1. Dada la siguiente estructura y declaración de variable de estructura:

```
struct CuentaCD {
  float saldo;
  float tasa_de_interes;
  int plazo;
  char inicial1;
  char inicial2;
};

struct CuentaCD cuenta;
```

   ¿Qué tipo de dato tiene cada una de las siguientes variables?

   a) `cuenta.saldo`
   b) `cuenta.tasa_de_interes`
   c) `CuentaCD.plazo`
   d) `cuenta_ahorros.inicial1`
   e) `cuenta.inicial2`
   f) `cuenta`

2. Crear una estructura apropiada para representar el tipo de dato número complejo. Un número complejo se representa como: `<parte_real> + <parte_imaginaria> * i` donde `i` representa $\sqrt{-1}$. Construir además funciones apropiadas para:

   a) Sumar dos números complejos.
   b) Multiplicar dos números complejos.
   c) Hallar el conjugado de un número complejo.
   d) Imprimir un número complejo con el formato `(r, i)`, siendo `r` la parte real y `i` la parte imaginaria.

3. Establecer una definición de una estructura que represente el tipo de dato rectángulo. A partir de la definición propuesta construir:

   a) Una función que calcule el área de un rectángulo.
   b) Una función que calcule el perímetro de un rectángulo.
   c) Una función que dados dos rectángulos, nos diga cual de ellos es mayor, teniendo en cuenta que el mayor es aquel que tiene mayor área.
   d) Una función que nos diga si dos rectángulos son idénticos. Ser idénticos implica que tienen el mismo área y el mismo perímetro.
   e) Una función que intercambie los valores entre dos variables de tipo rectángulo.
   f) Una función que dado un vector de rectángulos los ordene de mayor a menor.

4. Se tiene un vector de `ciudad` de tamaño `n`. `ciudad` es una estructura que almacena información sobre la posición relativa de una determinada ciudad con respecto a un determinado origen de coordenadas, siendo su descripción la siguiente:

```
struct ciudad{
  float x;
  float y;
  char nombre[50];
};
```

   Se pide construir una función que, a partir del vector de `ciudad` y dado el nombre de una ciudad, reordene ascendentemente el vector atendiendo a la distancia euclídea del resto de las ciudades con respecto a la elegida no pudiendo hacer uso de vectores auxiliares.

5. Un polígono puede representarse mediante la siguiente estructura de datos:

```
struct Punto{                    struct Poligono{
  float x;                         struct Punto ptos[100];
  float y;                         int num_ptos;
};                               };
```

donde `num_ptos` tiene el número de puntos del polígono. Los lados del polígono están constituidos por los pares de puntos (`ptos[i]`, `ptos[i+1]`) (i=0,...,`num_ptos`-2) y el par (`ptos[num_ptos-1]`,`ptos[0]`). Por otro lado, el área de un polígono convexo puede calcularse mediante la suma de un conjunto de triángulos que están inscritos en él. Los triángulos se obtienen uniendo los lados del polígono con un punto interior a la figura, mediante dos segmentos.

Construir la función `float AreaPoligono(const struct Poligono *P)` que calcule el área de un polígono `P` a través de la suma de las áreas de sus triángulos constituyentes. Para ello, primero hay que implementar una función `float AreaTriangulo(const struct Punto *pto1, const struct Punto *pto2, const struct Punto *pto3)` que devuelve el área del triángulo formado por los puntos `pto1`, `pto2`, y `pto3`. Se usará la siguiente fórmula:

$$Area = \sqrt{F(F-S1)(F-S2)(F-S3)},$$

siendo $F = (S1 + S2 + S3)/2$ donde $S1$, $S2$ y $S3$ son las longitudes de los lados del triángulo. El alumno hará uso de una función `Punto InteriorPoligono(const struct Poligono *P)` que devuelve un punto interior del polígono `P`. Esta función no tendrá que implementarse[1].

6. Se ha diseñado un método de codificación de caracteres que consiste en sustituir cada carácter por un par de números enteros positivos. Este método utiliza una lista de registros que almacenan las sustituciones asociadas a cada carácter. Cada registro consta de un identificador (un número entero positivo) que es único para cada registro y un vector de cinco caracteres que almacena cada carácter que permite codificar ese registro. La lista de registros se implementa como una estructura compuesta por un vector de registros y un valor entero que indica el número de registros que componen la lista. La descripción de dichas estructuras es la siguiente:

```
struct Registro {                struct Lista_Reg{
  int N;                           int N_Reg;
  char letra[5];                   struct Registro Clave[100];
};                               };
```

Un ejemplo de dicha lista de registros podría ser la siguiente:

(2)	t	s	r	b	a	(7)	c	e	m	s	x	···	(3)	e	n	l	d	a

Dada una definición de lista de registros como la anterior, un carácter se puede representar como un par de enteros $[a, b]$, donde a representa el identificador de registro donde se encuentra el carácter a codificar y b es la posición de dicho carácter dentro de ese registro. Por ejemplo, el par $[3, 0]$, representa al carácter que se encuentra en el registro 3, y ocupa la posición 0 en su vector asociado, es decir, el carácter *'e'*. Así, la secuencia $[3, 0][7, 4][2, 4][7, 2][7, 1][3, 1]$ representa la palabra *examen*. Nótese que un mismo carácter puede tener diferentes representaciones en el código, por ejemplo, *'e'* se codifica como $[3, 0]$ ó $[7, 1]$. La implementación de un código y de una secuencia de códigos es la siguiente:

```
struct Codigo{                   struct Secuencia{
  int id_reg;                      int tama;
  int posicion;                    struct Codigo caracter[100];
};                               };
```

Se pide:

a) Construir una función en C que permita ordenar ascendentemente los registros de una lista del tipo `struct Lista_Reg` por el identificador del registro (campo `N` del registro `struct Registro`).

[1] Si el alumno quisiera implementarla puede tener en cuenta que cualquiera de los vértices de un polígono puede considerarse interior al polígono.

b) Construir una función en C que dado un `código`, es decir, un par $[a, b]$, y una lista del tipo `struct Lista_Reg` devuelva el carácter que representa dicho código. Para ello, suponer que los registros de la lista se encuentran ordenados ascendentemente por el identificador de registro y que los identificadores de los registros no tienen porque representar números consecutivos. La cabecera de dicha función será la siguiente:

```
char Devolver_Caracter(const struct Codigo *C, const struct Lista_Reg *L)
```

c) Usando la función anterior, construir otra función en C que dada una secuencia `S`, es decir, un parámetro de tipo `Secuencia`, un carácter `ch` (un parámetro de tipo `char`) y una lista de registros `L` (de tipo `struct Lista_Reg`) que devuelva el número de veces que aparece el carácter `ch` en la secuencia `S` si usamos la lista de registros `L`.

7. Para almacenar información sobre un DVD utilizaremos la estructura:

```
struct DVD {
  int codigo;       /* < número del DVD. Mayor que cero */
  char titulo[257]; /* < string. Título de la película */
  int agno;         /* < año de estreno (cuatro dígitos). */
};
```

Para mantener nuestra colección de DVDs utilizaremos un vector de `DVD`. Se pide:

a) Hacer una función que dado un vector de `DVD` ordene dicho vector por título. El resultado se devolverá en el mismo vector.

b) Hacer una función que dado un vector `v` de `DVD`, que se sabe que está ordenado por título, y un título `titulo`, almacenado en un *string*, nos devuelva la posición en el vector del `DVD` cuyo título sea `titulo` o -1 si no se encuentra un `DVD` con ese título en el vector `v`. Se valorará la eficiencia de la solución propuesta.

c) Supuesto que tenemos un vector `v` ordenado por título hacer una función para cambiarle el título a un DVD (manteniendo el vector ordenado).

8. Se desea implementar un tipo de dato Conjunto que pueda contener números enteros usando la siguiente estructura:

```
struct Conjunto1000{
   int num_elem;
   int elementos[1000];
};
```

La estructura está compuesta por un vector de 1000 posiciones y un campo que indica el cardinal del conjunto, es decir, el número de elementos que actualmente forman parte del mismo. Además, el vector con los elementos se encuentra ordenado de forma ascendente, y después de cada modificación sobre la estructura se debe conservar dicha ordenación. Se pide implementar las siguiente funciones:

a) Una función booleana para determinar si un elemento se encuentra en el conjunto.

b) Una función para incorporar un nuevo elemento al conjunto.

c) Una función que calcula la unión de dos conjuntos definida como $A \cup B = \{x/x \in A \vee x \in B\}$

d) Una función que calcula la intersección de dos conjuntos definida como $A \cap B = \{x/x \in A \wedge x \in B\}$

e) Una función que calcula la diferencia de dos conjuntos definida como $A - B = \{x/x \in A \wedge x \notin B\}$