Laboratorio de Lenguajes Imperativos, repasando C

- **Objetivo**:

  Reforzar mediante una serie de ejercicios prácticos y un ejercicio final el uso de punteros, punteros y arreglos, punteros a funciones, retorno de punteros, estructuras y typedef en C

1. Repaso y ejercicios guiados

Realizar los siguientes ejercicios para entender cada concepto antes de avanzar a la siguiente parte.

1.2 Punteros en C

- Un puntero, es una variable que contiene la dirección de memoria, de un dato o de otra variable que contiene al dato. (El puntero apunta al espacio físico donde está el dato de la variable).
- Se usa para acceso eficiente a memoria y manipulación directa de datos.

1.3 Punteros básicos

```c
#include <stdio.h>

int main() {
    int a = 10;
    int *ptr = &a;  // Puntero apuntando a 'a'

    printf("Valor de a: %d\n", a);
    printf("Dirección de a: %p\n", &a);
    printf("Valor de ptr (direccion de a): %p\n", ptr);
    printf("Valor apuntado por ptr: %d\n", *ptr);

    return 0;
}
```

Ejercicio 1: Escribir un programa en C que use punteros para intercambiar los valores de dos variables enteras. Implementar una función intercambio () que tome dos punteros como parámetros y realice el intercambio. Imprimir los valores antes y después del intercambio.

2.1 Punteros y arreglos

- Un arreglo en C es una colección de datos almacenados en memoria contigua.
- El nombre de un arreglo es un puntero al primer elemento.

2.3 Recorriendo un arreglo con punteros

```
#include <stdio.h>

int main() {
    int arr[] = {10, 20, 30, 40, 50};
    int *ptr = arr; // Apunta al primer elemento del arreglo

    for (int i = 0; i < 5; i++) {
        printf("Elemento %d: %d\n", i, *(ptr + i));
    }

    return 0;
}
```

Ejercicio 2: Implementar una función invertirArreglo(int *arr, int tamaño) que reciba un puntero a un arreglo y su tamaño. Mostrar el arreglo original y el el arreglo invertido.

1. Funciones a funciones

- Un puntero a función almacena la dirección de una función y permite llamarla indirectamente.
- Se usa para callbacks y programación modular.

Ejemplo: Puntero a Función

```
#include <stdio.h>

void mensaje() {
    printf("Hola desde un puntero a función!\n");
}

int main() {
    void (*ptrFuncion)() = mensaje;
    ptrFuncion();

    return 0;
}
```

Ejercicio 3: Crear funciones que permitan realizar las siguientes operaciones (suma, resta, división (retorna un float), multiplicación) y usar un puntero a estas funciones para llamarla dinámicamente. Las operaciones se harán para solo dos valores. Usar el puntero a las funciones para definir un menú y dependiendo de la operación seleccionada ejecutar la operación indicada.

2. Retorno de punteros

- Se puede retornar un puntero desde una función para devolver estructuras de datos dinámicas.
- Se debe tener cuidado con el uso de memoria dinámica (malloc).

4.1 Retornar un puntero desde una función

```c
#include <stdio.h>
#include <stdlib.h>

int* crearNumero() {
    int *num = (int*) malloc(sizeof(int));
    *num = 100;
    return num;
}

int main() {
    int *ptr = crearNumero();
    printf("Numero creado: %d\n", *ptr);
    free(ptr);
    return 0;
}
```

Ejercicio 4: Implementar una función invertirArreglo(int *arr, int tamaño) que reciba un puntero a un arreglo y su tamaño, y retorne un puntero con un nuevo arreglo invertidoPunteros.

5 Eructuras y typedef

- Una estructura (**struct**) es una agrupación de variables bajo un mismo nombre.
- typedef permite crear alias de tipos.

5.1 Ejemplo: Uso de struct **y** typedef

```c
#include <stdio.h>

typedef struct {
    char nombre[50];
    int edad;
} Persona;

void imprimirPersona(Persona *p) {
    printf("Nombre: %s, Edad: %d\n", p->nombre, p->edad);
}

int main() {
    Persona p1 = {"Juan", 25};
    imprimirPersona(&p1);

    return 0;
}
```

Capsula: La función strcpy() en C se usa para copiar una cadena de caracteres de una variable a otra. Es parte de la librería estándar <string.h>.

char *strcpy(char *destino, const char *origen);

- destino: Puntero al arreglo donde se copiará la cadena.
- origen: Puntero a la cadena de origen que se copiará.
- Retorna un puntero al destino (destino), donde se copió la cadena.

```c
#include <stdio.h>
#include <string.h>
int main() {
    char origen[] = "Hola, mundo!";
    char destino[50];
    strcpy(destino, origen);
    printf("Cadena original: %s\n", origen);
    printf("Cadena copiada: %s\n", destino);
    return 0;
}
```

Ejercicio 5: Crear una estructura Estudiante con nombre, edad y promedio. Usar Typedef en la definición de la estructura y definir las siguientes funciones:

a. imprimir los datos del Estudiante
b. Modificar los valores del estudiante usando punteros.

6. Ejercicio final

Objetivo:

Implementar un programa en C que maneje un sistema de gestión de estudiantes, usando punteros, punteros a funciones, estructuras y memoria dinámica.

5.2 Requisitos:

1. Definir una estructura estudiante con nombre, edad e identificación.
2. Almacenar los estudiantes en un arreglo de máximo 10 estudiantes.
3. Mostrar información de todos los estudiantes.
4. Buscar un estudiante por identificación.
5. Modificar los valores de un estudiante por identificación
6. Implementar un menú interactivo usando punteros a funciones.
7. Incluir una opción para eliminar estudiantes

5.3 Ejemplo de Código Base:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
    char nombre[50];
    int edad;
    float promedio;
} Estudiante;

Estudiante* crearEstudiante(char *nombre, int edad, float promedio) {
    Estudiante *nuevo = (Estudiante*) malloc(sizeof(Estudiante));
    strcpy(nuevo->nombre, nombre);
    nuevo->edad = edad;
    nuevo->promedio = promedio;
    return nuevo;
}

void mostrarEstudiante(Estudiante *e) {
    printf("Nombre: %s, Edad: %d, Promedio: %.2f\n", e->nombre, e->edad, e->promedio);
}

int main() {
    Estudiante *e1 = crearEstudiante("Carlos", 22, 4.5);

    mostrarEstudiante(e1);

    free(e1); // Liberar memoria
    return 0;
}
```