

# Sesión 08: Más sobre tipos y sintaxis (incompleto)

## Hoja de problemas


Programación para Sistemas


Ángel Herranz

aherranz@fi.upm.es

Universidad Politécnica de Madrid

Otoño 2018

 **Ejercicio 1.** Repasa las transparencias de clase. Además de explorar los nuevos elementos (**enum**, **union**, y **typedef**), el aspecto más importante es la combinación de tantas cosas vistas en la sesiones anteriores y la exploración a fondo de ciertas partes de la sintaxis de C.

 **Ejercicio 2.** Tu primera tarea va a consistir en implementar un programa que lea figuras geométricas de la entrada estándar e imprima el área de cada una de ellas en la salida estándar.

Cada línea de la entrada estándar representa una de estas figuras. Cada línea empieza con un string que indica el tipo de figura, a saber: **circulo**, **triángulo**, **rectángulo**. Dependiendo del tipo, le seguirán ciertos parámetros:

- Para **circulo**: El punto central con dos coordenadas **int**  $x$  e  $y$  separadas por espacios<sup>1</sup> y el radio (**float**) también separado por espacios.
- Para **triangulo**: Los tres puntos que definan el triángulo, seis enteros separados por espacios. Dos de los puntos siempre coincidirán en la coordenada  $y$  y forman parte de la base del triángulo.
- Para **rectángulo**: Los puntos sudoeste y noreste, cuatro enteros separados por espacios.

Veamos algunos ejemplos:

```
circulo 0 0 2
triangulo -1 0 2 1 0 3
rectángulo -1 -2 3 2
```

---

<sup>1</sup>Todos los puntos se representarán así.

- Necesitarás un último struct para poder discriminar el dato que tienes almacenado en el union
- Tendrás que utilizar union para describir que en la variable puedes tener cualquiera de las tres figuras.
- Tendrás que declarar un struct por cada tipo de figura: círculo,

Un poquito de ayuda:

📄 **Ejercicio 3.** En la sesión de hoy, en clase, tienes múltiples ejercicios que han quedado propuestos. Te los recordamos aquí:

- Funciones sobre el tipo **enum** mes.
- Adaptar tu código a la convención de código `_t`, `_e`, `_s`, `,` `_u`.

📖 **Ejercicio 4.** En la sesión de hoy, en clase, se ha dejado propuesto un ejercicio de ordenación de enteros en ristas. Recordatorio:

- Escribe un programa que ordene ristas de enteros de menor a mayor
- Cada rista se representa de la siguiente forma en la entrada estándar:
  - Un entero positivo  $n$  en la primera línea
  - $n$  enteros en las  $n$  siguientes líneas
- Por cada rista, la salida de tu programa tiene los  $n$  enteros de la rista ordenados de menor a mayor
- Tu programa debe parar cuando lea una rista de 0 enteros

Veamos de nuevo un ejemplo de entrada (con dos ristas) y la salida esperada:

Entrada	Salida esperada
2	1
7	7
1	
3	1
1	2
4	4
2	
0	

Las restricciones para este ejercicio son:

- **Usa el módulo de árboles binarios**
- **Evita consumir más memoria de la necesaria**
- **Presta especial atención a los *memory leaks***

La función de inserción en orden ya la implementaste en clase. En este ejercicio tendrás que escribir una función que haga un recorrido apropiado del árbol e imprima los enteros en orden. También tendrás que recorrer el árbol adecuadamente para liberar toda la memoria consumida.

» **Ejercicio 5.** ¿Recuerdas las órdenes bash que generaban listas de enteros? Para probar el ejercicio anterior, te sugerimos que las modifiques para generar más de una rista de enteros.