Resumen de Conceptos Fundamentales de Programacion en C (8 a 10 paginas)

---

1. Modularizacion en C

La modularizacion consiste en dividir un programa grande en partes mas pequenas y organizadas llamadas modulos. Cada modulo se compone generalmente de:

- **.h (header)**: contiene declaraciones de funciones y estructuras.
- **.c (implementacion)**: contiene la logica de las funciones.
- **main.c**: funcion principal que coordina los modulos.

Ventajas:

- Facilita la lectura y el mantenimiento del codigo.
- Permite la reutilizacion.
- Mejora el trabajo en equipo.

Ejemplo:

```c
// operaciones.h
int sumar(int, int);

// operaciones.c
int sumar(int a, int b) {
    return a + b;
}

// main.c
#include "operaciones.h"
int main() {
    printf("%d", sumar(2, 3));
    return 0;
}
```

---

2. Listas Enlazadas

Son estructuras dinamicas que permiten almacenar elementos de forma lineal, a diferencia de los arrays, su tamanio puede variar en tiempo de ejecucion.

Tipos:

- Lista simplemente enlazada
- Lista doblemente enlazada
- Lista circular

Nodo de lista simple:

```c
struct Nodo {
    int dato;
    struct Nodo* siguiente;
};
```

Operaciones comunes:

- Insertar al inicio/final
- Eliminar nodo
- Buscar valor
- Mostrar lista

Ejemplo de insercion:

```c
void insertarInicio(struct Nodo** cabeza, int valor) {
    struct Nodo* nuevo = malloc(sizeof(struct Nodo));
    nuevo->dato = valor;
    nuevo->siguiente = *cabeza;
    *cabeza = nuevo;
}
```

---

3. Archivos en C

Permiten guardar datos permanentemente en el disco, a diferencia de los datos en memoria que se pierden al cerrar el programa.

Tipos de archivos:

- Texto: datos en formato legible.
- Binarios: datos en formato nativo (estructuras, enteros, etc).

Operaciones:

1. Abrir: `fopen()`
2. Leer: `fread()`, `fscanf()`, `fgets()`
3. Escribir: `fwrite()`, `fprintf()`, `fputs()`
4. Cerrar: `fclose()`

Ejemplo:

```c
FILE *f = fopen("datos.txt", "w");
if (f != NULL) {
    fprintf(f, "Hola mundo\n");
    fclose(f);
}
```

Estructura en archivo binario:

```
struct Persona {
    char nombre[20];
    int edad;
};

fwrite(&persona, sizeof(struct Persona), 1, archivo);
```

Lectura:

```
while (!feof(archivo)) {
    fread(&persona, sizeof(persona), 1, archivo);
    // Verificar cantidad leida
}
```

Funcion fread completa:

```
FILE *f = fopen("personas.dat", "rb");
struct Persona p;
while (fread(&p, sizeof(p), 1, f) == 1) {
    printf("%s tiene %d anios\n", p.nombre, p.edad);
}
fclose(f);
```

Funcion fwrite con multiples estructuras:

```
FILE *f = fopen("personas.dat", "wb");
struct Persona personas[2] = {{"Ana", 25}, {"Luis", 30}};
fwrite(personas, sizeof(struct Persona), 2, f);
fclose(f);
```

---

4. Arboles Binarios

Estructura jerarquica con nodos que tienen un hijo izquierdo y uno derecho.

Ejemplo de nodo:

```
struct Nodo {
    int dato;
    struct Nodo* izq;
    struct Nodo* der;
};
```

Operaciones basicas:

- Insertar nodo
- Recorridos: inorden, preorden, postorden
- Buscar elemento

Utilidad:

- Representar jerarquias
- Eficiencia en busqueda y ordenamiento

---

5. Punteros

Son variables que almacenan direcciones de memoria.

Sintaxis:

```
int *p;
int x = 10;
p = &x;
```

Operadores:

- `&` devuelve la direccion
- `*` accede al contenido

Usos comunes:

- Paso por referencia
- Manejo dinamico de memoria
- Estructuras complejas como listas

Ejemplo con malloc:

```
int *numeros = (int*) malloc(5 * sizeof(int));
if (numeros != NULL) {
    for (int i = 0; i < 5; i++) {
        numeros[i] = i * 2;
    }
    free(numeros);
}
```

Paso por referencia:

```c
void cuadrado(int *n) {
    *n = (*n) * (*n);
}
```

---

6. Recursividad

Funcion que se llama a si misma. Ideal para problemas que pueden dividirse en versiones mas pequenas del mismo.

Ejemplo: Factorial

```c
int factorial(int n) {
    if (n == 0) return 1;
    return n * factorial(n - 1);
}
```

Ejemplo: Serie de Fibonacci

```c
int fibonacci(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

Busqueda en arbol binario recursiva:

```c
int buscar(struct Nodo* raiz, int valor) {
    if (raiz == NULL) return 0;
    if (raiz->dato == valor) return 1;
    if (valor < raiz->dato)
        return buscar(raiz->izq, valor);
    else
        return buscar(raiz->der, valor);
}
```

---

7. Funciones y Prototipos

Permiten dividir el programa en bloques reutilizables.

Prototipo:

```c
int sumar(int, int);
```

Definicion:

```c
int sumar(int a, int b) {
    return a + b;
}
```

---

8. Manejo de errores en archivos

Verificar fopen:

```c
if ((f = fopen("archivo.txt", "r")) == NULL) {
    printf("Error al abrir archivo\n");
}
```

Cerrar archivo:

```c
if (fclose(f) != 0) {
    printf("Error al cerrar archivo\n");
}
```

Uso de feof(): Detecta fin de archivo.

```c
while (!feof(f)) {
    fgets(buffer, 100, f);
}
```

---

9. Comandos avanzados en archivos

- `fseek()` : mueve el puntero dentro del archivo
- `rewind()` : lleva el puntero al inicio
- `ftell()` : obtiene la posicion actual
- `remove()` : elimina un archivo

Ejemplo:

```c
fseek(f, 0, SEEK_END);
long tam = ftell(f);
rewind(f);
```

Ejemplo de lectura y edicion:

```
fseek(f, -sizeof(struct Persona), SEEK_CUR);
fwrite(&p_modificada, sizeof(struct Persona), 1, f);
```

---

10. Buenas practicas

- Modularizar el codigo
- Usar nombres significativos
- Liberar memoria con `free()`
- Cerrar archivos con `fclose()`
- Verificar errores siempre

Este resumen integra todos los conceptos esenciales de la programacion en C vistos en las clases sobre listas, archivos, modularizacion y arboles, e incluye ejemplos practicos para facilitar el aprendizaje. Se ha extendido con mas contenido de punteros, archivos y recursividad para alcanzar un nivel mas completo y detallado.