Departamento de Informática e Ingeniería de Sistemas
Universidad Zaragoza

Examen de Teoría

13 de enero de 2025

Fundamentos de Informática
Grado en Tecnologías Industriales
Duración: 3 horas

Entrega:

- Escribe tu nombre completo en **todas** las hojas que entregues como solución del examen.
- Responde a cada ejercicio comenzando **en una hoja diferente**, los ejercicios se entregan por separado.
- La resolución del examen no se puede escribir a lápiz ni con bolígrafo rojo. Lo que esté escrito a lápiz o en rojo será ignorado.

Ejercicio 1 [3.5 puntos]

Para muchos algoritmos de criptografía y seguridad es necesario calcular todos los números primos hasta un valor máximo `MAXNUM` (digamos, por ejemplo, 1.000.000), y además obtener cuál es el número primo más grande de todos ellos.

El método tradicional (o de *fuerza bruta*) de comprobar cada uno de los números con sus divisores es muy costoso e ineficiente, ya que se repiten los mismos cáculos para todos los números posibles.

El matemático griego Eratóstenes de Cirene (Cirene 276 aC – Alejandría 194 aC) diseñó un algoritmo más eficiente, conocido como la **Criba de Eratóstenes**:

1. Se comienza con una lista de todos los números entre 2 y `MAXNUM`.
2. Inicialmente todos los números se consideran candidatos a ser primos y se *marcan* como tales.
3. Se revisan todos los números de uno en uno y:
   (a) Si el número es candidato a ser primo (no lo hemos desmarcado):
      i. Se considera definitivamente como primo, y se queda marcado.
      ii. Se *desmarcan* como candidatos todos sus múltiplos.
   (b) En caso contrario, pasamos al siguiente.

Se pide:

1. Define el tipo para una estructura de datos `tpCriba`, que permita representar, para los números entre 2 y `MAXNUM`, si son primos o no:
2. Implementa un procedimiento `inicializa` que *marque* en la criba `cr` todos los números como posibles candidatos a ser primos.

   ```
   procedure inicializa(???? cr: tpCriba);
   ```

3. Implementa un procedimiento `quita_multiplos` que *desmarque* en la criba `cr` todos los múltiplos de `n` (pero no el propio `n`) como posibles candidatos a ser primos.

   ```
   procedure quita_multiplos(???? cr: tpCriba; n: ????);
   ```

4. Implementa una función `max_primo` que busque en la criba `cr` el mayor número que está marcado como primo, y lo devuelva. Ten en cuenta que encontrar el primo máximo es mucho más fácil si empiezas a buscar por el final.

   ```
   function max_primo(???? cr: tpCriba): ????;
   ```

5. Usando todos los subprogramas anteriores, implementa un programa completo (el programa principal) que nuestre por pantalla el mayor primo menor que 1.000.000.

Ejercicio 2 [3.5 puntos]

Una empresa de venta por internet desea conocer cuál es el grupo de edad al que más vende, para poder hacer promociones dirigidas a dicho grupo objetivo. Para ello, va a analizar las ventas de productos durante un mes.

Para cada venta realizada, se almacena un registro en un fichero secuencial de tipo `tpFichVentas`. Cada uno de dichos registros almacenados, de tipo `tpVenta`, contiene la siguiente información: nombre del cliente, dirección, edad, número de productos vendidos en esa venta, y un número de identificación de la venta. En Pascal, los registros y el fichero pueden representarse mediante los siguientes tipos de datos:

```
type
  tpVenta = record
    nombre     : string;
    direccion  : string;
    edad       : integer;
    nProductos : integer;
    idVenta    : integer;
  end;
  tpFichVentas = file of tpVenta;
  ...
```

Se establecen grupos de edad de 5 años, y los clientes pueden tener entre 18 y 97 años de edad (ambos incluidos); así, el primer grupo es [18, 22] años, el segundo es [23, 27] años, etc. Hay por tanto un total de 16 grupos.

Se pide:

1. Implementa un subprograma, de nombre `grupo`, que, teniendo como parámetro la edad del cliente, calcule y devuelva en qué grupo de edad está el cliente. El grupo de edad será un entero entre 1 (si la edad está en el rango [18, 22]) y 16 (si la edad está en el rango [93, 97]).
2. Define una estructura de datos `tpVentasGrupos` que almacene, para cada grupo de edad, cuántos productos (nótese que no es cuántas ventas sino cuántos productos) se han vendido a clientes de ese grupo de edad. Simplemente declara el tipo de datos.
3. Implementa un procedimiento `almacenarVentasGrupos` que, a partir de la información contenida en un fichero secuencial de registros de tipo `tpFichVentas`, almacene en un dato de tipo `tpVentasGrupos` cuántos productos se han vendido a cada grupo de edad.

   ```
   procedure almacenarGruposVentas(var f: tpFichVentas; var vg: tpVentasGrupos);
   ```

4. Implementa una función `grupoObjetivo` que, a partir de la información contenida en un dato de tipo `tpVentasGrupos`, calcule y devuelva a qué grupo de edad se han vendido más productos (y que será el grupo objetivo). La función devolverá el número del grupo correspondiente.

   ```
   function grupoObjetivo (const vg: tpVentasGrupos): integer;
   ```

5. Implementa un programa principal que a partir de un fichero secuencial de tipo `tpFichVentas`, conteniendo la información de las ventas del mes de diciembre de 2024 (de nombre '202412.dat'), calcule el grupo de edad objetivo, y luego escriba en otro fichero secuencial de tipo `tpFichVentas` ('202412_grupoObjetivo.dat') únicamente los registros que corresponden a ventas a clientes del grupo de edad objetivo. Nótese que el programa no mostrará nada por pantalla (ni requiere leer nada de teclado).

Nota: Se valorará el uso adecuado de procedimientos y funciones.

Ejercicio 3 [3 puntos]

Una imagen digital no es más que una **matriz bidimensional de elementos llamados píxeles**, cuyos valores numéricos indican el color de la correspondiente región de la foto. En el caso de una imagen en blanco y negro, estos valores numéricos van desde 0 (que indica negro) hasta 255 (que indica blanco). Los valores intermedios representan, distintas gradaciones de gris (ver Imagen 1)

Las técnicas de tratamiento digital se basan en alterar los datos de la matriz que representa la imagen para obtener distintos resultados. Por ejemplo, la *reducción de ruido* en una imagen consiste en aplicar un filtro que consiga reducir o eliminar la *nieva digital* que se produce por diferentes motivos al tomar la imagen. Uno de los filtros se llama *filtro de mediana*, que en su forma más simple consiste en **sustituir el valor original de cada píxel de la imagen por el que se obtiene** de la siguiente forma:

(1) Se toman los valores de ese píxel y los de sus vecinos a izquierda y derecha, (2) se ordenan por valor decreciente de sus valores, y (3) se selecciona el valor central, que será el que sustituya al original en la imagen mejorada. En el caso de los píxeles de la primera columna, y en los de la última columna, se toma duplicado el valor de ésta (véase el ejemplo de la Figura 2).

Sabiendo que **una imagen tiene un tamaño máximo** a 640(columnas)x480(filas), se pide:

1. Define la estructura de datos `tpImagen` y `tpPixel`, que permita guardar una imagen en memoria para su tratamiento posterior. Pon atención a cómo representar imágenes de menos de 640x480 pixels (no será válido rellenar el espacio sobrante con píxeles negros).
2. Implementa una función `mediana3` que, dados tres píxeles, devuelva el valor de la mediana de los tres.

   ```
   function mediana3(??? a,b,c:tpPixel): tpPixel;
   ```

3. Implementa un procedimiento `quitarRuido` que, dada una imagen original, aplique el filtro de mediana horizontalmente (cada pixel se suaviza con el de su derecha y el de su izquierda) y devuelva la imagen mejorada.

   ```
   procedure quitarRuido(??? original: tpImagen; ??? mejorada: tpImagen);
   ```

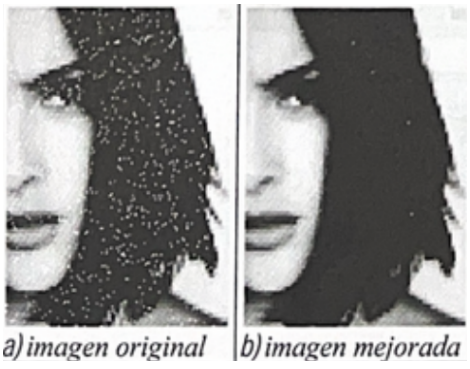

Figure 1: Ejemplo de mejora.

Imagen original:

```
[   1    4  211   13 ]
[  93  255    5   56 ]
[  67  128    0  202 ]
```

Imagen mejorada:

```
[   1    4   13   13 ]
[  93   93   56   56 ]
[  67   67  128  202 ]
```

La segunda fila se ha obtenido así:

```
mejorada[2,1] = mediana(  93,  93, 255) = 93
mejorada[2,2] = mediana(  93, 255,   5) = 93
mejorada[2,3] = mediana( 255,   5,  56) = 56
mejorada[2,4] = mediana(   5,  56,  56) = 56
```

Figure 2: Cálculo de la imagen mejorada usando el filtro de mediana.