NOMBRE.....

NÚM. de MATRÍCULA..... GRUPO.....

Examen Final. Informática. Junio 2007

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. **Diseña** los campos de bits **mínimos** necesarios para codificar en binario la fecha de nacimiento (día del mes, mes y año) de una persona nacida entre el año 1 y el año 10.000. Para ello debe indicarse con claridad la **codificación** y el significado de cada bit ó bits; y si un dato se codifica en varios bits el **número** de ellos. Utilícese el diseño para codificar la fecha de nacimiento del propio alumno.

Diseño	Fecha de Nacimiento	Codificación en Binario

2. Completa el programa para que **asigne valores** a las componentes de la matriz `mat` de M filas por N columnas. La matriz se rellena utilizando los números naturales por columnas. En la figura se muestran los valores que toma la matriz cuando es de 3x5.

```
Program RellenaMatriz;
Const M = {Numero arbitrario};
      N = {otro numero arbitrario};
TYPE Matriz = Array[1..M, 1..N] of Integer;
VAR mat: Matriz; i, j: Byte; aux: Integer;
BEGIN
```

Valores `mat` de 3x5

1	4	7	10	13
2	5	8	11	14
3	6	9	12	15

```
END.
```

3. Completa la función `numero_multiplos` para que devuelva el **número de elementos** de **una** matriz dada que cumplen que son **múltiplos** de **un** número entero dado. Se puede suponer que todos los elementos de la matriz son distintos de cero. Por ejemplo, se devuelve 3 para la matriz del ejemplo del problema anterior y el número 4.

```
Const M = 216; N = 250; TYPE Matriz = Array[1..M, 1..N] of Integer;
Function numero_multiplos
```

4. Completa el programa para que **guarde** en un archivo de **texto** la fila, la columna y el valor de las componentes de una matriz de MxN que cumplan que los valores son múltiplos de la última cifra del número de matrícula del alumno +2. Es requisito ineludible que **se pueda** recuperar la información grabada.

```
Program GrabaMatriz;
Const M = 212; N = 243; TYPE Matriz = Array[1..M, 1..N] of Integer;
VAR mat: Matriz; i, j: Byte; t: Text;
BEGIN
{ Se da valores a la matriz mat }
Assign(t,'Datos.txt');
```

```

```

```
END.
```

5. Completa la declaración y el procedimiento para que **grabe** la fila, la columna (enteros positivos) y el valor de las componentes de una matriz (parámetro `mat`) en un archivo **de tipo** record `Elemento`. Se guardan sólo las **filas** de índice **par**. El parámetro `ruta` es el nombre del archivo que se crea.

```
Const M = 158; N = 232; TYPE Matriz = Array[1..M, 1..N] of Integer;
Elemento = record
```

```

```

```
Archivo = File of Elemento;
Procedure grabar(mat: Matriz; ruta: String);
```

```

```

6. Completa la función `media_coc` para que devuelva la **media** del cociente de la **división entera** de los elementos de un vector `a` entre los elementos de otro vector `b`. Cada elemento de `a` se divide por el de `b` con el mismo índice. La función debe ignorar a todos los efectos los pares de elementos que pudiesen producir errores (no cuentan en la media). Se puede suponer que al menos 1 elemento de `b` es distinto de cero.

```
Const N = 200; Type Vector= Array[1..N] of Word;
Function media_coc (a,b:Vector): Real;
```

7. Completa la función **recursiva,** `coincide,` para que devuelva `true` si al menos una cifra en la representación en base *n+3* del número `num` (dado como parámetro) es **igual** al valor *n+1* y `false` en caso contrario. *n* es la última cifra de tu número de matrícula. Nota: se puede plantear el problema en base 10 y luego sustituir 10 por *n*+3.

```
Function coincide(num: word): Boolean;
Begin
```

```
End;
```

8. Declara los **tipos de datos** y la **variable** necesaria para guardar una estructura de datos que consista en una serie de listas de palabras indexadas u ordenadas por las letras de la ‘a’ a la ‘z’ sin incluir la ñ. Las palabras almacenadas tienen longitud máxima de 50. Las listas se indexan mediante letras porque todas las palabras de una lista empiezan por esa letra, pero esto no afecta a la declaración de los tipos.

```
TYPE
```

```
VAR
```

9. Dada una unidad cuya zona de `interface` se muestra a continuación y que sirve para controlar un haz láser que se mueve linealmente sobre una chapa, se pide **completar el programa** para que el haz se **desplace** hasta la posición dada por la decena del número de matrícula del alumno y haga **taladros** cuyo diámetro sea igual a la última cifra del número de matrícula hasta que se llegue al final de la chapa. Después de cada taladro hay que **incrementar** la posición en 1 unidad. La situación inicial del láser se desconoce, pero se puede obtener llamando a la función `position`. Además la función `position` devuelve el valor **999** para indicar que el haz láser se ha salido de la chapa.

```
Unit Laser;
Interface
Function position:integer;{Posición en unidades de longitud, el cero está en mitad
  de la chapa, la posición puede ser negativa, si devuelve 999 está fuera de la chapa}
Procedure move(inc: Boolean); {Cuando inc es true incrementa la posición 1 unidad
  de longitud y la reduce en 1 cuando es false}
Procedure drill(diam: Byte); {Realiza un taladro del diámetro dado }
Implementation
{ Implementación de las rutinas anteriores }
End.

Program OperacionTaladros; Uses Laser;
Begin
```

10. Completar el código para que el procedimiento `eliminar_igual_n` **elimine** y **libere** la memoria del primer elemento de una lista de enteros cuyo campo `num` coincida con un número entero `n` dado. Se puede suponer que la lista **no** está vacía. Si ningún elemento cumple la condición no hay que hacer nada. Es obligatorio usar el procedimiento `eliminar_cabeza` (la solución queda más sencilla). La solución recursiva es más corta, pero no es obligatoria.

```
TYPE Lista=^Elemento; Elemento = record num: Integer; sig: Lista; end;
Procedure eliminar_cabeza(var p: Lista); {Elimina el primer elemento}
Var aux: Lista;
Begin aux := p; p := p^.sig; dispose(aux); End;
Procedure eliminar_igual_n(n: Integer;
```