EJERCICIOS PARA RESOLVER

STRINGS

1. **(P1) (ocurrencias.cpp)** Escribir un programa en C/C++ que dada una cadena de caracteres permita sustituir todas las ocurrencias de una subcadena. Ejemplo:
   Cadena: Este problema es mas interesante que el problema anterior.
   Vieja Subcadena: problema
   Nueva Subcadena: programa
   Cadena Resultado: Este programa es más interesante que el programa anterior.
   Haz una función que, dada una cadena de caractéres y una palabra, deberá sustituir cada aparición de esta por una nueva pasada también como parámetro. Un prototipo adecuado podría ser:
   int BuscarReemplazar(string &frase, string viejacadena, string nuevacadena);
   La función devolverá el número de apariciones de 'viejacadena' en frase.

2. **(P2) (caracteres.cpp)** Escribir un programa en C/C++ que dada una cadena de caracteres permita obtener el número total de caracteres y palabras (nota: los espacios también son caracteres). Ejemplo:
   Cadena: El perro de San Roque no tiene rabo, Porque Ramón Ramírez se lo ha cortado.
   Salida:
   Palabras: 15
   Caracteres: 75

3. **(ahorcado.cpp)** Escriba un programa para jugar al ahorcado. El programa debe elegir una palabra (que pueda codificarse directamente en el programa) y mostrar lo siguiente:

   Adivine la palabra: xxxxx

   Cada X representa a una letra. Si el usuario adivina correctamente, el programa debe mostrar:

   ¡Felicidades! Usted pudo adivinar mi palabra. Desea jugar otra vez? si/no

   Debe introducir la respuesta si o no apropiada. Si el usuario no adivina correctamente, debe mostrarse la parte del cuerpo apropiada. Después de 7 intentos fallidos, el usuario debe ser colgado. La pantalla debe verse así:

4. **(palindroma.cpp)** Realizar un programa en C/C++ que pida una frase y nos indique si es o no palíndroma (se lee igual de izquierda a derecha que de derecha a izquierda). Resuélvelo usando funciones de strings. Por ejemplo:

   Entrada → dabale arroz a la zorra el abad

   Salida → cierto

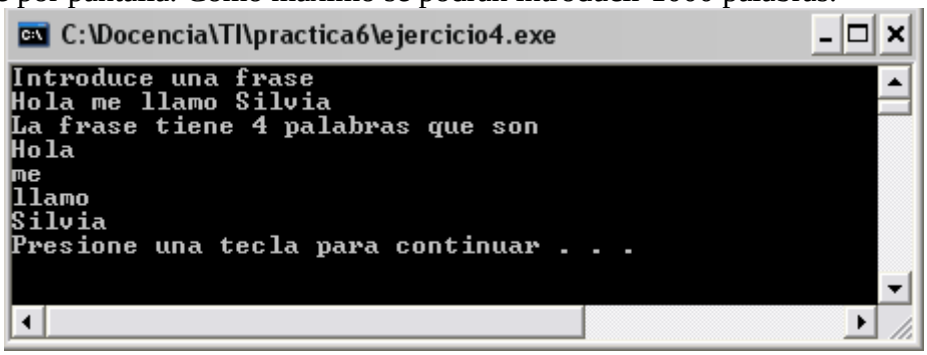
5. **(cuenta_pal.cpp)** Escribe un programa que cuente el número de palabras que aparecen en una sola línea de entrada. Se entiende por palabra una secuencia no vacía de letras mayúsculas o minúsculas delimitada por el principio de la línea, el final de ella o cualquier carácter no alfabético.

   Por ejemplo, si al ejecutar el programa el usuario introduce la línea que reproduce las primeras frases de La Regenta:

   La heroica ciudad dormia la siesta. El viento sur, caliente y<retorno>

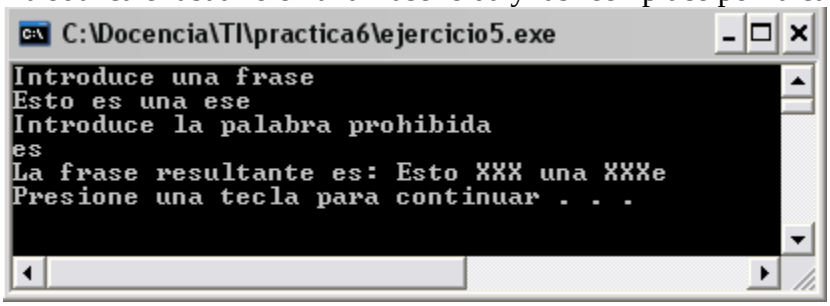
   el programa debe contabilizar 11 palabras.

6. **(separa.cpp)** Escribir un programa que guarde en un vector todas las palabras leídas de una frase y las muestre por pantalla. Como máximo se podrán introducir 1000 palabras.

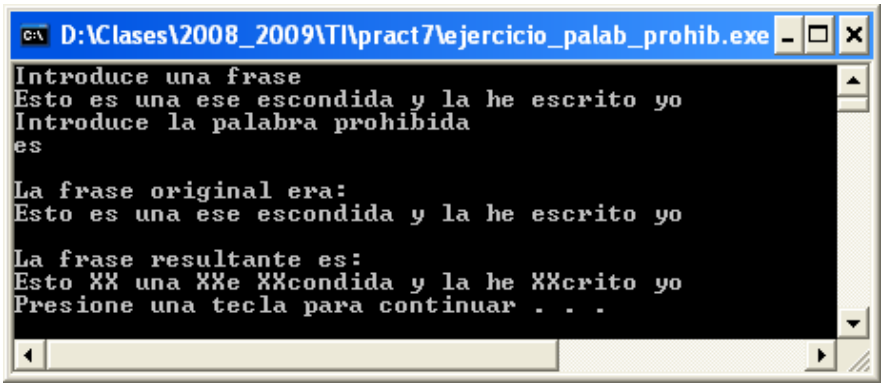


7. **(mayúsculas.cpp)** Escribe una función que transforme una cadena de caracteres de minúsculas a mayúsculas y úsala desde el programa principal.

8. **(P3)** **(pal_prohib.cpp)** Realizar un programa que busque todas las ocurrencias de la palabra prohibida que introduzca el usuario en una frase leída y las reemplace por la cadena “XXX”.



9. **(Op1)** **(pal_prohib2.cpp)** Realizar un programa que busque todas las ocurrencias de la palabra prohibida, que introduzca el usuario, en una frase leída y las reemplace por la cadena “XXX” (dónde el número de “X” que contiene la cadena deberá ser igual al número de caracteres que tiene la palabra prohibida).



10. **(anagrama.cpp)** 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 una función que compruebe si dos palabras son anagramas entre sí. Las palabras pueden contener letras mayúsculas y minúsculas. Haz una función que convierta mayúsculas a minúsculas.

11. **(vocales.cpp)** Escribir un programa en C++ que calcule el número de vocales que contiene una frase introducida por teclado y nos diga cuantas vocales de cada clase hay (cuantas 'a', cuantas 'e',...).

12. **(cuantas.cpp)** Escribir un programa que dada una frase nos diga el número total de caracteres, el número de vocales, el número de consonantes, el número de espacios en blanco y cuántos son "de otro tipo".

13. **(invertir.cpp)** Escribir un programa que, dada una frase, nos devuelva la frase invirtiendo el orden de las palabras. Entenderemos por palabra el conjunto de caracteres separados por un espacio.

Por ejemplo, si la frase original fuese: "Esto podria ser un ejemplo, como ya os dije."

La frase invertida sería: "dije. Os ya como ejemplo, un ser podria Esto"

14. **(cuenta.cpp)** Hacer un programa que lea una cadena desde el teclado y obtenga el número de vocales y consonantes que tiene. Haced una *funcion_vocales* que se le pase una cadena y devuelva un entero con el número de vocales. Implementa una función similar que devuelva el número de consonantes. El programa constara de un *menú* con las siguientes opciones y se ejecutará hasta que el usuario introduzca la opción de *Acabar*.

1. Leer cadena
2. Mostrar numero de vocales
3. Mostrar numero de consonates
4. Acabar

15. **(encriptado.cpp)** Queremos descifrar un mensaje que está encriptado. Nos dicen que en el mensaje se han remplazado la vocal 'a' por la 'e', la vocal 'e' por la 'i', la vocal 'i' por la 'o', la vocal 'o' por la 'u', y la vocal 'u' por la 'a'. Queremos hacer un programa que deshaga la encriptación, es decir, lea una cadena de entrada encriptada desde el teclado y muestre la original. Por ejemplo ante la entrada: *"hule, lu hes cunsigaodu"* debe mostrar *"hola, lo has conseguido"*. Haced una *funcion_desencripta* que tenga como entrada la cadena encriptada y devuelva la cadena desencriptada. El programa principal leerá una cadena, llamará a la función y mostrará la nueva cadena.

16. (Op2) **(ruido.cpp)** 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".

17. **(fecha.cpp)** Escribir un programa que convierta una fecha en formato "MMDDYYYY" al formato "DD de mes del YYYY".

Por ejemplo, para "12072006" debería devolver "7 de diciembre del 2006".

REGISTROS

18. **(P4)** **(complejos.cpp)** Realizar un programa en C/C++ que lea números complejos y realice operaciones sobre ellos: suma, resta, multiplicación y división. Los números complejos serán guardados en registros. Antes de aparecer el menú se pedirán dos números complejos, y tras cada operación se mostrará el resultado de la misma.

```
#include <iostream>
using namespace std;
struct complejo
{
   int re, im;
};
/* Definicion de prototipos */
void leer       ( struct complejo & );
void mostrar    ( struct complejo );
void suma       ( struct complejo, struct complejo, struct complejo & );
void multiplica ( struct complejo, struct complejo, struct complejo & );
/* Implementacion de las funciones */
...
int main ( void )
{
   int opcion;
   struct complejo a, b, res;

   cout << "Este programa realiza operaciones con complejos." ;
   cout << "Dame el primer complejo: " ;
   leer ( a );
   cout << "Dame el segundo complejo: " ;
   leer ( b );

   do
   {
      cout << " 1. Sumar \n" ;
      cout << " 2. Multiplicar \n" ;
      cout << " 0. Salir \n" ;

      cin >> opcion ;
      switch ( opcion )
      {
         case 1: sumar ( a, b, res );
                 mostrar ( res );
                 break;
         ...
      }
   }
   while ( opcion != 0 );
   return 0;
}
```

19. **(fraccion.cpp)** 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);

20. **(punto.cpp)** Un *punto* es un elemento compuesto por tres coordenadas *x*, *y* y *z*. Realizar un programa que pida dos puntos, los guarde en dos variables de tipo *punto* y calcule y muestre, sin menús, la suma y la distancia entre las dos variables.

21. **(particula.cpp)** Una *partícula* es un elemento compuesto por tres coordenadas *x*, *y* y *z*, y una masa *m*. Realizar un programa que pida dos partículas, las guarde en dos variables de tipo *partícula* y calcule y muestre, sin menús, la distancia entre las dos variables, y la masa total y el centro de masas del sistema formado por ellas.

22. (triangulos2D.cpp) 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;
Area = ½ * ((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)

23. (Op3) (mecanico.cpp) Crea una estructura de datos que almacene información acerca de componentes mecánicos de automóvil. En el programa debe ser capaces de almacenar un máximo de 10 piezas.

Cada ficha de un componente debe llevar la siguiente información:

a. Nombre de la pieza
b. unidades disponibles
c. precio de la pieza

Haz un programa que, mediante un menú, realice las siguientes acciones:

- Introducir una nueva pieza, tras comprobar que queda espacio.
- Eliminar una pieza corriendo una posición todas las posteriores. Debemos llevar un contador para saber el número de piezas almacenadas.
- Buscar por el nombre y listar por pantalla las características de una pieza dada.

Cada opción del menú debe ser realizada como un subprograma diferente pasando los parámetros que consideres necesarios.

24. (Op4) (imágenes.cpp) Vamos a crear un pequeño programa que nos "dibuje" imágenes binarias en la pantalla. Una imagen binaria es una matriz de nxm valores donde cada uno puede estar a 1 o a 0. Si el valor está a 1 esto indica que el pixel de la imagen es de color negro, si está a 0 es que el pixel es de color blanco.

Crea un tipo de registro adecuado para almacenar una imagen de, como máximo 80 x 24 pixeles. Aparte de los pixeles tendrás que almacenar en tamaño de la imagen en la matriz, que puede ser menor que 80 x 24 y también el nombre de la imagen.

Haz un procedimiento que pasándole un registro, grabe en él la imagen de una recta de –45 grados (esto es equivalente a poner '1' en la diagonal principal de una imagen cuadrada (misma dimensión de filas que de columnas). Además debe pedir el nombre y el resto de parámetros. Experimenta otros dibujos si quieres.

Haz un procedimiento que reciba un registro de este tipo y dibuje en la consola de texto, la imagen. Para ello bastará que si hay un 1 en la matriz, se escriba el carácter '*' y si hay un 0, se escriba el carácter ' '. Eso se puede hacer con un bucle "similar" a este:

```
for(i=0; i< MAXFILAS;i++)
{
   for(j=0;j< MAXCOLUMNAS;j++)
        if(<el valor es 1>)
            cout << '*';
        else
           cout << ' ';
   cout << endl;
}
```

25. **(P5)** **(ciudades.cpp)** Se dispone de un vector con información relativa a un conjunto de ciudades. Cada elemento del vector contiene datos sobre: nombre de la ciudad, número de habitantes y un código de la provincia a la que pertenece la ciudad (valor entero).

Definir las estructuras de datos necesarias y funciones que desempeñen estas tareas:

- Almacenar los datos del vector leyendo la información desde teclado (el máximo número de ciudades es 20).
- La suma total de habitantes de todas las ciudades que pertenecen a una provincia determinada (identificada por su código).
- Dar toda la información referente a la ciudad con mayor número de habitantes. En caso de que haya dos ciudades que repitan ese valor máximo, se dará la información de la ciudad que se corresponda con el menor índice del vector.

```
include <iostream.h>
using namespace std;

#define NUM 20

struct ciudad
{
      //definir campos
 };

/* Definicion de prototipos */
void leer ( ciudad &c );
void rellena_vector (ciudad v[NUM], int &tam );
int calcular_habit (ciudad v[NUM], int tam, int cod);
ciudad maximo_habitantes ((ciudad v[NUM], int tam);
int main ( void );

/* Implementacion de las funciones */
…..
```

26. **(futbol.cpp)** Hacer un programa para almacenar la información de una tabla de clasificaciones de un equipo de fútbol de 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 podrá mostrar la información completa sobre la clasificación.

27. **(calendario.cpp)** Crear un programa que implemente un calendario del 2011. 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 2011 y mostrar por pantalla esta semana. Implementad una función que devuelva el día de la semana de un día seleccionado.

28. (alumnos.cpp) Realizar un programa que lea los nombres y las notas de los 10 alumnos de una clase, calcule la media, y determine cuántos alumnos superan y cuántos están por debajo de la misma. La estructura del programa ya modularizado (dividido adecuadamente en subprogramas) es la siguiente:

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

#define ALUMNOS 10

struct alumno
{
    string nombre;
    float nota;
};

/* Definicion de prototipos */

void leer_notas ( alumno [ALUMNOS] );
float nota_media (alumno [ALUMNOS]);
void sup_inf_media (alumno [ALUMNOS], float, float &, float &, float & );
int main ( void );

/* Implementacion de las funciones */

void leer_notas (alumno datos[ALUMNOS] ))
{
   ...
}

float nota_media (alumno datos[ALUMNOS])
{
   ...
}
void sup_inf_media (alumno datos[ALUMNOS], float m, float &sup_m, float &inf_m )
{
   ...
}
int main ( void )
{
   alumno notas[ALUMNOS];
   float med, sup_med, inf_med;

   leer_notas ( notas );

   med = nota_media ( notas);
   sup_inf_media ( notas, med, sup_med, inf_med );

   cout << "media " << med << " Sup " << sup_med << " Inf " << inf_med << endl;

   system("pause");
   return 0;
}
```