EJERCICIO 1: GESTIÓN DE EMPLEADOS

Se desea crear un sistema básico para controlar empleados.

1. Defina un struct llamado Empleado con los siguientes campos:
    - int legajo;
    - char nombre[30];
    - float sueldo;
2. En el main:
    - Declare un arreglo de 5 empleados.
    - Cargue los datos manualmente (puede estar hardcodeado si lo desea).
3. Implemente una función void mostrarEmpleados(struct Empleado[], int); que imprima todos los empleados en formato:
    - 1001 - Juan - $45600.00
4. Implemente una función void buscarPorLegajo(struct Empleado[], int, int); que reciba un legajo y muestre los datos del empleado si existe. Si no lo encuentra, mostrar mensaje adecuado.
5. Implemente una función void aumentarSueldos(struct Empleado[], int, float porcentaje); que aumente el sueldo de todos los empleados según el porcentaje recibido (por ejemplo, 10.0 representa un aumento del 10%).

Ejercicio 2: Sistema de Contactos

Enunciado

Desarrollar un programa que maneje una agenda de contactos con las siguientes características:

Estructura requerida:

```
typedef struct {
    char nombre[50];
    char telefono[15];
    char email[50];
    int categoria;  // 1=Familia, 2=Trabajo, 3=Amigos
} Contacto;
```

Funciones a implementar:

1. void cargarContacto(Contacto *c)
    - Pide al usuario ingresar todos los datos del contacto
    - **Pista:** validar que la categoría esté entre 1-3
2. void mostrarContacto(Contacto c)
    - Muestra la información completa de un contacto
    - **Pista:** Usar un switch para mostrar la categoría como texto ("Familia", "Trabajo", "Amigos")
3. void buscarPorCategoria(Contacto agenda[], int cantidad, int cat)
    - Muestra todos los contactos de una categoría específica

- **Pista:** Recorrer el array y comparar el campo categoria

4. void ordenarPorNombre(Contacto agenda[], int cantidad)
    - Ordena los contactos alfabéticamente por nombre
    - **Pista:** Usar algoritmo burbuja con strcmp() para comparar nombres
    - **Pista:** Para intercambiar structs completos: temp = agenda[i]; agenda[i] = agenda[j]; agenda[j] = temp;
5. int contarPorCategoria(Contacto agenda[], int cantidad, int cat)
    - Retorna cuántos contactos hay de una categoría específica

Programa principal:

- Crear un array de máximo 10 contactos
- Cargar entre 3-5 contactos
- Mostrar menú con opciones:
    1. Mostrar todos los contactos
    2. Buscar por categoría
    3. Ordenar y mostrar por nombre
    4. Contar contactos por categoría
    5. Salir

Ejemplo de ejecución:

```
=== AGENDA DE CONTACTOS ===
Ingrese datos del contacto 1:
Nombre: Juan Pérez
Teléfono: 11-2345-6789
Email: juan@email.com
Categoría (1=Familia, 2=Trabajo, 3=Amigos): 1

--- MENÚ ---
1. Mostrar todos
2. Buscar por categoría
3. Ordenar por nombre
4. Contar por categoría
5. Salir
Opción: 2

Ingrese categoría (1=Familia, 2=Trabajo, 3=Amigos): 1
=== CONTACTOS DE FAMILIA ===
Juan Pérez - Tel: 11-2345-6789 - Email: juan@email.com
```

EJERCICIO 3: ANÁLISIS DE DATOS CON MATRICES (matrices + sumas parciales)

Una empresa lleva un registro de ventas mensuales en 3 sucursales durante 4 meses. Cada fila representa una sucursal y cada columna un mes.

1. Declare y cargue una matriz int ventas[3][4] con datos fijos o ingresados por teclado.
2. Calcule y muestre:
    - a) El total vendido por cada sucursal (suma por filas).
    - b) El total vendido por cada mes (suma por columnas).
    - c) La sucursal que más vendió en total.

- d) El promedio total de ventas.

3. Use funciones donde sea conveniente, por ejemplo:
    - void mostrarTotalesPorSucursal(int[3][4]);
    - void mostrarTotalesPorMes(int[3][4]);
    - int sucursalMaxVentas(int[3][4]);
    - float promedioVentas(int[3][4]);

EJERCICIO 4: VERDULERÍA

Desarrollar un programa en lenguaje **C** que simule el sistema de compras de una verdulería. El usuario podrá consultar productos, comprar hasta **4 artículos distintos** y obtener un **ticket final** detallado.

La verdulería vende **8 productos**, cada uno identificado por una letra (carácter) y un precio por kilo:

Producto	Letra	Precio por kilo ($)
Banana	b	120.50
Manzana	m	98.30
Pera	p	75.00
Naranja	n	85.00
Tomate	t	110.00
Lechuga	l	45.50
Papa	a	60.00
Zanahoria	z	50.00

El programa debe mostrar un menú principal con dos opciones:

1. Agregar producto

2. Terminar compra / Salir

Al seleccionar “1. Agregar producto”:

- Se deben mostrar los 8 productos con sus letras y precios.
- El usuario debe ingresar:
    - La letra del producto que desea comprar.
    - La cantidad de kilos que desea comprar.
- Esta operación se puede repetir (volver a elegir la opción 1) hasta que se hayan comprado **4 productos diferentes como máximo**.
- Si se intenta agregar un **quinto producto**, se debe mostrar un mensaje indicando que se alcanzó el límite, y continuar con la opción 2 automáticamente.
- **No se debe permitir comprar dos veces el mismo producto.**

Al seleccionar “2. Terminar compra / Salir”:

- El programa debe imprimir un **ticket de compra** usando una **matriz de 4 filas x 3 columnas** (float ticket[4][3]), donde cada fila representa un producto comprado y las columnas representan:
    1. Cantidad de kilos
    2. Precio por kilo
    3. Subtotal (kilos × precio)
- Además, se debe usar un arreglo adicional para almacenar las letras de los productos comprados (char productos_ticket[4]).
- El ticket debe mostrarse así (ejemplo):

Producto | Kilos | Subtotal

----

b | 2.00 | $241.00

t | 1.50 | $165.00

TOTAL A PAGAR: $406.00

Si el usuario no compró nada, debe mostrarse un mensaje:
"No se realizaron compras."

Validaciones obligatorias

- Si el usuario ingresa una letra de producto inválida, debe mostrarse un mensaje de error y volver a pedir un producto válido.
- Si el usuario intenta comprar el mismo producto más de una vez, también debe notificarse.
- El programa debe permitir volver a la opción 1 cuantas veces quiera el usuario (hasta el límite de 4 productos).

```
void mostrarMenu();                          // Muestra el menú principal
void mostrarProductos(char[], float[], int);       // Muestra productos y precios
int buscarIndiceProducto(char[], int, char);       // Busca índice del producto ingresado
int productoYaAgregado(char[], int, char);         // Verifica si ya se compró el producto
void mostrarTicket(char[], float[][3], int);       // Imprime el ticket de compra
```