Fundamentos de Programación
Relación de ejercicios 4 (funciones)

1. Encuentre los errores de las siguientes funciones:

```
int Suma5 (int entero) {
   if (0 == entero)
      return 0;
   else
      entero = entero + 5;
}

void Imprime(double valor) {
   double valor;
   cout << valor;
}

void Cuadrado (int entero) {
   return entero*entero;
}

bool EsPositivo(int valor) {
   if (valor > 0)
      return true;
}
```

   Finalidad: Familiarizarnos con la definición de funciones, el paso de parámetros y el ámbito de las variables. Dificultad Baja.

2. Determinar la salida por pantalla de los siguientes programas:

   a. Programa A:

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

void P(int i, int j) {

   i= i+3;
   j= j+3;
   cout << i << " " << j << endl;
}

int main() {
   int a,b;

   a= 2;
   b= 7;
   P(a,b);
   cout << a << " " << b << endl;
   return 0;
}
```

b. Programa B:

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

void P(int & i, int & j) {

   i= i+3;
   j= j+3;
   cout << i << " " << j << endl;
}

int main() {
   int a,b;

   a= 2;
   b= 7;
   P(a,b);
   cout << a << " " << b << endl;
   return 0;
}
```

Finalidad: Familiarizarnos con la definición de funciones, el paso de parámetros y el ámbito de las variables. Dificultad Baja.

3. Implementar en C++ un módulo para aceptar/cancelar mensajes de confirmación. El módulo deberá mostrar por pantalla el mensaje *¿Confirmar (S/N)?*, y únicamente aceptará las pulsaciones de las teclas S y N (mayúsculas o minúsculas). El módulo devolverá false si se pulsó la N o true si se pulsó S.

4. Implementar un módulo en C++ que tenga como entrada un caracter.

   a. Si el argumento es una letra mayúscula, deberá devolver su correspondiente minúscula.

   b. Si el argumento NO es una letra mayúscula, devolverá el mismo caracter.

5. Escribir en C++ un módulo `int MCD(int a, int b)`, que devuelva el Máximo Común Divisor de a y b.

6. Escribir en C++ un módulo `int MCM(int a, int b)`, que devuelva el Mínimo Común Múltiplo de a y b.

7. Escribir un módulo para saber, dado un año, mes y dia de entrada, si la fecha indicada es válida o no.

8. Escribir un módulo que tenga como entrada una fecha y devuelva como salida la fecha del dia siguiente.

9. Escribir un módulo que tenga como entrada una fecha y devuelva como salida la fecha del dia anterior.

10. Indicar y justificar qué escriben en pantalla los programas siguientes para las entrada 1, 3 y 7:

    a. Programa A:

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

void sumar(int &x, int &a, int &z, int sum) {

  cin>>x;
  cin>>a;
  cin>>z;
  sum= x+a+z;
}

main() {
  int a,b,c, sum= 0;

  sumar(a, b, c, sum);
  cout << a << "+" << b << "+" << c << "= " << sum << endl;
}
```

b. Programa B:

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

void sumar(int &x, int &a, int &z, int &sum) {

  cin>>x;
  cin>>a;
  cin>>z;
  sum= x+a+z;
}

main() {
  int a,b,c, sum= 0;

  sumar(a, b, c, sum);
  cout << a << "+" << b << "+" << c << "= " << sum << endl;
}
```

11. Dos números a y b se dice que son amigos si la suma de los divisores de a (salvo él mismo) coincide con b y viceversa. Implementa una función que determine si dos números naturales son amigos. Diseña un programa que tenga como entrada dos números naturales y que muestre en la pantalla todas las parejas de números amigos que existan en dicho intervalo.

12. Escribe un módulo que lea 2 números enteros positivos y un carácter. En función de este carácter leído efectuará las siguientes operaciones:

    - 'm': calcula el mínimo común múltiplo de ambos números.
    - 'd' : calcula el máximo común divisor de los mismos.
    - '+', '*', '-', '/' : realiza la operación correspondiente con los números.

13. Escribe una función que toma un valor entero y devuelve el entero con sus dígitos en orden inverso. Por ejemplo, si se le pasa el número 7631, la función devuelve 1367.

14. Escribe una función que muestre por pantalla todos los números perfectos (ver la relación 2 para la definición de número perfecto) hasta un número dado.

15. Escribe una función que lea una cadena, busque un carácter en ella y devuelva si se encuentra o no.

16. Implementar una función para calcular la moda de un vector de caracteres, es decir, el carácter que más veces se repite. Por ejemplo, si el vector fuese

    ```
    {'l','o','s',' ','d','o','s',' ','c','o','f','r','e','s'}
    ```

    los caracteres que más se repiten son 'o' y 's' con un total de 3 apariciones. La moda sería cualquiera de ellos, por ejemplo, el primero encontrado 'o'. El método devolverá un `struct` del tipo:

    ```
    struct FrecuenciaCaracter{
       char caracter;
       int frecuencia;
    }
    ```

    en el que el campo `caracter` contendrá el carácter en cuestión ('o') y en el campo `frecuencia` el conteo de la moda (3).

    Dificultad Media.

17. Implementar una función que, a partir de un vector (array) de enteros, obtenga dos vectores (arrays) que contienen, respectivamente, los números pares e impares del primer vector. Escribir un programa para probar la función.

    Finalidad: Salida de un vector por una función. Dificultad: Media.

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

    ```
    struct TipoPunto{
       double abscisa;
       double ordenada;
    };
    struct TipoPoligono{
       TipoPunto puntos[100];
       int num_puntos;
    };
    ```

    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])`.

    Implementar una función

    ```
    double AreaPoligono(TipoPoligono poligono)
    ```

    que calcule el área de un polígono convexo. Para ello, primero hay que implementar una función

    ```
    double AreaTriangulo( TipoPunto punto1,
             TipoPunto punto2, TipoPunto punto3)
    ```

que devuelve el área del triángulo formado por los puntos `punto1`, `punto2` y `punto3`. Se usará la siguiente fórmula:

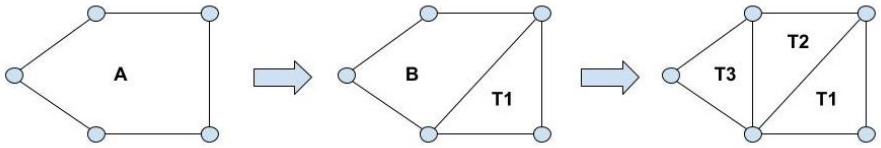
$$Area = \sqrt{F(F-S1)(F-S2)(F-S3)}$$

siendo $F = (S1 + S2 + S3)/2$, donde $S1$, $S2$ y $S3$ son las longitudes de los lados del triángulo.

Si un polígono A tiene n lados con $n > 3$, podemos calcular su área de la siguiente manera:

- Considerar tres vértices v_1, v_2, v_3 consecutivos del polígono A.
- Calcular el área del triángulo T_1 que forman esos lados.
- Formar un polígono B de $n-1$ lados eliminando el vértice v_2 (el intermedio) de A.
- El área de A es la suma de las áreas de T_1 y B.
- Si B no es un triángulo, aplicamos el proceso a B.

Repitiendo el proceso $n-2$ veces, el área de A es la suma de las áreas de $n-2$ triángulos.



De esta manera reducimos el cálculo del área de cualquier polígono al cálculo del área de varios triángulos.

19. El Instituto de Investigaciones Biológicas Avanzadas de Güejar Sierra dispone de una base de datos de ADN de diversas especies biológicas del Parque Nacional de Sierra Nevada codificadas como secuencias de caracteres (A, T, G y C) de distinta longitud según la especie. Esta base de datos se compone de 1000 registros donde cada registro contiene el nombre de la especie y la secuencia de su ADN, ambas representadas como cadenas de caracteres (tipo `string`). Ejemplos de la información asociada con distintas especies son:
Nombre: "La mosca del Vinagre", ADN: "ATAATGGACAAT"
Nombre: "La lombriz de tierra", ADN: "GGATACT"
Nombre: "La ameba verde", ADN: "AGAGAT"
El instituto necesita un programa de ordenador que les permita identificar a qué especie pertenece una toma de ADN que se ha recogido en el campo y analizado en el laboratorio. Teniendo en cuenta que las secuencias de ADN analizadas pueden comenzar en cualquier posición de la secuencia. Por ejemplo, las secuencias "GGATACT" y "ACTGGAT" pertenecen a la misma especie ("La lombriz de tierra"). Observad que la secuencia es la misma, aunque comienza en un punto distinto. Se pide:

    - Diseñar las estructuras de datos (tipos de datos, estructuras, clases, vectores, ect.) necesarios para almacenar en memoria principal esta base de datos.
    - Diseñar una función que, a partir de la base de datos y de una secuencia de ADN específica nos indique si está registrada en la base de datos y, en caso afirmativo, cuál es el nombre de la especie.

20. **Examen práctico 2 - Curso 18/19** Un cuadrado mágico es una matriz cuadrada con un número impar de filas y columnas, cuyas filas, columnas y diagonales principales suman el

mismo valor. Por ejemplo, las siguientes matrices son cuadrados mágicos

$$\begin{matrix} 6 & 1 & 8 \\ 7 & 5 & 3 \\ 2 & 9 & 4 \end{matrix} \qquad \begin{matrix} 11 & 24 & 7 & 20 & 3 \\ 4 & 12 & 25 & 8 & 16 \\ 17 & 5 & 13 & 21 & 9 \\ 10 & 18 & 1 & 14 & 22 \\ 23 & 6 & 19 & 2 & 15 \end{matrix}$$

Como puede verse, los números en cada fila, cada columna y cada diagonal principal suman 15 y 65, respectivamente. Implementar un módulo o una clase (lo que se prefiera) para determinar si una matriz cuadrada de números enteros positivos es un cuadrado mágico. Implementar un programa principal para probar la función o la clase. El programa principal NO se evalúa.

21. **Examen práctico 2 - Curso 18/19** Implementar un método que determine si dos palabras son iguales según el siguiente criterio: La primera letra de ambas palabras es la misma, la última letra de ambas palabras también es la misma, y el resto de letras son las mismas pero no necesariamente en las mismas posiciones. Por ejemplo, *Pepito* es "igual" que *Pipteo*. Implementar un programa principal para probar la función. El programa principal NO se evalúa.

22. Realizar un programa que, dado un vector de enteros positivos V y dos enteros positivos t y m, determine el subvector de V de tamaño t cuyo mayor valor sea m y que cumpla las siguientes propiedades:

    - No tiene elementos repetidos.
    - Contiene los valores $\{m, m-1, \ldots, m-t+1\}$
    - No se impone ningún orden en el subvector

    El programa debe devolver la posición de V donde comienza el subvector. Si no existe ninguno, el programa mostrará un mensaje informando sobre esta circunstancia. Si existiera más de un subvector se devolverá la información del primero (el que tiene menor posición inicial).

    Ejemplos de salida del programa:

t	m	V	Salida
3	7	17637564259	4
5	7	17637564259	3
3	6	17637564259	5
3	5	17637564259	No hay
4	6	17637564259	No hay

23. Implementar una función `Siguiente` que, dado un entero positivo n, devuelva el entero positivo m más pequeño que verifica que $m > n$ y que m y n tienen las mismas cifras. Si no existe tal número, devuelve n. Por ejemplo, `Siguiente(19)=91`, `Siguiente(3542)=4235`, `Siguiente(5432)=5432`, `Siguiente(58943)=59348`.

24. Un primo truncable a izquierda es un número primo que no contiene dígitos 0 y, cuando el primer dígito se elimina sucesivamente, el resultado siempre es primo. Un primo truncable a derecha es un número primo que no contiene 0 dígitos y, cuando el último dígito se elimina sucesivamente, el resultado siempre es primo. Implementar una función que tome un entero como argumento y:

    - Si el entero es un primo truncable a izquierda, devuelve "izquierda".
    - Si el entero es un primo truncable a la derecha, devuelve "derecha".

- Si el entero es ambos, devuelve ”ambos”.
- De lo contrario, devuelva ”ninguno”.

Por ejemplo,

```
truncable(9137)->"izquierda" // 9137, 137, 37 y 7 son primos.
truncable(5939)->"derecha" // 5939, 593, 59 y 5 son primos.
truncable(317)->"ambos" // 317, 17, 7; y 317, 31, 3 son primos.
truncable(5)->"ambos" // sólo tiene un dígito y es primo.
truncable(139)->"ninguno" // 1 y 9 no son primos
truncable(103)->"ninguno" // contiene un dígito 0
```

25. La secuencia de Ulam comienza con $\{1, 2\}$. El siguiente número en la secuencia es el número positivo más pequeño que es igual a la suma de 2 números distintos (que ya están en la secuencia) exactamente de una manera. Trivialmente, el siguiente es 3, ya que solo hay 2 números en la secuencia inicial. Luego tenemos $\{1, 2, 3\}$. El siguiente número es 4, que es la suma de 3 + 1. 4 también es 2 + 2, pero esta ecuación no cuenta, ya que los 2 sumandos tienen que ser distintos. Luego la sucesión de Ulam empieza por $\{1, 2, 3, 4\}$. El siguiente número no puede ser 5, ya que 5 = 1 + 4, y también 5 = 2 + 3. El siguiente número es 6 (2 + 4). Hay 2 formas de hacer 7 (1 + 6 ó 3 + 4), por lo que la siguiente es 8 (2 + 6). Y así, obtenemos los primeros términos

$$\{1, 2, 3, 4, 6, 8, 11, 13, 16, 18, 26, 28, 36, 38, 47, 48, 53, ...\}$$

Crear una función que tome un número n y devuelva el n-ésimo número en la secuencia de Ulam. Por ejemplo, `ulam (4)-> 4`, `ulam (9)->16`, `ulam (206)->1856`.

26. Implementar una función que devuelva `true` si una cadena consta de más de dos números ascendentes y consecutivos. Por ejemplo,

    ascending(”232425”)-¿ true

    ascending(”2324256”)-¿ false // No matter how this string is divided, the numbers are not consecutive.

    ascending(”444445”)-¿ true // Consecutive numbers 444 and 445.

27. Implementar una función que, para cada palabra formada por los símbolos `(` y `)`, devuelva `true` si los paréntesis están bien anidados, o `false` en caso contrario. Hacer el mismo ejercicio si se admiten diferentes tipos de paréntesis, como por ejemplo `(`, `)`, `[`, `]`, `{`, `}`.

28. Implementar una función que, dada una cadena de dígitos, devuelva la subcadena más larga con dígitos alternos pares/impares o impares/pares. Si dos o más subcadenas tienen la misma longitud, devuelve la subcadena que aparece primero. Por ejemplo,

```
longestSubstring("225424272163254474441338664823")-> "272163254"
// substrings = 254, 272163254, 474, 41, 38, 23

longestSubstring("594127169973391692147228678476")-> "16921472"
// substrings = 94127, 169, 16921472, 678, 476

longestSubstring("721449827599186159274227324466")-> "7214"
// substrings = 7214, 498, 27, 18, 61, 9274, 27, 32
// 7214 and 9274 have same length, but 7214 occurs first.
```

29. Implementar una función que, dada una cadena de caracteres, devuelva la subcadena más larga sin repeticiones. Si existen varias, la primera ocurrencia.