

EXAMEN RESUELTO FEB. 2006 TIPOS A,B,C,D:

Realizar un Tipo Abstracto de Datos (TAD) Cajero que dispondrá de billetes de 5, 10, 20, 50, 100 y 200 €. Las operaciones del TAD a codificar son: **Iniciar**: que vacía los contenedores de todos los billetes, **Cargar**: que incrementa el contenedor de un tipo de billete con el número de ellos que se desea cargar, **Sacar**: que dada una cantidad múltiplo de 5 e inferior a 500 €, devuelve el número de billetes de cada tipo que la satisface utilizando los billetes de mayor valor siempre que haya disponibles en el cajero y **EstadoySaldo**: que devuelve el saldo total del cajero y la disponibilidad de cada tipo de billete.

DEFINITION MODULE Cajero;

```
TYPE TipoBillete = (b5, b10, b20, b50, b100, b200);
TYPE TipoCajero = ARRAY TipoBillete OF CARDINAL;

PROCEDURE Iniciar(VAR C : TipoCajero);
PROCEDURE Cargar(VAR C : TipoCajero);
PROCEDURE Sacar(VAR C, desglose : TipoCajero);
PROCEDURE EstadoySaldo(C : TipoCajero);
```

END Cajero.

IMPLEMENTATION MODULE Cajero;

```
FROM InOut IMPORT WriteString, WriteLn, WriteCard, ReadCard;
```

PROCEDURE Iniciar(VAR C : TipoCajero);

```
VAR cont : TipoBillete;
BEGIN
FOR cont := b5 TO b200 DO
  C[cont] := 0;
END;
END Iniciar;
```

PROCEDURE Cargar(VAR C : TipoCajero);

```
VAR opcion, cantidad : CARDINAL;
BEGIN
REPEAT (*solo valores validos*)
  WriteString("Introduzca Billete a cargar (1=CINCO, 2=DIEZ, 3=VEINTE,
4=CINCUENTA 5:CIEN, 6=DOSCIENTOS):");
  ReadCard(opcion); WriteLn;
UNTIL (opcion > 0) AND (opcion < 7);
DEC(opcion);
WriteString("Introduzca el numero de billetes a cargar");
ReadCard(cantidad); WriteLn;
C[VAL(TipoBillete,opcion)] := C[VAL(TipoBillete,opcion)] + cantidad;
```

END Cargar;

PROCEDURE Sacar(VAR C, desglose : TipoCajero);

VAR cantidad : CARDINAL;

BEGIN

REPEAT (*solo valores validos*)

WriteString("Introduzca Cantidad a sacar (5-500)");

ReadCard(cantidad); WriteLn;

UNTIL (cantidad > 4) AND (cantidad < 501);

WHILE (cantidad >= 200) AND (C[b200] > 0) DO

DEC(C[b200]); INC(desglose[b200]); cantidad:=cantidad-200;

END;

WHILE (cantidad >= 100) AND (C[b100] > 0) DO

DEC(C[b100]); INC(desglose[b100]); cantidad:=cantidad-100;

END;

WHILE (cantidad >= 50) AND (C[b50] > 0) DO

DEC(C[b50]); INC(desglose[b50]); cantidad:=cantidad-50;

END;

WHILE (cantidad >= 20) AND (C[b20] > 0) DO

DEC(C[b20]); INC(desglose[b20]); cantidad:=cantidad-20;

END;

WHILE (cantidad >= 10) AND (C[b10] > 0) DO

DEC(C[b10]); INC(desglose[b10]); cantidad:=cantidad-10;

END;

WHILE (cantidad >= 5) AND (C[b5] > 0) DO

DEC(C[b5]); INC(desglose[b5]); cantidad:=cantidad-5;

END;

END Sacar;

PROCEDURE EstadoySaldo(C : TipoCajero);

VAR saldo : CARDINAL;

cont : TipoBillete;

BEGIN

saldo:=0;

FOR cont := b5 TO b200 DO

IF C[cont] > 0 THEN

CASE cont OF

b5 : WriteString("Billetes de cinco :");

WriteCard(C[cont],5); WriteLn; saldo:=saldo+(5*C[cont])

b10 : WriteString("Billetes de diez :");

WriteCard(C[cont],5); WriteLn; saldo:=saldo+(10*C[cont])

b20 : WriteString("Billetes de veinte :");

WriteCard(C[cont],5); WriteLn; saldo:=saldo+(20*C[cont])

b50 : WriteString("Billetes de cincuenta :");

WriteCard(C[cont],5); WriteLn; saldo:=saldo+(50*C[cont])

b100 : WriteString("Billetes de cien :");

WriteCard(C[cont],5); WriteLn; saldo:=saldo+(100*C[cont])

b200 : WriteString("Billetes de doscientos :");

WriteCard(C[cont],5); WriteLn; saldo:=saldo+(200*C[cont])

END;

```
    END;  
    END;  
    WriteString("Saldo en cajero :"); WriteCard(saldo,8); WriteLn;  
    END EstadoySaldo;
```

```
END Cajero.
```

```
MODULE ppal; (*No lo piden en el examen*)  
    FROM Cajero IMPORT TipoCajero, Iniciar ,Cargar, Sacar, EstadoySaldo;  
    FROM InOut IMPORT Read, WriteLn, WriteString, WriteCard;  
    VAR Caja,detalle : TipoCajero;  
        c : CHAR;  
BEGIN  
    Iniciar(Caja);  
    REPEAT  
        WriteString("Cargar Cajero (s/n) ?"); Read(c);WriteLn;  
        IF c <> "n" THEN Cargar(Caja) END;  
    UNTIL c="n";  
    Iniciar(detalle);  
    Sacar(Caja, detalle);  
    EstadoySaldo(detalle);  
    EstadoySaldo(Caja);  
END ppal.
```

EXAMEN RESUELTO FEB. 2006 TIPOS A,B,C,D:

Una heladeria ofrece cucuruchos con bolas de 25 gramos de 3 sabores distintos a escoger(chocolate, vainilla, fresa, limon, pistacho, menta).

Construir un TAD que almacene la cantidad disponible de cada sabor con:

1. IncrementaHelado: añade una cantidad de helado a un sabor.
2. EsPosibleCucurucho: recibe como entrada 3 sabores e indica si hay helado suficiente para el cucurucho.
3. CucuruchoDisponible: Imprime por pantalla que tipos de cucuruchos pueden confeccionarse con Los helados disponibles.

DEFINITION MODULE Helado;

```
TYPE sabores = (chocolate, vainilla, fresa, limon, pistacho, menta);  
TYPE Existencias = ARRAY [chocolate..menta] OF CARDINAL;
```

```
PROCEDURE IncrementaHelado(VAR stock:Existencias; sabor:sabores;  
gramos:CARDINAL);  
PROCEDURE EsPosibleCucurucho(stock:Existencias; sabor1,sabor2,sabor3:sabores):  
BOOLEAN;  
PROCEDURE CucuruchoDisponible(stock:Existencias);  
  
END Helado.
```

IMPLEMENTATION MODULE Helado;

```
FROM InOut IMPORT WriteString, WriteLn;
```

```
PROCEDURE IncrementaHelado (VAR stock:Existencias; sabor:sabores;  
gramos:CARDINAL);  
BEGIN  
    stock[sabor]:=stock[sabor]+gramos;  
END IncrementaHelado;
```

```
PROCEDURE EsPosibleCucurucho(stock:Existencias;  
sabor1,sabor2,sabor3:sabores): BOOLEAN;  
BEGIN  
    IF (stock[sabor1]<25) OR (stock[sabor2]<25) OR (stock[sabor3]<25) THEN  
        RETURN FALSE;  
    ELSE  
        RETURN TRUE;  
    END;  
END EsPosibleCucurucho;
```

PROCEDURE CucuruchoDisponible (stock:Existencias);

```
TYPE tNombre = ARRAY [0..10] OF CHAR;
VAR nombre : ARRAY [chocolate..menta] OF tNombre;
    s1,s2,s3:sabores;
BEGIN
    nombre[chocolate]:="Chocolate";
    nombre[vainilla]:="Vainilla";
    nombre[fresa]:="Fresa";
    nombre[limon]:="Limon";
    nombre[pistacho]:="Pistacho";
    nombre[menta]:="Menta";
    WriteLn;WriteString("Cucuruchos disponibles : ");
    FOR s1:=chocolate TO limon DO
        FOR s2:=VAL(sabores,ORD(s1)+1) TO pistacho DO
            FOR s3:=VAL(sabores,ORD(s2)+1) TO menta DO
                IF EsPosibleCucurucho(stock,s1,s2,s3) THEN
                    WriteLn;
                    WriteString(nombre[s1]);WriteString(" ");
                    WriteString(nombre[s2]);WriteString(" ");
                    WriteString(nombre[s3]);
                END;
            END;
        END;
    END;
    WriteLn; END CucuruchoDisponible;
```

END Helado.

MODULE ppal;

```
FROM Helado IMPORT sabores, Existencias, IncrementaHelado,
EsPosibleCucurucho,CucuruchoDisponible;
FROM InOut IMPORT WriteInt, WriteLn, WriteString, WriteCard;
VAR s1,s2,s3 : sabores;
    peso : CARDINAL;
    e1 : Existencias;
```

BEGIN

```
FOR s1:=chocolate TO menta DO
    e1[s1]:=200;
END;
s1:=chocolate;
peso:=50;
IncrementaHelado(e1,s1,peso);
s1:=chocolate; s2:=fresa; s3:=limon;
e1[limon]:=20;
CucuruchoDisponible (e1);
FOR s1:=chocolate TO menta DO
```

```
WriteString("sabor : ");WriteInt(ORD(s1),2);WriteCard(e1[s1],6);WriteLn;  
END;  
END ppal.
```

EXAMEN RESUELTO SEPT. 2005 TIPOS A,B,C:

Realizar un TAD para representar números enteros grandes (de hasta 100 dígitos). El TAD debe disponer de una operación de lectura de un entero grande introducido por teclado. Y una función que realice la suma: dados dos enteros grandes devuelve otro entero grande resultado de la suma, en caso de superar el resultado el rango de valores la función devolverá un número con todos los dígitos a cero.

DEFINITION MODULE GRANDES;

```
TYPE EnteroGrande = ARRAY [1..100] OF INTEGER;
  cadena = ARRAY[1..100] OF CHAR;
PROCEDURE Lee (VAR n : EnteroGrande);
PROCEDURE Suma (VAR n1,n2: EnteroGrande):EnteroGrande;
END GRANDES.
```

IMPLEMENTATION MODULE GRANDES;

```
FROM InOut IMPORT Read, Write;
```

PROCEDURE Lee (VAR n : EnteroGrande);

```
VAR i,j : INTEGER;
  repetir : BOOLEAN;
  c : CHAR;
  texto : cadena;
BEGIN
  i:=1; repetir := TRUE;
  WHILE repetir DO (*lo leo como string y luego lo paso al array*)
    Read(c);      (* de enteros *)
    IF (ORD(c) < ORD("0")) OR (ORD(c) > ORD("9")) THEN
      repetir := FALSE;
      texto[i]:=CHR(0);
    ELSE
      texto[i] := c;
      Write(texto[i]);
      INC(i);
    END;
  END;
END;
```

```
DEC(i); j:=100; (* crear array de enteros - ALINEAR A LA DCHA.*)
WHILE i>0 DO
  n[j]:= ORD(texto[i])-ORD("0");
  DEC(i); DEC(j);
END;
END Lee;
```

PROCEDURE Suma (VAR n1,n2: EnteroGrande):EnteroGrande;

```

VAR i,llevada,sum : INTEGER;
    rdo: EnteroGrande;
BEGIN
    llevada:=0;
    FOR i := 100 TO 1 BY -1 DO
        sum:= n1[i]+n2[i]+llevada;
        IF sum>9 THEN
            sum := sum MOD 10; rdo[i]:=sum; llevada := 1;
        ELSE
            rdo[i]:=sum; llevada := 0;
        END;
    END;
    IF llevada=1 THEN (* si hay rebasamiento poner todo a 0*)
        FOR i:=1 TO 100 DO rdo[i]:=0; END;
    END;
    RETURN rdo;
END Suma;
END GRANDES.

```

MODULE ppal;

```

FROM GRANDES IMPORT EnteroGrande, Lee, Suma;
FROM InOut IMPORT WriteString, WriteLn, WriteInt;
VAR numero1,numero2,total : EnteroGrande;
    j,i: INTEGER;
BEGIN
    Lee(numero1);
    WriteLn;
    Lee(numero2);
    WriteLn;
    total := Suma(numero1,numero2);
    i:=1;
    WHILE total[i]=0 DO INC(i); END; (*escribe el numero*)
    FOR j :=i TO 100 DO
        WriteInt(total[j],1);
    END;
END ppal.

```

EXAMEN RESUELTO SEPT. 2005 TIPO D:

Realice un tipo abstracto de datos para que un programa sea capaz de trabajar con números complejos. El modulo tendrá procedimientos para la suma de dos números complejos y el calculo del módulo de un numero complejo.

Sumar dos números $Z1+Z2$: $Z1(x1,x2) + Z2(y1,y2) = (x1+x2,y1+y2)$

Modulo de $Z1$: $|Z1| = \sqrt{(x1*x1 + x2*x2)}$

DEFINITION MODULE Complejo;

```
TYPE NComplejo = ARRAY [1..2] OF REAL;
```

```
PROCEDURE Suma (z1,z2 :NComplejo):NComplejo;
```

```
PROCEDURE Modulo (z1 : NComplejo):REAL;
```

```
END Complejo.
```

IMPLEMENTATION MODULE Complejo;

```
FROM MathLib0 IMPORT sqrt;
```

```
PROCEDURE Suma (z1,z2 :NComplejo):NComplejo;
```

```
VAR rdo : NComplejo;
```

```
BEGIN
```

```
rdo[1]:=z1[1]+z1[2];
```

```
rdo[2]:=z2[1]+z2[2];
```

```
RETURN rdo;
```

```
END Suma;
```

```
PROCEDURE Modulo (z1 : NComplejo):REAL;
```

```
VAR x : REAL;
```

```
BEGIN
```

```
x:=sqrt(z1[1]*z1[1]+z1[2]*z1[2]);
```

```
RETURN x;
```

```
END Modulo;
```

```
END Complejo.
```

Este modulo no lo pide en el examen, lo pongo aqui por que sirve para compilarlo y probar que el TAD funciona.

MODULE ppal;

```
FROM Complejo IMPORT NComplejo, Suma, Modulo;
```

```
FROM RealInOut IMPORT WriteReal;
```

```
FROM InOut IMPORT WriteString, WriteLn;
```

```
VAR complejo1, complejo2, complejo3 : NComplejo;
```

```
x: REAL;
```

```
BEGIN
```

```
complejo1[1]:=1.0; complejo1[2]:=2.0;
```

```
complejo2[1]:=3.0; complejo2[2]:=4.0;
complejo3:=Suma(complejo1,complejo2);
WriteString("SUMA : ");WriteReal(complejo3[1],5);
WriteString(",");WriteReal(complejo3[2],5); WriteLn;
x:=Modulo(complejo1);
WriteString("MODULO complejo1 : ");WriteReal(x,5);
END ppal.
```

FEB.2005 TIPOS A,B,C,D

Realice un tipo abstracto de datos para gestionar un array de fechas que no están ordenadas. El módulo será capaz de encontrar la fecha más antigua y la más actual de las que almacena. Las fechas serán registros con los siguientes campos: día, mes, año. Las operaciones que se tiene que realizar son: Comparar dos fechas, buscar la fecha antigua, buscar la fecha más reciente.

(Sólo alumnos de los planes antiguos). También se debe realizar la operación de mostrar todas las fechas del mes de Mayo.

DEFINITION MODULE fechas;

TYPE Tfecha = RECORD

 dia,mes,ano : INTEGER;

END;

PROCEDURE compara(f1,f2 : Tfecha):INTEGER;

(*f1=f2 devuelve 0; f1>f2 devuelve 1; f1

PROCEDURE masantigua(lista : ARRAY OF Tfecha):CARDINAL;

(*devuelve la posicion de la fecha mas antigua del array*)

PROCEDURE masreciente(lista : ARRAY OF Tfecha):CARDINAL;

(*devuelve la posicion de la fecha mas reciente del array*)

END fechas.

IMPLEMENTATION MODULE fechas;

PROCEDURE compara(f1,f2 : Tfecha):INTEGER;

BEGIN

 IF f1.ano>f2.ano THEN

 RETURN 1;

 ELSIF f1.ano

 RETURN -1;

 ELSIF f1.mes>f2.mes THEN

 RETURN 1;

 ELSIF f1.mes

 RETURN -1;

 ELSIF f1.dia>f2.dia THEN

 RETURN 1;

 ELSIF f1.dia

 RETURN -1;

```
ELSE RETURN 0;
END;
END compara;
```

```
PROCEDURE masantigua(lista : ARRAY OF Tfecha):CARDINAL;
  VAR antigua, i : CARDINAL;
BEGIN
  antigua:=0;
  FOR i:=1 TO HIGH(lista) DO
    IF compara(lista[antigua],lista[i])>0 THEN
      antigua:=i;
    END;
  END;
  RETURN antigua;
END masantigua;
```

```
PROCEDURE masreciente(lista : ARRAY OF Tfecha):CARDINAL;
  VAR reciente, i : CARDINAL;
BEGIN
  reciente:=0;
  FOR i:=1 TO HIGH(lista) DO
    IF compara(lista[reciente],lista[i])<0 THEN
      reciente:=i;
    END;
  END;
  RETURN reciente;
END masreciente;
END fechas.
```

```
MODULE ppal;
  FROM InOut IMPORT WriteInt, WriteLn;
  FROM fechas IMPORT Tfecha, compara, masantigua,masreciente;
  TYPE Tlista = ARRAY [0..2] OF Tfecha;
  VAR l : Tlista;
      res : INTEGER;
      ant, rec : CARDINAL;
BEGIN
  l[0].dia := 2; l[0].mes := 1; l[0].ano := 2006;
  l[1].dia := 1; l[1].mes := 1; l[1].ano := 2001;
  l[2].dia := 2; l[2].mes := 1; l[2].ano := 2005;
  res:=compara (l[1],l[2]);
  ant:=masantigua(l);
  rec:=masreciente(l);
  WriteLn; WriteInt(res,5);
  WriteLn; WriteInt(ant,5);
  WriteLn; WriteInt(rec,5);
END ppal.
```

EXAMEN RESUELTO SEPT. 2004 TIPOS A,B,C,D:

Realizar un tipo abstracto de datos para manejar y codificar mensajes de texto (compuestos únicamente por las 27 letras del alfabeto español) de usuarios de teléfono, con las operaciones:

- Generar_clave: la clave k será el resto de dividir el número de teléfono del usuario entre 27.
- Codificar_mensaje: sustituye cada letra del mensaje original por la que ocupa k posiciones más adelante en la secuencia del alfabeto.

DEFINITION MODULE Codifica;

```
PROCEDURE Generar_Clave(telefono:LONGCARD): CARDINAL;  
PROCEDURE Codificar_Mensaje(VAR mensaje : ARRAY OF CHAR; k :  
CARDINAL);  
END Codifica.
```

IMPLEMENTATION MODULE Codifica;

PROCEDURE Generar_Clave(telefono:LONGCARD): CARDINAL;

```
VAR temp : LONGCARD;  
k : CARDINAL;  
BEGIN  
temp:=telefono MOD 27L; (*LONGCARD acaban en L *)  
k:=VAL(CARDINAL,temp); (*lo convierte a CARDINAL*)  
RETURN k;  
END Generar_Clave;
```

PROCEDURE Codificar_Mensaje(VAR mensaje : ARRAY OF CHAR; k : CARDINAL);

```
(* Para simplificar supondre que los mensajes estan en MAYUSCULAS*)  
VAR cont,temp : CARDINAL;  
BEGIN  
cont := 0;  
WHILE (mensaje[cont]>="A") AND (mensaje[cont]<="Z") DO  
temp := ORD(mensaje[cont])+k;  
IF temp>ORD("Z") THEN  
temp:=temp-ORD("Z")+ORD("A")-1;  
END;  
mensaje[cont]:=CHR(temp);  
INC (cont);  
END;  
END Codificar_Mensaje;
```

```
END Codifica.
```

Este modulo no lo pide en el examen, lo pongo aqui por que sirve para compilarlo y probar que el TAD funciona.

```
MODULE ppal;
FROM InOut IMPORT WriteString, WriteCard, WriteLn;
FROM Codifica IMPORT Generar_Clave,Codificar_Mensaje;
  VAR temp : LONGCARD;
      clave : CARDINAL;
      texto : ARRAY [1..40] OF CHAR;
BEGIN
  temp:=985555555L;
  clave:=Generar_Clave(temp);
  WriteString("clave : "); WriteCard(clave,5); WriteLn;
  texto[1]:="A"; texto[2]:="Z"; texto[3]:="L";
  WriteString(texto); WriteLn;
  clave:=1; (*para probar facilmente que funciona*)
  Codificar_Mensaje(texto,clave);
  WriteString(texto); WriteLn;
END ppal.
```

SEPT. 2004 TIPO E

Desarrolle el Tipo Abstracto de Datos (TAD) Cesta de la Compra, que podrá contener los elementos: pan, bebida, carne, pescado y verdura. El TAD deberá disponer de las operaciones para:

- Vaciar la cesta
- Introducir un elemento en la cesta
- Sacar un elemento de la cesta
- Contestar si un elemento está en la cesta (Solo si es del Plan Viejo).
- Imprimir los elementos que contiene la cesta

DEFINITION MODULE Compra;

```
TYPE Telemento = ( pan, bebida, carne, pescado, verdura );
  Tcesta = SET OF Telemento;
PROCEDURE vacia(VAR cesta : Tcesta);
PROCEDURE mete(VAR cesta : Tcesta);
PROCEDURE saca(VAR cesta : Tcesta);
PROCEDURE escribe(cesta : Tcesta);
END Compra.
```

IMPLEMENTATION MODULE Compra;

```
FROM InOut IMPORT WriteString, WriteLn, ReadInt;
```

PROCEDURE vacia(VAR cesta : Tcesta);

```
BEGIN
  cesta:=Tcesta{};
END vacia;
```

PROCEDURE mete(VAR cesta : Tcesta);

```
  VAR opcion : INTEGER;
BEGIN
  REPEAT
    WriteLn;
    WriteString("ELEMENTO A INTRODUCIR (0=pan, 1=bebida, 2=carne,
3=pescado, 4=verdura) : ");
    ReadInt(opcion);
  UNTIL (opcion>=0) AND (opcion<5);
  INCL(cesta,VAL(Telemento,opcion));
END mete;
```

PROCEDURE saca(VAR cesta : Tcesta);

```
  VAR opcion : INTEGER;
BEGIN
```

```

REPEAT
  WriteLn;
  WriteString("ELEMENTO A SACAR (0=pan, 1=bebida, 2=carne, 3=pescado,
4=verdura) : ");
  ReadInt(opcion);
  UNTIL (opcion>=0) AND (opcion<5);
  EXCL(cesta,VAL(Telemento,opcion));
END saca;

```

PROCEDURE escribe(cesta : Tcesta);

```

  VAR articulo : Telemento;
BEGIN
  WriteLn;
  FOR articulo := pan TO verdura DO
    IF articulo IN cesta THEN
      CASE articulo OF
pan : WriteString("Pan"); WriteLn; |
bebida : WriteString("Bebida"); WriteLn; |
carne : WriteString("Carne"); WriteLn; |
pescado : WriteString("Pescado"); WriteLn; |
verdura : WriteString("Verdura"); WriteLn; |
      END;
    END;
  END;
END escribe;

```

END Compra.

MODULE ppal;

```

FROM InOut IMPORT WriteString, WriteLn;
IMPORT Compra;
  VAR arti : Compra.Telemento;
  cesta1 : Compra.Tcesta;
BEGIN
  Compra.vacia(cesta1);
  Compra.mete(cesta1);
  Compra.mete(cesta1);
  Compra.mete(cesta1);
  WriteLn; WriteString("Cesta con tres articulos:");WriteLn;
  Compra.escribe(cesta1);
  Compra.saca(cesta1);
  WriteLn; WriteString("Cesta con dos articulos:");WriteLn;
  Compra.escribe(cesta1);
  WriteLn; WriteString("Cesta vacia:");WriteLn;
  Compra.vacia(cesta1);
  Compra.escribe(cesta1);
END ppal.

```

EXAMEN RESUELTO SEPT. 2004 TIPO F:

Realizar un Tipo Abstracto de Datos para jugar a la lotería primitiva. Cada apuesta es una selección de 6 números del 1 al 49 en la que se debe guardar el nombre del apostante y su DNI. El número máximo de apuestas será de 1000. Las operaciones posibles serán:

- Iniciar sorteo para borrar todas las apuestas anteriores
- Realizar UNA apuesta y registrar el apostante
- Efectuar sorteo (Sólo si es del plan viejo).

NOTA: Suponer que existe una función predefinida RANDOM49(x) que cada vez que se la invoca devuelve un número comprendido entre el 1 y el 49

DEFINITION MODULE Loteria;

```
TYPE Apuesta = RECORD
    contador : INTEGER;
    Nombre: ARRAY [0..20] OF CHAR;
    DNI: ARRAY [0..12] OF CHAR;
    Numero: ARRAY [1..6] OF CARDINAL;
END;
TYPE Sorteo = ARRAY [1..1000] OF Apuesta;

PROCEDURE IniciarSorteo(VAR A:Sorteo);
PROCEDURE RealizarApuesta(VAR A:Sorteo);

END Loteria.
```

IMPLEMENTATION MODULE Loteria;

```
FROM InOut IMPORT WriteString,WriteCard,WriteLn,ReadString,ReadCard;
```

```
PROCEDURE IniciarSorteo(VAR A:Sorteo);
    VAR i : CARDINAL;
BEGIN
    FOR i := 1 TO 1000 DO
        A[i].contador := 0; (*para inicializar el array pongo a 0 el campo
        contador de todos los elementos*)
    END;
END IniciarSorteo;

PROCEDURE RealizarApuesta(VAR A:Sorteo);
    VAR i,j : CARDINAL;
```

```

BEGIN
  i := 1;
  WHILE A[i].contador <> 0 DO (*busco el ultimo elemto del array*)
    INC(i);
  END;
  A[i].contador:=i;
  FOR j:=1 TO 6 DO
    WriteLn; WriteString("numero ");
    WriteCard(j,2); WriteString("?: ");
    ReadCard(A[i].Numero[j]);
  END;
  WriteLn; WriteString("Nombre?: ");
  ReadString(A[i].Nombre);
  WriteLn;
  WriteString("DNI? (sin letra): ");
  ReadString(A[i].DNI); WriteLn;
END RealizarApuesta;
END Loteria.

```

MODULE ppal;

```

FROM InOut IMPORT Read, WriteString, WriteCard, WriteLn, WriteInt;
FROM Loteria IMPORT Sorteo, IniciarSorteo, RealizarApuesta;
VAR opcion :CHAR;
    i,j : CARDINAL;
    tabla : Sorteo;
BEGIN
  IniciarSorteo(tabla);
  REPEAT
    WriteString("nueva apuesta (s/n)?: "); Read(opcion); WriteLn;
    IF opcion <> "n" THEN
      RealizarApuesta(tabla);
    END;
  UNTIL opcion = "n";
  i:=1; (*escribe la tabla*)
  WriteString("N. / NOMBRE / DNI / NUMEROS"); WriteLn;
  WHILE tabla[i].contador <> 0 DO
    WriteInt(tabla[i].contador,4); WriteString(" / ");
    WriteString(tabla[i].Nombre); WriteString(" / ");
    WriteString(tabla[i].DNI); WriteString(" / ");
    FOR j := 1 TO 6 DO
      WriteCard(tabla[i].Numero[j],3);
    END;
    WriteLn;
  INC(i);
  END;
END ppal.

```

EXAMEN RESUELTO FEB. 2004 TIPOS A,B,C,D:

Un alumno de la UNED está identificado por un año de matrícula, un código de carrera y su NIF. Cada NIF (número de identificación fiscal) tiene asignada una letra. La asignación se realiza a partir del resto de la división del DNI entre 23. A cada uno de los posibles valores obtenidos le corresponde una letra del alfabeto (al 0 le corresponde la T, al 1 la R, al 2 la W, etc.). Se proporciona el módulo “NIF” con la función “AsignarLetra”, que nos da la letra que corresponde a cada uno de los restos posibles del 0 al 23.

Se pide:

- Realizar un tipo abstracto de datos, TipoAlumno, con procedimientos o funciones para:

- PedirAlumno: introducir por teclado la información necesaria (año, código y DNI) que identifica a un alumno.
- EscribirAlumno: escribir en la pantalla TODA (año, código y NIF) la información de un alumno.

DEFINITION MODULE TipoAlumno;

TYPE alumno = RECORD

 anno : CARDINAL;

 DNI : LONGCARD;

 letra : CHAR;

 cod_carrera: CARDINAL;

END;

PROCEDURE PedirAlumno(VAR registro:alumno);

PROCEDURE EscribirAlumno(registro:alumno);

END TipoAlumno.

IMPLEMENTATION MODULE TipoAlumno;

FROM InOut IMPORT

WriteLn, WriteLongCard, ReadLongCard, WriteString, Write, ReadCard;

IMPORT NIF;

PROCEDURE PedirAlumno(VAR registro:alumno);

 VAR resto : LONGCARD;

 restointeger: INTEGER;

BEGIN

 WriteLn;

 WriteString("Año de matrícula?: ");

 ReadCard(registro.anno);

 WriteLn;

 WriteString("DNI?: ");

 ReadLongCard(registro.DNI);

```
(* LongCard acaban en L, lo hago así para ilustrar el uso de LONGCARD*)
resto := registro.DNI MOD 23L;
restointeger := VAL(INTEGER,resto); (*lo convierte en integer*)
registro.letra:=NIF.AsignarLetra(restointeger);
WriteLn;
WriteString("Codigo de carrera?: ");
ReadCard(registro.cod_carrera);
WriteLn;
END PedirAlumno;
```

```
PROCEDURE EscribirAlumno(registro:alumno);
```

```
BEGIN
```

```
WriteString("Año de matricula: ");
WriteCard(registro.anno,5); WriteLn;
WriteString("Codigo de carrera: ");
WriteCard(registro.cod_carrera,6); WriteLn;
WriteString("NIF: ");
WriteLongCard(registro.DNI,10);
WriteString("-");
Write(registro.letra); WriteLn;
```

```
END EscribirAlumno;
```

```
END TipoAlumno.
```

EXAMEN RESUELTO FEB. 2003 TIPOS G,H,I:

Realizar un tipo abstracto de datos (TAD) de una ecuación de 2º grado de la forma ($ax^2 + bx + c = 0$) con los procedimientos de entrada_coeficientes y calculo_raices. Realizar un programa principal que utilice los procedimientos definidos en el TAD.

DEFINITION MODULE Ecuacion;

```
TYPE Tecuacion = ARRAY [1..3] OF REAL;
TYPE Tsolucion = ARRAY [1..2] OF REAL;
PROCEDURE entrada_coeficientes(VAR entrada:Tecuacion);
PROCEDURE calculo_raices(entrada:Tecuacion; VAR solucion:Tsolucion);
END Ecuacion.
```

IMPLEMENTATION MODULE Ecuacion;

```
FROM InOut IMPORT WriteString, WriteLn;
FROM MathLib0 IMPORT sqrt;
FROM RealInOut IMPORT ReadReal;
```

PROCEDURE entrada_coeficientes(VAR entrada:Tecuacion);

```
BEGIN
  WriteLn; WriteString("ax2+bx+c=0"); WriteLn;
  WriteString("coeficiente a? : "); ReadReal(entrada[1]); WriteLn;
  WriteString("coeficiente b? : "); ReadReal(entrada[2]); WriteLn;
  WriteString("coeficiente c? : "); ReadReal(entrada[3]); WriteLn;
END entrada_coeficientes;
```

PROCEDURE calculo_raices(entrada:Tecuacion; VAR solucion:Tsolucion);

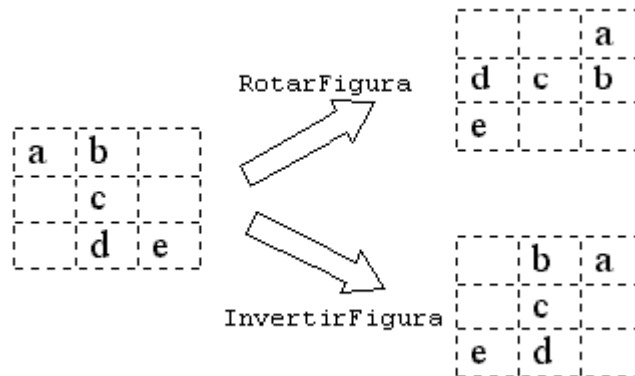
```
  VAR temp,raiz : REAL;
BEGIN
  temp:= ((entrada[2]*entrada[2])-(4.0*entrada[1]*entrada[3]));
  IF temp>0.0 THEN
    raiz:=sqrt(temp);
    solucion[1]:=(-entrada[2]+raiz)/(2.0*entrada[1]);
    solucion[2]:=(-entrada[2]-raiz)/(2.0*entrada[1]);
  ELSE
    WriteString("SIN SOLUCION");
  END;
END calculo_raices;
```

```
END Ecuacion.
```

```
MODULE ppal;  
  IMPORT Ecuacion;  
  FROM InOut IMPORT WriteInt, WriteLn, WriteString, WriteCard;  
  FROM RealInOut IMPORT WriteReal;  
  
  VAR ecuacion1 : Ecuacion.Tecuacion;  
      solucion1 : Ecuacion.Tsolucion;  
  
  BEGIN  
    Ecuacion.entrada_coeficientes(ecuacion1);  
    Ecuacion.calculo_raices(ecuacion1,solucion1);  
    WriteLn; WriteString("solucion 1 : "); WriteReal(solucion1[1],10);  
    WriteLn; WriteString("solucion 2 : "); WriteReal(solucion1[2],10);  
  END ppal.
```

EXAMEN RESUELTO FEB. 2002 TIPO E:

Construir un TAD (Tipo Abstracto de Datos) capaz de contener una figura como la del ejemplo en una cuadrícula de dimensión 3x3. El TAD dispondrá únicamente de los procedimientos RotarFigura e InvertirFigura, que transformarán la figura que reciban de entrada según se describe en el siguiente ejemplo:



DEFINITION MODULE Figura;

```
TYPE Cuadro = ARRAY [1..3],[1..3] OF CHAR;  
PROCEDURE RotarFigura (VAR entrada : Cuadro);  
PROCEDURE InvertirFigura (VAR entrada : Cuadro);  
END Figura.
```

IMPLEMENTATION MODULE Figura;

```
PROCEDURE RotarFigura (VAR entrada : Cuadro);  
  VAR fila : INTEGER;  
      temp : Cuadro;  
BEGIN  
  FOR fila := 1 TO 3 DO  
    temp[fila,1]:=entrada[3,fila];  
    temp[fila,2]:=entrada[2,fila];  
    temp[fila,3]:=entrada[1,fila];  
  END;  
  entrada:=temp;  
END RotarFigura;
```

```
PROCEDURE InvertirFigura (VAR entrada : Cuadro);  
  VAR fila : INTEGER;  
      temp : Cuadro;  
BEGIN  
  FOR fila := 1 TO 3 DO  
    temp[fila,1]:=entrada[fila,3];  
    temp[fila,2]:=entrada[fila,2];  
    temp[fila,3]:=entrada[fila,1];  
  END;
```

```
END;  
  entrada:=temp;  
END InvertirFigura;
```

```
END Figura.
```

Este modulo no lo pide en el examen, lo pongo aqui por que sirve para compilarlo y probar que el TAD funciona.

MODULE ppal;

```
FROM Figura IMPORT Cuadro, RotarFigura, InvertirFigura;  
FROM InOut IMPORT Write, WriteLn, WriteString, Read;  
VAR origen : Cuadro;  
  f,c : INTEGER;  
  opcion : CHAR;  
BEGIN  
  origen[1,1]:="a";origen[1,2]:="b";origen[1,3]:=" "  
  origen[2,1]:=" ";origen[2,2]:="c";origen[2,3]:=" "  
  origen[3,1]:=" ";origen[3,2]:="d";origen[3,3]:="e";  
  WriteString("ORIGINAL");WriteLn;  
  FOR f:=1 TO 3 DO  
    FOR c:=1 TO 3 DO  
      Write(origen[f,c]); Write(" ");  
    END;  
    WriteLn;  
  END;  
  WriteString("[R]otar o [I]nvertir ? :");Read(opcion);WriteLn;  
  IF opcion="R" THEN  
    RotarFigura(origen);  
    WriteString("ROTADO");WriteLn;  
  END;  
  IF opcion="I" THEN  
    InvertirFigura(origen);  
    WriteString("INVERTIDO");WriteLn;  
  END;  
  FOR f:=1 TO 3 DO  
    FOR c:=1 TO 3 DO  
      Write(origen[f,c]); Write(" ");  
    END;  
    WriteLn;  
  END;  
END ppal.
```

EXAMEN RESUELTO SEPT. 2001 TIPO C:

Implementar mediante un TAD (Tipo Abstracto de Datos) un vector de longitud 3 tal que en los extremos contenga los colores básicos (rojo, amarillo o azul), mientras que en el centro almacene la mezcla de dichos colores (rojo+amarillo=naranja, rojo+azul=violeta, amarillo+azul=verde). El TAD dispondrá del procedimiento *MezclarVector*, que al recibir un vector con colores en los extremos rellenara la posición central, tal como se indica en el ejemplo:



DEFINITION MODULE Color;

```
TYPE Colores = (rojo, amarillo, azul, naranja, violeta, verde);
VectorColores = ARRAY[1..3] OF Colores;
PROCEDURE MezclarVector (VAR vector : VectorColores);
END Color.
```

IMPLEMENTATION MODULE Color;

```
PROCEDURE MezclarVector (VAR vector : VectorColores);
BEGIN
  IF (vector[1]=rojo) AND (vector[3]=amarillo) THEN vector[2]:=naranja END;
  IF (vector[1]=rojo) AND (vector[3]=azul) THEN vector[2]:=violeta END;
  IF (vector[1]=azul) AND (vector[3]=rojo) THEN vector[2]:=violeta END;
  IF (vector[1]=azul) AND (vector[3]=amarillo) THEN vector[2]:=verde END;
  IF (vector[1]=amarillo) AND (vector[3]=rojo) THEN vector[2]:=naranja END;
  IF (vector[1]=amarillo) AND (vector[3]=azul) THEN vector[2]:=verde END;
END MezclarVector;
END Color.
```

Este modulo no lo pide en el examen, lo pongo aqui por que sirve para compilarlo y probar que el TAD funciona.

MODULE ppal;

```
FROM Color IMPORT Colores, VectorColores, MezclarVector;
FROM InOut IMPORT WriteInt;
VAR terna : VectorColores;
BEGIN
  terna[1]:=amarillo;
  terna[3]:=azul;
  MezclarVector(terna);
  WriteInt (ORD(terna[2]),3);
END ppal.
```

EXAMEN RESUELTO FEB. 2001 :

Dada una tabla de hasta 10 puntos en el plano (x,y). Realizar un tipo abstracto de dato con las operaciones siguientes:

1. Determinar si algún punto coincide con el primero devolviendo su posición en la tabla.
2. Suma de la longitud de los segmentos entre los puntos sucesivos de la tabla desde un punto inicial a otro final.
3. Perímetro del polígono cerrado formado por el punto inicial y los sucesivos puntos hasta el primer punto de la tabla que coincide con el punto inicial.

DEFINITION MODULE Exam;

```
TYPE TipoPunto = RECORD
    x,y:REAL
END;
TipoIndice=[0..9];
TipoTabla=ARRAY TipoIndice OF TipoPunto;
PROCEDURE Coincide(t:TipoTabla):TipoIndice;
PROCEDURE Suma(t:TipoTabla;p1,p2:TipoIndice):REAL;
PROCEDURE Poligono(t:TipoTabla):REAL;
END Exam.
```

IMPLEMENTATION MODULE Exam;

```
From MathLib0 IMPORT sqrt;
PROCEDURE Longitud(pto1,pto2:TipoPunto):REAL;
BEGIN
    RETURN sqrt((pto2.x-pto1.x)*(pto2.x-pto1.x)+
        (pto2.y-pto1.y)*(pto2.y-pto1.y));
END Longitud;
PROCEDURE Coincide(t:TipoTabla):TipoIndice;
    VAR encontrado:BOOLEAN;
    p:TipoIndice;
BEGIN
    encontrado:=FALSE; p:=HIGH(t);
    WHILE (p>0) OR (NOT encontrado) DO
        encontrado:=(t[0].x=t[p].x) AND (t[0].y=t[p].y);
        DEC(p)
    END;
    RETURN p+1;
END Coincide;
PROCEDURE Suma(t:TipoTabla;p1,p2:TipoIndice):REAL;
    VAR suma:REAL;
    i:TipoIndice;
BEGIN
    suma:=0;
    FOR i:=p1 TO p2-1 DO
```

```
    suma:=suma+Longitud(t[i],t[i+1]);  
END;  
RETURN suma;  
END Suma;  
PROCEDURE Poligono(t:TipoTabla):REAL;  
  VAR posi:TipoIndice;  
      perimetro:REAL;  
BEGIN  
  posi:=Coincide(t);  
  perimetro:=Suma(t,0,posi);  
  RETURN perimetro;  
END Poligono;  
END Exam.
```

EXAMEN RESUELTO FEB. 2000:

Definir en un modulo un tipo de dato que contenga la siguiente información altura, velocidad y rumbo (Norte, Sur, Este, Oeste) y los procesos LeerVelocidad, LeerAltura, LeerRumbo. Desarrollar solamente el módulo de definición. A continuación desarrollar un Programa que haga uso del tipo definido y que imprima por pantalla la información referente a 20 aviones.

DEFINITION MODULE VelAltRu;

TYPE

TipoRumbo=(Norte, Sur, Este, Oeste);

TipoVelAltRu = RECORD

Velocidad: CARDINAL;

Altura: CARDINAL;

Rumbo: TipoRumbo;

END;

PROCEDURE LeerRumbo(AvionaEvaluar: TipoVelAltRu): TipoRumbo;

PROCEDURE LeerVelocidad(AvionaEvaluar: TipoVelAltRu): CARDINAL;

PROCEDURE LeerAltura(AvionaEvaluar: TipoVelAltRu): CARDINAL;

END VelAltRu.

MODULE Aviones;

FROM InOut IMPORT WriteCard, WriteString, WriteLn, WriteInt;

IMPORT VelAltRu;

CONST NumAviones=20;

TYPE TipodatosAvion=ARRAY [1..NumAviones] OF VelAltRu.TipoVelAltRu;

VAR

Avion: TipodatosAvion;

Contador: INTEGER;

PROCEDURE ImprimirDatos(AvionaImprimir: VelAltRu.TipoVelAltRu, IdAvion:

INTEGER)

VAR

Altura, Velocidad: CARDINAL;

Rumbo: VelAltRu.TipoRumbo;

BEGIN

Altura:=VelAltRu.LeerAltura(AvionaImprimir);

Velocidad:= VelAltRu.LeerVelocidad(AvionaImprimir);

Rumbo:= VelAltRu.LeerRumbo(AvionaImprimir);

WriteInt(IdAvion,5); WriteCard(Altura,8);

WriteCard(Velocidad,8); WriteString(" ");

CASE Rumbo OF

VelAltRu.Norte: WriteString("Norte")|

VelAltRu.Sur: WriteString("Sur")|

VelAltRu.Este: WriteString("Este")|

VelAltRu.Oeste: WriteString("Oeste")| END;

WriteLn;

END ImprimirDatos;

BEGIN WriteString("Avion Altura Velocidad Rumbo"); WriteLn;

FOR Contador:=1 TO NumAviones DO

ImprimirDatos(Avion[Contador]; Contador);

END;
END;

EXAMEN RESUELTO FEB. 1999 2ª SEMANA:

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.

DEFINITION MODULE recinto;

```
PROCEDURE VacíaSala();
PROCEDURE Informe();
PROCEDURE OcupaButaca();
PROCEDURE LiberaButaca();
END recinto.
```

IMPLEMENTATION MODULE recinto;

```
FROM InOut IMPORT WriteInt, ReadInt, WriteString, WriteLn;
CONST filas = 5;
      butacas = 7;
VAR sala : ARRAY [1..filas],[1..butacas] OF BOOLEAN;
(* DATO ENCAPSULADO *)
```

PROCEDURE VacíaSala();

```
VAR i,j : INTEGER;
BEGIN
FOR i:=1 TO filas DO
FOR j:=1 TO butacas DO
sala[i,j]:=FALSE;
END;
END;
END VacíaSala;
```

PROCEDURE Informe();

```
VAR i,j,vacias : INTEGER;
BEGIN
vacias:=0;
FOR i:=1 TO filas DO
FOR j:=1 TO butacas DO
IF sala[i,j]=FALSE THEN INC(vacias); END;
END;
END;
```

```

    WriteString("libres : "); WriteInt(vacias,4);WriteLn
END Informe;
PROCEDURE OcupaButaca();
    VAR fila,butaca : INTEGER;
BEGIN
    WriteString("Fila : ");ReadInt(fila);WriteLn;
    WriteString("Butaca : ");ReadInt(butaca);WriteLn;
    sala[fila,butaca]:=TRUE;
END OcupaButaca;
PROCEDURE LibraButaca();
    VAR fila,butaca : INTEGER;
BEGIN
    WriteString("Fila : ");ReadInt(fila);WriteLn;
    WriteString("Butaca : ");ReadInt(butaca);WriteLn;
    sala[fila,butaca]:=FALSE;
END LibraButaca;
END recinto.

```

```

MODULE ppal;
    FROM InOut IMPORT WriteInt, ReadInt, WriteString, WriteLn;
    IMPORT recinto;
    VAR opcion : INTEGER;
BEGIN
    recinto.VaciaSala;
    REPEAT
        WriteString("opcion 0=fin 1=informe 2=ocupar 3=librar ?");
        ReadInt(opcion);WriteLn;
        CASE opcion OF
            1 : recinto.Informe|
            2 : recinto.OcupaButaca|
            3 : recinto.LibraButaca|
        ELSE
            END
    UNTIL opcion=0;
END ppal.

```