NOMBRE.....

NÚM. de MATRÍCULA..... GRUPO.....

Examen Final. Informática. Febrero 2004

Instrucciones

- El examen consta de un conjunto de **diez** preguntas, cada una de las cuales puntuará **cero** o **un** punto.
- Se calificará con **un punto** si la respuesta correcta se indica en la forma **más simple**.
- La duración total del examen será de **dos horas y media**
- Se debe prestar especial atención a los identificadores usados

---

1. Dada la secuencia de bits **1010 1101**, indicar cuál es el valor **decimal** representado

Valor entero con signo en complemento a dos	
Valor entero con signo en signo - magnitud	
Valor real en punto fijo (punto entre cuarto y quinto bit) y complemento a dos	
Valor real en coma flotante (4 primeros bits para mantisa y cuatro últimos para exponente, ambos en complemento a dos)	

2. Escribir la función `calcular` que **devuelva** la suma de A y B, a partir de los parámetros X e Y, de la siguiente forma:

$$X < Y \begin{cases} A = \dfrac{X}{|X| + |\text{sen}(Y)|} \\ B = \ln \dfrac{Y - X}{Y^2 - X^2} \end{cases} \qquad X \geq Y \begin{cases} \text{A es el resultado de llamar a la función } \texttt{uno}. \\ \text{B se obtiene como resultado en el primer parámetro de llamar al procedimiento } \texttt{dos}. \end{cases}$$

Se recuerda que en TurboPascal existen las funciones `abs(x)`, `sin(x)` y `ln(x)`.

```
{Se omiten los desarrollos de las rutinas uno y dos}
function uno(x,y: Real): Real;
procedure dos(var x: Real; y: Real);
```

```
function calcular(
```

3. Completar la función `decimal` que devuelva como resultado el valor **decimal** correspondiente a cualquier dígito **hexadecimal** (en mayúsculas) almacenado en el parámetro `c` de tipo `char`. Si el parámetro no corresponde a un valor hexadecimal la función debe devolver `99`.

```
function decimal(c:char):byte;
begin
  case c of
```

:
:

```
  end;
end;
```

4. Completar el **procedimiento** `RGB` que, a partir del parámetro formal `s` de tipo `cadena`, obtenga el **resultado** en un parámetro formal de tipo `vector`.
   - Los dos primeros caracteres de la cadena `s` representan un valor hexadecimal de dos dígitos que indican la concentración del color rojo, los dos caracteres siguientes la concentración de verde y los dos últimos la concentración de azul (código RGB *RedGreenBlue*).
   - Los valores asignados a las componentes del parámetro de tipo `vector` son las concentraciones de estos mismos colores expresados en decimal.
   - Se debe utilizar la función `decimal` construida en la pregunta anterior.
     Ejemplo: si `s` es '0F031A' entonces el vector generado es (15,3,26)

```
type vector=array[1..3] of byte;
     cadena=string[6];
```

```
procedure RGB(
```

5. Construir una función **recursiva** `escalar` que multiplique **escalarmente** dos vectores de $\boldsymbol{R^n}$ almacenados, cada uno de ellos, en una lista simple enlazada distinta, señalizada por los punteros `p` y `q` respectivamente. Si alguna de las listas está vacía el resultado debe ser cero.

```
type ptr=^componente;
     componente=record valor:real;sig:ptr end;
function escalar(p,q:ptr):real;
```

6. Una matriz es de *Toeplitz* cuando los elementos que pertenecen a una misma diagonal son iguales entre sí. Las diagonales que se consideran son la principal y las paralelas a ésta. Se pide completar el procedimiento que **almacene** en un archivo de **texto** de nombre '`datos.dta`' la información

mínima necesaria que permita (leyendo el archivo en otro programa) volver a reconstruir la matriz de dimensión N x N, con N ≤ 20. En cada línea del archivo se escribe la fila, la columna y el valor de un elemento.

```
Type Rango = 1..20;
     Matriz = array[Rango, Rango] of real;
  Procedure toeplitz(m: Matriz; N: Rango);
  var t:text; i:Rango;
  begin
```

Un ejemplo de este tipo de matrices es:

$$\begin{pmatrix} 2 & 5 & 8 & 6 \\ 3 & 2 & 5 & 8 \\ 9 & 3 & 2 & 5 \\ 4 & 9 & 3 & 2 \end{pmatrix}$$

```
  end;
```

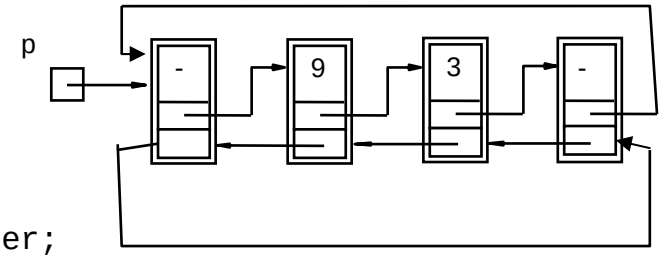
7. Dada una lista simple enlazada con un número **par** de elementos, a cuyo primer elemento apunta el parámetro puntero `prim`, construir un **procedimiento** que inserte un **nuevo** elemento en la mitad de la lista. Se debe considerar el caso en que la lista esté vacía.

```
TYPE ptr=^elemento; elemento=record dato:integer;sig:ptr end;

procedure insertar
```

8. Hacer la **declaración de tipos** necesaria para construir una lista de listas de enteros. Es decir, una lista **simple enlazada** en la que cada elemento de la lista contiene un puntero que señala el primer elemento de **otra** lista simple enlazada de números enteros.

9. Se dispone una lista **doble circular** (ejemplo de la figura) a cuyo primer elemento señala `p` y un archivo de números enteros. Construir una función que se mueva por la lista siendo los **incrementos** de posición los números que se guardan en el archivo. Si el número es cero no se mueve, si es negativo se mueve hacia atrás y si es positivo hacia delante, en ambos casos el valor absoluto indica el número de veces que se avanza. La función devuelve como **resultado** la suma de los datos de los elementos de destino. Se supone que el archivo existe y está abierto. Si el archivo o la lista está vacía el resultado es cero.




```
type ptr = ^elemento;
       elemento= record
       dato: real;
       sig,ant: ptr;
       end;
   archivoEnteros: file of Integer;

function suma_mov (p: ptr; var f: ArchivoEnteros): real;
```

10. En un pozo hay una escalera. Se representa el nivel de agua mediante un vector de componentes, `true` o `false`, que indica si hay agua o no para un índice que es el número de escalón. Los escalones se numeran de arriba abajo. Escribir un procedimiento **recursivo** que busque agua en el pozo. En cada escalón se debe mostrar por pantalla si hay agua o no y si se sube o se baja.
En la figura se muestra una salida de resultados con los parámetros indicados.
Se puede suponer que el pozo siempre tiene agua.
Si `v = (false, false, true, true, true).` La llamada: `buscar(1,v);`
Obtiene como salida por pantalla:

```
No hay agua en 1, bajo a 2
No hay agua en 2, bajo a 3
Hay agua en 3
Subo desde 3 hasta 2
Subo desde 2 hasta 1
```

```
Const N = 5;
Type Vector = Array [1..N] of Boolean;
Procedure buscar (esc: Byte; v: Vector);
Begin

End;
```