**FACULTAD DE INGENIERÍA Y CIENCIAS EXACTAS**
**Ingeniería en Informática**
**Programación I**

# Práctica Adicional 1: Funciones y Punteros

**Ejercicio 1:** Marque la respuesta correcta:

i. Los punteros se declaran en la misma forma que cualquier otra variable, pero en su declaración deben llevar el operador

   a) *
   b) &
   c) %

ii. Los tiempos de ejecución al usar punteros son mejores porque

   a) los datos son manipulados a través de su dirección
   b) los datos son manipulados a través de sus valores
   c) ninguno de los anteriores

iii. El operador de indirección (*) es usado para

   a) declaración e indirección
   b) declaración
   c) indirección
   d) Ninguno de los anteriores

**Ejercicio 2:** *// Dadas las siguientes declaraciones…*

```
int p=2,*k;
int *c=&p;
k=c;
```

*// …podría afirmarse que*

a) Los punteros c y k están apuntando a p
b) El puntero k apunta a p y a c
c) Sólo el puntero c apunta a p

**Ejercicio 3:** Escribir una función que reciba como parámetro un número real y lo redondee a una determinada cantidad de decimales, recibida también como parámetro de tipo entero. Observe que debe redondear y no truncar, y no deben usarse cadenas de caracteres.

**Ejercicio 4:** ¿Qué se imprime por pantalla cuando se ejecutan los siguientes fragmentos de código?

a)
```
int v1, v2, *a, *b;
v1 = 19;
v2 = 5;
a = &v1;
b = &v2;
a = b;
*b = 7;
printf("a: %d  b: %d  v1:%d  v2:%d\n", *a, *b, v1, v2 );
```

b)
```
int v1, v2, *a, *b;
v1 = 19;
v2 = 5;
a = &v1;
b = &v2;
*a = *b;
*b = 7;
printf("a: %d  b: %d  v1:%d  v2:%d\n", *a, *b, v1, v2 );
```

**Ejercicio 5:** Dados dos parámetros de tipo entero, calcular el valor de la multiplicación de los mismos usando sólo sumas.

**Ejercicio 6:** Dados dos parámetros A y B de tipo entero, devolver $A^B$ utilizando la función realizada en el ejercicio anterior.

**Ejercicio 7:** Dado un parámetro de tipo entero verificar si el número es par o impar, devolviendo 1 (VERDADERO) o 0 (FALSO).

**Ejercicio 8:** Dado un parámetro de tipo entero, devolver el resultado de calcular su factorial. Ejemplo: fact(4) = 4*3*2*1 = 24.

# Práctica Adicional 2: Arreglos

**Ejercicio 1:** Leer por teclado una serie de números enteros -de no más de 50 elementos- y guardarlos en un arreglo, finalizando la lectura al ingresar un valor -1 o cuando se hayan ingresado los 50 valores. Luego escribir una función para calcular la suma de los elementos del vector. Los parámetros que recibe la función son el arreglo y la cantidad de elementos que contiene.

**Ejercicio 2:** Realizar una función para determinar si un arreglo de números enteros es capicúa. La cantidad de elementos está declarada mediante una directiva #define.

**Ejercicio 3:** Escribir una función para invertir un arreglo.

**Ejercicio 4:** Cargar desde el teclado una matriz de enteros de M x N (declarados mediante directivas #define) e imprimirla por pantalla respetando el formato de la matriz. La impresión deberá realizarse exclusivamente mediante el uso de punteros.

**Ejercicio 5:** Dada una matriz de enteros de dimensión M x N (declarados mediante directivas #define), escriba las siguientes funciones:

a. Cargue una matriz con números al azar, cuidando que ninguno se repita. Los números al azar debe situarse en el intervalo [0, M*N).
b. Sume una columna dada.
c. Sume una fila dada.
d. Calcule el promedio de la diagonal primaria.
e. Calcule el promedio de la diagonal secundaria.

# Práctica Adicional 3: Caracteres y cadenas

*En los siguientes ejercicios desarrolle la estrategia desde cero, sin utilizar funciones de biblioteca:*

**Ejercicio 1:** Desarrollar una función para eliminar los espacios en blanco ubicados al principio y al final de una cadena de caracteres. Los espacios intermedios no deben alterarse.

**Ejercicio 2:** Hacer una función que permita reemplazar todas las secuencias de más de un espacio por uno solo en una cadena de caracteres cualquiera.

**Ejercicio 3:** Separar las palabras de una cadena de caracteres, imprimiendo cada una en una línea de pantalla.

*Utilice funciones de biblioteca para codificar las siguientes funciones:*

**Ejercicio 4:** Escribir una función `strpos(c,s)` que devuelva la posición del carácter c dentro de la cadena s. Si el carácter no se encuentra deberá devolverse -1.

**Ejercicio 5:** Codificar una función que devuelva la cantidad de palabras (separadas por uno o más espacios) que tiene una frase.

**Ejercicio 6:** Desarrollar una función que permita convertir un número entero positivo cualquiera a palabras, tal como se realiza en los cheques al escribir el importe en números y en letras. Para ello se recibe como parámetros el número y una cadena donde se devolverá el resultado.

**Ejercicio 7:** Codificar una función `strsub(s,d,n)` que devuelva una subcadena de la cadena de caracteres s, indicando en d la posición inicial y en n la cantidad de caracteres deseada. Ejemplo, dada la secuencia: "El número de teléfono es 4356-7890." se desea obtener el teléfono, que está a partir de la posición 26 y tiene 9 caracteres.

## Práctica Adicional 4: Estructuras

**Ejercicio 1:** Escribir un programa que utilice una estructura para representar números racionales (compuestos por un numerador y un denominador enteros), y permita operar con los mismos a través de funciones. Las operaciones a implementar son la suma, la resta, la multiplicación y la división de números racionales.

**Ejercicio 2:** Realizar un programa que declare una estructura llamada cuadrilatero con tres campos enteros: Tipo, Lado1, Lado2. El programa deberá leer los tres datos del teclado e invocar a una función que reciba la estructura como parámetro y devuelva la superficie del cuadrilátero, de acuerdo con la siguiente tabla:

| | |
|---|---|
| 0: Cuadrado | Sup=Lado1 * Lado1 |
| 1: Rectángulo | Sup=Lado1 * Lado2 |
| 2: Rombo | Sup=(Lado1 * Lado2) / 2 (en este caso los lados corresponden a la medida de las diagonales) |
| 5: Paralelogramo | Sup=Lado1 * Lado2 |

## Práctica Adicional 5: Archivos Binarios

**Ejercicio 1:** Codificar una función `int buscarArch(char nombre[], int valor)` que reciba el nombre de un archivo de enteros y un número e indique en que posición del archivo se encuentra ese número. En caso de no encontrarlo devolver -1. El archivo no está ordenado.

**Ejercicio 2:** Desarrollar un programa para ordenar un archivo binario de enteros utilizando el método de intercambio o burbujeo. Recuerde que no se permite cargar el archivo completo en memoria.

**Ejercicio 3:** Repetir el ejercicio 1 considerando que el archivo ha sido ordenado mediante el programa del ejercicio 2. Utilizar el método de búsqueda binaria.

**Ejercicio 4:** Un banco dispone de un archivo de saldos con la siguiente estructura:

```
struct ssaldo {
        int numerocliente;
        int saldo;
        int limitedecredito;
};
```

Debido a una crisis financiera se ha decidido reducir al 50% el limite de crédito autorizado para los clientes, por lo que tal vez algunos de ellos se excederán del nuevo máximo establecido. Se solicita escribir un programa que actualice el archivo de saldos con los nuevos límites de crédito permitidos, y que informe en un listado por pantalla los números de cliente que se encuentren excedidos de acuerdo a los nuevos límites. Tener en cuenta que los clientes que tienen deuda con el banco son solamente aquellos con saldo negativo.

Nota: Crear primero un archivo con la estructura indicada y generar los registros con datos obtenidos al azar.

**FACULTAD DE INGENIERÍA Y CIENCIAS EXACTAS**
**Ingeniería en Informática**
**Programación I**

## Práctica Adicional 6: Archivos de Texto

**Ejercicio 1:** Desarrollar una función `grabarNotas()` que permita grabar en un archivo de texto las notas de los parciales de cada alumno con el siguiente formato:

Nombre
Nota1
Nota2

**Ejercicio 2:** Realizar una función `grabarPromedios()` que permita leer el archivo generado en el ejercicio anterior y generar otro archivo de texto con los promedios de los alumnos con el siguiente formato:

Nombre
Promedio

**Ejercicio 3:** Escribir una función `mostrarMejores()` que lea el archivo del ejercicio anterior y permita mostrar los alumnos cuyo promedio supere al promedio general del curso.

**Ejercicio 4:** A través de la función system() (declarada en stdlib.h) es posible ejecutar comandos del sistema operativo o incluso cualquier programa del equipo. El único parámetro que lleva es una cadena de caracteres con el programa a ejecutar, por ejemplo system("notepad"); o system("cls"); También pueden aprovecharse los comandos de redirección del sistema operativo (">") para dirigir a un archivo de texto la salida que normalmente iría a la pantalla:

system("dir > lista.txt"); *// lista el directorio actual en un archivo de texto llamado lista.txt*

Utilizando esta técnica se solicita resolver los siguientes problemas:

a. Imprimir por pantalla los nombres de los diez archivos de mayor tamaño existentes en cualquier lugar del disco. (utilizar `dir c:\ /s > lista.txt`)

b. Imprimir por pantalla los nombres de aquellos archivos que tengan extensión .TXT ubicados en cualquier lugar del disco. (utilizar `dir c:\*.txt /s > lista.txt`)

c. Imprimir por pantalla los nombres de las carpetas existentes en el directorio raíz del disco. (utilizar `dir c:\ /ad /s > lista.txt`)

Puede utilizar "dir /h" desde una ventana de símbolo del sistema para obtener ayuda sobre los parámetros soportados por el comando dir.

## Práctica Adicional 7: Punteros y Listas

**Ejercicio 1:** Escribir una función que borre de una lista simple el primer nodo cuyo campo valor coincida con un número pasado como parámetro. Devolver un valor de verdad indicando si el borrado tuvo éxito o si el nodo no existía.

**Ejercicio 2:** Escribir una función que borre de una lista simple un nodo cuyo número de orden se pasa como parámetro. Devolver un valor de verdad indicando si el borrado tuvo éxito o si el nodo no existía.

**Ejercicio 3:** Se dice que una lista es circular cuando el puntero a siguiente del último nodo apunta al primer nodo de la misma. Escribir un programa que permita ingresar números enteros desde el teclado y los almacene en una lista circular utilizando inserción al final, terminando la carga con -1. Luego imprimir la lista por pantalla y finalmente eliminar todos sus nodos.

**Ejercicio 4:** Normalmente las listas enlazadas deben ser recorridas secuencialmente a fin de buscar datos. Esto puede provocar grandes demoras en listas que contengan una cantidad considerable de nodos. Para paliar este problema existen las *listas indexadas*, que son listas enlazadas simples que se relacionan con otra lista enlazada que hace las veces de índice.

Por ejemplo, una lista ordenada de clientes podría tener una lista índice de 26 elementos, uno para cada letra del alfabeto. Cada nodo de la lista índice cuenta no sólo con un puntero al siguiente nodo de la misma sino también con un puntero adicional que indica a partir de qué nodo de la lista principal aparecen los apellidos con esa letra. Las estructuras de los nodos podrían tener el siguiente aspecto:

```
typedef struct snodoprincipal {
    char nombre[40];
    char domicilio[50];
    double saldo;
    struct snodoprincipal *ant,*sig;
} tnodoprincipal;
```

```
typedef struct snodoindex {
    char letra:
    struct snodoprincipal *comienzo;
    struct snodoindex *sig;
} tnodoindex;
```

Escribir un programa que permita insertar nodos en orden en una lista indexada con las características descritas precedentemente, y que pueda realizar búsquedas aprovechando el índice. Implementar también una función para eliminar nodos de la lista principal de acuerdo al nombre del cliente. Utilizar un archivo binario adecuado para cargar la lista y para resguardar su contenido luego de las operaciones realizadas sobre la misma.

**Nota:** El habitual puntero a la cabeza de la lista principal aquí no se utiliza, ya que es reemplazado por un puntero al primer nodo de la lista índice.