CBS Department

Linked Lists

Soma Soleiman Zadeh

Data Structures & Algorithms (CBS 216)

Spring 2025 - 2026

Week 4 – Week 9

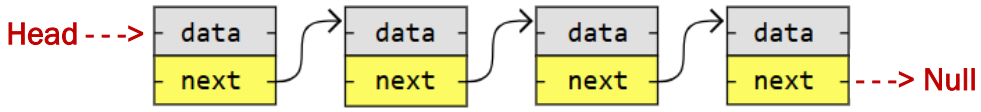
February 22, April 2, 2026

Outline

- Concept of **Linked List**
- Types of Linked Lists
  - **Single** Linked List
  - **Double** Linked List
  - **Circular** Linked List
  - **Circular Double** Linked List

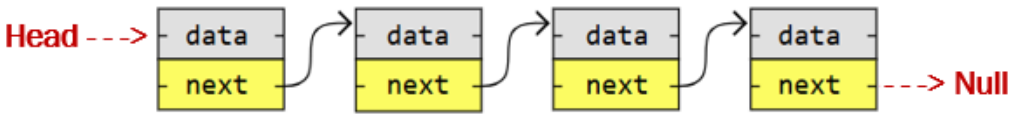
What is Linked List?

- A **linked list** is a linear data structure consisting of nodes (elements). Each node typically has two parts:
  - A **data field** and
  - a **reference (link)** to the next node in the list.
- **Linked Lists** are used to create **deques**, **trees** and **graphs**.



Head of a Linked List

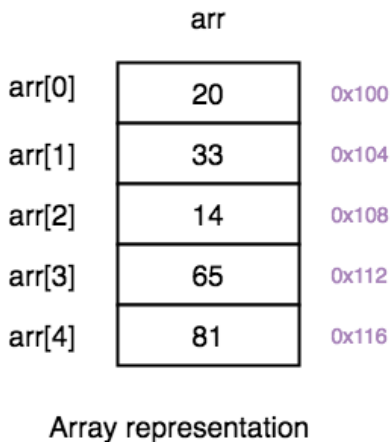
- The ***'Head'*** of the Linked List is a pointer that contains the address of the first element in the Linked List.
  - ➢ If the list is empty, then the head is a null reference.
- The last node has a reference to ***Null***.



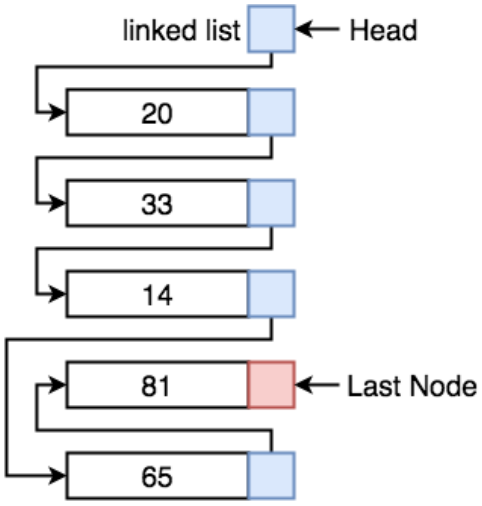
Linked List vs. Array

- Difference between **Linked List** and **Array**:
  - **Array** → Allocate memory for all elements in **one block of memory**.
  - **Linked List** → **Allocate memory for each element separately** (Each element in different memory space).

Linked List vs. Array



Array representation

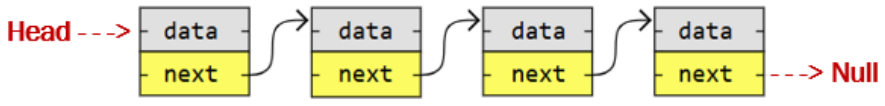


Linked List vs. Array

Feature	Array	Linked List
A ready data structure in the programming language	Yes	No
Fixed size in memory	Yes	No
Elements are stored right after each other in memory	Yes	No
Memory usage is low	Yes	No
Elements can be accessed directly	Yes	No
Elements can be inserted or deleted in constant time, no shifting operations in memory needed.	No	Yes

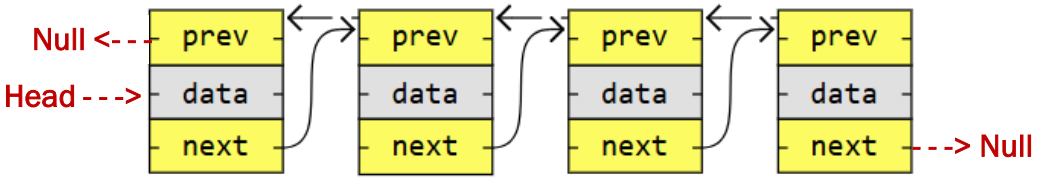
Single Linked List

- A **single linked list** is the simplest kind of linked lists.
- It takes up less space in memory because each node has only one address to the next node.
- **Head:** A pointer pointing to the first node.
- The last node will have no next node → null



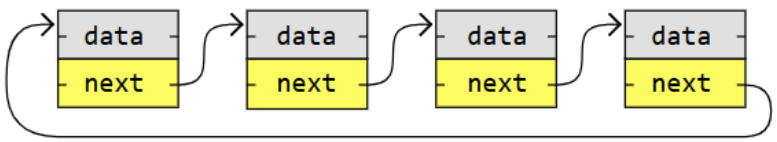
Double Linked List

- A **double linked list** has nodes with addresses to both the previous and the next node.
- **Double linked list** takes up more memory than **single linked list**.
- Double linked lists are good if you want to be able to move both forward and backward in the list.



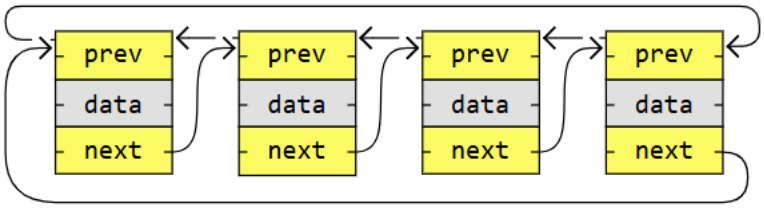
Circular Linked List

- A **circular linked list** is like a **single linked list** with the first node, and the last node connected.
  - Last node references the first node.
  - Every node has a next node.
  - No node in a circular linked list references to NULL.



Circular Double Linked List

- A **circular double linked list** is a **double linked list** with the first node, and the last node connected.
  - Last node references the first node.
  - Every node has a next node and a previous node.
  - No node in a circular linked list references to NULL.



Linked List Operations

- Create a Node and a Linked List
- **Traversal** of a Linked List
- **Search** for a Node
- **Add** a Node
- **Remove** a Node
- **Sort** a Linked List
- …

Node Class

- To create a single linked list, first we need to create the main component of a list, which is a ***node***.
- Remember that each node has two parts: ***data*** and a ***reference*** to the next node.

```
class Node:
    def __init__(self,data):
        self.data = data
        self.next = None
```

Linked List Class

- After creating the Node class, we can create the Linked List class.
- We will use nodes, link them together to make a linked list.
- Rather than the nodes, the linked list has a ***head*** that references to the first element in the linked list.

```
class linkedList:
    def __init__(self):
        self.head = None
```

- Creating an empty linked list, as the head is None. It means the head references to nothing.
- In order to be able to add/remove nodes to the linked list, we will add methods to this class.

Some Linked List Methods

- **isEmpty( )** → Returns a Boolean indicating if linked list is empty.
- **append(item)** → Inserts a node containing **item** at the end of the linked list.
- **preappend(item)** → Inserts a node containing **item** at the beginning of the linked list.
- **remove(item)** → Removes the node containing **item** from the linked list.
- **display( )** → Prints the data values of nodes in the linked list.
- **size( )** → Returns the number of nodes in the linked list.
- **search(item)** → Searches for **item** and Returns a Boolean indicating if item is found in the linked list.