1. Introduction to Linked Lists

1.1 What is a Linked List?

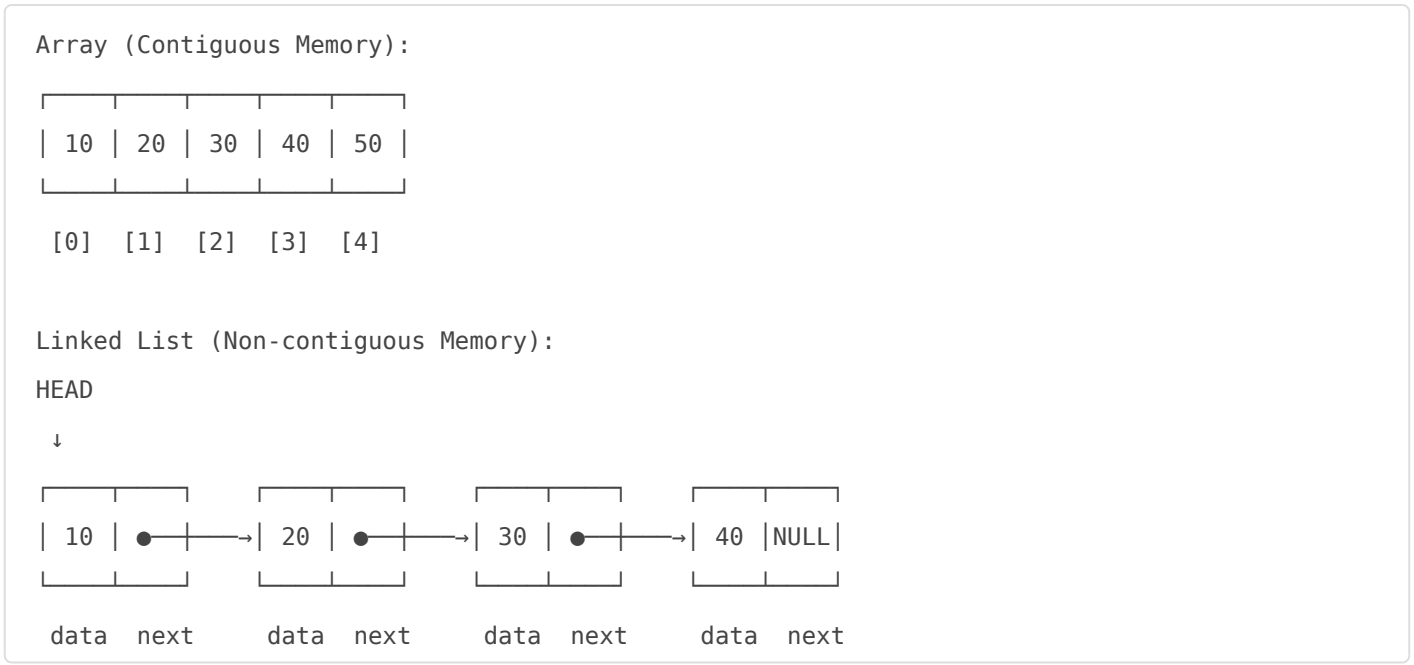
A **linked list** is a linear data structure where elements are not stored in contiguous memory locations. Instead, each element (called a **node**) contains:

1. **Data:** The actual value stored
2. **Pointer(s):** Reference to the next (and possibly previous) node

Analogy: Think of a linked list like a treasure hunt:

- Each clue (node) contains information (data)
- Each clue also tells you where to find the next clue (pointer)
- You start at the first clue (head)
- The last clue says "End of hunt" (NULL pointer)

Visual Representation:



1.2 Why Use Linked Lists?

Advantages:

1. **Dynamic Size:** Can grow or shrink at runtime
2. **Easy Insertion/Deletion:** No need to shift elements

3. **Memory Efficient:** Allocate memory only when needed
4. **Flexible Structure:** Can implement stacks, queues, graphs

Disadvantages:

1. **No Random Access:** Must traverse from beginning
2. **Extra Memory:** Requires space for pointers
3. **Sequential Access:** Slower than arrays for direct access
4. **Cache Performance:** Poor cache locality

1.3 Types of Linked Lists

1.4 Linked List vs Array

Feature	Array	Linked List
Memory Allocation	Contiguous	Non-contiguous
Size	Fixed	Dynamic
Access Time	O(1)	O(n)
Insertion (beginning)	O(n)	O(1)
Insertion (end)	O(1)	O(n) or O(1) with tail
Deletion (beginning)	O(n)	O(1)
Memory Usage	Less (no pointers)	More (pointers)
Cache Performance	Better	Worse

Feature	Array	Linked List
Random Access	Yes	No

Revision #1
Created 2025-10-27 04:59:45 UTC by DS
Updated 2025-10-27 05:00:52 UTC by DS