NOMBRE.....

NÚM. de MATRÍCULA..... GRUPO.....

Examen Final. Informática. Febrero 2005

Instrucciones

- El examen consta 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**

---

1. Se quiere diseñar un formato de representación **binaria** en coma o **punto fijo** que, utilizando el **MÍNIMO** número de bits, cumpla las tres condiciones siguientes:
   a) debe representar números reales con un valor absoluto de, al menos, hasta $\mathbf{10^3}$ (incluido).
   b) el valor representado por el dígito binario **menos** significativo es **0.125**.
   c) debe emplear la representación en **signo-magnitud**.

   Indicar el **número MÍNIMO total** de dígitos **binarios** necesarios: [ ]

2. Completar la función `esAnterior` para que devuelva `true` o `false` si la fecha almacenada en el parámetro `a` es **anterior** o no a la almacenada en `b`. Nota: Si las fechas coinciden la función ha de devolver el valor falso.

```
type fecha=record
     dia,mes: byte; anho:word;
     end;
function esAnterior(a,b:fecha):boolean;
  begin
  esAnterior:= [                    ]
                [                    ]
                [                    ]
                [                    ]
  end;
```

3. Se define como número de **cifras crecientes** a todo número natural N tal que $N=d_m d_{m-1}... d_1 d_0$ tal que $d_{i+1} \leq d_i$ para $0 \leq i < m$. Por ejemplo: 359, 1227, 5, 139, 88 y 0 son todos números de cifras crecientes, mientras que 1329, 21 y 187 no lo son. Completar la función **recursiva** `cc` para que devuelva **verdadero** si el valor numérico del parámetro `n` es de cifras crecientes o **falso** en caso contrario.

```
function cc(n:word):boolean;
  begin
  [                    ]
  [                    ]
  [                    ]
  [                    ]
  end;
```

4. Completar la función `cadiguales` para que compare dos cadenas de caracteres: debe devolver **verdadero** o **falso** si las secuencias de caracteres almacenadas respectivamente en los parámetros `s1` y `s2` son **iguales** sin considerar mayúsculas/minúsculas. Por ejemplo, si `s1` vale `'Casa'` y `s2` vale `'Casa'`, `'casa'` o `'CASA'` la función debe devolver `true`, mientras que si `s1` vale `'Casa'` y `s2` vale `'Cas'`, `'Casas'`, `'Pasa'` o `'Caso'` la función debe devolver `false`. Nota: la función estándar `upcase(c:char):char;` devuelve el carácter en mayúsculas correspondiente al parámetro `c`.

```
function cadiguales(s1, s2:string):boolean;
```

5. Dadas dos variables `v1` y `v2` de tipo `vector`, construir una rutina llamada `desencriptar` para que, tras la llamada a `desencriptar(v1,v2);`, se almacene en `v2` el valor **desencriptado** correspondiente al número de `N` dígitos asignado a `v1` (valor encriptado) antes de la llamada. **Cada dígito** debe almacenarse en un elemento del *array* correspondiente. El proceso de **encriptación** del número **original** ha sido el siguiente: (a) se reemplaza cada dígito por el **resto** de la división entera entre 10 de la suma de dicho dígito más 6 y (b) se intercambia el **primer** dígito, componente de índice 1, por el **penúltimo**. Nota: El valor desencriptado resultante debe coincidir con el número original.

   Esquema del proceso total del que se pide la **desencriptación** y ejemplo de valores en cada caso:

Encriptación		**Desencriptación**
Valor original ---------- >	Valor encriptado (**v1**)	---------- > Valor desencriptado (**v2**)
71919514	77575130	71919514

```
const N=8;
type indice=1..N; digito=0..9; vector=array[indice] of digito;
```

6. En el archivo de nombre `s` se almacena una secuencia de ternas (grupos de 3 elementos) de valores de tipo `word`. Completar la función `ntp` para que devuelva **verdadero** si en el archivo existe, al menos, un *Triple Pitagórico* (la suma de los cuadrados de los dos primeros valores de una terna es igual al cuadrado del tercero) o **falso** en caso contrario. Nota: considerar el caso de archivo vacío, en cuyo caso, la función debe devolver el valor falso.

```
function ntp(s:string):boolean;
```

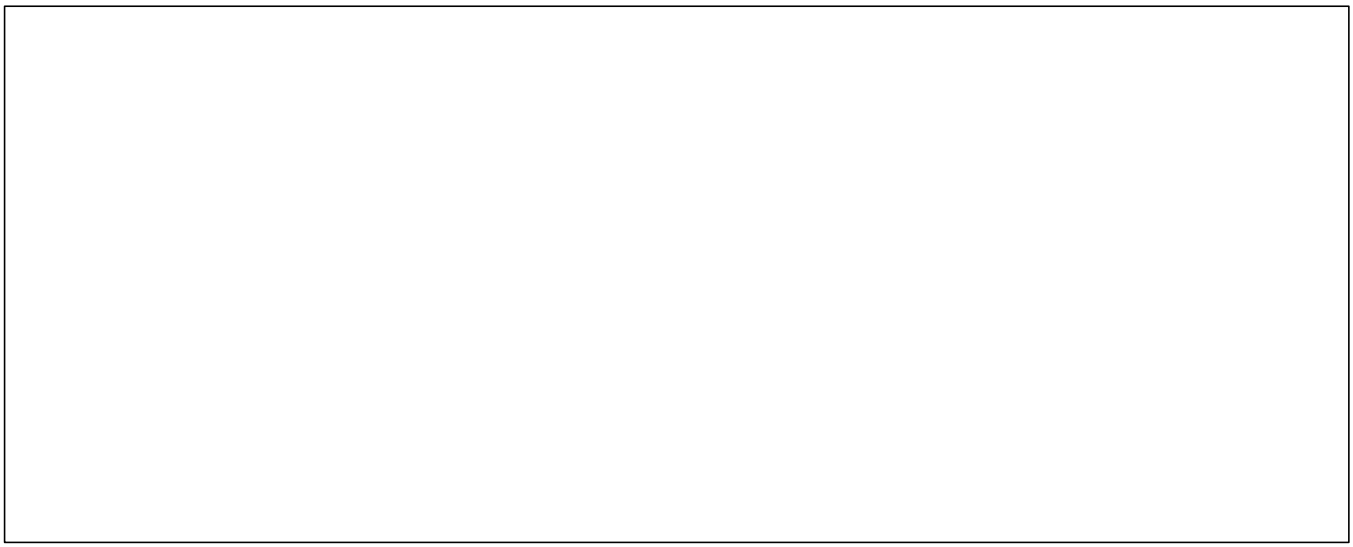
7. Completar el procedimiento `media` para que escriba en el archivo con tipo, de nombre `'media.dat'`, la **media aritmética** de los números reales que se encuentran **en cada línea** de un archivo de texto de nombre dado por el parámetro `s`. Nota: Los valores reales de cada línea se encuentran separados por caracteres de espacios en blanco. En cada línea hay un número arbitrario de valores reales. Si una línea está vacía entonces no se debe escribir la correspondiente media aritmética. Si el archivo de texto está vacío el archivo generado debe estar completamente vacío.

```
type reales = file of real;
procedure media(s: string);
  var f: text; g: reales; n: word; x,suma: real;
  begin
  assign(f,s); reset(f); assign(g,'media.dat'); rewrite(g);
```

```
   close(f); close(g)
   end;
```

8. Escribir una rutina que construya una matriz **simétrica** a partir de los elementos por encima de la **diagonal principal** de una matriz dada (a). Se debe dar como resultado la **dirección** de memoria de una **nueva** variable dinámica con la matriz simétrica. Nota: se define una matriz simétrica como aquella que cumple para todo i, j: m[i,j]=m[j,i].

```
const n=100;
type matriz=array[1..n,1..n] of real;
     pmatriz=^matriz;
procedure simetrica(a:pmatriz; var b:pmatriz);
```



9. Completar la unidad `poly` que declare en la zona pública e implemente un procedimiento `insertar` que **inserta** un monomio, dado por su coeficiente y grado, en un polinomio, es decir, una lista de monomios ordenada de forma creciente por el grado. Se *recomienda* dar la solución **recursiva**. Se debe considerar el caso de lista vacía. Se supone que en el polinomio no existe un monomio del mismo grado que el monomio a insertar.

```
type polinomio = ^monomio;
     monomio = record c: real; g: word: sig: polinomio end;
procedure insertar (var p: polinomio; c: real; g: word);
```

10. Completar el procedimiento `suma_pol` para que **modifique** un polinomio (dado por `prim`) de forma que, al ejecutarse, se almacene la **suma** de dicho **polinomio** y un **monomio** (dado por `m`) en el polinomio `prim`. Se recomienda usar la unidad declarada en el ejercicio anterior.

```
program suma;
```

```
procedure suma_pol(var prim: polinomio; m: monomio);
```

```
begin   { del programa principal }
...
end.    { del programa principal }
```