EXAMENES RESUELTOS DE PROGRAMACION I

Comentarios por Jose Antonio Vaqué

EXAMEN DE Febrero de 1999, segunda semana

SOLUCIONES OFICIALES

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

A.- Correcto es de tipo BOOLEAN
B.- Correcto y grupo son de tipos compatibles
C.- La función calculo devuelve un tipo BOOLEAN
D.- grupo es de tipo enumerado

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, y que retornará un valor BOOLEAN, que se asigna a la variable **Correcto**. La respuesta B es no válida, **Correcto** debe ser BOOLENA, mientras que **grupo** es de tipo conjunto. La respuesta C es incorrecta, ya que debe devolver un tipo elemento del conjunto Sobre la respuesta D, **grupo** es de tipo conjunto.

2. En la sentencia:
Pagina.Imprimir;

A.- Pagina es un módulo
B.- Pagina es un registro
C.- Imprimir es un campo variante
D.- Imprimir es una función

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

Como nos dice que es una sentencia, debe tener sentido en si misma, no sería lo mismo si nos dices que es parte de una sentencia, ya que desconocemos la otra parte, y puede añadirse lo que queramos. En A se está usando la forma cualificada de acceder a un elemento de un módulo, importado completamente sin especificar que se importa. B, C y D solo serían válidos si fueran parte de una sentencia, no una sentencia completa.

3. El invariante de una iteración es la condición que se debe cumplir siempre:

A.- Antes y después de cada nueva repetición
B.- Sólo después de cada repetición
C.- Sólo antes de cada repetición
D.- En cualquier punto del bucle iterativo

Texto base, apartado 6.3.4 Razonamiento sobre bucles: invariante, terminación, página 143.

> "Para analizar el comportamiento de un fragmento de programa correspondiente a un esquema de iteración habremos de identificar las condiciones que deben cumplirse siempre inmediatamente antes de examinar la condición de repetición. Estas condiciones constituyen el llamado invariante del bucle…deduciremos que al comienzo de cada repetición de la acción del bucle habrá que cumplirse el invariante y además la condición de repetición; y que al terminar las repeticiones y salir del bucle se cumplirá el invariante y además no se cumplirá la condición de repetición."

Estudiar es bueno, pero entender las cosas es mucho mejor. Esto solo es un adelanto de Programación II, y no queda claro en el libro.

4. Dado el siguiente fragmento de código:

```
TYPE INTEGER=[0..10];
VAR K: ARRAY [10..20] OF INTEGER;
…
K[15]:= 10;
```

A.- Es correcto.
B.- El tipo de datos [10..20] utilizado en la declaración es incorrecto.
C.- El acceso al elemento 15 es incorrecto.
D.- La asignación del valor 10 produce un error de incompatibilidad de tipos.

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 crea un tipo nuevo de datos, de nombre INTEGER (una redefinición legal al no ser una palabra reservada), de tipo rango, que admite valores entre cero y diez.
- Se crea una variable **K**, que es un vector de 11 elementos del nuevo tipo anteriormente definido, por lo que cada elemento del vector es del tipo rango del cero al diez.
- Se asigna al elemento 15 de **K** un valor que está dentro del rango definido.

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?).

5. Dado el siguiente fragmento de código:

```
TYPE INTEGER=('0','1','2','3','4','5');
VAR K: ARRAY [2..4] OF INTEGER;
…
K[3]:= 2;
```

A.- La redefinición del tipo INTEGER es incorrecta.
B.- La declaración de la variable K es incorrecta.
C.- K[3] accede al tercer elemento del vector K.
D.- Es correcto.

Lo mismo de la pregunta anterior
Texto base, Tema 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 }

Aunque sea muy similar al anterior, lo que está mal es la redefinición del tipo en sí, pues una enumeración requiere identificadores, no pueden usarse ni números ni, como en este caso, cadenas de caracteres.

6. Dado el siguiente fragmento de código:

```
VAR a:INTEGER;
TYPE CHAR=SET OF ['1'..'9'];
VAR b:CHAR;
…
b:=CHAR{'1'};
a:=VAL(CHAR, '1');
```

A.- La asignación del valor a la variable a es incorrecta.
B.- Es correcto y la variable a toma el valor 0.
C.- Es correcto y la variable a toma el valor 1.
D.- Es correcto y la variable a toma el valor '1'.

Lo mismo de las dos preguntas anteriores
Texto base, Tema 9.2.2 Uso de tipos enumerados, Página 224 y siguientes

Insisten en redefiniciones. Entre otras cosas VAL solo se puede usar con enumeraciones, no con subrangos, de por si solo es suficiente, pero además se usa incorrectamente, pues no puede usarse en una asignación.

7. Dada la siguiente declaración
VAR T:tipodias
Del siguiente fragmento de código
dato=dias{T}
podemos decir que

A.- es una expresión condicional
B.- dato debe ser una constante
C.- es una sentencia de asignación
D.- es una definición de un tipo

En A se pregunta si **dato** es igual al subconjunto de **dias** enumerado por **T**. La respuesta B no es posible, ya que una constante solo puede definirse usando otras constantes, y **T** es una variable. La C es evidente, le faltan los dos puntos. La D es similar a la B, una variable no se puede usar en una definición de tipos.

8. Dada la siguiente declaración
VAR p1:POINTER TO INTEGER;
p2:POINTER TO REAL;
BEGIN
NEW(p1); NEW(p2);
…
La sentencia correcta será

A.- p1^:=TRUNC(p2^);
B.- p1:=TRUNC(p2);
C.- p1:=p2;
D.- p2:=FLOAT(p1);

Texto base, Tema 13.3.1 Punteros, Página 363 y siguientes

Una variable de tipo puntero solo se pude usar para apuntar a una variable, así en B y D se intenta asignar a un puntero un valor, lo que no es correcto. En C se asigna un puntero a otro, pero de es diferente tipo. En A se asigna a lo que apunta **p1** (una entero), la conversión a entero de lo que apunta **p2** (un real).

9. Cuál de las siguientes parejas de operadores tienen siempre resultados del mismo tipo

A.- AND , IN
B.- * , +
C.- *, IN
D.- OR, INCL

Veamos cada una de las respuestas:

a) Los operadores AND e IN devuelven un valor BOOLEANO
b) Por y Mas pueden devolver tipos enteros, reales o de conjuntos
c) Por * puede devolver tipos enteros, reales o de conjuntos, mientras que IN es BOOLEANO
d) OR devuelven un valor BOOLEANO, pero INCL solo incluye un elemento en un conjunto.

10. Dada la siguiente declaración
VAR dato:ARRAY[1..10] OF INTEGER;
Con la siguiente sentencia
FOR cont:=1 TO 10 DO
dato[cont]:=dato[cont+1]
END

A.- Cometemos un error de acceso a los elementos del vector
B.- Trasladamos los elementos del vector una posición a la izquierda
C.- Trasladamos los elementos del vector una posición a la derecha
D.- Manipulamos el índice del vector por referencia

Cuando **cont** alcance el valor 10, asignamos a **dato[10]** el valor de **dato[11]**, pero **dato** solo tiene como máximo el elemento 10, hay un error en el acceso.

EJERCICIO DE PROGRAMACIÓN

Realizar un **programa completo** en Modula 2 que gestione la asignación de butacas de un recinto en el que hay 5 filas de 7 butacas cada una. El recinto será un **dato encapsulado** con las siguientes operaciones posibles ante la solicitud de un cliente (1 / 2 / 3):

1. Informe de la ocupación por pantalla. (Por ejemplo: Quedan 7 butacas vacías).
2. Asignación de una butaca libre.
3. Liberación de una butaca ocupada.

He añadido dos cosas que no están en lo que se pide. Por un lado, hay que inicializar la sala antes de empezar. Esto se hace en el módulo de implementación, y aunque no lo pidan, es imprescindible para que funcione el programa.

Por otro lado, he añadido algunos detalles, como el imprimir la sala completa, o que tras cada ocupación o liberación la imprima. Esto son cuatro instrucciones más, pero el resultado es muy diferente.

No he ubicado comentarios, ya que el programa creo que está muy claro, pero en el examen habría que poner unos cuantos.

```
DEFINITION MODULE Sala;
   PROCEDURE Quedan();
   PROCEDURE Ocupar(fila,columna:INTEGER);
   PROCEDURE Libera(fila,columna:INTEGER);
END Sala.
```

```
IMPLEMENTATION MODULE Sala;
FROM InOut IMPORT WriteString,WriteInt,WriteLn;
   CONST Fil=5;  Col=7;
   TYPE  Tsala = ARRAY [1..Fil],[1..Col] OF INTEGER;
   VAR i,j,libres:INTEGER;
   VAR butaca:Tsala;
   PROCEDURE Quedan();
   BEGIN
      libres := 0;
      FOR i:=1 TO Fil DO
         FOR j:=1 TO Col DO
            IF (butaca[i,j] = 0) THEN
               WriteString("[ ] ");   libres := libres + 1;
            ELSE
               WriteString("[X] ");
            END;
         END;
         WriteLn;
      END;
      WriteLn;WriteString("Quedan ");WriteInt(libres,3);WriteString(" butacas vacias");
      WriteLn;
   END Quedan;
   PROCEDURE Ocupar(fil,col:INTEGER);
   BEGIN
      IF (fil < 1) OR (fil > Fil) OR (col < 1) OR (col > Col) OR (butaca[fil,col] = 1) THEN
            WriteString("Butaca ocupada o fuera de rango");WriteLn;WriteLn;
      ELSE
            butaca[fil,col] := 1;   Quedan();
      END;
   END Ocupar;
   PROCEDURE Libera(fil,col:INTEGER);
   BEGIN
      IF (fil < 1) OR (fil > Fil) OR (col < 1) OR (col > Col) OR (butaca[fil,col] = 0) THEN
            WriteString("Butaca libre o fuera de rango");WriteLn;WriteLn;
      ELSE
             butaca[fil,col] := 0;   Quedan();
      END;
   END Libera;
BEGIN (* Aqui se inicializa la sala cuado carga el programa *)
   FOR i:=1 TO Fil DO   FOR j:=1 TO Col DO   butaca[i,j] := 0;   END;   END;
END Sala.
```

```
MODULE Examen;
FROM InOut IMPORT WriteString,Write,WriteLn,ReadInt,Read;
IMPORT Sala;
VAR fila,columna:INTEGER;   opc:CHAR;
BEGIN
   REPEAT
      WriteLn;WriteString("Opción (1=Ver,2=Ocupar,3=Liberar,0=Fin) ");
      Read(opc);Write(opc);WriteLn;
      IF (opc='1') THEN
         Sala.Quedan;
      ELSIF (opc='2') THEN
         WriteLn;WriteString("Fila...: ");ReadInt(fila);
         WriteLn;WriteString("Columna: ");ReadInt(columna);WriteLn;
         Sala.Ocupar(fila,columna);
      ELSIF (opc='3') THEN
         WriteLn;WriteString("Fila...: ");ReadInt(fila);
         WriteLn;WriteString("Columna: ");ReadInt(columna);WriteLn;
         Sala.Libera(fila,columna);
      END
   UNTIL (opc='0');
END Examen.
```