📘 FUNCIONES EN C – Transcripción Ordenada

🧠 Definición de Función

Una **función** es un subprograma que realiza una tarea determinada y acotada. Se le pasan datos (**parámetros**) y devuelve datos (valor de retorno). Solo se ejecuta cuando se la llama.

✅ Ventajas de Usar Funciones

- Simplifican la comprensión del programa.
- Permiten aislar errores y depurar módulos.
- Facilitan modificaciones futuras.
- Reutilización sin necesidad de revisar nuevamente.
- Independencia del código por módulos.

📦 Modularización en Programación

- Divide problemas complejos en partes más pequeñas.
- Estas partes se llaman **módulos**.
- Cada módulo debe resolver **una única tarea** → **cohesión alta**.
- Si un módulo tiene más de un verbo activo, debería dividirse.

🔗 Acoplamiento

- Mide cuán relacionado está un módulo con otros.
- **Bajo acoplamiento = mejor diseño.**
- Se logra reduciendo interacciones y efectos colaterales.

🔁 Funcionamiento de main

- Toda aplicación en C debe tener una función main().
- El programa comienza ejecutando main.
- main puede llamar a otras funciones.
- Las definiciones deben ser independientes entre sí.

🎯 Argumentos y return

- **Argumentos (o parámetros)**: datos que recibe una función.
- **Return**: devuelve un único valor.
- Ejemplo:

```
return [expresión];
```

🧰 Tipos de Funciones

- **Predefinidas**: vienen con el lenguaje. Ej: pow().
- **Definidas por el usuario**: se crean cuando no hay funciones predefinidas que resuelvan el problema.

🔧 Procedimientos

- En C se los implementa como funciones tipo void.
- No devuelven valor y pueden no tener return.

🕳️ Variables

- **Locales**: definidas dentro de funciones. Solo existen allí.
- **Globales**: definidas fuera de funciones. Visibles en todo el programa.

📤 Pasaje de Parámetros

- **Por valor**: se copia el valor.
- **Por referencia**: se pasa la dirección (punteros).

📦 Estructura de una Función

```
tipo_de_dato nombre_funcion(tipo param1, tipo param2) {
  // instrucciones
  return valor;
}
```

🔠 Ejemplos de Código

📄 **Sin Función**

```
#include <stdio.h>
#include <conio.h>

int main() {
  int x, y, z;
  printf("Ingrese número: ");
  scanf("%d", &x);
  printf("Ingrese número: ");
  scanf("%d", &y);
  z = x + y;
  printf("La suma es %d", z);
  getch();
  return 0;
}
```

📄 **Con Función**

```
#include <stdio.h>
#include <conio.h>

int suma(int a, int b);

int main() {
  int x, y, z;
  printf("Ingrese número: ");
  scanf("%d", &x);
  printf("Ingrese número: ");
  scanf("%d", &y);
  z = suma(x, y);
  printf("La suma es %d", z);
  getch();
```

```
  return 0;
}

int suma(int a, int b) {
  int total = a + b;
  return total;
}
```

Función con parámetros por referencia

```
int suma(int *a, int *b) {
  int total = *a + *b;
  return total;
}
```

Procedimiento con void

```
void imprimeValor() {
  int contador = 5;
  printf("El valor de contador es: %d\n", contador);
}
```

Función que suma un arreglo

```
#define max 9

int sumoarreglo(int a[], int n) {
  int suma = 0;
  for (int i = 0; i < n; i++) suma += a[i];
  return suma;
}
```

📄 **Función promedio**

```
float promedio(int a[], int n) {
  int suma = 0;
  float prom = 0;
  for (int i = 0; i < n; i++) suma += a[i];
  prom = suma / n;
  return prom;
}
```

📄 **Resumen de "Registro.pdf"**

Tema principal: Uso de estructuras (registros) en C para manejar datos compuestos.

✅ **Conceptos Clave:**

- **Estructura (struct):** Agrupa variables de distintos tipos bajo un solo nombre.
- **Campo o miembro:** Cada variable dentro de una estructura.
- **Arreglos de estructuras:** Permiten manejar múltiples registros (ej. varios amigos).

🛠️ **Operaciones comunes con estructuras:**

- Cargar, recorrer, buscar, acceder, insertar, eliminar y ordenar elementos dentro de arreglos de estructuras.

🧪 **Ejemplos de código:**

1. **Estructura básica de un amigo:**

c

CopiarEditar

```
struct estructura_amigo {
  char nombre[30];
  char apellido[40];
  char telefono[10];
  int edad;
};
```

2. **Cargar y mostrar datos de un amigo con scanf.**
3. **Arreglo de estructuras para varios amigos:**
    - Usando arrays y bucles para almacenar y mostrar múltiples registros.
4. **Calcular el promedio de edad de amigos.**
5. **Manejo de cuentas bancarias:**
    - Identifica clientes deudores y acreedores según su saldo.

📄 **Resumen de "Unidad 4-Arreglos-2025-3.pdf"**

Tema principal: Métodos de ordenamiento de arreglos.

✅ **Clasificación:**

- **Ordenamiento Interno:** Datos en memoria principal.
- **Ordenamiento Externo:** Datos en almacenamiento secundario.

🔁 **Tipos de Algoritmos:**

1. **Intercambio:**
    - Compara elementos dos a dos e intercambia si es necesario.
    - Ejemplos: BubbleSort, QuickSort.
2. **Inserción:**
    - Inserta cada elemento en su lugar correspondiente.
    - Ejemplos: InsertionSort, ShellSort.
3. **Selección:**
    - Busca el menor/mayor y lo coloca en su sitio.
    - Ejemplo: Selección Directa.
4. **Enumeración:**
    - Cuenta cuántos elementos son menores para determinar la posición final.
5. **Árbol, MergeSort, RadixSort:** Otros métodos internos.
6. **Métodos Externos:**
    - Natural merging, Polyphase sort, etc.