Day 2: Linked List Operations - Push Back and Traversal

Date: February 28, 2025

Topic: Singly Linked List Operations in Rust

Goal: Master adding nodes to the end (`push_back`) and understand mutable traversal.

Recap of Day 1

- **Date:** February 27, 2025
- **Achievements:**
    - Built a singly linked list with `Node<T>` and `LinkedList<T>`.
    - Implemented `push` (add to front), `print` (display list), and `pop` (remove from front).
    - Explored `self.head.take()` for ownership movement.
- **Base Code:**

```rust
struct Node<T> {
    data: T,
    next: Option<Box<Node<T>>>,
}

struct LinkedList<T> {
    head: Option<Box<Node<T>>>,
}

impl<T> LinkedList<T> {
    fn new() -> Self {
        LinkedList { head: None }
    }

    fn push(&mut self, data: T) {
        let new_node = Box::new(Node {
            data,
            next: self.head.take(),
        });
        self.head = Some(new_node);
    }

    fn print(&self) where T: std::fmt::Display {
        let mut current = &self.head;
        while let Some(node) = current {
            print!("{} -> ", node.data);
            current = &node.next;
        }
        println!("None");
    }

    // Your pop (assumed implementation)
    fn pop(&mut self) -> Option<T> {
        self.head.take().map(|node| {
            self.head = node.next;
            node.data
        })
    }
}
```

Day 3: Linked List Operations - Insert and Delete at Position

Date: March 1, 2025

Topic: Singly Linked List Operations in Rust

Goal: Implement insertion and deletion at a specific index in the list.

Recap of Day 2

- **Date:** February 28, 2025
- **Achievements:**
  - Explored `push_back` (add to end) with mutable traversal.
  - Mastered `while let Some` with `ref mut` for mutable references vs. immutable traversal in `print`.
  - Reinforced `pop` and `push` from Day 1.
- **Base Code (Updated from Day 2):**

```rust
struct Node<T> {
    data: T,
    next: Option<Box<Node<T>>>,
}

struct LinkedList<T> {
    head: Option<Box<Node<T>>>,
}

impl<T> LinkedList<T> {
    fn new() -> Self {
        LinkedList { head: None }
    }

    fn push(&mut self, data: T) {
        let new_node = Box::new(Node {
            data,
            next: self.head.take(),
        });
        self.head = Some(new_node);
    }

    fn print(&self) where T: std::fmt::Display {
        let mut current = &self.head;
        while let Some(node) = current {
            print!("{} -> ", node.data);
            current = &node.next;
        }
        println!("None");
    }

    fn pop(&mut self) -> Option<T> {
        self.head.take().map(|node| {
            self.head = node.next;
            node.data
        })
    }

    fn push_back(&mut self, data: T) {
        let new_node = Box::new(Node { data, next: None });
        if self.head.is_none() {
            self.head = Some(new_node);
            return;
        }
        let mut current = self.head.as_mut().unwrap();
        while let Some(ref mut next_node) = current.next {
            current = next_node;
        }
        current.next = Some(new_node);
    }
}
```