Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza

3 de julio de 2018

Examen de **Fundamentos de Informática**

Grado en Tecnologías Industriales

Duración total del examen: **3 horas**

NOTA. Responde a cada ejercicio en una hoja diferente. No utilices la misma hoja para dos (o tres) ejercicios.

Ejercicio 1 [3 puntos]

Un *número de Fermat* es un número entero y positivo que se puede calcular con la siguiente fórmula:

$$F_n = 2^{2^n} + 1$$

donde n es un entero no negativo. Así, dando valores a n, se pueden ir obteniendo números de Fermat como sigue:

$$F_0 = 2^{2^0} + 1 = 3$$
$$F_1 = 2^{2^1} + 1 = 5$$
$$F_2 = 2^{2^2} + 1 = 17$$
$$F_3 = 2^{2^3} + 1 = 257$$
$$F_4 = 2^{2^4} + 1 = 65537$$

Fermat conjeturó que todos los números calculados mediante esta fórmula eran primos. Sin embargo, esto no es cierto.

Se pide desarrollar un programa en Pascal que muestre por pantalla el primer número de Fermat que no es primo (y su valor n correspondiente). El programa no solicitará nada al usuario. Se recomienda implementar funciones adicionales para la resolución.

Nota: Puedes asumir que todos los enteros son representables con el tipo `integer`.

Ejercicio 2 [3.5 puntos]

Una organización guarda en un fichero de registros `contabilidad.dat` todos los asientos contables de su **contabilidad** anual. Cada asiento contable (registro) incluye el código de cuenta del *debe*, el código de cuenta del *haber*, la cantidad del asiento (un número real en euros) y el concepto (una cadena de texto). Los códigos de cuenta son números naturales de **exactamente tres cifras**.

Algunas cuentas corresponden a conceptos que la organización no desea que sean de dominio público. Los códigos y conceptos de dichas cuentas están en un fichero de registros `cuentasB.dat`, con el código (natural de 3 cifras) y el concepto (una cadena de texto). El código es **único** por cada cuenta. **Se pide** desarrollar un programa en Pascal que separe los asientos del `contabilidad.dat` en dos ficheros de registros:

- `contabilidadB.dat` que contenga todos los asientos contables que involucren alguna cuenta del fichero `cuentasB.dat` (debe o haber)
- `contabilidadA.dat` donde permanezcan el resto de asientos, de dominio público.

Ejemplo de ejecución:

contabilidad.dat	**cuentasB.dat**
126 501 11689.15 Pago deuda proveedores 126 721 60000.00 Paga extra en sobre 425 214 121050.00 Venta inmueble 126 845 305000.00 Contrato obra sede 845 215 105000.00 Donacion	721 Sobresueldos 215 Empresa asociada Panama

contabilidadA.dat	**contabilidadB.dat**
126 501 11689.15 Pago deuda proveedores 425 214 121050.00 Venta inmueble 126 845 305000.00 Contrato obra sede	126 721 60000.00 Paga extra en sobre 845 215 105000.00 Donacion

Ejercicio 3 [3.5 puntos]

Un **Diccionario Bilingüe** sirve para buscar la traducción de una palabra de un idioma a otro. Esa traducción puede no ser única, y la palabra original de un idioma se puede traducir con distintas acepciones dependiendo del contexto.

Partiendo de que representamos las palabras mediante el tipo de datos `String` de Pascal, define los siguientes tipos y estructuras de datos para representar el diccionario:

- `tpAcepcion`, que permita representar la traducción de una palabra y su significado para una acepción dada (el significado puede almacenarse en un `String`).
- `tpEntrada`, que represente una entrada del diccionario que contenga la palabra original en el idioma de partida, y un conjunto de acepciones en el idioma de destino, hasta un máximo de 10.
- `tpDiccionario`, que contenga un grupo de entradas hasta un máximo de 10000.

Usando esos tipos de datos, implementa los siguientes procedimientos o funciones:

- Una función que, tomando como entrada una palabra y un diccionario, devuelva el índice de la palabra dentro del diccionario o el valor **0** en caso de que no la encuentre:

```
function indiceEnDiccionario(??? palabra:     String;
                             ??? diccionario: tpDiccionario): integer
```

- Un procedimiento que, tomando como entrada una palabra y un diccionario, muestre por pantalla todas las traducciones y acepciones de dicha palabra, o un mensaje informando de que no existen:

```
procedure traduce(??? palabra:     String;
                  ??? diccionario: tpDiccionario);
```

Para comparar palabras puedes usar la siguiente función de Pascal (no tienes que implementarla):

```
function CompareStr(S1: String; S2: String): integer;
```

que devuelve 0 si las dos cadenas son iguales (y otro valor distinto de 0 si no lo son).

Nota: Se valorará que NO haya código repetido, por lo que puedes definir los procedimientos y funciones auxiliares que creas necesario.