PRÁCTICA 7: Cadenas y Estructuras.

Objetivo de la práctica:
- Utilización y manipulación de cadenas de texto. Conocer y aplicar el tipo de dato estructurado en C++.

Cadenas

Las cadenas ('string') son un tipo especial de datos que nos facilitan las operaciones con agrupaciones de caracteres. Un string sería equivalente a un vector de caracteres, pero con funcionalidades extendidas.

Por ejemplo, con vectores de caracteres no podríamos usar simples sentencias de asignación o comparación:

```
char opcion1[10], opcion2[10];
opcion1 = opcion2;          -- Sentencia inválida
if (opcion1 == 'Si')        -- Sentencia inválida
```

Este tipo de operaciones básicas con cadenas, junto con un amplio conjunto de funcionalidades adicionales, es posible realizarlo mediante el tipo string.

Operaciones básicas de la clase string

```
string s  = "abc def abc", s2 = "abcde uvwyz";
char c;
```

Stream input	`cin >> s;`	Cambia el valor de s por el valor introducido. El valor se interrumpe al primer espacio en blanco.
Stream output	`cout << s;`	Escribe el string en la salida
Entrada de línea	`getline (cin,s);`	Asigna el valor introducido por teclado (hasta antes del carácter de final de línea) al string s
Asignación	`s = s2; s = "abc";`	Asignaciones entre variables string
Subscript	`s[1] = 'c'; c = s[1];`	s cambia a "acc def abc" c cambia a b
Longitud	`i = s.length();`	i pasa a valer el número de caracteres de s
Vacio	`if ( s.empty() ) i++;` `if ( s == "" ) i++;`	En ambos ejemplos se incrementa en 1 el valor de i si la cadena está vacía
Concatenación	`s2 = s2 + "x";` `s2 += "x";`	En ambos ejemplos se añade "x" al final de s2
Substring	`s = s2.substr(1,4);` `s = s2.substr(1,50);`	El primer ejemplo guarda en s la subcadena de s2 que comienza en el carácter 1 y tiene una longitud de 4, "bcde". En el segundo ejemplo s guarda "bcde uvwxyz". Si la longitud indicada en el substring es mayor que los caracteres restantes se detiene en el último carácter. La primera posición de un string es la 0.
Sustitución de substrings	`s.replace(4,3,"x");`	Sustituye los 3 caracteres de s comenzando por la posición 4 con el carácter "x". La variable s queda con el valor "abc x abc"
Eliminación de substrings	`s.erase(4,5);` `s.erase(4);`	Borra los 5 caracteres de s desde la posición 4 (s="abc bc") Borra desde la posición 4 hasta el final (s="abc "
Búsqueda de patrones	`s.find("ab",4);`	Retorna la posición (en este caso 8) del patrón ab en la cadena s comenzando la búsqueda en la posición 4

Ejercicios

1. Escribir un programa que dada una frase nos diga el número total de letras, el número de vocales, el número de consonantes y el número de espacios en blanco.

2. Diseñar un programa que lea una cadena y la invierta.

3. Escribir un programa que limpie de ruidos una señal de entrada. La señal de entrada será una cadena con letras y números y la salida será la misma cadena eliminando los números. Por ejemplo para la cadena "Es2to0 3es u9na se88ñal c0on ru1id2os" debe devolver "Esto es una señal con ruidos".

4. Escribir un programa que convierta una fecha en formato "MM-DD-YYYY" al formato "DD de mes del YYYY". Por ejemplo, para "12-07-2006" debería devolver "7 de diciembre del 2006".

Estructuras

Los registros, también llamados *estructuras*, son tipos compuestos de datos heterogéneos, es decir, que agrupan datos que pueden ser de diferentes tipos.

Consideremos por ejemplo una ficha de venta de un libro para una librería. En ella se guarda la siguiente información:

- Titulo del libro
- Autor
- Editorial
- Número de páginas
- Precio

Toda esta información puede guardarse en una estructura de datos de tipo registro. Para definirla se utiliza la palabra reservada *struct* y tiene la siguiente sintaxis:

```
struct ficha{
      string titulo;
      string autor;
      string editorial;
      int num_pag;
      float precio;
};
```

Esta declaración debe entenderse como una declaración de tipo (de ahí el punto y coma final) y no como un subprograma ni como una declaración de variables. Para declarar una variable de tipo ficha deberemos hacerlo dentro de un subprograma de la manera habitual:

```
int main(){
      ficha venta;
      ....
      return 0;
}
```

Ahora la variable venta es de tipo ficha. A cada uno de los componentes de un registro se denomina campo. Así, los datos titulo, autor, editorial, num_pag, precio, son los campos del registro. Para acceder a cada uno de los campos de un registro se utiliza el operador punto ‘.’.

```
venta.titulo = “El conde de Montecristo”;
```

Inicialización de un registro

Un registro puede inicializarse en el momento de su declaración como variable:

```
struct dimensiones_libro{
      float alto;
      float ancho;
      };
      int main(){
      dimensiones_libro dim={12.4, 20.8};
      ....
      return 0;
}
```

Como caso aparte, si algún campo del registro es de tipo string, no puede inicializarse de la anterior manera. La alternativa es la inicialización campo a campo que podemos hacer en cualquier punto del programa:

```
int main(){
      ficha libro;
      ...
      libro.titulo = "El conde de Montecristo";
      libro.autor = "Alejandro Dumas";
      libro.editorial = "Aguilar";
      libro.num_pag = 324;
      libro.precio = 12.35;
      ...
      return 0;
}
```

Asignación entre registros y paso como parámetro a un subprograma

En C++ está permitida la asignación entre registros:

```
dimensiones_libro libroA, libroB;
libroA.alto = 13.56;
libroA.ancho = 12.56;
libroB = libroA;      //ahora libroB tiene el mismo contenido que libroA
```

Como consecuencia de esta propiedad, **un registro puede pasarse por valor o por referencia como parámero a una función. Igualmente una función puede devolver un registro**.

Uso de registros en estructuras de datos complejas

Un registro puede tener como campo otro registro. Para ello el registro que hace de campo debe declararse antes que en el momento de su uso como tipo de dato de un campo de otro registro.

```
struct dimensiones_libro{
      float alto;
      float ancho;
};
struct ficha_modif{
      string titulo;
      string autor;
      string editorial;
      int num_pag;
      float precio;
```

```
        dimensiones_libro tamanyo;
};
```

El acceso a los campos del registro *tamanyo* se hace usando dos veces el operador punto '.'.

```
ficha_modif libro;
cout << "El libro mide " << libro.tamanyo.alto << "cm de alto \n";
```

El campo registro tiene todas las propiedades de los registros (asignación,...).

Vectores de registros

Esta estructura de datos es muy útil para hacer colecciones de datos . Por ejemplo para tener los datos de todos los libros en venta de una librería, podemos guardar las fichas de los libros en un vector.

```
const int MAX_TAM= 500;
typedef ficha todos_los_libros[MAX_TAM];
todos_los_libros inventario;
```

La variable inventario guardaría hasta un máximo de 500 fichas de libros en venta.

Ejercicios

5. Escribir un programa que implemente la estructura *fraccion* con dos campos, numerador y denominador. El programa incluirá la función de suma que deberá adecuarse al siguiente prototipo:

```
fraccion suma ( fraccion, fraccion );
```

6. Crear un programa que implemente un calendario del 2007. Para cada día el calendario guardará el número de día, el mes, el día de la semana, si es festivo o no y el nombre de la festividad en el caso de que lo sea. Inicializar el calendario con la primera semana de enero del 2007 y mostrar por pantalla esta semana. Implementad una función que devuelva el día de la semana de un día seleccionado.

7. Realizar un programa que realice un filtrado de mensajes electrónicos recibidos en un servidor de correo.

El filtro marcará como inválidos siguientes mensajes:
- Los que contengan en el título del mensaje palabras ofensivas
- Los que contengan archivos con extensión no permitida (.exe,.zip,...)
- Los enviados desde direcciones que se consideran origen de spam

El programa deberá utilizar una estructura *mensaje* que contenga los campos remitente, destinatario, adjunto, título y cuerpo del mensaje. El esqueleto básico del programa debe adecuarse a la siguiente estructura:

```
#include <iostream>
using namespace std;

#include <string>

const int MAX_TAM = 5;

typedef struct {
....
} mensaje;

void introduce_mensaje ( mensaje & );
bool mensaje_ofensivo ( mensaje, string [MAX_TAM] );
bool mensaje_adjunto_peligroso ( mensaje, string [MAX_TAM] );
```

```
bool mensaje_spam ( mensaje, string [MAX_TAM] );

int main ( void )
{
  char op;
  mensaje correo;
  string palabras_ofensivas[MAX_TAM] = {"xxx","yyy"};
  string adjuntos_peligrosos[MAX_TAM] = {".exe",".zip"};
  string direcciones_spam[MAX_TAM] =
{"spam@yahoo.es","pepe@gmail.com","lola@hotmail.com"};

  do
  {
        introduce_mensaje(correo);

        if ( mensaje_ofensivo(correo,palabras_ofensivas) ||
mensaje_adjunto_peligroso(correo,adjuntos_peligrosos) ||
mensaje_spam(correo,direcciones_spam) )
                cout << endl << "El mensaje es incorrecto";
        else
                cout << endl << "El mensaje es correcto";

        cout << endl << endl << "¿Quiere introducir un nuevo mensaje (s/n)?: ";
        cin >> op;
  } while ( op != 'n' );

  cout << endl << endl << "Final del programa" << endl << endl;
  return 0;
}
```