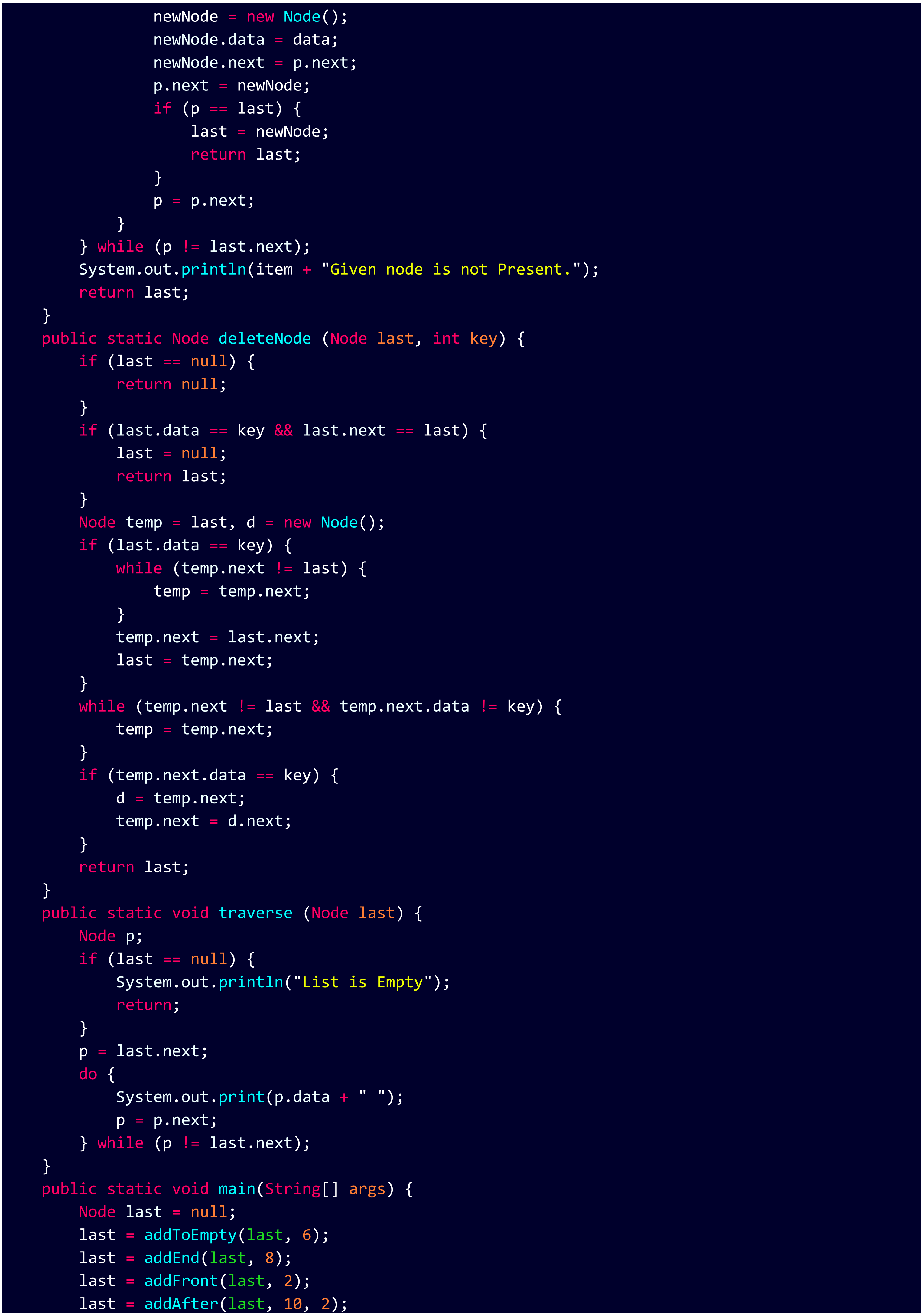
# Linked List

1. Add in the Middle

```
public class AddInMiddle {
    public static class Node {
        int data;
        Node next;
        public Node (int data) {
            this.data = data;
            this.next = null;
        }
    }
    public static Node head;
    public static Node tail;
    public void addFirst (int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = tail = newNode;
            return;
        }
        newNode.next = head;
        head = newNode;
    }
    public void addLast (int data) {
        Node newNode = new Node(data);
        if (head == tail) {
            head = tail = newNode;
            return;
        }
        tail.next = newNode;
        tail = newNode;
    }
    public void add (int idx, int data) {
        if (idx == 0) {
            addFirst(data);
            return;
        }
        Node newNode = new Node(data);
        Node temp = head;
        int i = 0;
        while (i < idx -1) {
            temp = temp.next;
            i++;
        }
        newNode.next = temp.next;
        temp.next = newNode;
    }
    public void print() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + "->");
            temp = temp.next;
        }
        System.out.println("null");
    }
    public static void main(String[] args) {
        AddInMiddle ll = new AddInMiddle();
        ll.addFirst(2);
        ll.addFirst(1);
        ll.addLast(3);
        ll.addLast(4);
        ll.add(2, 9); // Adding an node at the given index
        ll.print();
    }
}
```

2. Circular Linked List

```
public class CircularLL {
    public static class Node {
        int data;
        Node next;
    }
    public static Node addToEmpty (Node last, int data) {
        if (last != null) {
            return last;
        }
        Node newNode = new Node();
        newNode.data = data;
        last = newNode;
        newNode.next = last;
        return last;
    }
    public static Node addFront (Node last, int data) {
        if (last == null) {
            return addToEmpty(last, data);
        }
        Node newNode = new Node();
        newNode.data = data;
        newNode.next = last.next;
        last.next = newNode;
        return last;
    }
    public static Node addEnd (Node last, int data) {
        if (last == null) {
            return addToEmpty(last, data);
        }
        Node newNode = new Node();
        newNode.data = data;
        newNode.next = last.next;
        last.next = newNode;
        last = newNode;
        return last;
    }
    public static Node addAfter (Node last, int data, int item) {
        if (last == null) {
            return null;
        }
        Node newNode, p;
        p = last.next;
        do{
            if (p.data == item) {
```



```
                newNode = new Node();
                newNode.data = data;
                newNode.next = p.next;
                p.next = newNode;
                if (p == last) {
                    last = newNode;
                    return last;
                }
                p = p.next;
            }
        } while (p != last.next);
        System.out.println(item + "Given node is not Present.");
        return last;
    }
    public static Node deleteNode (Node last, int key) {
        if (last == null) {
            return null;
        }
        if (last.data == key && last.next == last) {
            last = null;
            return last;
        }
        Node temp = last, d = new Node();
        if (last.data == key) {
            while (temp.next != last) {
                temp = temp.next;
            }
            temp.next = last.next;
            last = temp.next;
        }
        while (temp.next != last && temp.next.data != key) {
            temp = temp.next;
        }
        if (temp.next.data == key) {
            d = temp.next;
            temp.next = d.next;
        }
        return last;
    }
    public static void traverse (Node last) {
        Node p;
        if (last == null) {
            System.out.println("List is Empty");
            return;
        }
        p = last.next;
        do {
            System.out.print(p.data + " ");
            p = p.next;
        } while (p != last.next);
    }
    public static void main(String[] args) {
        Node last = null;
        last = addToEmpty(last, 6);
        last = addEnd(last, 8);
        last = addFront(last, 2);
        last = addAfter(last, 10, 2);
        traverse(last);
        deleteNode(last, 8);
        traverse(last);
    }
}
```

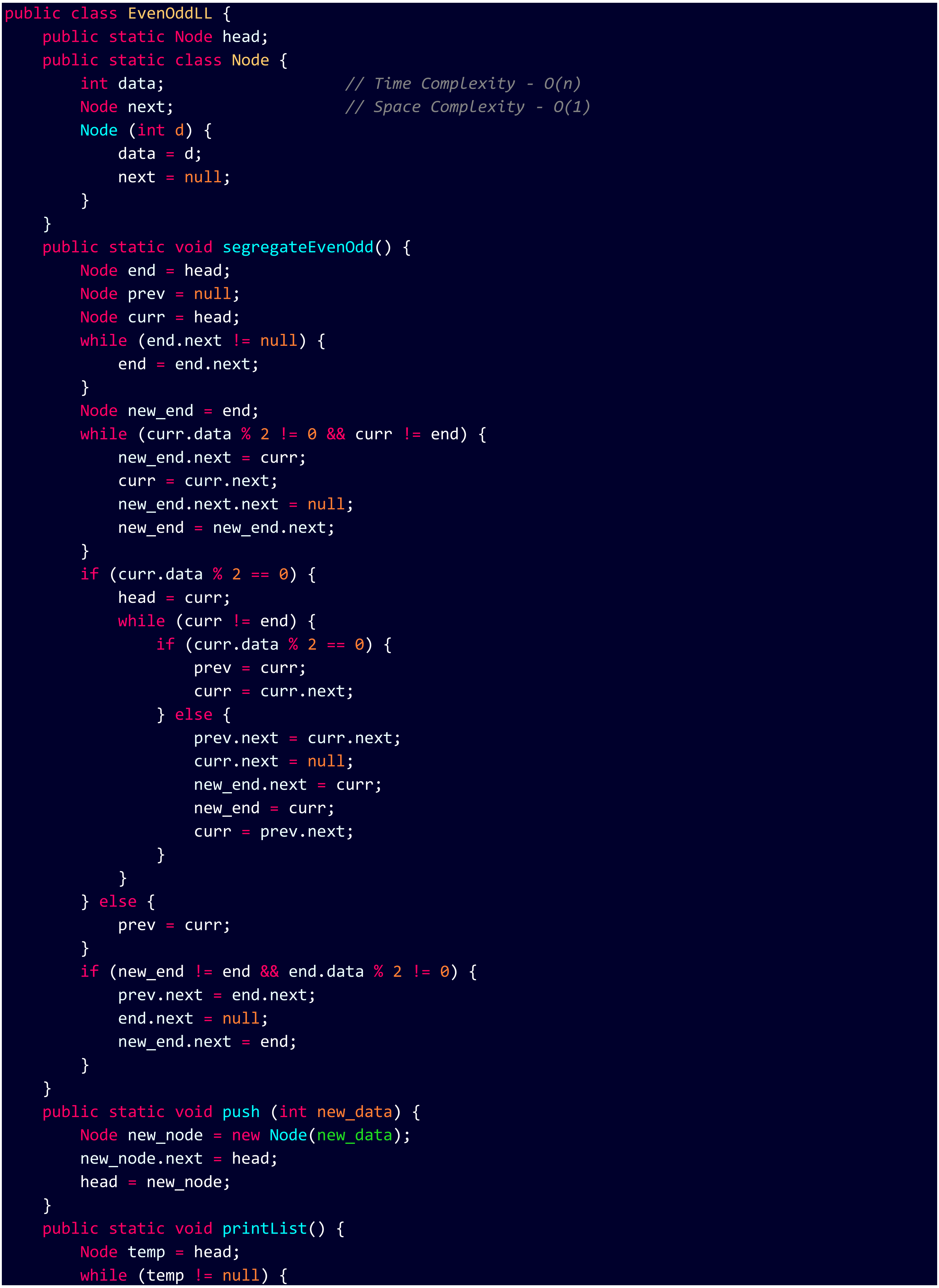
3. Delete N Nodes After M Nodes

```
public class DeleteNnodesAfterMnodes {
    public static class Node {
        int data;                    // Time Complexity - O(n)
        Node next;                   // Space Complexity - O(1)
    }
    public static Node push (Node head_ref, int new_data) {
        Node new_node = new Node();
        new_node.data = new_data;
        new_node.next = (head_ref);
        (head_ref) = new_node;
        return head_ref;
    }
    public static void printList (Node head) {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }
    public static void skipMdeleteN (Node head, int M, int N) {
        Node curr = head, t;
        int count;
        while (curr != null) {
            for (count = 1; count < M && curr != null; count++) {
                curr = curr.next;
            }
            if (curr == null) {
                return;
            }
            t = curr.next;
            for (count = 1; count <= N && t != null; count++) {
                Node temp = t;
                t = t.next;
            }
            curr.next = t;
            curr = t;
        }
    }
    public static void main(String[] args) {
        Node head = null;
        int M = 2, N = 3;
        head = push(head, 10);
        head = push(head, 9);
        head = push(head, 8);
        head = push(head, 7);
        head = push(head, 6);
        head = push(head, 5);
        head = push(head, 4);
        head = push(head, 3);
        head = push(head, 2);
        head = push(head, 1);
        System.out.println("Original Linked List: ");
        printList(head);
        skipMdeleteN(head, M, N);
        System.out.println("Deleted Linked list: ");
        printList(head);
    }
}
```

4. Doubly Linked List

```
public class DoublyLL {
    public class Node{
        int data;
        Node next;
        Node prev;
        public Node (int data) {
            this.data = data;
            this.next = null;
            this.prev = null;
        }
    }
    public static Node head;
    public static Node tail;
    public static int size;
    public void addFirst (int data) {
        Node newNode = new Node(data);
        size++;
        if (head == null) {
            head = tail = newNode;
            return;
        }
        newNode.next = head;
        head.prev = newNode;
        head = newNode;
    }
    public void addLast (int data) {
        Node newNode = new Node(data);
        size++;
        if (head == null) {
            head = tail = newNode;
            return;
        }
        tail.next = newNode;
        newNode.prev = tail;
        tail = newNode;
    }
    public int removeFirst() {
        if (head == null) {
            System.out.println("DLL is Empty.");
            return Integer.MIN_VALUE;
        }
        if (size == 1) {
            int val = head.data;
            head = tail = null;
            size--;
            return val;
        }
        int val = head.data;
        head = head.next;
        head.prev = null;
        size--;
        return val;
    }
    public void removeLast() {
        if (head == null) {
            System.out.println("DLL is Empty.");
            return;
        }
        if (size == 1) {
            head = tail = null;
            size--;
            return;
        }
        tail = tail.prev;
        tail.next = null;
        size--;
    }
    public void print() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + "<->");
            temp = temp.next;
        }
        System.out.println("null");
    }
    public static void main(String[] args) {
        DoublyLL dll = new DoublyLL();
        dll.addFirst(3);
        dll.addFirst(2);
        dll.addFirst(1);
        dll.addLast(4);
        dll.addLast(5);
        dll.addLast(6);
        System.out.print("Original Doubly LL: ");
        dll.print();
        System.out.println("Size = " + size);
        dll.removeFirst();
        dll.removeLast();
        System.out.print("After doing operation, DLL: ");
        dll.print();
        System.out.println("Size = " + size);
    }
}
```

5. Even Odd Linked List



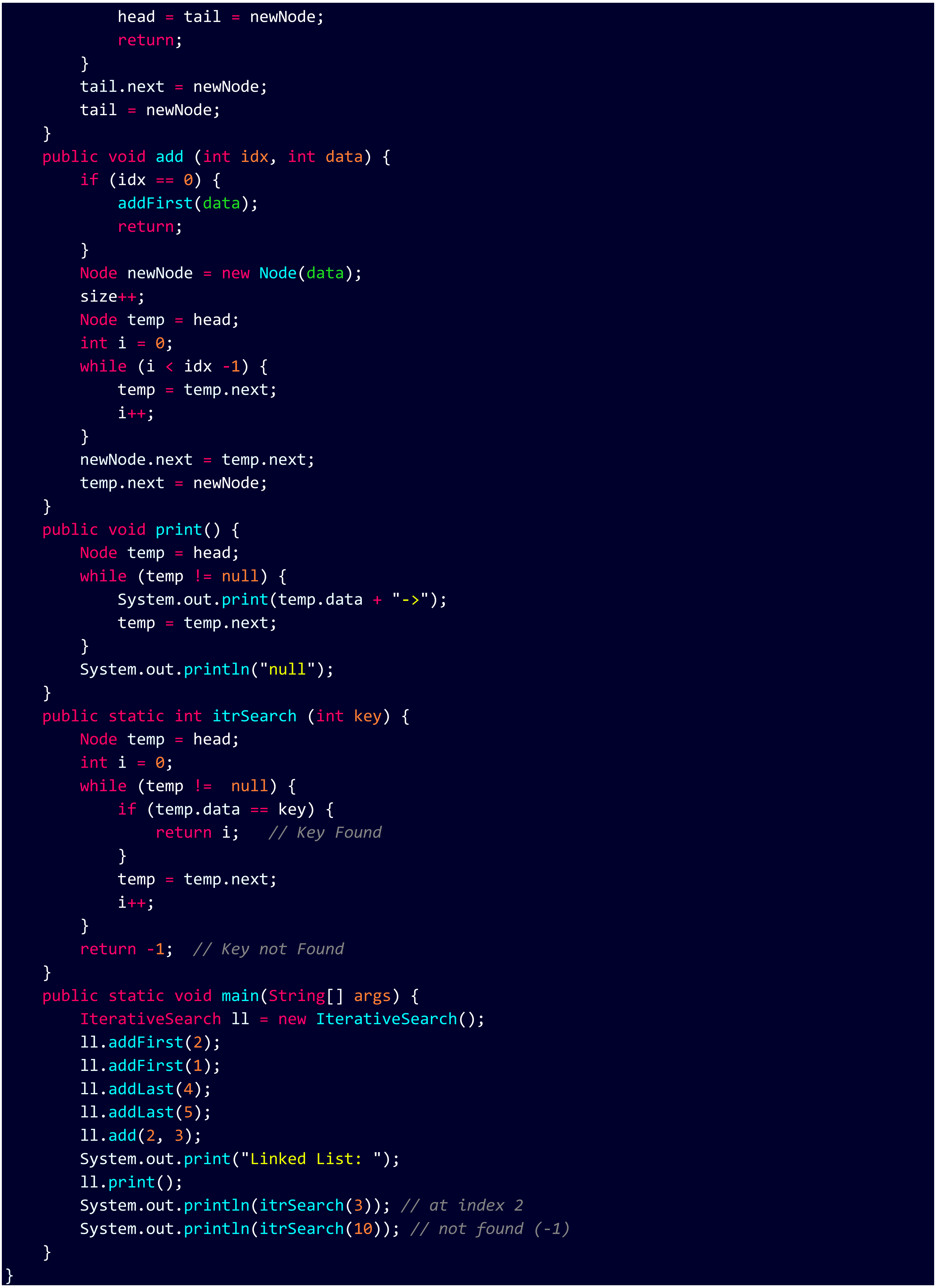
```java
public class EvenOddLL {
    public static Node head;
    public static class Node {
        int data;                       // Time Complexity - O(n)
        Node next;                      // Space Complexity - O(1)
        Node (int d) {
            data = d;
            next = null;
        }
    }
    public static void segregateEvenOdd() {
        Node end = head;
        Node prev = null;
        Node curr = head;
        while (end.next != null) {
            end = end.next;
        }
        Node new_end = end;
        while (curr.data % 2 != 0 && curr != end) {
            new_end.next = curr;
            curr = curr.next;
            new_end.next.next = null;
            new_end = new_end.next;
        }
        if (curr.data % 2 == 0) {
            head = curr;
            while (curr != end) {
                if (curr.data % 2 == 0) {
                    prev = curr;
                    curr = curr.next;
                } else {
                    prev.next = curr.next;
                    curr.next = null;
                    new_end.next = curr;
                    new_end = curr;
                    curr = prev.next;
                }
            }
        } else {
            prev = curr;
        }
        if (new_end != end && end.data % 2 != 0) {
            prev.next = end.next;
            end.next = null;
            new_end.next = end;
        }
    }
    public static void push (int new_data) {
        Node new_node = new Node(new_data);
        new_node.next = head;
        head = new_node;
    }
    public static void printList() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }
    public static void main(String[] args) {
        push(1);
        push(2);
        push(3);
        push(4);
        push(5);
        push(6);
        push(7);
        push(8);
        push(9);
        push(10);
        System.out.print("Linked List: ");
        printList();
        segregateEvenOdd();
        System.out.print("Updated Linked List: ");
        printList();
    }
}
```

6. Intersection of Two Linked List

```
public class IntersectionOfTwoLL {
    public static class Node {
        int data;                       // Time Complexity - O(m*n)
        Node next;                      // Space Complexity - O(n)
        Node (int d) {
            data = d;
            next = null;
        }
    }
    public static Node getIntersectionNode (Node head1, Node head2) {
        while (head2 != null) {
            Node temp = head1;
            while (temp != null) {
                if (temp == head2) {
                    return head2;
                }
                temp = temp.next;
            }
            head2 = head2.next;
        }
        return null;
    }
    public static void main(String[] args) {
        // Creating two linked lists
        Node head1, head2;

        // Initialize the first and second linked list with one node
        head1 = new Node(10);
        head2 = new Node(3);
        // Adding more nodes to the second linked lists
        Node newNode = new Node(6);
        head2.next = newNode;
        newNode = new Node(9);
        head2.next.next = newNode;

        // Adding intersection node(same node in both lists)
        newNode = new Node(15);
        head1.next = newNode; // Link node 15 to first list
        head2.next.next.next = newNode; // Link node 15 to the second list

        // Add one more node to the first linked list
        newNode = new Node(30);
        head1.next.next = newNode;
        head1.next.next.next = null; // End of the first linked list

        // Call the method to find intersection point
        Node interscetionPoint = getIntersectionNode(head1, head2);

        // Output the result
        if (interscetionPoint == null) {
            System.out.print("No Intersection Point");
        } else {
            System.out.print("Intersection Point: " + interscetionPoint.data);
        }
    }
}
```

7. Iterative Search

```
public class IterativeSearch {
    public static class Node {
        int data;
        Node next;
        public Node(int data) {
            this.data = data;
            this.next = null;
        }
    }
    public static Node head;
    public static Node tail;
    public static int size;
    public void addFirst (int data) {
        Node newNode = new Node(data);
        size++;
        if (head == null) {
            head = tail = newNode;
            return;
        }
        newNode.next = head;
        head = newNode;
    }
    public void addLast (int data) {
        Node newNode = new Node(data);
        size++;
        if (head == tail) {
```



```java
            head = tail = newNode;
            return;
        }
        tail.next = newNode;
        tail = newNode;
    }
    public void add (int idx, int data) {
        if (idx == 0) {
            addFirst(data);
            return;
        }
        Node newNode = new Node(data);
        size++;
        Node temp = head;
        int i = 0;
        while (i < idx -1) {
            temp = temp.next;
            i++;
        }
        newNode.next = temp.next;
        temp.next = newNode;
    }
    public void print() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + "->");
            temp = temp.next;
        }
        System.out.println("null");
    }
    public static int itrSearch (int key) {
        Node temp = head;
        int i = 0;
        while (temp !=  null) {
            if (temp.data == key) {
                return i;   // Key Found
            }
            temp = temp.next;
            i++;
        }
        return -1;  // Key not Found
    }
    public static void main(String[] args) {
        IterativeSearch ll = new IterativeSearch();
        ll.addFirst(2);
        ll.addFirst(1);
        ll.addLast(4);
        ll.addLast(5);
        ll.add(2, 3);
        System.out.print("Linked List: ");
        ll.print();
        System.out.println(itrSearch(3)); // at index 2
        System.out.println(itrSearch(10)); // not found (-1)
    }
}
```

## 8. Linked List JCF

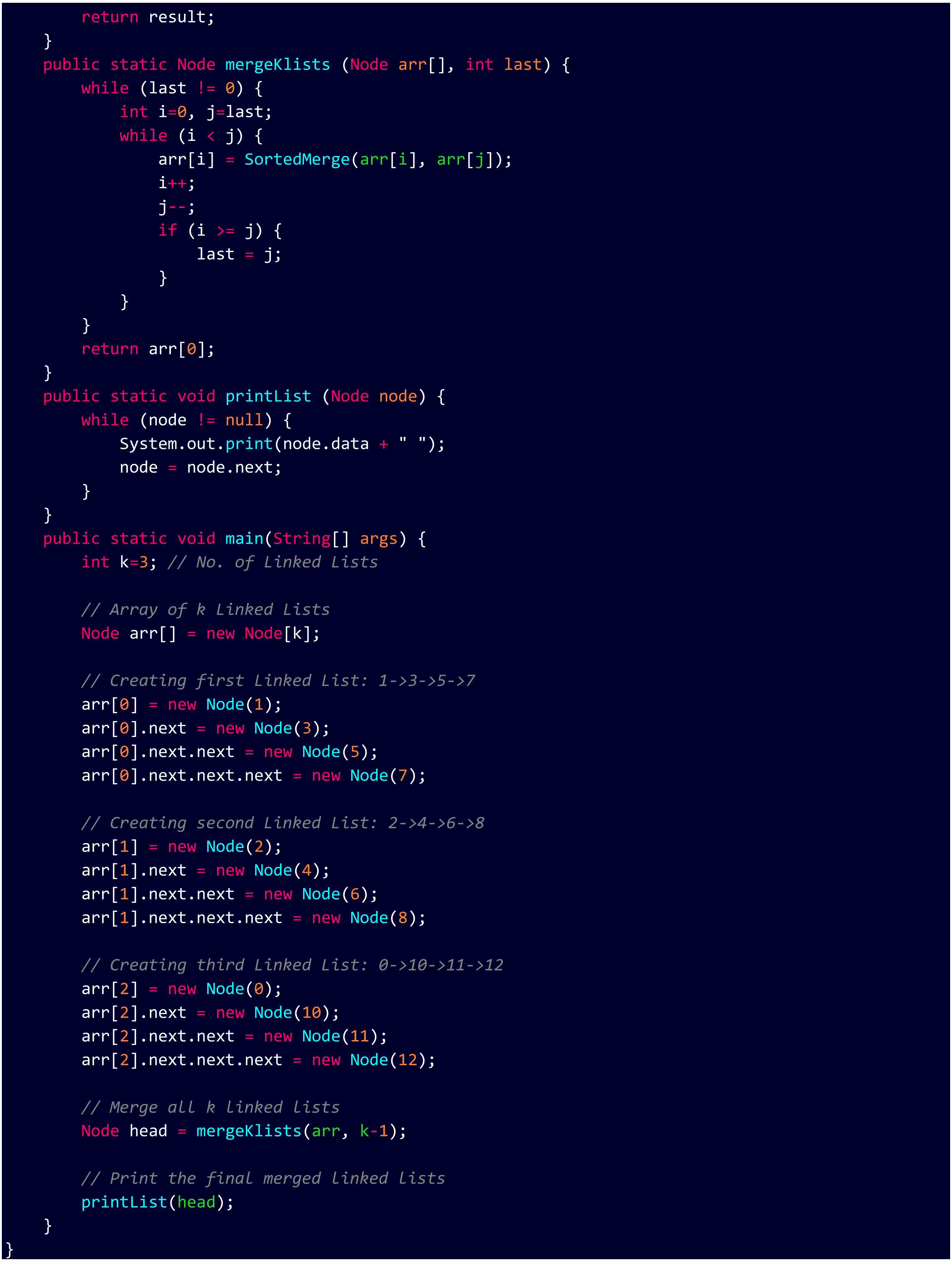
```java
import java.util.LinkedList;
public class LLinJCF {          // Java Collection Framework
    public static void main(String[] args) {
        LinkedList<Integer> ll = new LinkedList<>();    // Creation
        //Add
        ll.addLast(1);
        ll.addLast(2);
        ll.addFirst(0);
        System.out.println(ll); // [0, 1, 2]
        //Remove
        ll.removeLast();
        ll.removeFirst();
        System.out.println(ll); // [1]
    }
}
```

## 9. Loop in Linked List

```java
public class LoopInLL {
    public static class Node {
        int data;
        Node next;
        public Node(int data) {
            this.data = data;
            this.next = null;
        }
    }
    public static Node head;
    public static Node tail;
    public static boolean isCycle() {   // Floyd's Cycle Finding Algorithm
        Node slow = head;
        Node fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next; //+1
            fast = fast.next.next; //+2
            if (slow == fast) {
                return true; //cycle exists
            }
        }
        return false; //cycle doesn't exist
    }
        // This print function will give infinite loop.
    /*  public void print() {
            Node temp = head;
            while (temp != null) {
                System.out.print(temp.data + "->");
                temp = temp.next;
            }
            System.out.println("null");
        } */
    public void print() {
        Node temp = head;
        Node slow = head;
        Node fast = head;
        boolean cycleDetected = false;
        while (temp != null) {
            System.out.print(temp.data + "->");
            if (slow != null) {
                slow = slow.next;
            }
            if (fast != null && fast.next != null) {
                fast = fast.next.next;
            }
            // Detect the cycle
            if (slow == fast) {
                cycleDetected = true;
                break;
            }
            temp = temp.next;
        }
        if (cycleDetected) {
            System.out.println("Cycle Detected at Node with value: " + temp.data);
        } else {
            System.out.println("null");
        }
    }
    public static void main(String[] args) {
        LoopInLL ll = new LoopInLL();
        head = new Node(1);
        head.next = new Node(2);
        head.next.next = new Node(3);
        head.next.next.next = head;
        System.out.println("Cycle exists: " + isCycle());
        ll.print();
    }
}
```

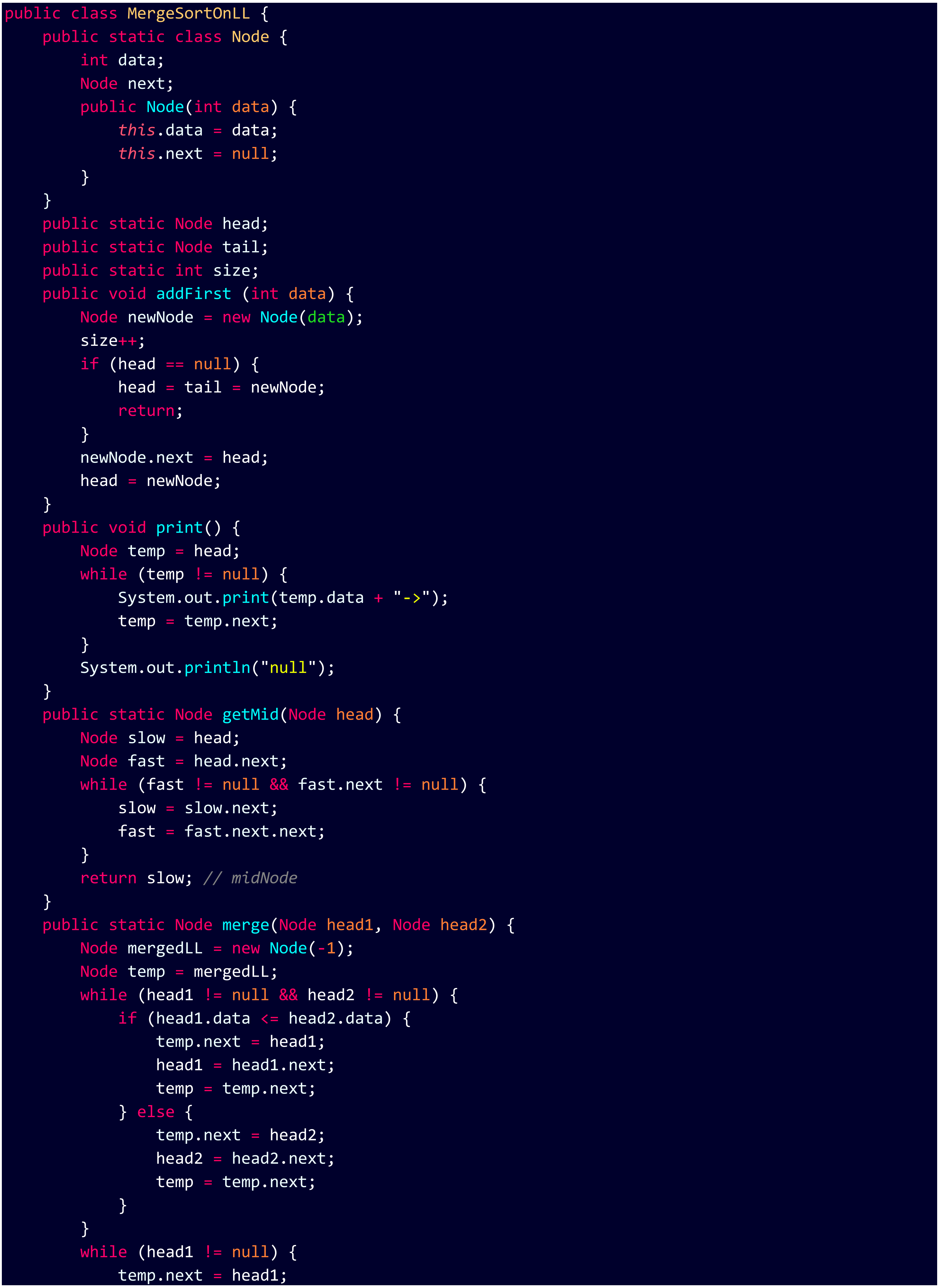
10. Merge K Sorted

```
public class MergeKsortedLists {
    public static class Node {
        int data;                          // Time Complexity, O(nlogk)
        Node next;                         // Space Complexity, O(n)
        Node (int data) {
            this.data = data;
        }
    }
    public static Node SortedMerge (Node a, Node b) {
        Node result = null;
        if (a == null) {
            return b;
        } else if (b == null) {
            return a;
        }
        if (a.data <= b.data) {
            result = a;
            result.next = SortedMerge(a.next, b);
        } else {
            result = b;
            result.next = SortedMerge(a, b.next);
        }
```



```
        return result;
    }
    public static Node mergeKlists (Node arr[], int last) {
        while (last != 0) {
            int i=0, j=last;
            while (i < j) {
                arr[i] = SortedMerge(arr[i], arr[j]);
                i++;
                j--;
                if (i >= j) {
                    last = j;
                }
            }
        }
        return arr[0];
    }
    public static void printList (Node node) {
        while (node != null) {
            System.out.print(node.data + " ");
            node = node.next;
        }
    }
    public static void main(String[] args) {
        int k=3; // No. of Linked Lists

        // Array of k Linked Lists
        Node arr[] = new Node[k];

        // Creating first Linked List: 1->3->5->7
        arr[0] = new Node(1);
        arr[0].next = new Node(3);
        arr[0].next.next = new Node(5);
        arr[0].next.next.next = new Node(7);

        // Creating second Linked List: 2->4->6->8
        arr[1] = new Node(2);
        arr[1].next = new Node(4);
        arr[1].next.next = new Node(6);
        arr[1].next.next.next = new Node(8);

        // Creating third Linked List: 0->10->11->12
        arr[2] = new Node(0);
        arr[2].next = new Node(10);
        arr[2].next.next = new Node(11);
        arr[2].next.next.next = new Node(12);

        // Merge all k linked lists
        Node head = mergeKlists(arr, k-1);

        // Print the final merged linked lists
        printList(head);
    }
}
```

11. Merge Sort on Linked List



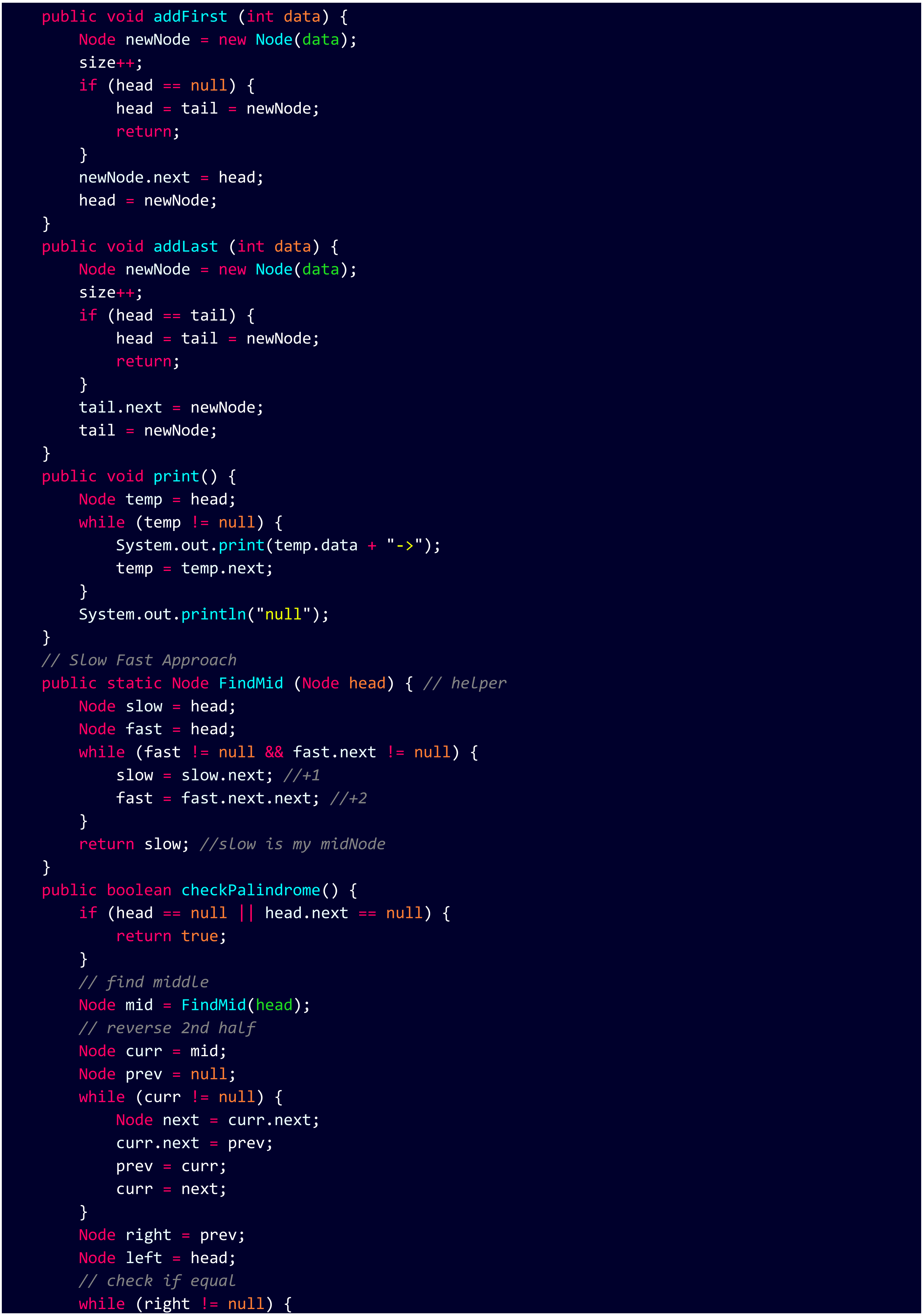
```
public class MergeSortOnLL {
    public static class Node {
        int data;
        Node next;
        public Node(int data) {
            this.data = data;
            this.next = null;
        }
    }
    public static Node head;
    public static Node tail;
    public static int size;
    public void addFirst (int data) {
        Node newNode = new Node(data);
        size++;
        if (head == null) {
            head = tail = newNode;
            return;
        }
        newNode.next = head;
        head = newNode;
    }
    public void print() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + "->");
            temp = temp.next;
        }
        System.out.println("null");
    }
    public static Node getMid(Node head) {
        Node slow = head;
        Node fast = head.next;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow; // midNode
    }
    public static Node merge(Node head1, Node head2) {
        Node mergedLL = new Node(-1);
        Node temp = mergedLL;
        while (head1 != null && head2 != null) {
            if (head1.data <= head2.data) {
                temp.next = head1;
                head1 = head1.next;
                temp = temp.next;
            } else {
                temp.next = head2;
                head2 = head2.next;
                temp = temp.next;
            }
        }
        while (head1 != null) {
            temp.next = head1;
            head1 = head1.next;
            temp = temp.next;
        }
        while (head2 != null) {
            temp.next = head2;
            head2 = head2.next;
            temp = temp.next;
        }
        return mergedLL.next;
    }
    public static Node mergeSort (Node head) {
        if (head == null || head.next == null) {
            return head;
        }
        // Find Mid
        Node mid = getMid(head);
        // Left & Right MergeSort
        Node rightHead = mid.next;
        mid.next = null;
        Node newLeft = mergeSort(head);
        Node newRight = mergeSort(rightHead);
        // Merge
        return merge (newLeft, newRight);
    }
    public static void main(String[] args) {
        MergeSortOnLL ll = new MergeSortOnLL();
        ll.addFirst(1);
        ll.addFirst(2);
        ll.addFirst(3);
        ll.addFirst(4);
        ll.addFirst(5);
        System.out.print("Original Linked List: ");
        ll.print(); // 5->4->3->2->1->null
        head = mergeSort(head);
        System.out.print("Sorted Linked List: ");
        ll.print();
    }
}
```

12. Operation on Linked List

```
public class OperationOnLL {
    public static class Node {
        // Making a Linked List
        int data;
        Node next;
        public Node (int data) {
            this.data = data;
            this.next = null;
        }
    }
    public static Node head; // HEAD
    public static Node tail; // TAIL
    public void addFirst (int data) {
        // Adding at beginning of Linked list
        //S1: Create new node
        Node newNode = new Node(data);
        if (head == null) {
            head = tail = newNode;
            return;
        }
        //S2: New node next = head
        newNode.next = head;
        //S3: Head = newNode
        head = newNode;
    }
    public void addLast (int data) {
        // Adding at last of Linked List
        Node newNode = new Node(data);
        if (head == tail) {
            head = tail = newNode;
            return;
        }
        tail.next = newNode;
        tail = newNode;
    }
    public void print() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + "->");
            temp = temp.next;
        }
        System.out.println("null");
    }
    public static void main(String[] args) {
        OperationOnLL ll = new OperationOnLL();
        ll.print();
        ll.addFirst(2);
        ll.print();
        ll.addFirst(1);
        ll.print();
        ll.addLast(3);
        ll.print();
        ll.addLast(4);
        ll.print();
    }
}
```

13. Palindrome Linked List

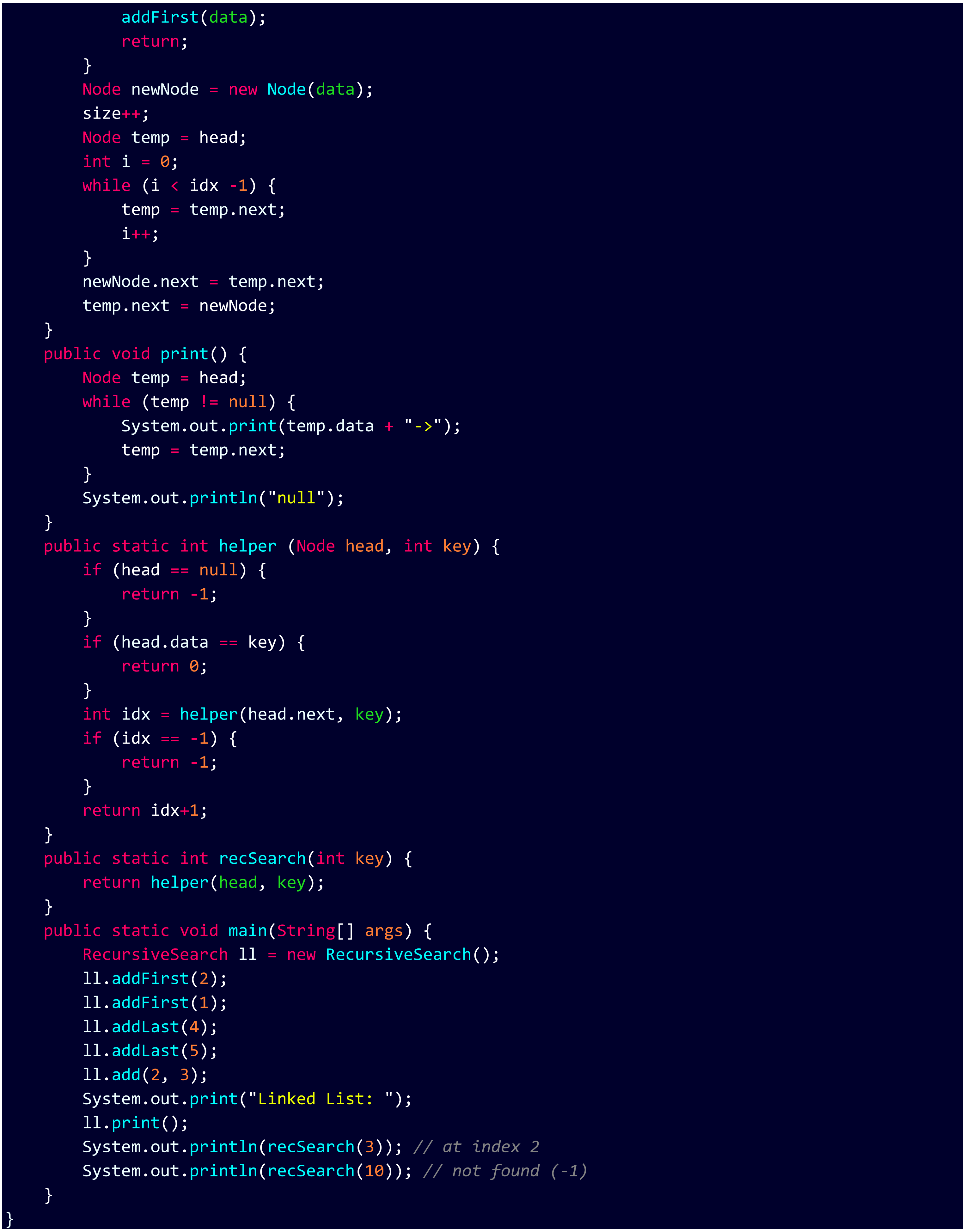
```
public class PalindromeLL {
    public static class Node {
        int data;
        Node next;
        public Node (int data) {
            this.data = data;
            this.next = null;
        }
    }
    public static Node head;
    public static Node tail;
    public static int size;
```



```
    public void addFirst (int data) {
        Node newNode = new Node(data);
        size++;
        if (head == null) {
            head = tail = newNode;
            return;
        }
        newNode.next = head;
        head = newNode;
    }
    public void addLast (int data) {
        Node newNode = new Node(data);
        size++;
        if (head == tail) {
            head = tail = newNode;
            return;
        }
        tail.next = newNode;
        tail = newNode;
    }
    public void print() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + "->");
            temp = temp.next;
        }
        System.out.println("null");
    }
    // Slow Fast Approach
    public static Node FindMid (Node head) { // helper
        Node slow = head;
        Node fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next; //+1
            fast = fast.next.next; //+2
        }
        return slow; //slow is my midNode
    }
    public boolean checkPalindrome() {
        if (head == null || head.next == null) {
            return true;
        }
        // find middle
        Node mid = FindMid(head);
        // reverse 2nd half
        Node curr = mid;
        Node prev = null;
        while (curr != null) {
            Node next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }
        Node right = prev;
        Node left = head;
        // check if equal
        while (right != null) {
            if (left.data != right.data) {
                return false;
            }
            left = left.next;
            right = right.next;
        }
        return true;
    }
    public static void main(String[] args) {
        PalindromeLL ll = new PalindromeLL();
        ll.addFirst(2);
        ll.addFirst(1);
        ll.addLast(2);
        ll.addLast(1);
        System.out.print("Linked List: ");
        ll.print();
        System.out.println("Above LL is Palindrome. -- " + ll.checkPalindrome());
    }
}
```

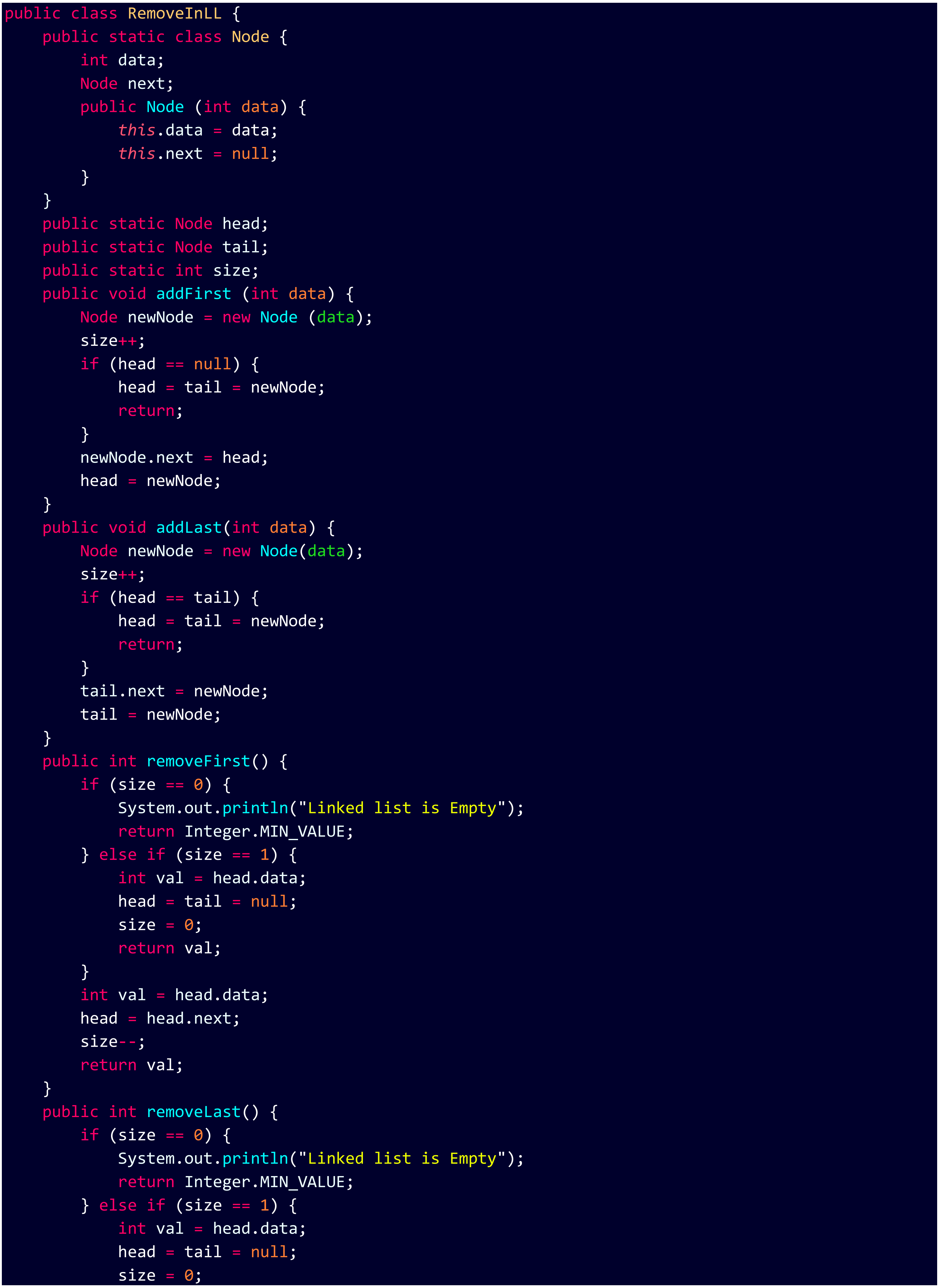
14. Recursive Search

```
public class RecursiveSearch {
    public static class Node {
        int data;
        Node next;
        public Node (int data) {
            this.data = data;
            this.next = null;
        }
    }
    public static Node head;
    public static Node tail;
    public static int size;
    public void addFirst (int data) {
        Node newNode = new Node(data);
        size++;
        if (head == null) {
            head = tail = newNode;
            return;
        }
        newNode.next = head;
        head = newNode;
    }
    public void addLast (int data) {
        Node newNode = new Node(data);
        size++;
        if (head == tail) {
            head = tail = newNode;
            return;
        }
        tail.next = newNode;
        tail = newNode;
    }
    public void add (int idx, int data) {
        if (idx == 0) {
```



```
            addFirst(data);
            return;
        }
        Node newNode = new Node(data);
        size++;
        Node temp = head;
        int i = 0;
        while (i < idx -1) {
            temp = temp.next;
            i++;
        }
        newNode.next = temp.next;
        temp.next = newNode;
    }
    public void print() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + "->");
            temp = temp.next;
        }
        System.out.println("null");
    }
    public static int helper (Node head, int key) {
        if (head == null) {
            return -1;
        }
        if (head.data == key) {
            return 0;
        }
        int idx = helper(head.next, key);
        if (idx == -1) {
            return -1;
        }
        return idx+1;
    }
    public static int recSearch(int key) {
        return helper(head, key);
    }
    public static void main(String[] args) {
        RecursiveSearch ll = new RecursiveSearch();
        ll.addFirst(2);
        ll.addFirst(1);
        ll.addLast(4);
        ll.addLast(5);
        ll.add(2, 3);
        System.out.print("Linked List: ");
        ll.print();
        System.out.println(recSearch(3)); // at index 2
        System.out.println(recSearch(10)); // not found (-1)
    }
}
```

15. Remove in Linked List



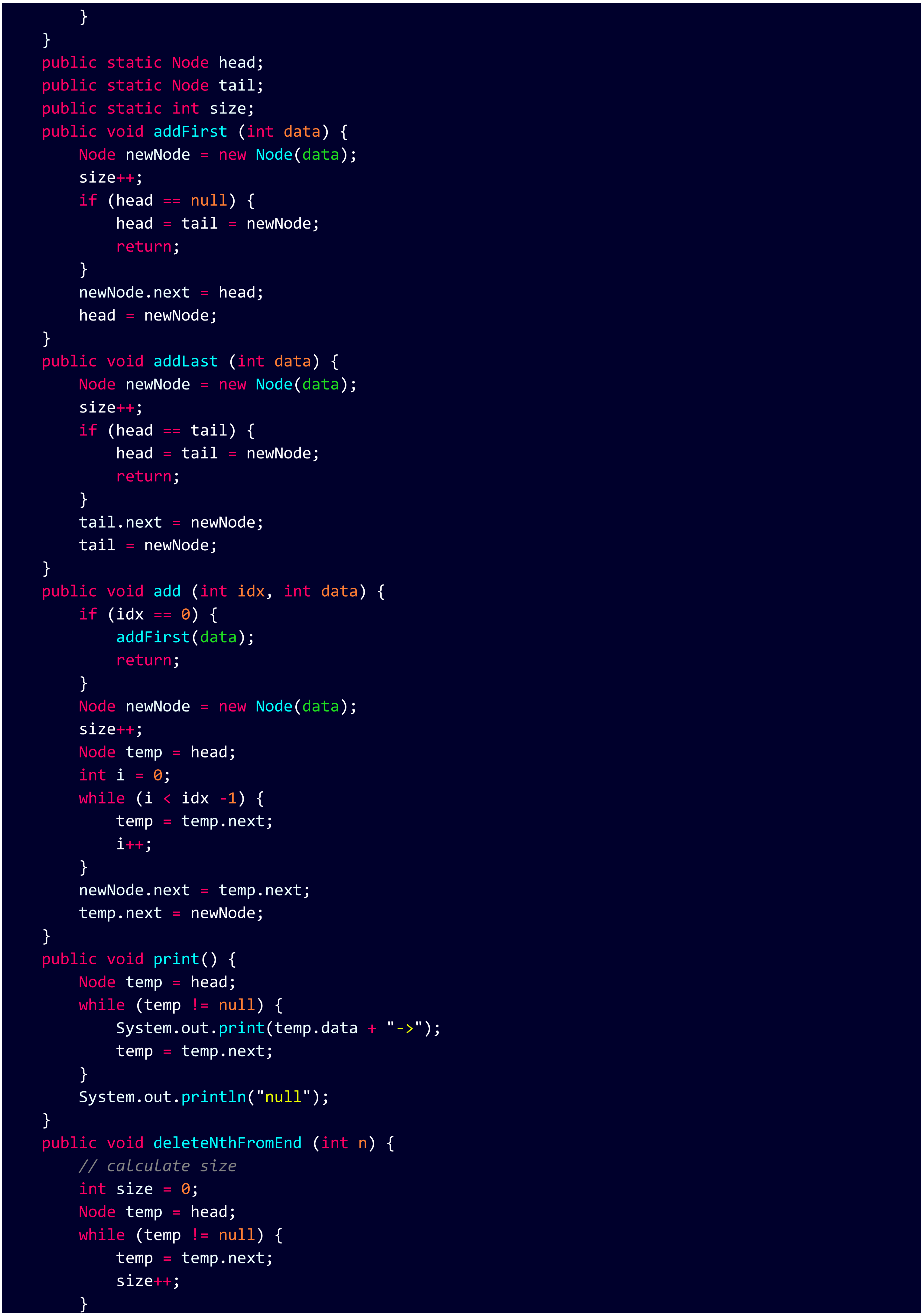
```java
public class RemoveInLL {
    public static class Node {
        int data;
        Node next;
        public Node (int data) {
            this.data = data;
            this.next = null;
        }
    }
    public static Node head;
    public static Node tail;
    public static int size;
    public void addFirst (int data) {
        Node newNode = new Node (data);
        size++;
        if (head == null) {
            head = tail = newNode;
            return;
        }
        newNode.next = head;
        head = newNode;
    }
    public void addLast(int data) {
        Node newNode = new Node(data);
        size++;
        if (head == tail) {
            head = tail = newNode;
            return;
        }
        tail.next = newNode;
        tail = newNode;
    }
    public int removeFirst() {
        if (size == 0) {
            System.out.println("Linked list is Empty");
            return Integer.MIN_VALUE;
        } else if (size == 1) {
            int val = head.data;
            head = tail = null;
            size = 0;
            return val;
        }
        int val = head.data;
        head = head.next;
        size--;
        return val;
    }
    public int removeLast() {
        if (size == 0) {
            System.out.println("Linked list is Empty");
            return Integer.MIN_VALUE;
        } else if (size == 1) {
            int val = head.data;
            head = tail = null;
            size = 0;
            return val;
        }
        Node prev = head;
        for (int i=0; i<size-2; i++) {
            prev = prev.next;
        }
        int val = prev.next.data;
        prev.next = null;
        tail = prev;
        size--;
        return val;
    }
    public void print() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + "->");
            temp = temp.next;
        }
        System.out.println("null");
    }
    public static void main(String[] args) {
        RemoveInLL ll = new RemoveInLL();
        ll.addFirst(3);
        ll.addFirst(2);
        ll.addFirst(1);
        ll.addLast(4);
        ll.addLast(5);
        ll.addLast(6);
        System.out.print("Original Linked List: ");
        ll.print();
        System.out.println("Size of Linked List: " + size);
        ll.removeFirst();
        ll.removeLast();
        System.out.print("After Removing, Linked List: ");
        ll.print();
        System.out.println("Size of Linked List: " + size);
    }
}
```

16. Remove Loop in Linked List

```
public class RemoveLoopInLL {
    public static class Node {
        int data;
        Node next;
        public Node(int data) {
            this.data = data;
            this.next = null;
        }
    }
    public static Node head;
    public static Node tail;
    public static boolean isCycle() {   // Floyd's Cycle Finding Algorithm
        Node slow = head;
        Node fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next; //+1
            fast = fast.next.next; //+2
            if (slow == fast) {
                return true; //cycle exists
            }
        }
        return false; //cycle doesn't exist
    }
    public static void removeCycle() {
        // Detect Cycle
        Node slow = head;
        Node fast = head;
        boolean cycle = false;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
            if (fast == slow) {
                cycle = true;
                break;
            }
        }
        if (cycle == false) {
            return;
        }
        // finding meeting point
        slow = head;
        Node prev = null;
        while (slow != fast) {
            prev = fast;
            slow = slow.next;
            fast = fast.next;
        }
        // remove cycle -> last.next = null
        prev.next = null;
    }
    public static void main(String[] args) {
        head = new Node(1);
        Node temp = new Node(2);
        head.next = temp;
        head.next.next = new Node(3);
        head.next.next.next = temp;
        System.out.println(isCycle()); // 1->2->3->2
        removeCycle();
        System.out.println(isCycle()); // 1->2->3->null
    }
}
```

17. Remove nth Node from End

```
public class RemoveNthNodeFromEnd {
    public static class Node {
        int data;
        Node next;
        public Node (int data) {
            this.data = data;
            this.next = null;
```



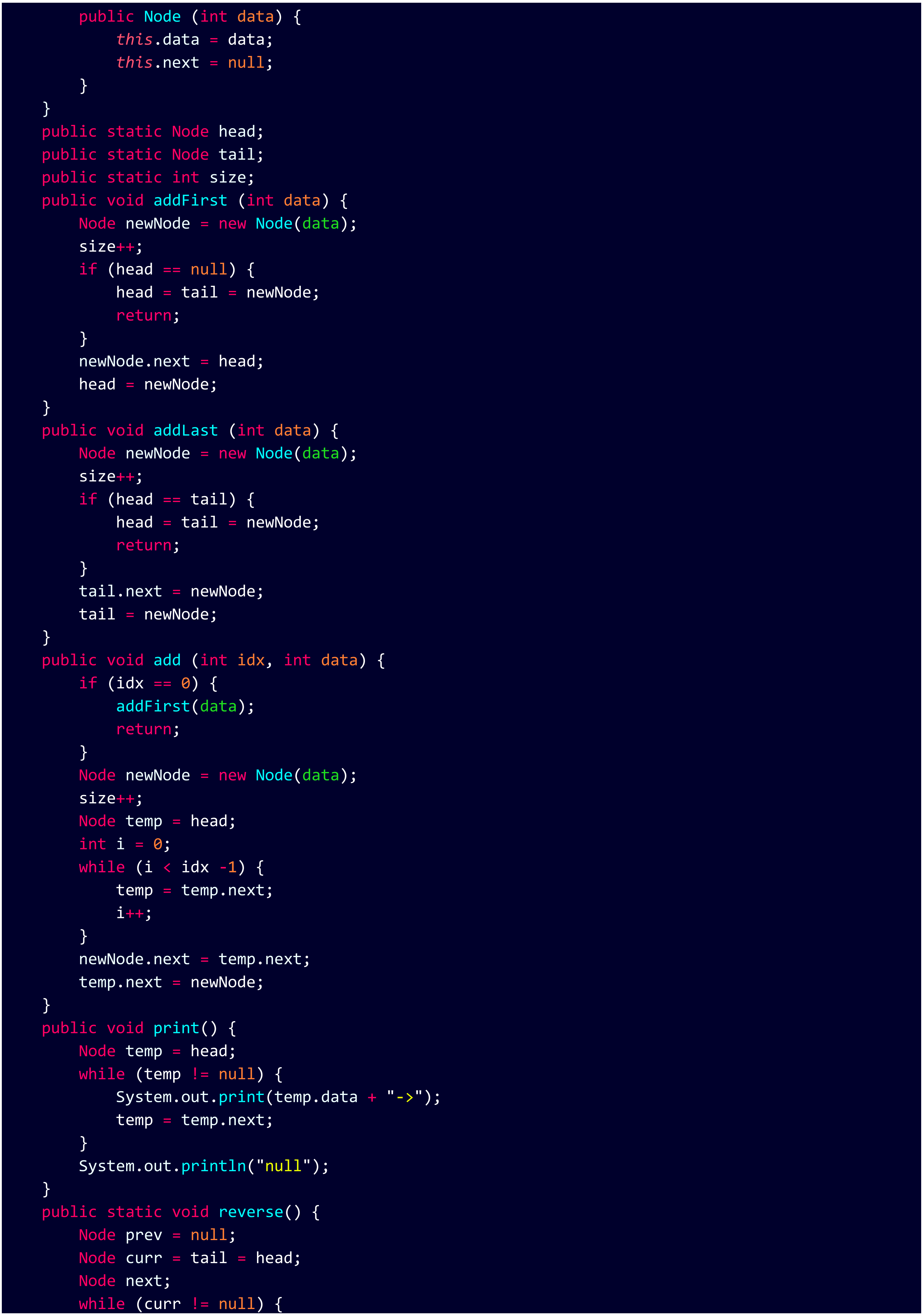
```
        }
    }
    public static Node head;
    public static Node tail;
    public static int size;
    public void addFirst (int data) {
        Node newNode = new Node(data);
        size++;
        if (head == null) {
            head = tail = newNode;
            return;
        }
        newNode.next = head;
        head = newNode;
    }
    public void addLast (int data) {
        Node newNode = new Node(data);
        size++;
        if (head == tail) {
            head = tail = newNode;
            return;
        }
        tail.next = newNode;
        tail = newNode;
    }
    public void add (int idx, int data) {
        if (idx == 0) {
            addFirst(data);
            return;
        }
        Node newNode = new Node(data);
        size++;
        Node temp = head;
        int i = 0;
        while (i < idx -1) {
            temp = temp.next;
            i++;
        }
        newNode.next = temp.next;
        temp.next = newNode;
    }
    public void print() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + "->");
            temp = temp.next;
        }
        System.out.println("null");
    }
    public void deleteNthFromEnd (int n) {
        // calculate size
        int size = 0;
        Node temp = head;
        while (temp != null) {
            temp = temp.next;
            size++;
        }
        if (n == size) {
            head = head.next;   //remove first
            return;
        }
        // size-n
        int i = 1;
        int iToFind = size - n;
        Node prev = head;
        while (i < iToFind) {
            prev = prev.next;
            i++;
        }
        prev.next = prev.next.next;
        return;
    }
    public static void main(String[] args) {
        RemoveNthNodeFromEnd ll = new RemoveNthNodeFromEnd();
        ll.addFirst(2);
        ll.addFirst(1);
        ll.addLast(4);
        ll.addLast(5);
        ll.add(2, 3);
        ll.print(); // 1->2->3->4->5->null
        ll.deleteNthFromEnd(3);
        ll.print(); // 1->2->4->5->null
    }
}
```

18. Reverse Doubly Linked List

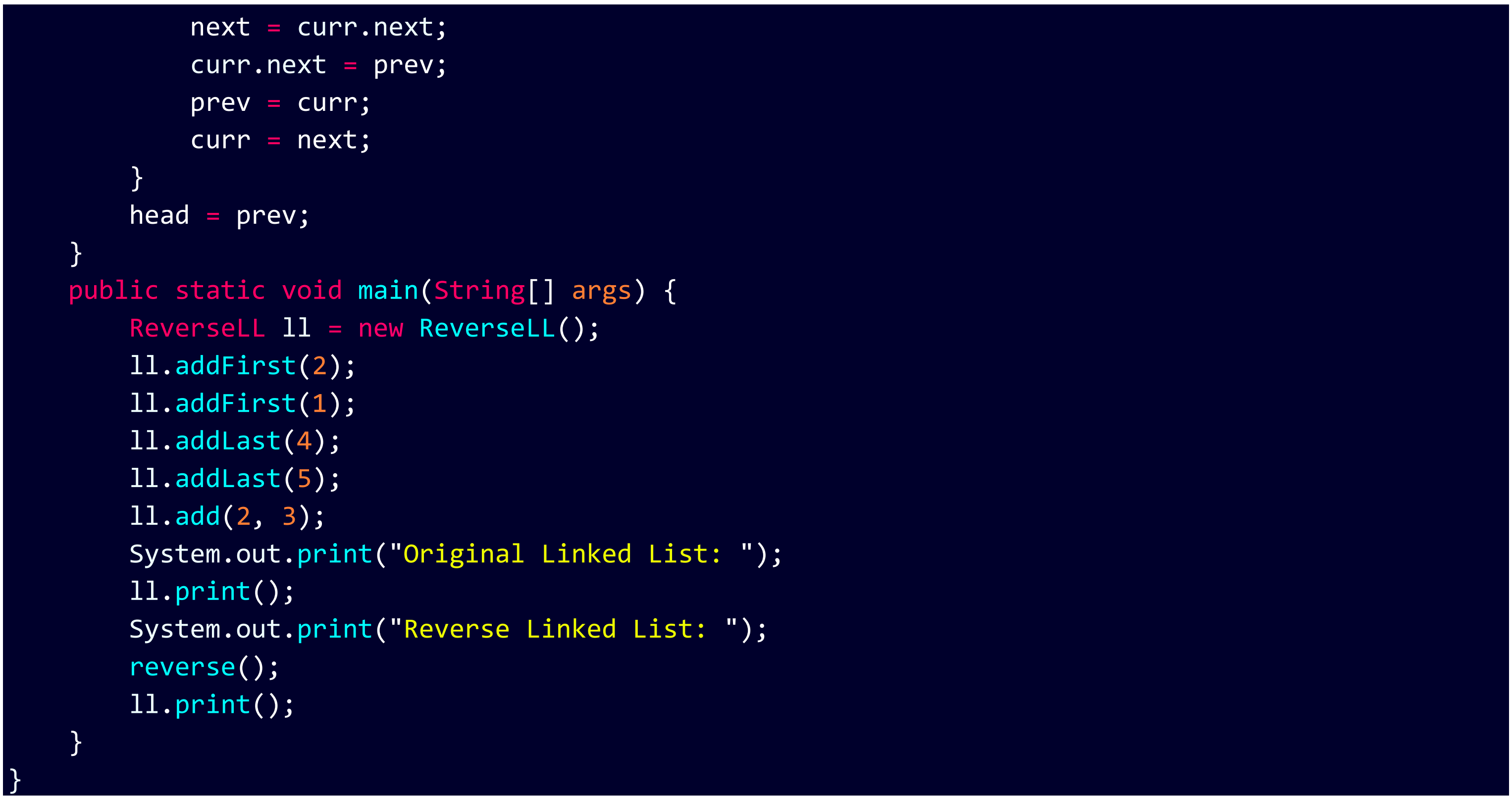
```
public class ReverseDLL {
    public class Node{
        int data;
        Node next;
        Node prev;
        public Node (int data) {
            this.data = data;
            this.next = null;
            this.prev = null;
        }
    }
    public static Node head;
    public static Node tail;
    public static int size;
    public void addFirst (int data) {
        Node newNode = new Node(data);
        size++;
        if (head == null) {
            head = tail = newNode;
            return;
        }
        newNode.next = head;
        head.prev = newNode;
        head = newNode;
    }
    public void addLast (int data) {
        Node newNode = new Node(data);
        size++;
        if (head == null) {
            head = tail = newNode;
            return;
        }
        tail.next = newNode;
        newNode.prev = tail;
        tail = newNode;
    }
    public void reverse() {
        Node curr = head;
        Node prev = null;
        Node next;
        while (curr != null) {
            next = curr.next;
            curr.next = prev;
            curr.prev = next;
            prev = curr;
            curr = next;
        }
        head = prev;
    }
    public void print() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + "<->");
            temp = temp.next;
        }
        System.out.println("null");
    }
    public static void main(String[] args) {
        ReverseDLL dll = new ReverseDLL();
        dll.addFirst(3);
        dll.addFirst(2);
        dll.addFirst(1);
        dll.addLast(4);
        dll.addLast(5);
        dll.addLast(6);
        System.out.print("Original Doubly LL: ");
        dll.print();
        System.out.println("Size = " + size);
        dll.reverse();
        System.out.print("Reversed Doubly LL: ");
        dll.print();
    }
}
```

19. Reverse Linked List

```
public class ReverseLL {
                                                // Iterative Approach
    public static class Node {
        int data;
        Node next;
```

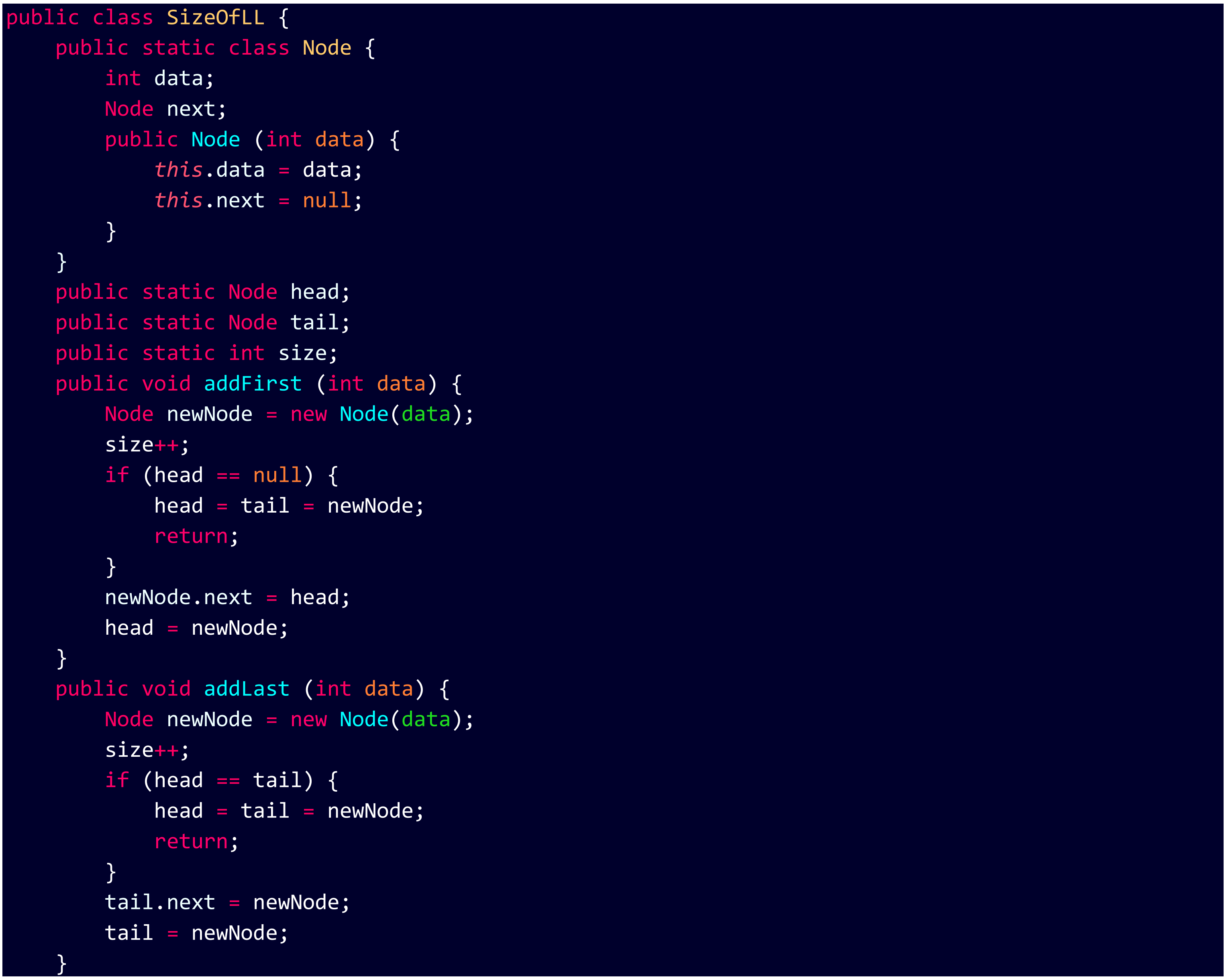


```
        public Node (int data) {
            this.data = data;
            this.next = null;
        }
    }
    public static Node head;
    public static Node tail;
    public static int size;
    public void addFirst (int data) {
        Node newNode = new Node(data);
        size++;
        if (head == null) {
            head = tail = newNode;
            return;
        }
        newNode.next = head;
        head = newNode;
    }
    public void addLast (int data) {
        Node newNode = new Node(data);
        size++;
        if (head == tail) {
            head = tail = newNode;
            return;
        }
        tail.next = newNode;
        tail = newNode;
    }
    public void add (int idx, int data) {
        if (idx == 0) {
            addFirst(data);
            return;
        }
        Node newNode = new Node(data);
        size++;
        Node temp = head;
        int i = 0;
        while (i < idx -1) {
            temp = temp.next;
            i++;
        }
        newNode.next = temp.next;
        temp.next = newNode;
    }
    public void print() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + "->");
            temp = temp.next;
        }
        System.out.println("null");
    }
    public static void reverse() {
        Node prev = null;
        Node curr = tail = head;
        Node next;
        while (curr != null) {
```

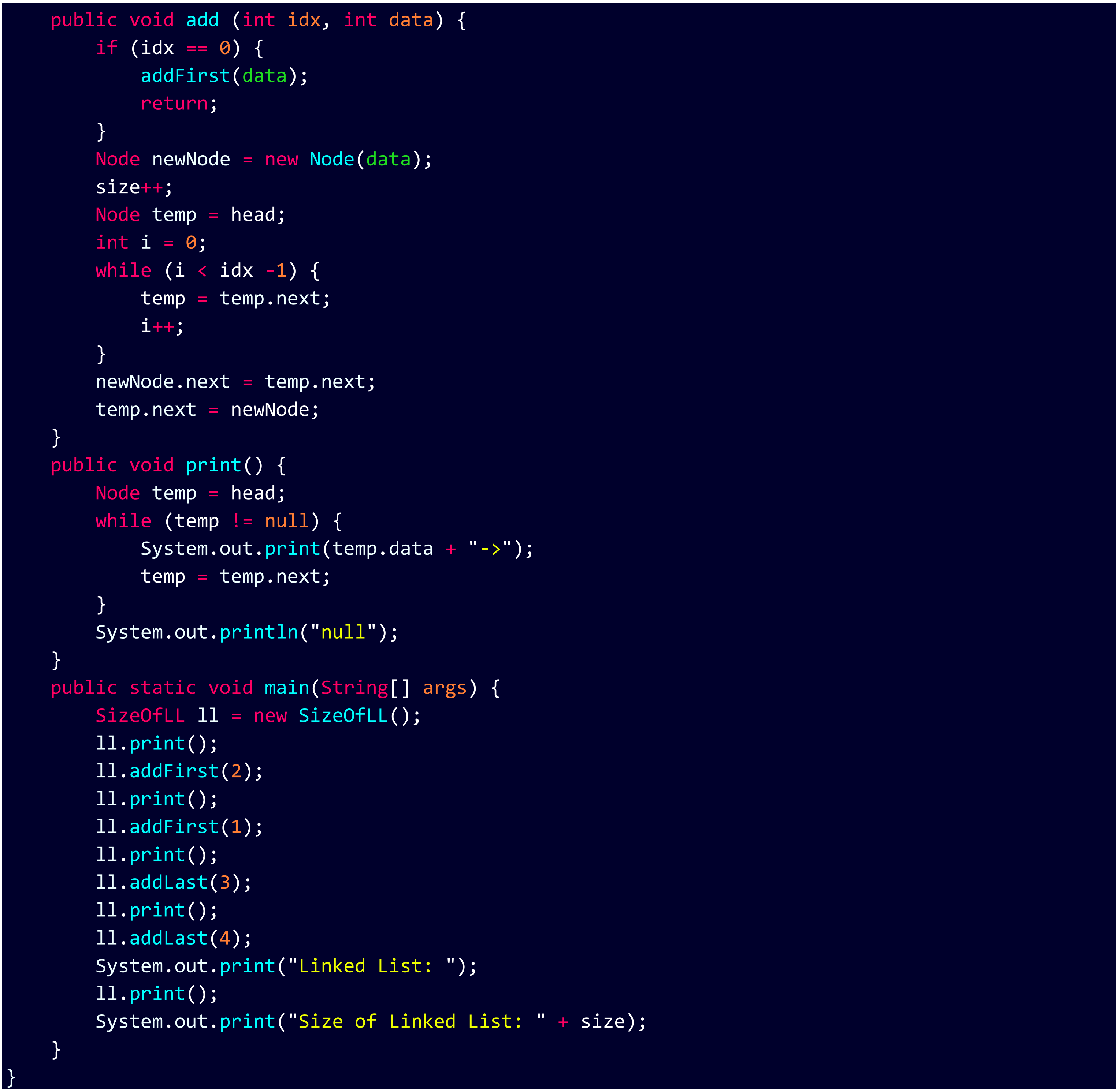


```java
            next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }
        head = prev;
    }
    public static void main(String[] args) {
        ReverseLL ll = new ReverseLL();
        ll.addFirst(2);
        ll.addFirst(1);
        ll.addLast(4);
        ll.addLast(5);
        ll.add(2, 3);
        System.out.print("Original Linked List: ");
        ll.print();
        System.out.print("Reverse Linked List: ");
        reverse();
        ll.print();
    }
}
```

20. Size of Linked List

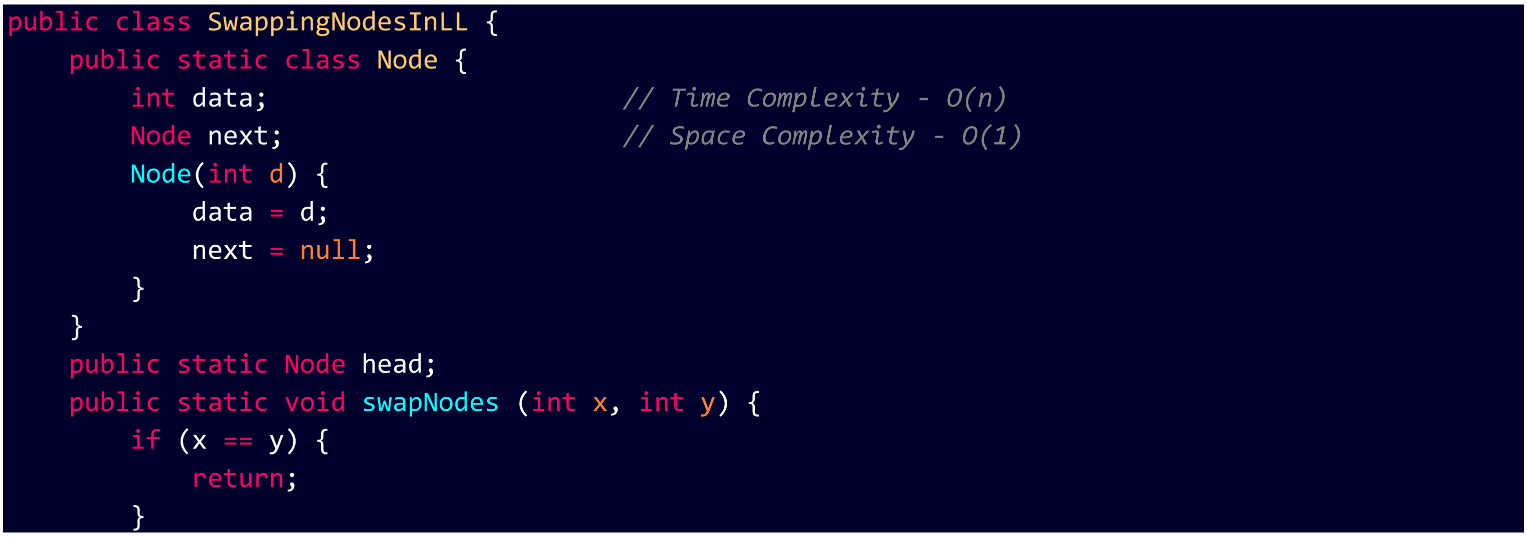


```java
public class SizeOfLL {
    public static class Node {
        int data;
        Node next;
        public Node (int data) {
            this.data = data;
            this.next = null;
        }
    }
    public static Node head;
    public static Node tail;
    public static int size;
    public void addFirst (int data) {
        Node newNode = new Node(data);
        size++;
        if (head == null) {
            head = tail = newNode;
            return;
        }
        newNode.next = head;
        head = newNode;
    }
    public void addLast (int data) {
        Node newNode = new Node(data);
        size++;
        if (head == tail) {
            head = tail = newNode;
            return;
        }
        tail.next = newNode;
        tail = newNode;
    }
```

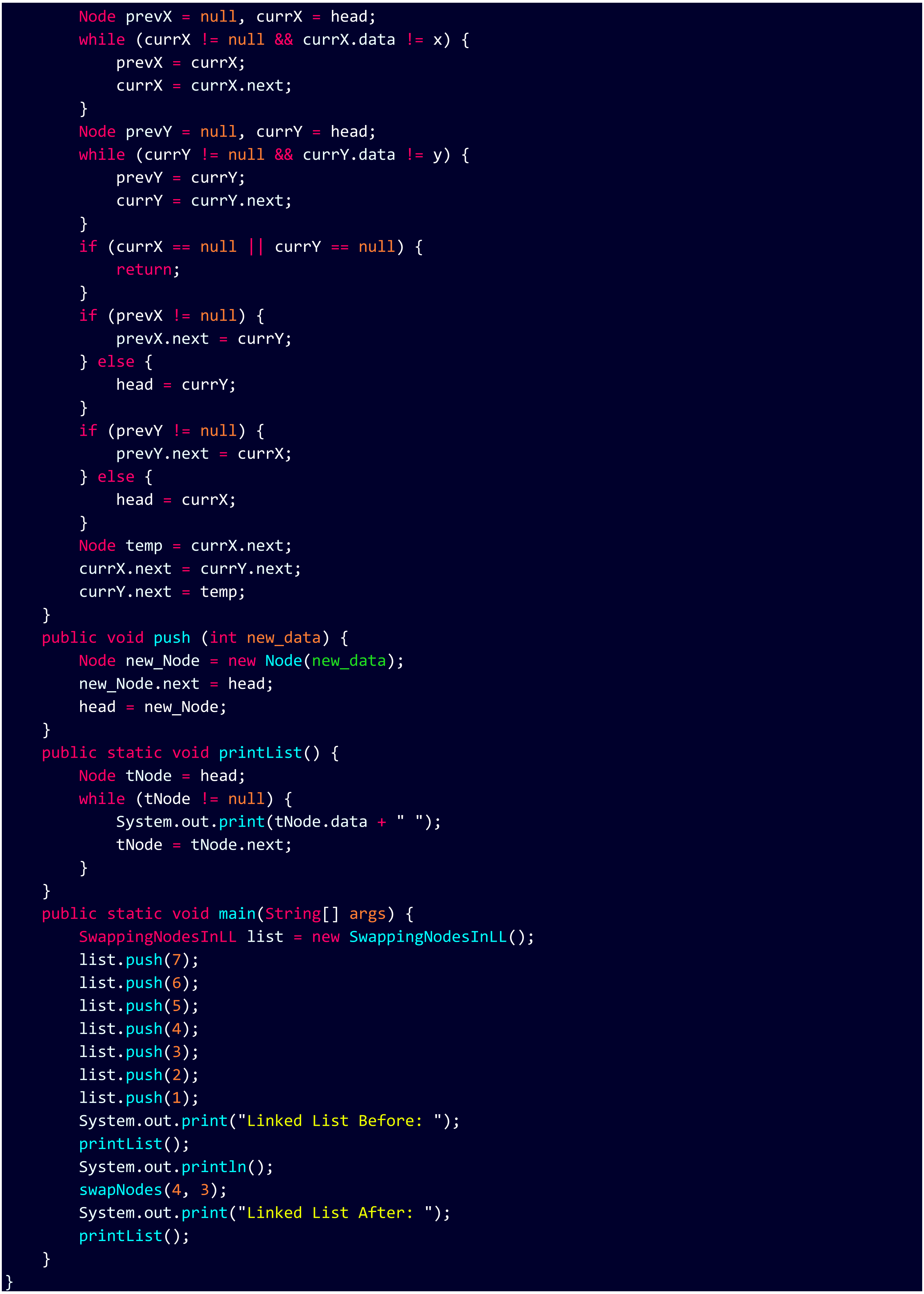


```java
    public void add (int idx, int data) {
        if (idx == 0) {
            addFirst(data);
            return;
        }
        Node newNode = new Node(data);
        size++;
        Node temp = head;
        int i = 0;
        while (i < idx -1) {
            temp = temp.next;
            i++;
        }
        newNode.next = temp.next;
        temp.next = newNode;
    }
    public void print() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + "->");
            temp = temp.next;
        }
        System.out.println("null");
    }
    public static void main(String[] args) {
        SizeOfLL ll = new SizeOfLL();
        ll.print();
        ll.addFirst(2);
        ll.print();
        ll.addFirst(1);
        ll.print();
        ll.addLast(3);
        ll.print();
        ll.addLast(4);
        System.out.print("Linked List: ");
        ll.print();
        System.out.print("Size of Linked List: " + size);
    }
}
```

21. Swapping Nodes in Linked List



```java
public class SwappingNodesInLL {
    public static class Node {
        int data;                               // Time Complexity - O(n)
        Node next;                              // Space Complexity - O(1)
        Node(int d) {
            data = d;
            next = null;
        }
    }
    public static Node head;
    public static void swapNodes (int x, int y) {
        if (x == y) {
            return;
        }
```



```
        Node prevX = null, currX = head;
        while (currX != null && currX.data != x) {
            prevX = currX;
            currX = currX.next;
        }
        Node prevY = null, currY = head;
        while (currY != null && currY.data != y) {
            prevY = currY;
            currY = currY.next;
        }
        if (currX == null || currY == null) {
            return;
        }
        if (prevX != null) {
            prevX.next = currY;
        } else {
            head = currY;
        }
        if (prevY != null) {
            prevY.next = currX;
        } else {
            head = currX;
        }
        Node temp = currX.next;
        currX.next = currY.next;
        currY.next = temp;
    }
    public void push (int new_data) {
        Node new_Node = new Node(new_data);
        new_Node.next = head;
        head = new_Node;
    }
    public static void printList() {
        Node tNode = head;
        while (tNode != null) {
            System.out.print(tNode.data + " ");
            tNode = tNode.next;
        }
    }
    public static void main(String[] args) {
        SwappingNodesInLL list = new SwappingNodesInLL();
        list.push(7);
        list.push(6);
        list.push(5);
        list.push(4);
        list.push(3);
        list.push(2);
        list.push(1);
        System.out.print("Linked List Before: ");
        printList();
        System.out.println();
        swapNodes(4, 3);
        System.out.print("Linked List After: ");
        printList();
    }
}
```

22. Zig Zag in Linked List

```
public class ZigZagLL {
    public static class Node {
        int data;
        Node next;
        public Node (int data) {
            this.data = data;
            this.next = null;
        }
    }
    public static Node head;
    public static Node tail;
    public static int size;
    public void addFirst (int data) {
        Node newNode = new Node(data);
        size++;
        if (head == null) {
            head = tail = newNode;
            return;
        }
        newNode.next = head;
        head = newNode;
    }
    public void addLast (int data) {
        Node newNode = new Node(data);
        size++;
        if (head == tail) {
            head = tail = newNode;
            return;
        }
        tail.next = newNode;
        tail = newNode;
    }
    public void print() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + "->");
            temp = temp.next;
        }
        System.out.println("null");
    }
    public static void zigZag() {
        // find mid
        Node slow = head;
        Node fast = head.next;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        Node mid = slow;
        // reverse 2nd half
        Node curr = mid.next;
        mid.next = null;
        Node prev = null;
        Node next;
        while (curr != null) {
            next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }
        Node left = head;
        Node right = prev;
        Node nextL, nextR;
        while (left != null && right != null) {
            nextL = left.next;
            left.next = right;
            nextR = right.next;
            right.next = nextL;
            left = nextL;
            right = nextR;
        }
    }
    public static void main(String[] args) {
        ZigZagLL ll = new ZigZagLL();
        ll.addFirst(3);
        ll.addFirst(2);
        ll.addFirst(1);
        ll.addLast(4);
        ll.addLast(5);
        ll.addLast(6);
        System.out.print("Original Linnked List: ");
        ll.print();
        zigZag();
        System.out.print("Zig-Zag Linked List: ");
        ll.print();
    }
}
```