Deusto
Facultad de Ingeniería

EXAMEN PROGRAMACIÓN IV

(Parcial lenguaje C, 29 de abril de 2014)

Nombre y apellidos: _____

Código individual: _____

INSTRUCCIONES PARA EL EXAMEN

- La entrega del examen resuelto se realiza a través de la plataforma ALUD2. Existe una tarea para la entrega del examen llamada **Entrega examen 29/04/2014**
- El alumno debe crear un único **archivo ZIP** que contenga TODO el proyecto C y los ficheros requeridos, y subirlo a la plataforma. Solamente se corregirán los archivos entregados. Comprobar contenido del ZIP.
- Es imprescindible que en el fichero correspondiente al programa principal se incluya en la parte superior, como comentario, el nombre del alumno y el **código individual** que se entregará en el examen.
- La duración del examen es de **1 hora y 30 minutos**.
- Se puede hacer uso de la referencia de la librería de C contenida en ALUD2
- Se permite el uso de cualquier material en el examen (prácticas y apuntes).

Se parte de un **módulo "punto"** ya creado (con su fichero de cabecera **"punto.h"** y de implementación **"punto.c"**) y un fichero **"main.c"** que contiene el programa principal. Este programa principal crea un `Punto` y lo imprime por pantalla. Antes de comenzar el examen, prueba que este mínimo programa funciona correctamente de modo que estés seguro que el entorno de desarrollo se encuentra bien configurado. Una vez hecha esta comprobación, el examen consta de tres partes que deberás ir resolviendo incrementalmente.

Nota: los parámetros que se indican para las distintas funciones a implementar en el examen son orientativos, es decir, no se detalla en ningún caso si éstos deben ser punteros o estructuras. Será el alumno quien deba decidir y actuar en consecuencia. En muchos casos existirán varias alternativas correctas.

PARTE 1 (0,5 puntos)

1.1 Dentro de tu programa principal crea dos puntos con las siguientes coordenadas X e Y:

```
p1 = (1, 2)
p2 = (3, 4)
```

1.2 Añade dentro del módulo "punto" estas dos funciones:

`float distancia(Punto p1, Punto p2)`: calcula la distancia geométrica entre dos puntos. La distancia se calcula implementando la siguiente operación:

$$\text{Distancia} = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

Para implementar la raíz cuadrada puede serte de utilidad la función `sqrt()` que se encuentra en la librería <math.h>

`void trasladarXY(Punto p1, Punto p2, int x, int y)`: realiza un desplazamiento relativo de ambos puntos (p1, p2), incrementando las coordenadas X e Y de ambos en las unidades contenidas en el tercer y cuarto argumento (x, y).

1.3 En el programa principal usa estas dos funciones sobre los dos puntos (p1, p2) creados e imprime los puntos, antes y después de realizar la traslación, así como la distancia existente entre ellos. Deberás comprobar que efectivamente las coordenadas de los puntos han sido trasladadas y que la distancia entre ellos no ha cambiado.

Posible **juego de ensayo.** Asumiendo una traslación de X=5 e Y=6, una vez ejecutada la parte 1, un posible resultado por pantalla sería el siguiente:

```
(1, 2)
(3, 4)
```

```
Distancia = 2.828427
Después de trasladar los puntos...
Distancia = 2.828427
(6, 8)
(8, 10)
```

Opcional (0,25 puntos adicionales). Hacer que las coordenadas X y Y de los dos puntos (p1, p2) en lugar de insertarlas en el código, se lean de los argumentos que recibe el programa.

PARTE 2 (0,75 puntos)

2.1 Crea un nuevo módulo "polígono" (con su correspondiente fichero de cabecera y de implementación), el cual defina la estructura **`Poligono`**. Esta estructura deberá contener el número de vértices que conforman el polígono, así como todos los puntos que conforman estos vértices (deberás hacer uso de la estructura **`Punto`** con la que trabajaste anteriormente).

```
int numVertices
Punto *vertices
```

Ten presente que el número de vértices no es un dato conocido a priori. Podemos estar hablando de un triángulo (3 vértices y 3 puntos), un cuadrado (4 vértices y 4 puntos) o cualquier otro polígono.

2.2 Añade dentro del módulo "polígono", al menos, estas tres funciones:

> **`void imprimirPoligono(Poligono poli)`**: imprime por pantalla los valores de los puntos correspondientes a los vértices del polígono.
>
> **`float perimetro(Poligono poli)`**: calcula el perímetro del polígono. El perímetro se corresponde con la suma de la longitud de todos los lados del polígono. Por tanto, usando la función que calcula la distancia entre dos puntos, puedes ir sumando las distancias entre los distintos vértices del polígono. Todo punto (vértice) debe considerar la distancia con su adyacente y ten en cuenta que el último punto lo debe hacer con el primero.
>
> **`void liberar(Poligono poli)`**: libera la memoria dinámica reservada para ese polígono.

2.3 En el programa principal crea un polígono e inicialízalo. Para ello, pide por teclado el número de vértices del polígono y posteriormente los valores de tantos puntos como vértices vaya a tener[1].

Una vez creado el polígono, usa las funciones anteriores sobre él para imprimir sus vértices y visualizar su perímetro. Deberás comprobar que efectivamente los vértices del polígono se corresponden con los puntos introducidos por teclado. Finalmente, no olvides liberar la memoria.

[1] En caso de que tengas problemas para leer por teclado, no dediques excesivo tiempo a ello en el examen. Puedes inicializar desde código el polígono (número de vértices y sus puntos) de modo que puedas continuar realizando el resto de apartados.

Posible **juego de ensayo.** Asumiendo que se insertan por teclado los 4 puntos correspondientes, una vez ejecutada la parte 2, un posible resultado por pantalla sería el siguiente:

```
Vertice 0 = (1, 2)
Vertice 1 = (3, 4)
Vertice 2 = (5, 6)
Vertice 3 = (7, 8)
Perimetro = 16.970562
```

PARTE 3 (0,75 puntos)

3.1 Añade dentro del módulo "polígono" estas dos funciones:

`void anadirVertice(Poligono poli, Punto p)`: añade un nuevo vértice (punto) al polígono ya existente. Ten presente que implica incrementar en uno el número de vértices y volver a reservar memoria para poder contener todos los puntos anteriores más el nuevo vértice.

`void copiarPoligono(Poligono poli1, Poligono poli2)`: copia en el polígono del primer argumento (poli1) el contenido del segundo polígono (poli2). Una vez invocada la función, el polígono copiado no debe compartir ningún dato en memoria con el polígono original.

3.2 En el programa principal y sobre el polígono que ya tenías creado, usa la función que permite añadir un nuevo vértice. Posteriormente usa la función que te permite copiar polígonos para obtener un nuevo polígono igual que el anterior. Finalmente, imprime los valores de ambos polígonos para comprobar que todo funciona correctamente.

Posible **juego de ensayo.** Asumiendo que el nuevo vértice insertado es el punto (4, 4), una vez ejecutada la parte 3, un posible resultado por pantalla sería el siguiente:

```
Después de añadir un vertice...
Vertice 0 = (1, 2)
Vertice 1 = (3, 4)
Vertice 2 = (5, 6)
Vertice 3 = (7, 8)
Vertice 4 = (4, 4)
Después de copiar el poligono...
Vertice 0 = (1, 2)
Vertice 1 = (3, 4)
Vertice 2 = (5, 6)
Vertice 3 = (7, 8)
Vertice 4 = (4, 4)
```