Video13 (1)

Manejo de Memoria con Listas en C

Conceptos Principales

1. a
2. b
3. c

Notas

Conceptos clave

- En **C**, cuando creamos nodos con `malloc`, la memoria se asigna en el **heap** (montículo).
- Cuando declaramos variables dentro de una función (ej. `struct Nodo n1`), se asignan en la **pila (stack)**.
- **Punteros en la pila** → solo almacenan **direcciones de memoria** (4 bytes típicamente).
- **Heap** almacena los datos reales (estructura completa).
- `NULL` en un puntero = dirección `0x0` (indica que no apunta a ningún lado).

Escenario del ejemplo

- Creamos **tres punteros** `n1`, `n2`, `n3` en la pila (stack).
- Cada puntero ocupa 4 bytes.
- Les asignamos memoria dinámica con `malloc(sizeof(struct Nodo))` → esto crea 3 bloques en el **heap**.
- Cada bloque tiene:
  - `valor` (ej. 27, 10, 115)
  - `siguiente` (puntero al próximo nodo)

Cómo se enlazan

- Cada `malloc` devuelve una **dirección de memoria** (ej. `0xA1`, `0xB2`, `0xC3`).
- Cuando hacemos:

```
n1->siguiente = n2;
n2->siguiente = n3;
```

→ **NO copiamos el bloque de memoria**, **solo copiamos la dirección** del siguiente nodo.

- Esto permite crear la estructura de lista enlazada:

```
n1 -> n2 -> n3 -> NULL
```

Asignación de valores

- Se asignan así:

```
n1->valor = 27;
n2->valor = 10;
n3->valor = 115;
```

- Después enlazamos:

```
n1->siguiente = n2;
n2->siguiente = n3;
n3->siguiente = NULL;
```

Recorrido de la lista

Ejemplo de algoritmo:

```
Nodo* actual = n1;
while (actual != NULL) {
    printf("%d\n", actual->valor);
    actual = actual->siguiente;
}
```

- **Explicación paso a paso**:
  1. `actual` apunta a la dirección de `n1` (ej. `0xA1`).
  2. Imprime el valor (`27`).
  3. Salta al siguiente: `actual = actual->siguiente;` → ahora apunta a `n2` (`0xB2`).
  4. Imprime `10`.
  5. Salta al siguiente (`n3`).
  6. Imprime `115`.
  7. Llega a `NULL` → termina.

Detalles importantes

- **Diferencia entre copiar memoria y copiar dirección**:
  - `p = *q;` → copia el contenido.
  - `p = q;` → copia la dirección (puntero).
- El **stack** tiene espacio limitado → no usarlo para listas grandes.
- Siempre que uses `malloc`, **hay que hacer `free` después** para liberar memoria.