Faculdade de Design,
Tecnologia e Comunicação
Universidade Europeia

Introduction to Data Structures

Data Structures

Fernando Marson

fernando.marson@universidadeeuropeia.pt

What is a data structure?

- Any ideas?

What is a data structure?

- Concepts
- Uses in programming
- Specific uses in games

Data structure - Concepts

- Data structures organize and store information efficiently.
- Allow quick access and manipulation of data.
- Fundamental for the development of efficient algorithms.
- Different types meet different storage needs.

Data structures - In game uses

- How to use data structures in game development?
- Some data structures:
  - Arrays
  - Lists
  - Stacks
  - Queues
  - Trees
  - Hash Tables
  - Graphs
  - Other types: dictionary, sortedList, set, sortedSet, …

Data structures - In game uses - Assessment

- How to use data structures in game development?
  - 20 minutes in pairs
  - Pick one game that both of you know well. You may also consider a second game — the research can include up to two games.
  - Discuss and write down at least 5 different examples of how data structures could be used in the chosen game(s).

Data structures - In game uses - Assessment

- How to use data structures in game development?
  - **Desired answer**
    - Name of the data structure (just the name).
    - Where/for what in the game(s) it would be used (one simple sentence).
    - Why this structure seems to make sense in that case, using only general concepts (1–2 short sentences).``

Data structures - In game uses - Assessment

- Some use cases:
  - Item inventory.
  - Sequential management of events.
  - Action history for undo/revert.
  - Progressive choice of skills.
  - Score table/leaderboard.
  - Map or path representation.
  - Spatial organization for collisions.
  - Fast lookup of entities by identifier.
  - Ordering renders by distance.
  - Buffer for audio/input.
  - Management/reuse of enemy instances.

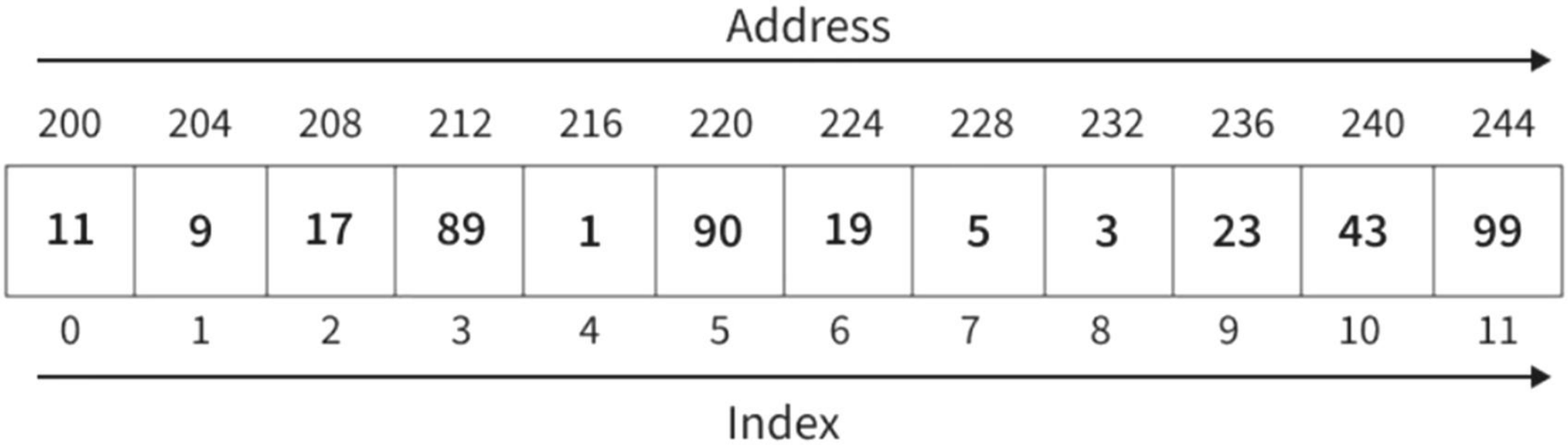
Faculdade de Design,
Tecnologia e Comunicação
Universidade Europeia

Arrays

Arrays - Concepts

- Arrays are collections of elements with a fixed size, indexed by integers.
- They store multiple values of the same data type in a single variable, allowing efficient access to elements through their indices.
- All elements in an array must be of the same type.
- It's possible to have arrays from **1** to **n** dimensions.
- An array with only one dimension also is called as a vector.

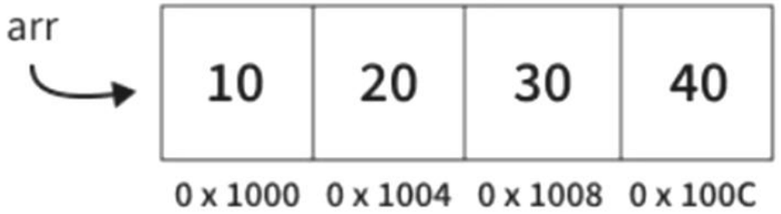
1D Arrays - Concepts



1D Arrays - Concepts

Primitive Array

int []arr = {10, 20, 30, 40}



arr [0] → 10

Primitive arrays stores the values directly in the memory

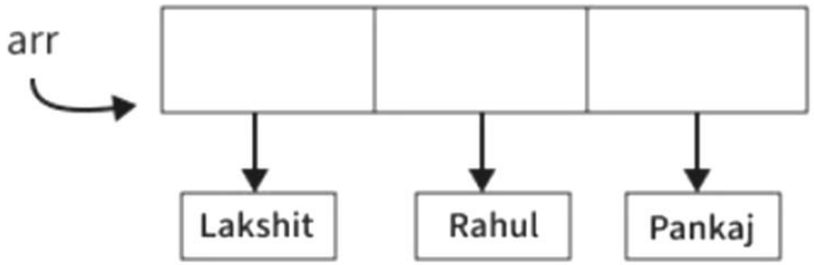
Object Array

String []arr = new String [3]

arr [0] = “Lakshit”

arr [1] = “Rahul”

arr [2] = “Pankaj”

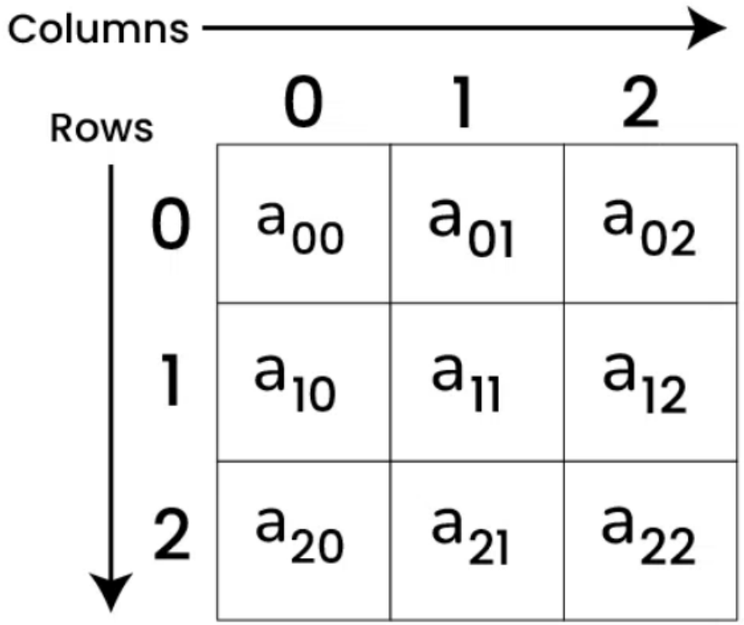


Each element of the object array stores a reference to separate string object

2D Arrays - Concepts

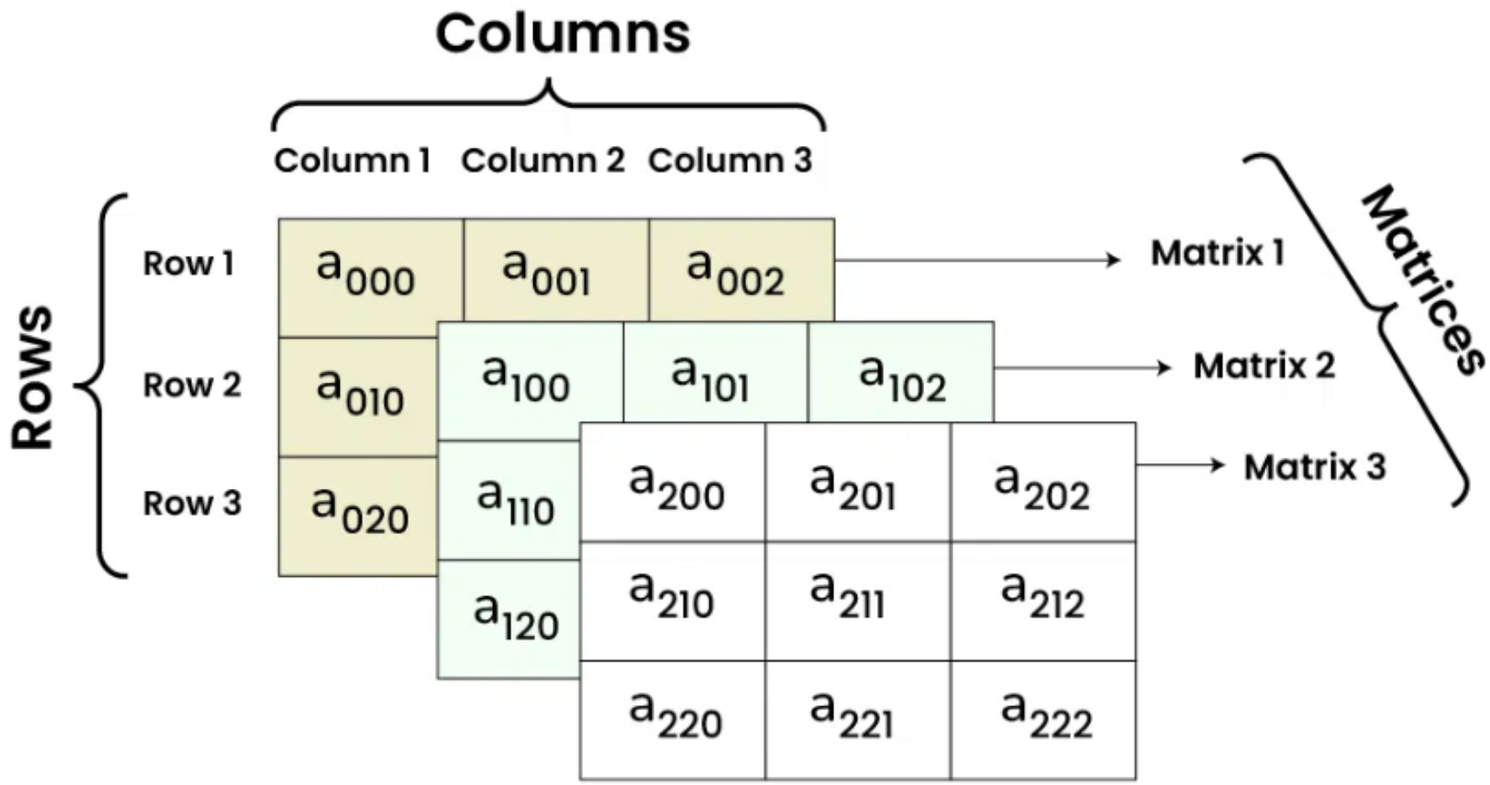
- Matrices are two-dimensional collections of elements arranged in rows and columns, indexed by two integers (row, column).
- They store multiple values of the same data type in a rectangular grid, enabling representation of tabular data and linear transformations.
- Common operations include addition, scalar multiplication, matrix multiplication, transpose, and computing determinants or inverses (when defined).
- An array with more than one dimension also is called as a matrix.

2D Arrays - Concepts



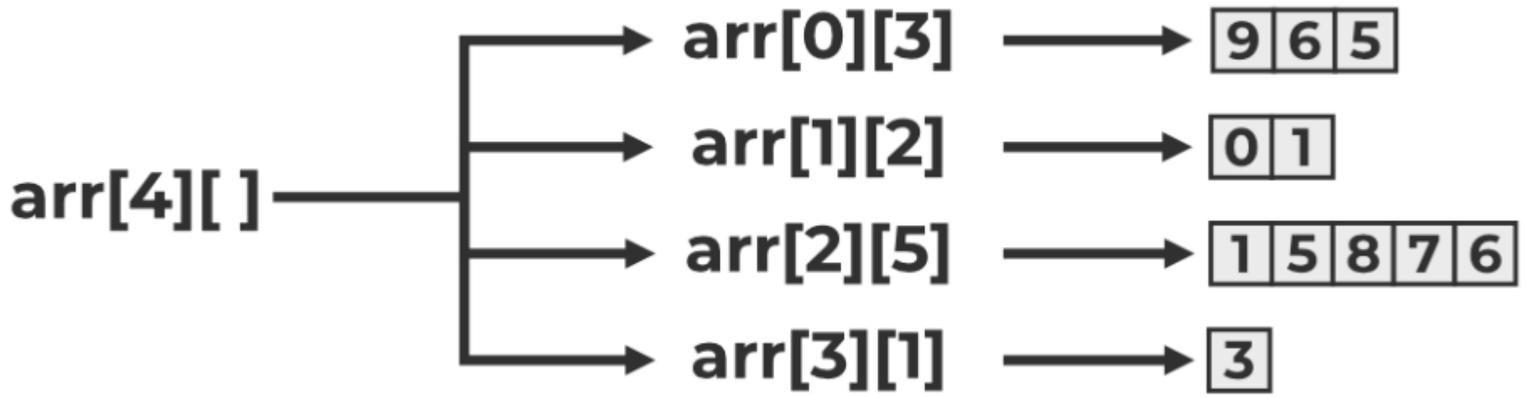
Source: https://www.geeksforgeeks.org/

3D Arrays - Concepts



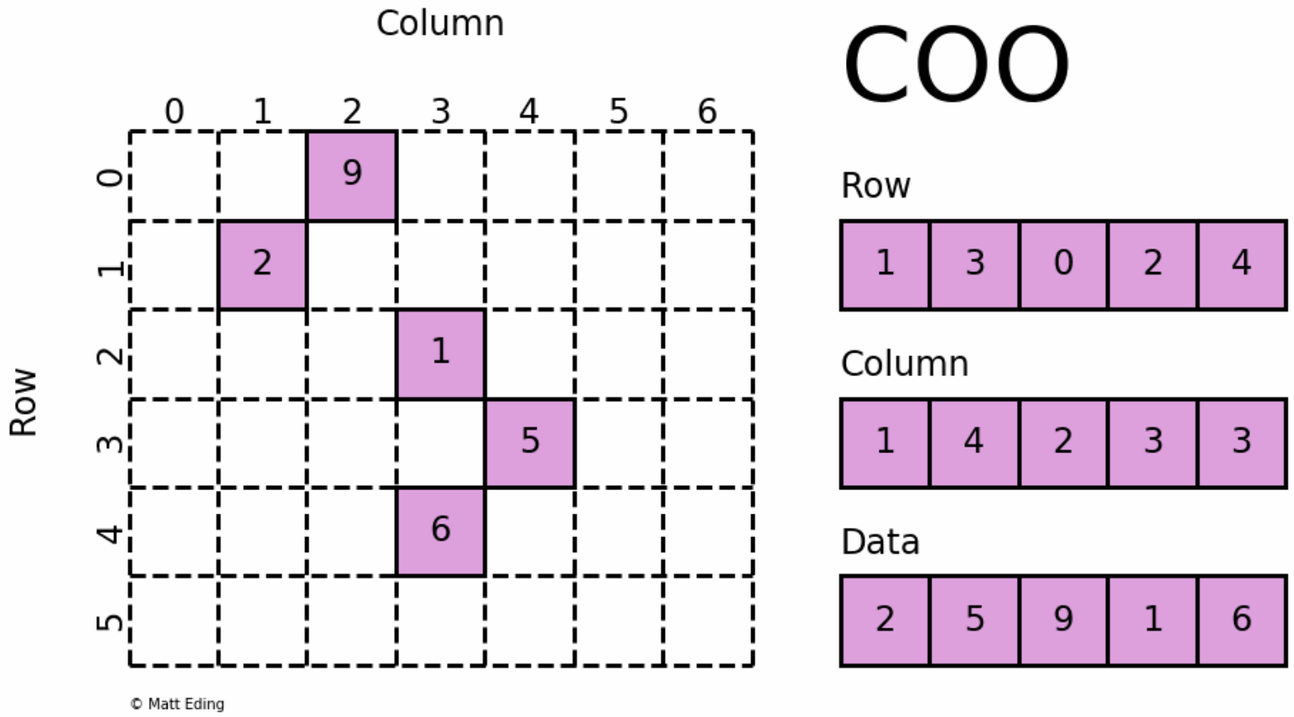
Source: https://www.geeksforgeeks.org/

Jagged Arrays - Concepts



Source: https://www.geeksforgeeks.org/

Jagged Arrays - Concepts



Source: https://matteding.github.io/

Arrays - Declaration and access

1D Array

```
int[] primes = new int[] { 2, 3, 5, 7, 11, 13 };
string[] fruits = new string[] { "apple", "banana", "cherry" };


Console.WriteLine("1D Array from Initialization:\n");
for (int i = 0; i < primes.Length; i++) {
    Console.WriteLine($"primes[{i}] = {primes[i]}");
}

for (int i = 0; i < fruits.Length; i++) {
    Console.WriteLine($"fruits[{i}] = {fruits[i]}");
}
```

Arrays - Declaration and access

1D Array

```
int size = 6;

int[] numbers = new int[size];
string[] words = new string[size];

for (int i = 0; i < size; i++) {
    Console.Write($"Enter integer for numbers[{i}]: ");
    numbers[i] = int.Parse(Console.ReadLine() ?? "1");
}

for (int i = 0; i < size; i++) {
    Console.Write($"Enter string for words[{i}]: ");
    words[i] = Console.ReadLine() ?? string.Empty;
}

for (int i = 0; i < numbers.Length; i++) {
    Console.WriteLine($"numbers[{i}] = {numbers[i]}");
}

for (int i = 0; i < words.Length; i++) {
    Console.WriteLine($"words[{i}] = {words[i]}");
}
```

Arrays - Declaration and access

2D Array

```
int rows = 4;
int cols = 6;
var rnd = new Random();

int[,] matrix = new int[rows, cols];

for (int r = 0; r < rows; r++)
{
    for (int c = 0; c < cols; c++)
    {
        matrix[r, c] = rnd.Next(0, 100);
    }
}

for (int r = 0; r < rows; r++)
{
    for (int c = 0; c < cols; c++)
    {
        Console.Write(matrix[r, c].ToString().PadLeft(4));
    }
    Console.WriteLine();
}
```

Arrays - Declaration and access

3D Array

```
int depth = 3;
int[,,] cube = new int[rows, cols, depth];
for (int r = 0; r < rows; r++)
{
    for (int c = 0; c < cols; c++)
    {
        for (int d = 0; d < depth; d++)
        {
            cube[r, c, d] = rnd.Next(0, 100);
        }
    }
}
for (int d = 0; d < depth; d++)
{
    Console.WriteLine($"Depth {d}:");
    for (int r = 0; r < rows; r++)
    {
        for (int c = 0; c < cols; c++)
        {
            Console.Write(cube[r, c, d].ToString().PadLeft(4));
        }
        Console.WriteLine();
    }
    Console.WriteLine();
}
```

Arrays - Declaration and access

Jagged Array

(irregular number of columns)

```
int jaggedRows = 4;
int[][] jaggedArray = new int[jaggedRows][];
for (int r = 0; r < jaggedRows; r++)
{
    int jaggedCols = rnd.Next(1, 7);
    jaggedArray[r] = new int[jaggedCols];
    for (int c = 0; c < jaggedCols; c++)
    {
        jaggedArray[r][c] = rnd.Next(0, 100);
    }
}

// Print jagged array
for (int r = 0; r < jaggedRows; r++)
{
    for (int c = 0; c < jaggedArray[r].Length; c++)
    {
        Console.Write(jaggedArray[r][c].ToString().PadLeft(4));
    }
    Console.WriteLine();
}
```

Lists

Concept

- Lists are dynamic collections.
- Unlike arrays, lists can grow and shrink in size as needed.
- They allow flexible manipulation of elements, with methods to add, remove, and insert items.

Lists

There are two different types:

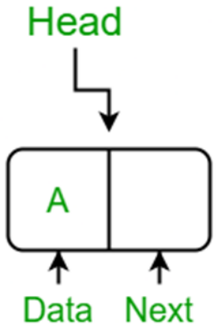
- Single linked
- Double linked

Single Linked Lists

Concept

- A singly linked list is a linear data structure consisting of nodes. Each node contains two parts: the **data** and a **reference** (or link) to the next node in the sequence.

- The list starts with a **head** node, and it ends when a node's **next** reference points to **null**.

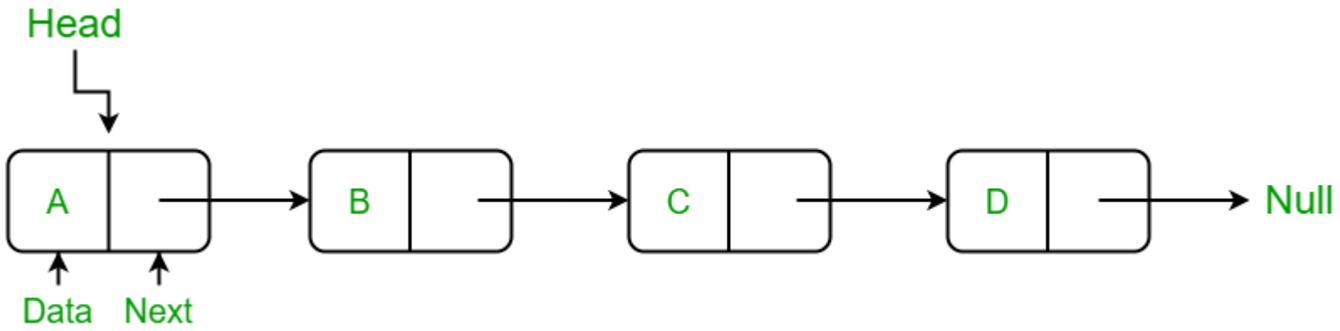


Single Linked Lists

Concept

- A singly linked list is a linear data structure consisting of nodes. Each node contains two parts: the **data** and a **reference** (or link) to the next node in the sequence.

- The list starts with a **head** node, and it ends when a node's **next** reference points to **null**.

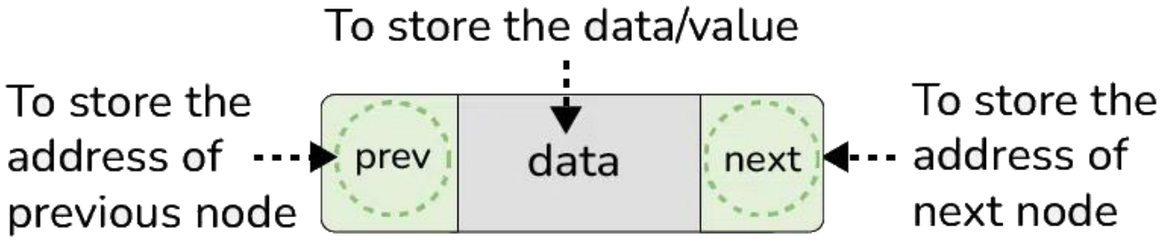


Double Linked Lists

Concept

- A doubly linked list is like a singly linked list but with an additional reference in each node, allowing traversal in both forward and backward directions.

- Each node contains three parts: the **data**, a reference to the **previous node**, and a reference to the **next node**.

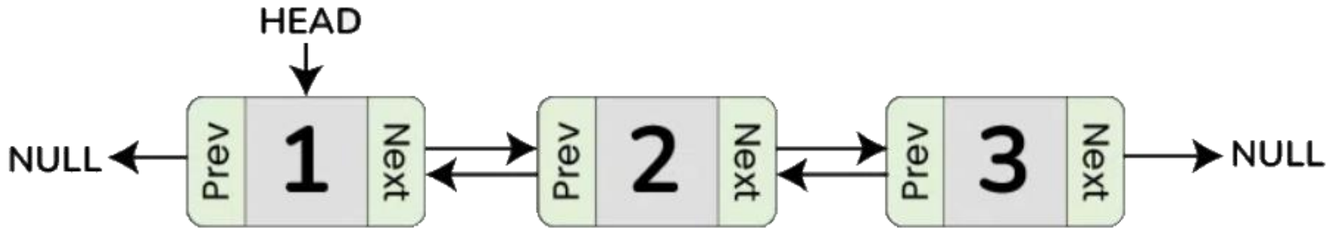


Double Linked Lists

Concept

- A doubly linked list is like a singly linked list but with an additional reference in each node, allowing traversal in both forward and backward directions.

- Each node contains three parts: the **data**, a reference to the **next node**, and a reference to the **previous node**.



Linked Lists - Operations

Operations

- Insertion
  - At the beginning
  - At the end
  - At a given position
- Deletion
  - From the beginning
  - From the end
  - From a given position
- Traversal
- Searching
- Updating (modifying the value of a node)
- Checking for list emptiness
- Counting the number of nodes
- Reversing the list

Linked Lists - Operations

Operations

- Insertion
  - At the beginning
  - At the end
  - At a given position
- Deletion
  - From the beginning
  - From the end
  - From a given position
- Traversal
- Searching
- Updating (modifying the value of a node)
- Checking for list emptiness
- Counting the number of nodes
- Reversing the list