Code With Coffee

```c
// Definition for singly-linked list.
struct ListNode {
   int val;
   struct ListNode *next;
};

struct ListNode* reverse(struct ListNode* head) {
   struct ListNode* prev = NULL;
   struct ListNode* curr = head;
   struct ListNode* next = NULL;
   while (curr) {
      next = curr->next;
      curr->next = prev;
      prev = curr;
      curr = next;
   }
   return prev;
}

void reorderList(struct ListNode* head) {
   if (!head || !head->next) return;

   // 1. Find middle
   struct ListNode* slow = head;
   struct ListNode* fast = head;
   while (fast->next && fast->next->next) {
      slow = slow->next;
      fast = fast->next->next;
   }

   // 2. Reverse second half
   struct ListNode* second = reverse(slow->next);
   slow->next = NULL;

   // 3. Merge
   struct ListNode* first = head;
   while (first && second) {
      struct ListNode* temp1 = first->next;
      struct ListNode* temp2 = second->next;
      first->next = second;
      second->next = temp1;
      first = temp1;
      second = temp2;
   }
}
```

Code With Coffee

```
// Definition for singly-linked list.
struct ListNode {
    int val;
    ListNode *next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode *next) : val(x), next(next) {}
};

class Solution {
public:
    ListNode* reverse(ListNode* head) {
        ListNode* prev = nullptr;
        while (head) {
            ListNode* next = head->next;
            head->next = prev;
            prev = head;
            head = next;
        }
        return prev;
    }

    void reorderList(ListNode* head) {
        if (!head || !head->next) return;

        // 1. Find middle by Tortoise and Hare method
        ListNode* slow = head, *fast = head;
        while (fast->next && fast->next->next) {
            slow = slow->next;
            fast = fast->next->next;
        }

        // 2. Reverse second half
        ListNode* second = reverse(slow->next);
        slow->next = nullptr;

        // 3. Merge
        ListNode* first = head;
        while (first && second) {
            ListNode* t1 = first->next;
            ListNode* t2 = second->next;
            first->next = second;
            second->next = t1;
            first = t1;
            second = t2;
        }
    }
};
```

Code With Coffee

```
# Definition for singly-linked list.
class ListNode:
    def _init_(self, val=0, next=None):
        self.val = val
        self.next = next

class Solution:
    def reverse(self, head):
        prev, curr = None, head
        while curr:
            nxt = curr.next
            curr.next = prev
            prev = curr
            curr = nxt
        return prev

    def reorderList(self, head: ListNode) -> None:
        if not head or not head.next:
            return

        # 1. Find middle
        slow, fast = head, head
        while fast.next and fast.next.next:
            slow = slow.next
            fast = fast.next.next

        # 2. Reverse second half
        second = self.reverse(slow.next)
        slow.next = None

        # 3. Merge
        first = head
        while first and second:
            t1, t2 = first.next, second.next
            first.next = second
            second.next = t1
            first, second = t1, t2
```

Code With Coffee

```java
// Definition for singly-linked list.
class ListNode {
    int val;
    ListNode next;
    ListNode() {}
    ListNode(int val) { this.val = val; }
    ListNode(int val, ListNode next) { this.val = val; this.next = next; }
}

class Solution {
    private ListNode reverse(ListNode head) {
        ListNode prev = null, curr = head;
        while (curr != null) {
            ListNode next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }

    public void reorderList(ListNode head) {
        if (head == null || head.next == null) return;

        // 1. Find middle
        ListNode slow = head, fast = head;
        while (fast.next != null && fast.next.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }

        // 2. Reverse second half
        ListNode second = reverse(slow.next);
        slow.next = null;

        // 3. Merge
        ListNode first = head;
        while (first != null && second != null) {
            ListNode t1 = first.next;
            ListNode t2 = second.next;
            first.next = second;
            second.next = t1;
            first = t1;
            second = t2;
        }
    }
}
```