

## 1... কোর কনসেপ্ট + রিয়েল লাইফ উদাহরণ

### **Linked List কী?**

Linked List হলো একটি **Linear Data Structure**, যেখানে ডেটা গুলো **Node** আকারে থাকে, এবং প্রতিটি Node তার পরের Node-এর address ধরে রাখে।

### **সহজ ভাষায়:**

Linked List = Data + Link (পরের Node-এর ঠিকানা)

***Here we know what is node:***

### **Node কী? (Bangla সহজ ব্যাখ্যা)**

Node হলো একটি ছোট ডাটা-ঘর (data container), যা Linked List এর মূল অংশ।

সহজভাবে বললে —

### **Node = Data + Next (পরবর্তী ঠিকানা)**

---

### **Node এর ভিতরে কী থাকে?**

একটা Node সাধারণত দুইটা জিনিস রাখে:

- 1 Data → যেকেনো তথ্য (যেমন: 10, 25, "Abir")
  - 2 Next → পরবর্তী Node কোথায় আছে তার ঠিকানা
- 

## 🎯 খুব সহজ উদাহরণ

ধরো তুমি একটা ট্রেন কল্পনা করছো 🚂

- প্রতিটা বগি (coach) = একটা Node
- বগির ভিতরের যাত্রী = Data
- বগির পেছনের ছক যা পরের বগির সাথে যুক্ত = Next

যদি শেষ বগি হয়, তাহলে তার পেছনে আর কিছু নেই →  
তাই তার next = null

---

## 💻 Java তে Node কেমন হয়?

```
class Node {  
    int data; // ডাটা রাখবে  
    Node next; // পরের node কে point করবে
```

```
Node(int data) {  
    this.data = data;  
    this.next = null;  
}  
}
```

## 🔥 এক লাইনে মনে রাখার ট্রিক

Node হলো এমন একটি বাক্স যেখানে

- 📦 ডাটা আছে
- 🔗 আর পরের বাক্সের ঠিকানা আছে

## 📌 Array vs Linked List (মূল পার্থক্য)

### ◆ 1 Array কী?

Array হলো এমন একটি data structure যেখানে সব element **continuous memory location** এ থাকে।

📦 উদাহরণ:

ধরো একটা ফ্ল্যাটের ৫টা রুম একসাথে পাশাপাশি আছে।  
সব রুম একই লাইনে, কোনো gap নেই।

Index: 0 1 2 3

Value: 10 20 30 40

- সবগুলো একসাথে memory তে থাকে
- index দিয়ে সরাসরি access করা যায়

Here we know what is pointer..

### ❖ Pointer কী?

Pointer হলো এমন একটি variable যা কোনো data এর মান না রাখে, সেই data যেখানে আছে তার ঠিকানা (address) সংরক্ষণ করে।

### 👉 সহজভাবে:

Pointer নিজে data রাখে না,  
data কোথায় আছে সেটা দেখায়।

---

## খুব সহজ উদাহরণ

ধরো তুমি বন্ধুর বাসায় যাবে।

- বন্ধুর নাম = Data
- বন্ধুর বাসার ঠিকানা = Address
- যে কাগজে তুমি ঠিকানা লিখে রাখলে = Pointer

মানে, pointer নিজে বাসা না — শুধু বাসার ঠিকানা জানে।

---

## আরেকটা রিয়েল লাইফ উদাহরণ

ধরো একটা বাড়ি আছে।

- বাড়ির ভিতরে মানুষ = Data
- বাড়ির location = Memory Address
- Google Map link = Pointer

Google Map link মানুষ না, কিন্তু মানুষ কোথায় আছে দেখায়।

## Linked List এ Pointer কিভাবে কাজ করে?

Linked List এ প্রতিটা Node এ একটা pointer থাকে:

- **class Node {**  
*int data;*  
**Node next; // এইটা pointer**  
**}**

এখানে next হলো pointer

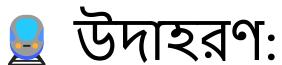
এটা পরবর্তী Node এর address রাখে।

### 🎯 এক লাইনের সংজ্ঞা (Exam এর জন্য)

Pointer হলো এমন একটি variable যা অন্য variable এর memory address সংরক্ষণ করে।

## ◆ 2 Linked List কী?

**Linked List** হলো এমন data structure যেখানে element গুলো memory তে ছড়ানো থাকে কিন্তু pointer দিয়ে যুক্ত থাকে।



উদাহরণ:

ট্রেনের বগি একটার সাথে আরেকটা যুক্ত আছে, কিন্তু আলাদা আলাদা বানানো।

***10 -> 20 -> 30 -> 40 -> null***

- প্রতিটা অংশ = Node
- Node = Data + Next Address



### Array vs Linked List (Comparison Table)

বিষয়	Array	Linked List
Memory	Continuous	Non-Continuous
Size	Fixed (আগেনির্ধারিত)	Dynamic (বাড়ানো/কমানো যায়)
Access Speed	Fast ( $O(1)$ )	Slow ( $O(n)$ )
Insert/Delete	কঠিন(shift করতে হয়)	সহজ(pointer change)
Memory Usage	কম	বেশি(extra pointer লাগে)

## সহজ উদাহরণ দিয়ে পার্থক্য

- ◆ ধরো তোমার ৫ জন বন্ধুর নাম রাখতে হবে

### Array হলে:

- . আগে থেকেই ৫ সাইজ declare করতে হবে
- . মাঝখানে নতুন বন্ধু এলে shift করতে হবে

### Linked List হলে:

- . যত খুশি নতুন Node যোগ করা যাবে
- . শুধু last node এর next change করলেই হবে

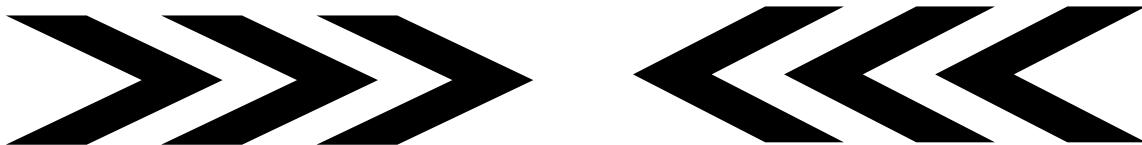
## কখন কোনটা ব্যবহার করবো?

- ◆ যদি দ্রুত index access দরকার → **Array** ব্যবহার করো
  - ◆ যদি বারবার insert/delete করতে হয় → **Linked List** ব্যবহার করো
-

## 💡 এক লাইনে পার্থক্য মনে রাখো

Array দ্রুত কিন্তু rigid

Linked List flexible কিন্তু একটু slow



Here we again summary this topic

## 📌 Array vs Linked List (মূল পার্থক্য)

### ● Array (Contiguous Memory)

- সব element একসাথে পাশাপাশি মেমোরিতে থাকে
- Index দিয়ে সরাসরি access করা যায়
- Size fixed

উদাহরণ:

- **Index: 0 1 2 3**

**Value: 10 20 30 40**

মেমোরিতে:

1000 1004 1008 1012

সব পাশাপাশি।

---

### **Linked List (Dynamic Memory)**

- Node গুলো মেমোরিতে ছড়িয়ে থাকতে পারে
- প্রতিটি Node পরের Node-এর address ধরে রাখে
- Size dynamic (যত খুশি বাড়ানো যায়)

মেমোরিতে এমন হতে পারে:

1000 → 5000 → 8000 → 2000

---

### **Real-Life উদাহরণ (খুব সহজ)**

ধরো তুমি তোমার জিমের বন্ধুদের একটা লিস্ট করছো 🤝

প্রথম জন জানে তার পরের জন কে।

Abir → Asif → Medha → null

এখানে:

- Abir = Head

- প্রতিটি ব্যক্তি জানে তার পরের ব্যক্তি কে
- শেষ জন কাউকে জানে না → null

👉 এটাই Singly Linked List



Contiguous vs Dynamic Memory (Comparison)

বিষয়	Contiguous Memory	Dynamic Memory
Memory Type	একটার পাশে একটা	ছড়ানো থাকতে পারে
Size	আগে fixed	Runtime এ পরিবর্তনযোগ্য
Example	Array	Linked List
Flexibility	কম	বেশি
Insert/Delete	কঠিন	সহজ

🎯 সহজ ভাষায় মনে রাখার ট্রিক

📌 Contiguous = পাশাপাশি বসা ছাত্ররা

📌 Dynamic = আলাদা আলাদা জায়গায় দাঁড়ানো মানুষ,  
কিন্তু ফোনে connected

## Exam এর জন্য সংজ্ঞা

### **Contiguous Memory:**

যে memory block গুলো একটার পাশে আরেকটা ধারাবাহিকভাবে থাকে তাকে Contiguous Memory বলে।

### **Dynamic Memory:**

Program execution চলাকালীন সময়ে প্রয়োজন অনুযায়ী memory allocate করাকে Dynamic Memory বলে।



## **Singly Linked List কী?**

**Singly Linked List** হলো এমন একটি Data Structure যেখানে প্রতিটা Node শুধু তার **পরবর্তী (next) Node** কে জানে।

 **সহজভাবে:**

প্রতিটা Node = Data + Next Address

আর Next শুধু সামনে দিকে ইঙ্গিত করে (এক দিকেই চলা যায়)

## ◆ Structure কেমন?

একটা Singly Linked List দেখতে এমন হয়:

- $10 \rightarrow 20 \rightarrow 30 \rightarrow 40 \rightarrow \text{null}$
- 10 হলো প্রথম Node (Head)
- 40 হলো শেষ Node
- শেষ Node এর next = null (মানে আর কেউ নেই)

## ◆ Node এর গঠন (Java Example)

- `class Node {  
 int data; // ডাটা  
 Node next; // পরবর্তী node এর ঠিকানা`
- `Node(int data) {`

```
this.data = data;  
this.next = null;  
}  
}
```

এখানে:

- data → তথ্য রাখে
- next → পরবর্তী Node কে point করে

### ◆ Real Life Example (খুব সহজ)

#### 👉 ট্রেনের উদাহরণ

- প্রতিটা বগি = একটা Node
- বগির ভিতরের যাত্রী = Data
- বগির পেছনের হুক = Next pointer

কিন্তু সমস্যা হলো:

- 👉 তুমি শুধু সামনে যেতে পারবে
- 👉 পিছনে যেতে পারবে না

এটাই **Singly Linked List**।

## ◆ Singly Linked List কিভাবে কাজ করে?

ধরো আমরা ৩টা Node বানালাম:

- *Head*

↓

**[10 | •] → [20 | •] → [30 | null]**

- ✓ Head প্রথম Node কে ধরে রাখে
  - ✓ প্রতিটা Node তার পরের Node এর ঠিকানা রাখে
- 

## ◆ কেন ব্যবহার করবো?

### ✓ সুবিধা

- ✓ Dynamic size (আগে থেকে size জানার দরকার নেই)
- ✓ Insert/Delete সহজ
- ✓ Memory waste কম (Array এর মতো extra জায়গা লাগে না)

### ✗ অসুবিধা

- ✗ Random access করা যায় না
- ✗ শুধু সামনে যাওয়া যায়

## ◆ Array vs Singly Linked List

বিষয়	Array	Singly Linked List
Memory	Continuous	Non-Continuous
Access	Fast	Slow
Insert	কঠিন	সহজ
Size	Fixed	Dynamic

### 🔥 এক লাইনে মনে রাখো

Singly Linked List হলো এমন একটি লিস্ট যেখানে প্রতিটা Node শুধু তার পরবর্তী Node কে চেনে।

## Node আর্কিটেকচার (Memory Level বুঝো)

Singly Linked List-এ একটি Node এর ২টা অংশ থাকে:

- **[ Data | Next ]**

### Data

→ আসল ভ্যালু (যেমন: 10)

### Next

→ পরের Node-এর reference

---

ধরো Node A:

Address: 1000

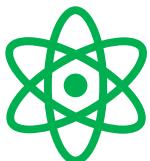
[ 10 | 5000 ]

মানে:

- Data = 10
- Next = 5000 (পরের Node-এর address)

## STEP 2: Node Structure

```
class Node {  
    int data;  
    Node next;  
  
    Node(int data) {  
        this.data = data;  
        this.next = null;  
    }  
}
```



- কেন next null?
  - কেন Node type?
  - কেন int না?
- 
- ◆ কনস্ট্রাক্টরে this.next = null; কেন দেওয়া হয়েছে?
- মূল ধারণা:

next মানে হলো → “পরের Node কোথায়?”

যখন নতুন একটা Node তৈরি করি, তখন সে জানে না তার  
পরে কে আসবে।

তাই শুরুতে তাকে বলি — তোমার পরে কেউ নেই → null

---



## ট্রেইন উদাহরণ

ভাবো একটা ট্রেইনের বগি।

- প্রতিটি বগি = এক একটা Node
- বগির ভিতরে যাত্রী = data
- বগির সাথে পরের বগির সংযোগ = next

শেষ বগির পরে কি থাকে?

👉 কিছুই না

👉 তাই null

[10] → [20] → [30] → null

শেষে null না দিলে বুঝবো কিভাবে যে ট্রেইন শেষ হয়েছে?

---



## Treasure Hunt উদাহরণ

ভাবো তুমি গুপ্তধন খুঁজছো।

প্রতিটি ক্লু বলছে:

"পরের ক্লু কোথায় আছে"

শেষ ক্লুতে লেখা থাকবে:

"আর কোনো ক্লু নেই"

এই "আর কোনো ক্লু নেই" = null

---

 **তাই `this.next = null;` কেন?**

- নতুন Node তৈরি হলে
- তার পরে কেউ নেই
- তাই `next = null`
- এটা Linked List এর শেষ বোঝায়

◆ **`Node next;` এর ডেটা টাইপ Node কেন?**

 **মূল ধারণা:**

কারণ `next` এমন একটা ভেরিয়েবল যা পরের Node কে  
ধরে রাখবে

মানে:

একটা Node → আরেকটা Node কে পয়েন্ট করবে  
তাই তার টাইপও Node হতে হবে।

---



## ট্রেন উদাহরণ

একটা বগি যদি পরের বগির সাথে যুক্ত হয়,  
তাহলে সে কি একটা সংখ্যা ধরে রাখবে?

না ✗

সে ধরে রাখবে পুরো একটা বগি

তাই:

Node next;

মানে:

এই ভেরিয়েবলটা আরেকটা Node ধরে রাখবে

---



## চেইন উদাহরণ

Node A → Node B → Node C

A এর next = B

B এর next = C

তাহলে A এর next কি?

👉 একটা Node (B)

তাই টাইপ হবে Node।

---

◆ next এর টাইপ int বা অন্য সাধারণ ডেটা টাইপ  
কেন হলো না?

এটাই সবচেয়ে গুরুত্বপূর্ণ প্রশ্ন 🔥

---

✖ যদি আমরা লিখতাম:

int next;

তাহলে কী হতো?

তাহলে আমরা শুধু একটা সংখ্যা রাখতাম।

কিন্তু Linked List এ দরকার হচ্ছে:

"পরের Node এর ঠিকানা"

সংখ্যা না,

👉 পুরো একটা অবজেক্ট দরকার।

---

## খুব গুরুত্বপূর্ণ কথা

Linked List এ next ভেরিয়েবল আসলে ধরে রাখে:

পরের Node এর memory address (reference)

এটা কোনো সংখ্যা না

এটা একটা reference

---



## Treasure Map উদাহরণ

তুমি যদি শুধু লিখো:

next = 5

তাহলে তুমি জানো না ক্লু কোথায়।

কিন্তু যদি লেখো:

next = nextClueObject

তাহলে তুমি সরাসরি পরের ক্লুতে চলে যেতে পারো।

---



## Linked List এর আসল ম্যাজিক

Node A



Node B



Node C



null

এখানে প্রতিটা Node এর ভিতরে আছে:

- data (সংখ্যা)
- next (আরেকটা Node এর reference)

প্রশ্ন

this.next = null; কেন?

সহজ উত্তর

কারণ নতুন Node এর পরে কেউ নেই

Node next; কেন?

কারণ এটা পরের Node কে ধরবে

int next; কেন না?

কারণ আমাদের সংখ্যা না, পুরো Node দরকার

## 💡 মনে রাখার গোল্ডেন লাইন

👉 Linked List এ next মানে সংখ্যা না

👉 এটা হচ্ছে “পরের Node এর ঠিকানা”

👉 তাই তার টাইপ হবে Node

## 👉 Singly Linked List-এ একটি Node এর ২টা অংশ

একটি Node এর ভিতরে থাকে:

- [ Data | Next ]

এখন আমরা এটা Memory Level এ বুঝবো।

---

### ◆ Data Part (ডাটা অংশ)

এটা হলো আসল তথ্য।

উদাহরণ:

```
int data = 10;
```

ধরো এই data memory তে address 1000 এ রাখা আছে।

**Address: 1000**

**Value: 10**

এই অংশে শুধু মান (value) থাকে।

### ◆ Next Part (Pointer অংশ)

এটা সবচেয়ে গুরুত্বপূর্ণ অংশ 🔥

Next নিজে কোনো value না।  
এটা রাখে → **পরবর্তী Node** কোথায় আছে তার  
**memory address**।

ধরো:

**Node A → data = 10**

**Node B → data = 20**

Memory তে এমন থাকতে পারে:

**Address 1000 → 10**

**Address 1004 → 2000 (Next pointer)**

**Address 2000 → 20**

**Address 2004 → null**

এখন বুঝো কী হলো 👇

Node A এর:

. data = 10

. next = 2000

মানে Node A বলছে:

👉 “আমার পরের node আছে 2000 address এ।”

---

## 🔥 সম্পূর্ণ Node Memory Layout

ধরো Linked List:

***10 → 20 → 30 → null***

Memory তে এমন হতে পারে (বাস্তবে ছড়ানো থাকে):

<b>Address</b>	<b>Data</b>	<b>Next</b>
<b>1000</b>	<b>10</b>	<b>5000</b>
<b>5000</b>	<b>20</b>	<b>9000</b>
<b>9000</b>	<b>30</b>	<b>null</b>

দেখো গুরুত্বপূর্ণ জিনিস:

- ! Memory contiguous না
  - ! কিন্তু next pointer দিয়ে সব connected
- এটাই Singly Linked List এর আসল শক্তি।

---

## 🧠 Visualization Trick

Node কে ভাবো একটা দুই রুমের বাড়ি হিসেবে:

- 🏠 রুম ১ → Data
- 🏠 রুম ২ → পরের বাড়ির ঠিকানা

যদি ঠিকানা না থাকে → null



## Java Level Deep Understanding

- **class Node {**  
    **int data;**  
    **Node next;**  
**}**

এখানে:

- int data → 4 byte memory (সাধারণত)
- Node next → এটা একটা reference (address ধরে)

Java তে সরাসরি address দেখা যায় না  
কিন্তু concept একই।

---

### ⚡ কেন Next Node type?

অনেকে এখানে ভুল করে।

Node next;

কেন int next; না?

কারণ next কোনো সংখ্যা না  
এটা আরেকটা Node এর ঠিকানা ধরে।

Node → Node কে point করে।

---

## 🎯 সবচেয়ে গুরুত্বপূর্ণ বিষয়

Linked List এ:

Data যতটা গুরুত্বপূর্ণ

Next তার চেয়ে বেশি গুরুত্বপূর্ণ।

কারণ:

- Data বদলালে শুধু মান বদলায়
  - Next বদলালে পুরো structure বদলে যায়
- 

## 👉 Pointer Change মানেই Structure Change

ধরো:

**10 → 20 → 30**

যদি তুমি 10 এর next কে null করে দাও:

10 → null

20 → 30 (Disconnected)

পুরো list ভেঙে গেল।

---

### 🏆 Final Memory-Level Definition

Singly Linked List এর একটি Node দুইটি অংশ নিয়ে  
গঠিত —

একটি অংশে data সংরক্ষিত থাকে এবং অন্য অংশে  
পরবর্তী Node এর memory address সংরক্ষিত থাকে।



## Clean Java Implementation

```
1. // Node class (single node structure)
2. class Node {
3.     int data;    // data store করবে
4.     Node next;  // next node কে point করবে
5.     // Constructor
6.     Node(int data){
7.         this.data = data;
8.         this.next = null; // শুরুতে next null
9.     }
}
```

```

10. }
11. // Singly Linked List class
12. class SinglyLinkedList {
13.     Node head; // first node
14.     // Add method (insert at end)
15.     void add(int value) {
16.         Node newNode = new Node(value); // নতুন node বানালাম
17.         // যদি list empty হয়
18.         if (head == null) {
19.             head = newNode;
20.             return;
21.         }
22.         // না হলে শেষ পর্যন্ত traverse করবো
23.         Node temp = head;
24.         while (temp.next != null) {
25.             temp = temp.next;
26.         }
27.         // শেষ node এর next এ newNode বসাই
28.         temp.next = newNode;
29.     }
30.     // Print method
31.     void printList() {
32.         Node temp = head;
33.         while (temp != null) {
34.             System.out.print(temp.data + " -> ");
35.             temp = temp.next;
36.         }
37.         System.out.println("null");
38.     }
39. }
40. public class Main {
41.     public static void main(String[] args) {
42.         SinglyLinkedList list = new SinglyLinkedList();
43.         list.add(10);
44.         list.add(20);
45.         list.add(30);
46.         list.printList();
47.     }
48. }

```

**Now we explain it step by step :**

## Step 1: Node Class (একদম basic)

```
// Single node structure
class Node {
    int data; // value store করবে
    Node next; // next node কে point করবে

    Node(int data) { // constructor
        this.data = data;
        this.next = null; // প্রথমে next null
    }
}
```

### ব্যাখ্যা:

- *data* → node এর value
- *next* → পরের node কে point করবে
- *constructor* → node তৈরি করার সময় value set করবে

## Step 2: Linked List Head + Manual Node

### *Creation*

এখন আমরা মনে মনে একটা list বানাবো *manually*

- ```
public class Main {  
    public static void main(String[] args) {  
  
        // Step 1: head node তৈরি  
        Node head = new Node(10); // প্রথমnode  
        System.out.println(head.data); // 10  
    }  
}
```

- **Memory Visualization:**

**head**

↓

**[10 | null]**

এখন **list** এ শুধু ১টা **node** আছে

**next null** → কোন পরের **node** নেই



### **Step 3: Add Node Manually (Without Method)**

- ```
public class Main {  
    public static void main(String[] args) {  
  
        Node head = new Node(10); // first node  
  
        Node second = new Node(20); // second node  
        head.next = second;      // link first -> second  
  
        Node third = new Node(30); // third node  
    }  
}
```

```

second.next = third; // link second -> third

// Print manually
System.out.println(head.data); // 10
System.out.println(head.next.data); // 20
System.out.println(head.next.next.data); // 30
}
}

```

## **Memory Visualization:**

**head → [10 | • ] → [20 | • ] → [30 | null]**

🔥 ধাপে ধাপে link করা হয়েছে

## **Full code**

```

class node {
    int data;
    node next;
    node(int data) {
        this.data = data;
        this.next = null;
    }
}

```

```
}

}

public class Main {

public static void main(String[] args) {

node head = new node(10);

node second = new node(20);

node third = new node(30);

node fourth = new node(40);

node fifth = new node(50);

head.next = second;

second.next = third;

third.next = fourth;

fourth.next = fifth;

fifth.next = null;

System.out.println(head.data);

System.out.println(head.next.data);

System.out.println(head.next.next.data);

System.out.println(head.next.next.next.data);

System.out.println(head.next.next.next.next.data);

}

}
```

## *Traverse Linked List (Loop)*

এখন আমরা *loop* দিয়ে *print* করতে শিখব

### Step 1: Traverse মানে কি?

*Traverse* মানে হলো:

- 👉 *Head* থেকে শুরু করে
- 👉 *next pointer* ধরে ধরে
- 👉 একদম *null* পর্যন্ত যাওয়া

```
class node {  
    int data;  
    node next;  
    node(int data) {  
        this.data = data;  
        this.next = null;  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {
```

```
node head = new node(10);
node second = new node(20);
node third = new node(30);
node fourth = new node(40);
node fifth = new node(50);

head.next = second;
second.next = third;
third.next = fourth;
fourth.next = fifth;
fifth.next = null;

node temp = head;

while (temp != null) {
    System.out.print(temp.data + ">>");
    temp = temp.next;
}

System.out.println("null");

int count = 0;
temp = head;

while (temp != null) {
    count++;
    temp = temp.next;
}

System.out.println("Total count is = " + count);
```

}

}