante
C.- dias es el nombre del tipo referencial
D.- dato no tiene tipo

Si nos fijamos un poco, vemos que no se utiliza la asignación [:=], por lo que no es una asignación en el cuerpo del programa. Tampoco puede ser parte de una sentencia IF, ya que acaba en punto y coma. Por tanto debe estar en la parte de declaración. El igual se utiliza para asignar valores a las constantes o a los tipos definidos. No puede ser un tipo definido, ya que no se ajusta a ninguna definición de tipo, por lo que solo nos queda un uso de constante. Como ejemplo se puede compilar este programa:

```
MODULE Ejemplo;
TYPE
        dias = SET OF (A,L,M,T);
CONST
        dato=dias{L};

END Ejemplo
```

9. Dado el siguiente módulo

```
DEFINITION MODULE Segundo;
    PROCEDURE Previo( VAR dato:tipodato):
END Segundo.
```

A.- Es erróneo, se necesita la declaración de tipodato
B.- Es correcto, tipodato es un tipo opaco
C.- Es erróneo tipodato es un tipo anónimo
D.- Es erróneo dato debería ser por valor

Texto base, Tema 14.2.2. Módulos de definición, Páginas 401 y siguientes

Para que una función pueda compilarse, hay que tener definidos los tipos de las variables que le pasamos como argumentos, y en este caso no se ha realizado así.

10. La cabecera del subprograma WriteString para imprimir cadenas de caracteres podría ser

A.- PROCEDURE WriteString(a:ARRAY OF CHAR);
B.- PROCEDURE WriteString(VAR a:ARRAY[1..1000]OF CHAR);
C.- PROCEDURE WriteString(a:CHAR);
D.- PROCEDURE WriteString(a:POINTER TO CHAR);

Texto base, Tema 11.5 Vector de caracteres: Ristra (String), Páginas 308 y siguientes
Texto base, Tema 11.6 Argumentos de tipo vector abierto, Páginas 310 y siguientes

La función WriteString imprime una cadena de cualquier longitud, para ello se utilizan los vectores abiertos, en los que no se pasa la longitud del vector. La respuesta B queda limitada por el tamaño, la C solo imprime un carácter, y en la D habría que pasar un puntero, no una cadena.

EJERCICIO DE PROGRAMACIÓN

En el módulo Juegos, se dispone del tipo abstracto de datos CartaBaraja, que representa una carta de la baraja española. También se dispone de dos operaciones asociadas: PonerPalo y PonerTriunfo.
PonerPalo, establece el palo de la carta: oros, copas, espadas o bastos.
Por ejemplo: PonerPalo(carta,oros).
PonerTriunfo, establece el triunfo de la carta: as, dos,…, sota, caballo o rey.
Por ejemplo: PonerTriunfo(carta,rey)
Se pide crear en el módulo principal una baraja de 40 cartas y la operación Vencer. Una carta vence a otra cuando su triunfo es mayor, excepto cuando una de las cartas es de la "pinta" que entonces gana aún cuando su triunfo sea menor. Si dos cartas tienen igual triunfo vence cualquiera de las dos.
Ejemplo: carta1 es el dos de copas, carta2 es el rey de bastos y la "pinta" es copas.
Vencer(carta1,carta2,copas) devuelve cierto; Vencer(carta2,carta1,copas) devuelve falso; Vencer(carta1,carta2,bastos) devuelve falso.

LA SIGUIENTE SOLUCION ES LA QUE HA APORTADO EL EQUIPO DOCENTE EN SUS CD.ROM, NO ES MIA:

```
MODULE Examen;

FROM Juegos IMPORT CartaBaraja, PonerPalo, PonerTriunfo, TipoPalo, TipoTriunfo, PedirPalo,
PedirTriunfo;

TYPE
   Baraja=ARRAY[1..40] OF CartaBaraja;

VAR
   B1:Baraja;
   C1:TipoPalo;
   C2:TipoTriunfo;

PROCEDURE Vencer(Carta1,Carta2:CartaBaraja;Pinta:TipoPalo);
BEGIN
   IF (PedirPalo(Carta1)=PedirPalo(Carta2)) THEN
      IF (PedirTriunfo(Carta1)> PedirTriunfo(Carta2)) THEN
         RETURN TRUE;
      ELSE
         RETURN FALSE;
      END;
   ELSE
      IF (PedirPalo(Carta1)=Pinta) THEN
         RETURN TRUE;
      ELSIF (PedirPalo(Carta2)=Pinta ) THEN
         RETURN FALSE;
      ELSE
         RETURN TRUE;
      END;
   END;
END Vencer;

BEGIN
   (* Rellenado de las cuarenta cartas de la baraja *)
   FOR C1:= oros TO bastos DO
      FOR C2:= as TO rey DO
         PonerPalo (B1[ORD(C2)+1+ORD(C1)*10],C1);
         PonerTriunfo (B1[ORD(C2)+1+ORD(C1)*10],C2);
      END;
   END;
END Examen.
```