EXAMENES RESUELTOS DE PROGRAMACION I

Comentarios por Jose Antonio Vaqué

EXAMEN DE Febrero 2005 - 1ª Semana - Tipo C

Soluciones no oficiales

1.- Las variables de tipo puntero utilizan:

a) El procedimiento NEW para su declaración.
b) El procedimiento NEW para reservar memoria
c) La función NEW para liberar memoria.
d) La función NEW para reservar memoria.

Texto base, apartado 13.3.1 Uso de variables dinámicas, página 364:
" NEW(puntero)
Este procedimiento crea una variable dinámica…"

NEW es un procedimiento, no una función, por lo que descartamos c y d. Los punteros se declaran con VAR, y se reserva memoria para ellos con NEW

2.- En Modula-2, en la sentencia ReadInt(x);

a) X solo puede ser una variable entera
b) X puede ser cualquier expresión entera o variable entera
c) X puede ser cualquier expresión o variable
d) X solo puede ser una expresión entera

Texto base, apartado 3.6.1 Procedimientos ReadInt, ReadCard y ReadReal, página 65.
"ReadInt(Variable_entera)

A un procedimiento que cambia el valor de una variable, solo se le puede pasar una variable, y en este caso, de tipo entera.

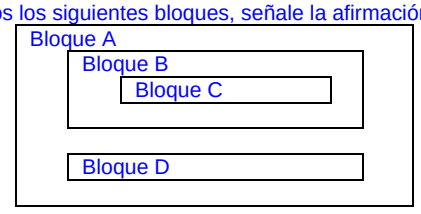
3.- La siguiente declaración en Modula-2
VAR z:COMPLEX

a) No es correcta nunca
b) Es correcta, porque COMPLEX es un tipo predefinido de Modula-2
c) Es correcta si COMPLEX es una constante
d) Es correcta si COMPLEX es un tipo de datos definido previamente

Texto base, apartado 3.2 El vocabulario de Modula-2, página 53

Esta es directa, COMPLEX no está definido en Modula, ni es una palabra reservada o palabra clave, por lo que podemos usarma para definir nuestros tipos. Si hemos definido antes ese tipo, lo podremos usar, si no está definido, no podremos.

4.- Dados los siguientes bloques, señale la afirmación verdadera



a) Desde B se puede acceder a C
b) Desde C se puede acceder a D
c) Desde A se puede acceder a D
d) Desde C se puede acceder a B.

Texto base, apartado 7.5.,Visibilidad, estructura de bloques, página 172.

Esta es fácil, desde dentro se accede a lo de fuera, peor no al contrario. Por tanto (y en la página 175 está así casi literalmente) desde A solo se accede a A, desde B se accede a A y B, desde C se accede a A, B, C, y desde D se accede a A y D.

5.- Las versiones mas estrictas de Modula-2 imponen que…

a) La realización de tipos opacos sea con punteros.
b) <u>La especificación de tipos opacos sea con punteros.</u>
c) La realización de tipos abstractos de datos sea con punteros
d) La especificación de formaciones abiertas sea con punteros.

Texto base, apartado 14.4.2., página 415:
"Las versiones mas estrictas de Modula-2 exigen que los tipos opacos se definan internamente como punteros"

Este tipo de pregunta se debería suprimir del examen. El saber esto, primero que no sirve realmente para nada, y segundo que no significa otra cosa que te sabes el libro de memoria. Yo creía que era un simple comentario del autor, pero me llamó la atención cuando estudiaba el tema, por lo que recordaba que para los tipos opacos se usaban punteros. Pero de ahí a recordar si era solo la definición o la implementación, en fin, adiós al diez.

6.- Un bucle indefinido:

a) Siempre necesita un EXIT
b) <u>Se puede hacer con LOOP, WHILE o REPEAT</u>
c) Solo se puede hacer con un LOOP
d) No es necesario en programación imperativa

Puedo hacer un bucle indefinido de todas estas formas:

```
LOOP          WHILE TRUE     REPEAT          FOR i:=0 TO 1 DO
  …             …              …               …
END;          END;           UNTIL FALSE;      i:=0;
                                               END;
```

La respuesta a no es buena, ya que no o puedo usar con un WHILE. La b es cierta, incluso se puede hacer de otra forma con un FOR (aunque no es la mas recomendable). La c es falsa, hay mas formas. La d es para rellenar, se usan mucho para hacer menús por ejemplo.

7.- En Modula-2, la declaración:
TYPE vocales = ("A","E","I","O","U");

a) Es un subrango de caracteres
b) Es una enumeración de caracteres
c) Es un conjunto de caracteres
d) <u>Es incorrecta</u>

Texto base, apartado 9.2. Tipos enumerados, páginas 223 y siguientes

Aunque parezca una enumeración, no es correcta, una enumeración debe definirse usando nombres para los elementos, por lo tanto dará un error.

8.- Marque la expresión válida según la gramática

A::= B [C] [D]
B::= E {E}
C::= e B
D::=(f | g) B
E::=a | b | c

a) <u>bbaeaac</u>
b) eabcabc
c) fababcc
d) abgaeab

Texto base, apartado 2.1 Notación BNF, página 27:

Si vamos desarrollando, vemos que A es obligatoriamente un B, seguido opcionalmente de un C, seguido opcionalmente de un D. Como B es una serie de uno o varios E, y E es a, b o c, toda secuencia válida comienza por un conjunto de 'a', 'b', 'c'. lo que descarta b) y c). Tras la secuencia B, en C usamos una letra 'e', seguida de una B, que es una serie de E, que son letras 'a', 'b' o 'c', por lo que descartamos d), al seguir una 'g' a la 'ab'

9.- Cuanto vale la variable x después de ejecutar el siguiente código:

```
x:=0;
FOR i:= 1 TO 10 DO
        FOR j:= 1 TO 20 DO
                FOR k:= 1 TO 30 DO
                        INC(x);
                END
        END
END;
```

a) x vale 30
b) x vale 6000
c) x vale 60
d) ninguna de las anteriores

Es muy sencillo, un FOR que se ejecuta 10 veces, dentro otro que se ejecuta 20 veces, por lo tanto se ejecuta 10 veces 20 = 200, y dentro otro que se ejecuta 30 veces, por lo que se ejecuta 200 veces 30 = 6000 veces, por lo tanto x se incremente 6000 veces.

10.- Dado el siguiente código:

```
TYPE Cadena = ARRAY [0..20] OF CHAR;
VAR nombre : Cadena;
VAR alias: ARRAY [0..20] OF CHAR;
…..
nombre := ‘juan’;
alias := ’juanito’;
nombre := alias;
```

e) <u>Existe incompatibilidad de tipos</u>
f) Nombre vale ‘juanjuanito’
g) Nombre vale ‘juanito’
h) No se pueden hacer asignaciones entre arrays

Pregunta para despistar, lógicamente parecería que la respuesta es la c), ya que la b) solo es para despistar, y sabemos que un array se puede asignar a otro igual, simplemente igualándolos. Pero resulta que nombre es del tipo Cadena, pero alias es de tipo anónimo, por lo que no son del mismo tipo, aunque su estructura interna sea idéntica.

EJERCICIO

Realice un tipo abstracto para gestionar un array de fechas, que no están ordenadas. El módulo será capaz de encontrar la fecha mas antigua y la mas actual de las que almacena. Las fechas serán registros con los siguientes campos: día, mes, año. Las operaciones que se tienen que realizar son: compara dos fechas, buscar la fecha antigua, buscar la fecha más reciente.
(Solo alumnos de planes antíguos). También se debe realizar la operación de mostrar todas las fechas del mes de Mayo.

```
    DEFINITION MODULE Fechas;
        TYPE TDia=[1..31]; TMes=[1..12]; TAnno=CARDINAL;
        TYPE TFecha = RECORD
                nula : BOOLEAN;
                dia  : TDia;
                mes  : TMes;
                anno : TAnno;
              END;
        TYPE TComparacion = (fmenor, figuales, fmayor, fnula);

        PROCEDURE CompararFechas(f1,f2:TFecha):TComparacion;
        PROCEDURE FechaMasAntogua(F:ARRAY OF TFecha):TFecha;
        PROCEDURE FechaMasReciente(F:ARRAY OF TFecha):TFecha;
    END Fechas.
```

```
IMPLEMENTATION MODULE Fechas;
        TYPE TModo = (Antigua, Reciente);

    PROCEDURE CompararFechas(f1,f2:TFecha):TComparacion;
    BEGIN
         IF (f1.nula = TRUE) OR (f2.nula = TRUE) THEN
            RETURN (fnula);
         ELSE
             IF (f1.anno > f2.anno) THEN
                RETURN (fmayor);
             ELSIF (f1.anno < f2.anno) THEN
                RETURN (fmenor);
             ELSE
                IF (f1.mes > f2.mes) THEN
                   RETURN (fmayor);
                ELSIF (f1.mes < f2.mes) THEN
                   RETURN (fmenor);
                ELSE
                   IF (f1.dia > f2.dia) THEN
                      RETURN (fmayor);
                   ELSIF (f1.dia < f2.dia) THEN
                      RETURN (fmenor);
                   ELSE
                      RETURN (figuales);
                   END;
                END;
             END;
         END;
    END CompararFechas;

    PROCEDURE FechaMasAntogua(F:ARRAY OF TFecha):TFecha;
    BEGIN
         RETURN (CmpFechas(F,Antigua));
    END FechaMasAntogua;

    PROCEDURE FechaMasReciente(F:ARRAY OF TFecha):TFecha;
    BEGIN
         RETURN (CmpFechas(F,Reciente));
    END FechaMasReciente;

    PROCEDURE CmpFechas(F:ARRAY OF TFecha;modo:TModo):TFecha;
    VAR ind  :CARDINAL;     (* Para seguir el array *)
        fecha:TFecha;       (* La fecha encontrada *)
        cmp  :TComparacion; (* Para comparar dos fechas *)
    BEGIN
       FOR ind := 1 TO HIGH(F) DO
          cmp := CompararFechas(fecha,F[ind]);
          IF ((modo = Antigua)  AND (cmp = fmayor)) OR
             ((modo = Reciente) AND (cmp = fmenor)) THEN

             fecha := F[ind];
          END;
       END;
       RETURN (fecha);
    END CmpFechas;
END Fechas.
```

No pongo todos los comentarios que puse en el examen, ya que creo que no hacen mucha falta, lo entenderéis igual. Para los del plan antiguo, la putada es que no cabe todo en una hoja, por lo que hay que cambiar de táctica, y comparar dos fechas usando otra opción, menos bonita, pero mucho mas corta (OJO: ES SOLO UNA IDEA SIN PROBAR, NO SE SI FUNCIONA BIEN EXACTAMENTE ASI):

```
PROCEDURE CompararFechas(f1,f2:TFecha):TComparacion;
VAR a1,a2:REAL;
BEGIN
   a1 := REAL(f1.anno) * 10000.0 + REAL(ORD(f1.mes)+1) * 100.0 + REAL(ORD(f1. dia) + 1);
   a2 := REAL(f2.anno) * 10000.0 + REAL(ORD(f2.mes)+1) * 100.0 + REAL(ORD(f2. dia) + 1);
   IF (a1 > a2)    THEN RETURN (fmayor);
   ELSIF (a1 < a2) THEN RETURN (fmenor);
   ELSE                 RETURN (figual);
   END;
END CompararFechas;
```