

APELLIDOS:	NOMBRE:
ESPECIALIDAD:	TURNO:
DNI:	Duración: 2:30 h.

Ejercicio 1	Ejercicio 2	Ejercicio 3	Ejercicio 4	TOTAL

El examen se aprueba con una nota igual o superior a 50 puntos. De ellos, un mínimo de 8 se deben obtener en el Ejercicio 1, un mínimo de 5 en el Ejercicio 2, un mínimo de 10 en el Ejercicio 3 y un mínimo de 10 en el Ejercicio 4.

Ejercicio 1: [20 puntos: respuesta acertada = +2, respuesta incorrecta = -0.75]. Marque claramente la opción correcta. Si existe más de una marca, o no queda clara la opción escogida, el ejercicio se puntuará como respuesta incorrecta

1.1. Dada la siguiente definición de nodo:

```
TPtrNodo = ^TNodo;  
TNodo = RECORD  
  codigo : integer;  
  psig: TPtrNodo; {puntero al siguiente elemento de la lista}  
END; {TNodo}
```

Y el siguiente procedimiento, que recibe como argumento la dirección de memoria del primer nodo de una lista enlazada

```
PROCEDURE muestraElementos (pAux: TPtrNodo);  
BEGIN  
  IF pAux<>NIL THEN BEGIN  
    muestraElementos (pAux^.pSig);  
    writeln (pAux^.codigo);  
  END  
END;  
END;
```

Indicar cual de las siguientes afirmaciones es correcta:

- (a) El procedimiento no muestra nada ya que *writeln* está detrás de una llamada recursiva, por lo que cuando se va a ejecutar no estamos apuntando a ningún dato y hemos llegado al final de la lista.
- (b) El procedimiento nos muestra el dato almacenado en cada nodo de la lista desde el final hasta el principio.
- (c) El procedimiento nos muestra el dato almacenado en cada nodo de la lista desde el principio hasta al final.
- (d) Si la lista está vacía el procedimiento genera recursión infinita.

1.2. ¿Cuál de las siguientes funciones no puede utilizarse con ficheros binarios?

- (a) EOF
- (b) EOLN
- (c) read
- (d) write

1.3. La función FILESIZE devuelve:

- (a) El tamaño en bytes del fichero
- (b) El tamaño en bits del fichero
- (c) El número de registros que contiene el fichero
- (d) El valor del puntero

1.4. Dado el siguiente código:

```
TYPE
  TipoDia = 1..31;
  TipoMes = 1..12;
  TipoAnio = 1900..2010;
  TipoRango = 1..100;
  TipoCoche = (FORD, SEAT, RENAULT, CITROEN);
  TipoNombre = string[30];
  TipoFecha = RECORD
    dia: TipoDia;
    mes: TipoMes;
    anio: TipoAnio;
  END;
  TipoCliente = RECORD
    numCliente : integer;
    nombre: TipoNombre;
    fechaNac: TipoFecha;
    vehiculo: TipoCoche;
    sueldo: longint;
  END;
  TipoLista = ARRAY [TipoRango] OF TipoCliente;
VAR
  clientes: TipoLista;
  c: TipoCliente;
  f: TipoFecha;
  i: TipoCoche;
```

Indica que sentencia es incorrecta:

- (a) write (c.vehiculo);
- (b) clientes[3].vehiculo := succ(c.vehiculo);
- (c) clientes[10].fechaNac := f;
- (d) Ninguna respuesta es correcta ya que el código no compila;

1.5. Fíjese en la siguiente definición de tipos y sentencias. ¿Qué respuesta es correcta?

```
TYPE
  TNombre = string[50];
  TCuenta = string[20];
  TEdad = 1..99;
  TCodigo = 1..1000;
  TCliente = RECORD
    nombre: TNombre;
    cuenta: TCuenta;
    edad: TEdad;
    saldo: real;
  END;
  TClientes = array [1..1000] OF TCliente;
  TSucursal = RECORD
    codigo: TNombre;
    clientes: TClientes;
  END;
VAR
  cliente: TCliente;
  sucursal1, sucursal2: TSucursal;
BEGIN
  ...
  sucursal1[9] := cliente;
  sucursal2.clientes[50].saldo := cliente.saldo;
  ...
END;
```

- (a) Las dos instrucciones son correctas.
- (b) Ninguna instrucción es correcta, son tipos incompatibles.
- (c) La primera instrucción es correcta y la segunda es incorrecta.
- (d) La primera instrucción es incorrecta y la segunda correcta.

1.6. Indica que sentencia es incorrecta en el siguiente fragmento de código

```
TYPE
  Tregistro = RECORD
    edad: integer;
    nombre: string[10];
  END;
  Tfichero = FILE OF Tregistro;
VAR
  reg: Tregistro;
  fich: Tfichero;
  ed: integer;
BEGIN
  ...
  ASSIGN(fich, 'fichero.dat');
  REWRITE(fich);
  ...
END;
```

- (a) `reg.edad := ed;`
- (b) `write(fich, reg.edad);`
- (c) `readln (reg.nombre);`
- (d) Todas son correctas.

1.7. Hablando de strings, indica cuál de las siguientes afirmaciones es correcta:

- (a) La posición cero de string contiene el carácter ASCII cuyo valor ordinal indica el número de caracteres del string.
- (b) La posición cero almacena un carácter ASCII que será el primer elemento del string almacenado, es decir, no contiene ninguna información especial.
- (c) Para saber la longitud de un string se utiliza la función *len*.
- (d) No existe posición cero en los strings de PASCAL.

1.8. Dado el siguiente código, indique lo que se muestra por pantalla:

```
PROGRAM EXAMEN;
TYPE
  TEmpleado = RECORD
    codigo : integer;
    nombre: string[10];
  END;
  TPtrEmpleado = ^TEmpleado;
VAR
  p1: TPtrEmpleado;

PROCEDURE recogerDatos (VAR p: TPtrEmpleado);
VAR
  r: TEmpleado;
BEGIN
  p:=@r;
  p^.nombre:='examen';
  p^.codigo:=10;
END;

PROCEDURE mostrarDatos (p: TPtrEmpleado);
BEGIN
  writeln('Nombre:', p^.nombre, 'Codigo: ', p^.codigo);
END;

BEGIN {P. PRINCIPAL}
  recogerDatos (p1);
  mostrarDatos (p1);
END.
```

- (a) Muestra datos erróneos que no tienen nada que ver con lo que se intentaba almacenar porque la carga de datos del registro r se hace por medio del puntero.
- (b) Muestra datos erróneos que no tienen nada que ver con lo que se intentaba almacenar porque el registro r es local a recogerDatos y por lo tanto pierde su vigencia al finalizar el subprograma.
- (c) Nombre: examen Codigo: 10
- (d) Muestra datos erróneos que no tienen nada que ver con lo que se intentaba almacenar porque el procedimiento mostrarDatos debería recoger el puntero p por referencia.

1.9. El proceso de fusión requiere:

- (a) Que los ficheros a fusionar se copien íntegramente a memoria principal del ordenador.
- (b) Que el formato de los registros de los ficheros a fusionar sea el mismo y además los datos almacenados en cada registro sean siempre de un tipo ordinal.
- (c) Que alguno de los ficheros tenga el doble de registros que la suma de registros del resto de ficheros.
- (d) Que los ficheros a fusionar estén ordenados por el mismo campo utilizando además el mismo criterio de ordenación.

1.10. (SÓLO GESTIÓN) ¿Qué criterios deben tenerse en cuenta al descomponer un algoritmo en módulos?

- (a) Debe maximizarse la cohesión y el acoplamiento.
- (b) Debe maximizarse la cohesión y reducirse el acoplamiento.
- (c) Debe minimizarse la cohesión y maximizarse el acoplamiento.
- (d) Debe minimizarse la cohesión y el acoplamiento

1.10. (SÓLO SISTEMAS) En el siguiente fragmento de código, ¿qué afirmación es correcta?

```
TYPE
  TEvoluciongripe = (Sano, Moqueo, Estornudos, Tos, Doloresmusculares,
  Fiebre, Nausias, Cansancio);
VAR
  a, b : TEvoluciongripe;
  c : boolean;
BEGIN
  ...
  a:=succ(Moqueo);
  b:=pred(pred(Doloresmusculares));
  c:=ord(a) < ord(b);
  IF c THEN
    write('NO, ');
    writeln('A está más enfermo');
  ...
END.
```

- (a) Muestra por pantalla el mensaje “No, A está más enfermo”.
- (b) No compila, ya que no se puede aplicar el operador “<” con la función *ord*.
- (c) Muestra por pantalla el mensaje “A está más enfermo”.
- (d) La instrucción en la que se asigna valor a la variable “c” provoca un error en tiempo de ejecución ya que no se puede averiguar si un valor es mayor que otro cuando son ordinales.

Ejercicio 2:

[20 puntos]

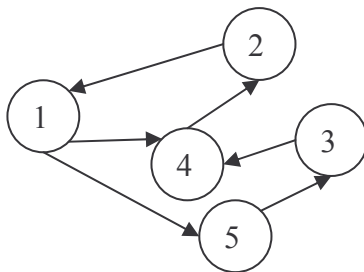
El grafo de la figura (a) se puede representar mediante una matriz $g[1..5,1..5]$ de booleanos, en la que el elemento $g[i,j]$ es *true* si y solo si existe un arco que va del nodo i al nodo j .

En el ejemplo de la figura, los nodos están numerados del 1 al 5. Desde el nodo 1 parten arcos hacia el nodo 4 y el 5, desde el nodo 2 se puede acceder al 1, etc.

Definición: Diremos que una permutación (un vector $v[1..n]$) de los n nodos de un grafo $g[1..n,1..n]$ es un camino hamiltoniano si se pueden recorrer todos los nodos del grafo en ese orden.

Ejemplo: $[1,5,3,4,2]$ es un camino hamiltoniano, porque existen los arcos $(1,5)$, $(5,3)$, $(3,4)$ y $(4,2)$.

Se pide: Construir un subprograma que compruebe si un vector, v , es un camino hamiltoniano para un grafo g . El subprograma devolverá *true* si es un camino hamiltoniano, *false* en caso contrario.



(a) Un grafo g

	1	2	3	4	5
1	F	F	F	T	T
2	T	F	F	F	F
3	F	F	F	T	F
4	F	T	F	F	F
5	F	F	F	F	F

(b) Matriz asociada a g

Solución:

```
FUNCTION hamiltoniano (v:Tvector; M:Tmatriz):boolean;
{ pre: vector con número de posiciones pares.
  Los valores almacenados en el vector han de estar comprendidos
entre 1..N}
{ post: True si es hamiltoniano, false en caso contrario.}
VAR
  i,j,k : integer;
  hamil:boolean;
BEGIN
  k:=1;
  i:=v[k];
  j:=v[k+1];
  k:=2;
  hamil:=true; {supongo que el grafo si es hamiltoniano}
  {mientras este en rango y siga siendo hamiltoniano...}
  while (hamil) and (k<=N) do begin
    if not m[i,j] then
      hamil:=false;
    i:=j;
    k:=k+1;
    j:=v[k];
  end;
  END;
hamiltoniano := hamil;
END;
```

Ejercicio 3:

[30 puntos]

Realizar un subprograma que reciba como argumento dos punteros índice, correspondientes a dos listas enlazadas de valores de tipo entero y nos devuelva un puntero a otra lista enlazada que contendrá (sin valores repetidos) los elementos existentes en las dos listas ordenados de manera ascendente.

Por ejemplo si recibimos dos listas con los siguientes valores.

Pindice1 → 5 → 7 → 3 → 8 → 2 → NIL

Pindice2 → 3 → 9 → 4 → 1 → NIL

Se generará una lista que contendrá:

pListaSalida → 1 → 2 → 3 → 4 → 5 → 7 → 8 → 9 → NIL

A modo de ayuda, se muestra la definición de nodo de la lista:

```
TPtrNodo = ^TNodo;
TNodo = RECORD
  codigo : integer;
  psig: TPtrNodo; {puntero al siguiente elemento de la lista}
END; {TNodo}
```

```
PROCEDURE generarOrdenada(p1, p2: TPtrNodo; VAR p: TPtrNodo);
{genera, a partir de las listas apuntadas por p1 y p2, una nueva lista
apuntada por p que contendrá los elementos no repetidos de las anteriores
de manera ordenada}
VAR {generar ordenada}
  paux: TPtrNodo; {para recorrer las listas}

PROCEDURE tratar(pTratada: TPtrNodo; VAR p: TPtrNodo);
{trata todos los elementos de la lista apuntada por pTratada insert ndolos
en la lista apuntada por p}
VAR {tratar}
  paux: TPtrNodo;

FUNCTION estaEnLista(p: TPtrNodo; n: integer):boolean;
{comprueba si el valor n est en la lista apuntada por p}
VAR
  paux: TPtrNodo;
  encontrado: boolean;
BEGIN
  paux:=p;
  encontrado:=FALSE;
  WHILE (paux<>NIL) AND (NOT ENCONTRADO) DO BEGIN
    IF pAux^.codigo = n THEN
      encontrado:=TRUE;
    paux:=paux^.psig;
  END;
  estaEnLista:=encontrado;
END; {estaEnLista}
```

```

PROCEDURE insertarOrdenado(VAR pIni:TptrNodo; n: integer);
{realiza la inserci3n de manera ordenada de un nuevo nodo que contendr
el valor n en la lista apuntada por pIni}
VAR
  pnuevo, pant, paux: TPtrNodo;
BEGIN
  new(pnuevo); {pedimos memoria para el nuevo elemento}
  pnuevo^.codigo:=n; {copiamos los datos del nodo}
  pnuevo^.psig:=NIL; {por si acaso terminase el 3ltimo de la lista}
  if (pIni=NIL) then {si la lista esta vac;a}
    pIni:=pnuevo
  else begin {lista no vac;a, hay q buscar posici3n de inserci3n}
    paux:=pIni;
    pant:=NIL;
    while((paux<>NIL) AND (paux^.codigo<pnuevo^.codigo)) do begin
      pant:=paux;
      paux:=paux^.psig;
    end; {while}
    {al acabar la b3squeda comprobamos}
    if (pant=NIL) then
      begin {inserci3n en primera posici3n}
        pnuevo^.psig:=pIni;
        pIni:=pnuevo;
      end
    else
      begin {inserci3n en posici3n intermedia o final}
        pnuevo^.psig:=paux;
        pant^.psig:=pnuevo;
      end
    end; {else}
  END; {procedure}

BEGIN {tratar}
  paux:=pTratada; {nos situamos puntero ;ndice lista a tratar}
  WHILE paux<>NIL DO BEGIN {recorremos todos sus elementos}
    IF NOT estaEnLista(p, paux^.codigo) THEN {si no est  en lista destino}
      insertarOrdenado(p,paux^.codigo); {insertamos}
      paux:=paux^.psig; {y pasamos al siguiente}
    END; {WHILE}
  END; {tratar}

BEGIN {generar Ordenada}
  p:=NIL;
  tratar(p1,p); {tratamos la lista 1}
  tratar(p2,p); {tratamos la lista 2}
  END; {generar Ordenada}

```

Ejercicio 4:

[30 puntos]

Se desea realizar un programa que gestione informaci3n sobre una base de datos de m3sica. La informaci3n almacenada ser3 el t3tulo de la canci3n, el int3rprete y una calificaci3n que se le otorga a cada canci3n.

Dada la siguiente definici3n de tipos de datos y constantes:

```

CONST
    MAXINTERPRETE = 50;
    MAXTITCANCION = 100;

TYPE
    TTitCancion = string[MAXTITCANCION];
    TInterprete = string[MAXINTERPRETE];
    TCancion = RECORD
        interprete: TInterprete;
        titCancion: TTitCancion;
        calificacion: real;
    END;
    TBDCanciones = FILE OF TCancion;

```

Se pide realizar un subprograma que permita crear un fichero binario de canciones a partir de los datos almacenados en un fichero de texto. El subprograma recibirá como parámetros el nombre físico del fichero de texto con la información a insertar, y el nombre físico del fichero binario a crear. La información de cada una de las líneas del fichero de texto es la siguiente:

- Título de la canción
- Calificación de la canción
- Nombre del intérprete

Un ejemplo de líneas en el fichero de texto de entrada es:

```

...
Aire 4.62 Mecano
Nieve_Negra 5.1 Tahures_Zurdos
...

```

Cada línea está formada por el título de la canción, la calificación y el intérprete.

Se pide:

- (a) Realizar un subprograma que se encargue de leer los datos de una línea del fichero de texto y guarde el resultado en un registro de tipo TCancion. ¡OJO! La lectura de los campos se deberá hacer de la siguiente manera:
 - Lectura del título de la canción leyendo carácter a carácter.
 - Lectura de la calificación.
 - Lectura del intérprete de la canción leyendo carácter a carácter.
- (b) Realizar un subprograma que genere el fichero binario de canciones a partir de los datos del fichero de texto (utilizando el subprograma del apartado a). Este subprograma deberá ir leyendo línea a línea los datos del fichero de texto de entrada e ir insertando en el fichero binario, que contendrá la información de nuestra base de datos de música. Si existiera el fichero binario, el subprograma eliminaría los datos anteriores.

a)

```
FUNCTION leeCadenaCaracteres (VAR ftexto: text): string;
VAR
    cad: string;
    c: char;
BEGIN
    cad := '';
    read(ftexto, c);
    WHILE ((c <> ' ') AND NOT EOLN(ftexto)) DO
    BEGIN
        cad:= cad + c;
        read(ftexto, c);
    END;

    IF (EOLN(ftexto)) THEN
        cad := cad + c;
    leeCadenaCaracteres := cad;
END;

PROCEDURE leeDatosCancion (VAR ftexto: text; VAR c: tCancion);
VAR
    esp : char;
BEGIN
    c.titCancion := leeCadenaCaracteres(ftexto);
    read(ftexto, c.calificacion);
    read(ftexto, esp);
    c.interprete := leeCadenaCaracteres(ftexto);
END;
```

b)

```
PROCEDURE generaBDCanciones (nombrefich: string; nombrebd: string);
VAR
    cancion: tCancion;
    ftexto: text;
    fbd: tBDCanciones;
BEGIN
    ASSIGN(ftexto, nombrefich);
    ASSIGN(fbd, nombrebd);
    RESET(ftexto);
    REWRITE(fbd);
    WHILE NOT EOF(ftexto) DO
    BEGIN
        leeDatosCancion(ftexto, cancion);
        write(fbd, cancion);
    END;
    CLOSE(ftexto);
    CLOSE(fbd);
END;
```