Objetivo de la práctica:

Aprender a utilizar el tipo string y las funciones definidas sobre él.
Declarar y utilizar el tipo de dato estructurado registro (o *struct*)

Conceptos básicos: tipo *string*

Las cadenas de caracteres (strings) permiten la manipulación de textos. Un string se puede considerar un vector de caracteres con unas operaciones añadidas. En C++, las cadenas de caracteres se representan mediante el tipo *string*. Para su uso es preciso utilizar **`#include <string>`**, no **`<string.h>`**.

Operaciones básicas definidas para *string*:

- Creación de variables:

```
string palabra, frase;
```

- Asignación:

```
frase = palabra;
frase = "hola";
```

- Acceso a los caracteres (como con vectores):

```
palabra[0]
```

- Comparación lexicográfica (**`==`**, **`!=`**, **`<`**, **`>`**):

```
frase == palabra
frase > palabra
```

- Lectura/escritura:

```
cin >> palabra;
getline (cin, frase);    //lee frase de cin hasta encontrar el fin de línea
cout << frase << endl;
```

- Manipulación de textos (suponer **`unsigned int i;`**):

```
// n° de caracteres de palabra
i = palabra.length();

// inserta palabra en la posición 3 de frase
frase.insert(3, palabra);

// concatena (une) palabra y "hola" y almacena el  resultado en frase
frase = palabra + "hola";

// concatena (añade al final) palabra a frase
frase += palabra;

// borra 7 caracteres de frase desde la posición 3
frase.erase (3,7);

// sustituye (reemplaza) 6 caracteres de frase, empezando en la posición 1,
// por la cadena palabra
frase.replace (1, 6, palabra);

//busca palabra como una subcadena dentro de frase, devuelve la posición
//donde la encuentra
i = frase.find(palabra);
```

```
//devuelve la subcadena formado por 3 caracteres desde la posición 3 de
//frase
palabra = frase.substr(5,3);
```

Conceptos básicos: Registros

Una estructura es una colección de elementos de información, denominados campos, que representan información relevante sobre una entidad. Cada uno de estos campos puede pertenecer a un tipo de dato diferente y el acceso se realiza mediante el nombre del campo (y no mediante un índice como en los vectores). Ejemplo:

```
// información de interes sobre un empleado de una empresa
struct empleado
{
   string nombre;
   long int salario;    // salario anual en pesetas
   string num_telefono;
};
```

Operaciones básicas

- Creación de una variable estructurada.

```
empleado jose, antonio;
```

- Asignación de estructuras.

```
jose = antonio;
```

- Acceso a los datos contenidos en la estructura (operador 'punto').

```
jose.nombre = "Jose";
```

- La lectura y escritura de la información de la estructura se debe hacer campo a campo.

```
cin >> jose.nombre;
cin >> jose.salario;
cin >> jose.telefono;
cout << antonio.nombre << antonio.telefono << endl;
```

Una estructura puede tener como campos otras estructuras:

```
struct departamento
{
   string nombre;
   int numero_empleados;
   empleado jefe;
};
```

También es posible definir *arrays* de estructuras:

```
typedef empleado plantilla [30];
```

EJERCICIOS

(*) 1. Escribir un programa en C++ que diga cuantas veces se repite una determinada palabra en una frase. Ejemplo:

```
Palabra: problema
Frase: Este problema es mas interesante que el problema anterior.
Número de repeticiones: 2
```

(*) 2. Escribir un programa que elimine todos los espacios en blanco de una frase introducida por teclado.

(*) 3. Escribir un programa que convierta un string a entero. El programa devolverá también un bool indicando si la conversión se ha podido realizar con éxito o no. Para convertir un carácter a entero se le resta el código del carácter '0'.

4. Se dice que la palabra α es un anagrama de la palabra β si es posible obtener α cambiando el orden de las letras de β. Por ejemplo, MORA y ROMA son anagramas de RAMO.

Realizar un programa que diga si dos palabras introducidas por teclado son anagramas entre si. Considerar que las palabras solo contienen letras mayúsculas.

(*) 5. Hacer un programa para calcular operaciones sobre puntos en un espacio 2-D. Los puntos tendrán coordenadas (x, y) almacenadas en un registro para cada punto. El programa pedirá el punto1 y el punto2 y calculará la distancia, el vector suma y el vector resta, y los mostrará por pantalla.

(*) 6. Crear un programa para realizar cálculos sobre triángulos en un espacio 2-D. El programa debe permitir la introducción de las coordenadas (x, y) para los tres vértices del triángulo, que deberán ser almacenados en un registro. Posteriormente, se deberá poder calcular el área del triángulo, el perímetro del triángulo y el punto medio del triángulo.

```
Area = base * altura / 2     ó
Area = 1/2*((x3.x-x1.x)*(x1.y-x2.y)+(x3.y-x1.y)*(x2.x-x1.x))
Perimetro = (distancia(x1,x2) + distancia(x2,x3) + distancia(x3,x1))
Punto medio = ((x1.x + x2.x + x3.x) / 3, (x1.y + x2.y + x3.y) / 3)
```

(**) 7. Hacer un programa para almacenar la información de una tabla de clasificación de un equipo de fútbol en un campeonato. Se utilizará un registro para almacenar el nombre del equipo, los puntos obtenidos, los partidos ganados, empatados y perdidos, los goles a favor y los goles en contra. Se utilizará un vector para almacenar la información de los 10 equipos del campeonato. Se generarán datos aleatorios para rellenar los valores de cada equipo para hacer pruebas. El programa indicará, después de los cálculos aleatorios, cuales son los equipos primero y último, mostrando la información completa. También se podrá mostrar información completa sobre la clasificación.

8. Crear un programa que gestione una agenda de direcciones. El programa debe permitir añadir nuevas entradas, borrar entradas existentes y buscar entradas por apellido. La información de cada entrada de la agenda debe constar de nombre, apellidos, dirección, teléfono y e-mail. La implementación de la agenda se realizará mediante un vector de estructuras.