Examen de **Fundamentos de Informática**
Grado de Tecnologías Industriales
Escuela de Ingeniería y Arquitectura
Primera convocatoria: 23 de Junio de 2017

Departamento de Informática e Ingeniería de Sistemas
Universidad Zaragoza

Duración total del examen: ***3 horas***

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

Ejercicio 1 [3 puntos]

En un futuro no distópico todos los ciudadanos llevamos un chip subcutáneo que permite tenernos localizados en todo momento mediante tecnología GPS, aunque a nadie parece importarle porque así pueden saber cuántos pasos dan al día o localizar los restaurantes más cercanos. El chip subcutáneo guarda diariamente para cada ciudadano toda la información del recorrido que ha realizado ese día en un fichero secuencial de registros almacenado en la nube con la siguiente estructura Pascal:

```
tpPaso = record
   x, y: Real; {Coordenadas x, y en metros sobre la superficie de la ciudad.}
   t: Real;    {Instante temporal en segundos desde las 00:00. }
end;
tpRecorrido = file of tpPaso;
```

El ayuntamiento de Zaragoza necesita procesar los ficheros de todos sus ciudadanos para extraer cierta información, y te han encargado a ti que hagas el programa correspondiente. **Se pide** desarrollar un programa en Pascal que le pida un nombre del fichero generado por el chip al usuario y muestre por pantalla la siguiente información:

1. La **distancia total** recorrida en metros (con un decimal). Se considerará que entre dos pasos del fichero el ciudadano ha ido en línea recta (distancia euclídea).
2. La **velocidad media** de todo el día (en metros por segundo y con un decimal) o "error" si eso no es posible. Nota: Se puede calcular globalmente y no paso a paso.
3. Si el ciudadano es (o no) **subversivo**. Un ciudadano es subversivo si ha conseguido esconder información (alterando el chip o interceptando su señal) de tal forma que no ha guardado un paso en diez minutos desde el paso anterior o en diez minutos desde el inicio del día a las 00:00.

Se puede asumir que el fichero tiene al menos un elemento y que está ordenado por el instante temporal.

Ejemplo de ejecución:

EJEMPLO 1	**Fichero**	x=35,y=30,t=20 | x=35,y=80,t=45
	Pantalla	Distancia total: 50.0 m Velocidad media: 2.0 m/s No es subversivo
EJEMPLO 2	**Fichero**	x=98,y=132,t=1000
	Pantalla	Distancia total: 0.0 m Velocidad media: error Es subversivo
EJEMPLO 3	**Fichero**	x=35,y=30,t=20 | x=75,y=30,t=60 | x=75,y=130,t=160 | x=775,y=130,t=860 | x=775,y=110,t=880
	Pantalla	Distancia total: 860 m Velocidad media: 1 m/s Es subversivo

Ejercicio 2 [3.5 puntos]

El **triángulo de Pascal**, también conocido como triángulo de Tartaglia, es una representación de los coeficientes binomiales ordenados en forma de triángulo, tal y como muestra la Figura 1.

Para construirlo, se parte de un uno en el vértice superior del triángulo. A partir de ahí, cada número del triángulo se puede obtener como la suma de los dos números adyacentes al mismo en la fila superior (o un único número adyacente si se trata de un número del borde izquierdo o derecho del triángulo, resultando en que todos los elementos de los bordes izquierdo y derecho toman valor 1).

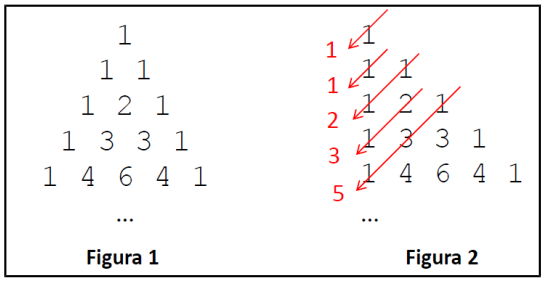


Figura 1 Figura 2

El triángulo de Pascal tiene además importantes propiedades matemáticas. Una de ellas es la posibilidad de calcular la sucesión de Fibonacci a partir de la suma de las diagonales del triángulo, tal y como se muestra (en rojo) la Figura 2 para los cinco primeros números de la sucesión.

Se pide, para un triángulo de Pascal de **19 líneas:**

a) Determinar la estructura de datos más apropiada para representar el triángulo.

b) Escribir un procedimiento que calcule los valores de las 19 líneas del triángulo.

```
procedure construirTriangulo( ??? );
```

c) Escribir un procedimiento que muestre por pantalla el triángulo anterior.

```
procedure mostrarTriangulo( ??? );
```

La salida puede mostrarse en forma de triángulo rectángulo, es decir, la salida por pantalla para las 5 primeras líneas (de las 19 que pedimos) será:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

d) Escribir un procedimiento que muestre por pantalla los 16 primeros números de la sucesión de Fibonacci, calculados a partir del triángulo de Pascal construido en el apartado anterior.

```
procedure mostrarSucFibonacci( ??? );
```

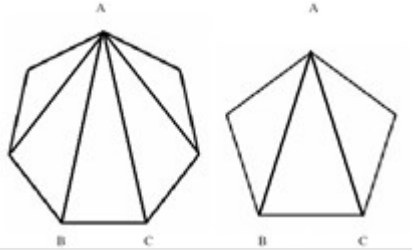
Así, la salida por pantalla que mostrará este procedimiento será:

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

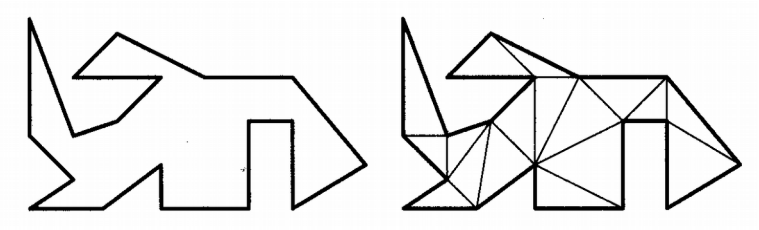
Observaciones: Todos los números que aparecen en el triángulo de 19 líneas pueden representarse con el tipo `integer`. Además, la línea *n* del triángulo está formada por *n* números (la línea 1 tiene un número, la línea 2 tiene dos números, y la línea 19 tiene diecinueve números).

Ejercicio 3 [3.5 puntos]

En la implementación de un programa de dibujo se necesita representar **polígonos** de cualquier forma, para luego convertirlos en triángulos. Convertir un polígono convexo en triángulos es fácil, pero no lo es tanto si el polígono es cóncavo:



Triangulación de dos polígonos convexos



Triangulación de un polígono cóncavo

Un polígono está definido mediante puntos en 2 dimensiones (con coordenadas x e y) denominados vértices, que se agrupan en una secuencia en el orden seguido para dibujar los lados de un vértice al siguiente. El polígono se cierra mediante un lado que conecta el último vértice con el primero. Para **triangular** polígonos cóncavos se utiliza el algoritmo de *ear-clipping* (recorte de orejas):

- Se busca un vértice central que con el anterior y el siguiente formen una oreja (ángulo convexo, sobresale hacia el exterior)
- Se define un triángulo con esos tres vértices
- Se genera un nuevo polígono con todos los vértices originales excepto el vértice central que se ha usado para definir el triángulo (el vértice anterior y siguiente se mantienen).

Sobre este tema **se pide**:

1. Define las estructuras de datos necesarias para representar un vértice de un polígono o de un triángulo (`tpVertice`), un polígono definido como una secuencia de vértices hasta un máximo de 32 (`tpPoligono`) y un triángulo, formado por exactamente 3 vértices (`tpTriangulo`).

2. Suponiendo que existe una función

   ```
   function esConvexo(anterior, central, siguiente: tpVertice) : boolean
   ```

   que nos dice si el vértice `central` define un ángulo convexo (oreja) con el `anterior` y el `siguiente`, implementa la siguiente función que devuelva el índice del primer vértice convexo del polígono:

   ```
   function primerVerticeConvexo(p : tpPoligono) : integer
   ```

3. Utilizando la función anterior, construye un procedimiento que reciba un polígono y devuelva como resultado otro polígono con un vértice menos y un triángulo, resultado de aplicar el recorte sobre el primer vértice convexo del polígono:

   ```
   procedure recorta( ??? original: tpPoligono;
                      ??? recortado: tpPoligono; ??? tri : tpTriangulo)
   ```