Video12 (1)

Insertar al final en C

Conceptos Principales

1. a
2. b
3. c

Notas

Definición de la estructura

```
struct lista {
    int valor;
    struct lista *siguiente;
};
```

- `struct lista` → define el tipo **nodo**.
- No se reserva memoria aquí, solo se define cómo sería.
- Para reservar memoria se usa **`calloc`** (o `malloc`).
- **`sizeof()`** calcula el tamaño exacto de la estructura (varía según arquitectura).

Proceso general

1. **Validaciones**:
   - Verificar que la lista no sea `NULL` (evitar *segmentation fault*).
2. **Caso especial**: lista vacía.
   - Crear nodo con `crearNodo(valor)`.
   - Asignar `lista->inicio = nuevoNodo;`.
3. **Caso general**:
   - Recorrer lista hasta encontrar el último nodo (`while(actual != NULL)`).
   - Cuando `actual->siguiente == NULL`, insertar ahí.
4. **Retorno**:
   - `0` = éxito, `1` = error.

Código base explicado

- **Función**: `insertarAlFinal(lista, valor)`

- Parámetros:
  - `lista`: puntero a la lista.
  - `valor`: dato a insertar.
- **Pasos**:
  - Si la lista no está inicializada → error.
  - Si está vacía:
    - Crear nodo.
    - Asignar como primer elemento (`inicio`).
  - Si no está vacía:
    - Recorrer con `actual` hasta que `actual->siguiente == NULL`.
    - Crear nodo nuevo y asignarlo a `actual->siguiente`.
- **Detalle importante**:
  - **Siempre copiamos direcciones de memoria**, no movemos bloques.
  - Ejemplo: `actual->siguiente = nn;` solo copia la **dirección**.

Consideraciones de diseño

- Se usan **punteros** porque trabajamos con direcciones en memoria dinámica (heap).
- Evitar `return` múltiples (buena práctica → 1 solo return).
- Manejo de errores: retornar códigos numéricos (0 = OK, -1 = error).
- Crear funciones auxiliares como `crearNodo()` para modularizar.

Ejemplo simplificado

```
int insertarAlFinal(Lista *lista, int valor) {
    if(lista == NULL) return -1;

    Nodo *nuevo = crearNodo(valor);
    if(lista->inicio == NULL) {
        lista->inicio = nuevo;
        return 0;
    }

    Nodo *actual = lista->inicio;
    while(actual->siguiente != NULL) {
        actual = actual->siguiente;
    }
    actual->siguiente = nuevo;
    return 0;
}
```

Puntos clave

- **Heap vs Stack**: los nodos están en **heap**, pero variables locales (punteros) están en la **pila**.

- **Direcciones, no datos**: se copian punteros, no contenido.
- **Encapsulación**: el usuario solo llama la función, no sabe cómo se implementa la lista.