Linked list Insertion:

🧩 Insert a Node at the Beginning

Problem Statement:

You are given a singly linked list. Write a program to insert a new node at the **beginning** of the linked list.

Input:

- The first line contains the number of nodes `n` in the existing linked list.
- The next line contains `n` space-separated integers representing node values.
- The next line contains an integer `x`, the value to be inserted at the beginning.

Output:

- Print the linked list after insertion using the format:
  `data1 → data2 → data3 → ... → NULL`

Example:

Input

```
4
10 20 30 40
5
```

Output

```
5 → 10 → 20 → 30 → 40 → NULL
```

🧩 Insert a Node at the End

Problem Statement:

Write a program to insert a new node at the **end** of the singly linked list.

Input:

- Number of nodes `n`.
- Next line: `n` space-separated integers.
- Next line: integer `x`, the value to be inserted at the end.

Output:

- Print the linked list after insertion using `>` format.

Example:

Input

```
3
10 20 30
40
```

Output

```
10 → 20 → 30 → 40 → NULL
```

🧩 Insert a Node at a Given Position

Problem Statement:

You are given a singly linked list and an integer `pos`. Insert a new node with a given value `x` **at the specified position** (1-based index).

Input:

- Number of nodes `n`.

- List elements.
- Integer `x` (value to insert).
- Integer `pos` (position to insert).

Output:

- Linked list after insertion using `>`.

Example:

Input

```
4
10 20 30 40
25
3
```

Output

```
10 → 20 → 25 → 30 → 40 → NULL
```

🧩 Insert After a Given Value

Problem Statement:

Write a program to insert a node with a given value `x` **after a node with a specific value** `key` in the singly linked list.

If `key` is not found, print “Value not found”.

Input:

- Number of nodes `n`.
- List elements.
- Integer `key` (value after which new node is inserted).
- Integer `x` (value to insert).

Output:

- Updated linked list in `>` format or “Value not found” if key doesn’t exist.

Example:

Input

```
5
10 20 30 40 50
30
35
```

Output

```
10 → 20 → 30 → 35 → 40 → 50 → NULL
```

🧩 Insert Before a Given Value

Problem Statement:

Write a program to insert a node with a given value `x` **before a node with a specific value** `key`.

If `key` is not found, print “Value not found”.

Input:

- Number of nodes `n`.
- List elements.
- Integer `key` (value before which new node is inserted).
- Integer `x` (value to insert).

Output:

- Updated linked list in `>` format or “Value not found”.

Example:

Input

```
5
10 20 30 40 50
30
25
```

Output

```
10 → 20 → 25 → 30 → 40 → 50 → NULL
```

🧩 Insert in a Sorted Linked List

Problem Statement:

Given a **sorted singly linked list**, insert a new node with a value `x` in such a way that the linked list remains sorted.

Input:

- Number of nodes `n`.
- `n` sorted integers (ascending order).
- Integer `x` (value to insert).

Output:

- Sorted linked list after insertion using `>`.

Example:

Input

```
5
10 20 30 40 50
35
```

Output

```
10 → 20 → 30 → 35 → 40 → 50 → NULL
```

🧩 Insert in the Middle of Linked List

Problem Statement:

Insert a new node with value `x` in the **middle** of a given singly linked list.

If the number of nodes is even, insert the new node **after the first half**.

Input:

- Number of nodes `n`.
- List elements.
- Integer `x` (value to insert).

Output:

- Linked list after inserting `x` in the middle (use `>` format).

Example:

Input

```
5
10 20 30 40 50
25
```

Output

```
10 → 20 → 25 → 30 → 40 → 50 → NULL
```

🧩 Merge and Insert Alternate Nodes

Problem Statement:

You are given two singly linked lists. Write a program to **merge** them by inserting nodes of the second list **alternately** into the first list.

Input:

- Number of nodes `n1` in the first list followed by its elements.
- Number of nodes `n2` in the second list followed by its elements.

Output:

- Merged linked list after alternate insertion using `>`.

Example:

Input

```
3
10 20 30
3
40 50 60
```

Output

```
10 → 40 → 20 → 50 → 30 → 60 → NULL
```

🧩 Insert Nodes Such That Even and Odd Nodes Are Separated

Problem Statement:

Write a program to rearrange and insert nodes in such a way that **all even-valued nodes appear after odd-valued nodes**, while maintaining their original order.

Input:

- Number of nodes `n`.
- List elements.

Output:

- Linked list with all odd values first, followed by even values (use `>` format).

Example:

Input

```
7
10 15 20 25 30 35 40
```

Output

```
15 → 25 → 35 → 10 → 20 → 30 → 40 → NULL
```
