|  |  |
| --- | --- |
| **OddEven in C++** | |
| #include <iostream>  using namespace std;  // Node class definition  class Node {  public:  int data;  Node\* next;  Node(int val) {  data = val;  next = nullptr;  }  };  // LinkedList class definition  class LinkedList {  public:  Node\* head;  Node\* tail;  int size;  LinkedList() {  head = nullptr;  tail = nullptr;  size = 0;  }  // Method to add a node at the end of the list  void addLast(int val) {  Node\* newNode = new Node(val);  if (size == 0) {  head = tail = newNode;  } else {  tail->next = newNode;  tail = newNode;  }  size++;  }  // Method to display the elements of the list  void display() {  Node\* temp = head;  while (temp != nullptr) {  cout << temp->data << " ";  temp = temp->next;  }  cout << endl;  }  // Method to remove the first node from the list  void removeFirst() {  if (size == 0) {  cout << "List is empty" << endl;  } else if (size == 1) {  head = tail = nullptr;  size = 0;  } else {  head = head->next;  size--;  }  }  // Method to get the data of the first node  int getFirst() {  if (size == 0) {  cout << "List is empty" << endl;  return -1;  } else {  return head->data;  }  }  // Method to add a node at the beginning of the list  void addFirst(int val) {  Node\* newNode = new Node(val);  newNode->next = head;  head = newNode;  if (size == 0) {  tail = newNode;  }  size++;  }  // Method to segregate odd and even nodes in the list  void oddEven() {  LinkedList odd;  LinkedList even;  while (size > 0) {  int val = getFirst();  removeFirst();  if (val % 2 == 0) {  even.addLast(val);  } else {  odd.addLast(val);  }  }  if (odd.size > 0 && even.size > 0) {  odd.tail->next = even.head;  head = odd.head;  tail = even.tail;  size = odd.size + even.size;  } else if (odd.size > 0) {  head = odd.head;  tail = odd.tail;  size = odd.size;  } else if (even.size > 0) {  head = even.head;  tail = even.tail;  size = even.size;  }  }  };  int main() {  // Initialize LinkedList  LinkedList l1;  // Add elements to the LinkedList  l1.addLast(2);  l1.addLast(8);  l1.addLast(9);  l1.addLast(1);  l1.addLast(5);  l1.addLast(4);  l1.addLast(3);  // Display original list  cout << "Original List: ";  l1.display();  // Perform odd-even segregation  l1.oddEven();  // Display list after odd-even segregation  cout << "List after Odd-Even Segregation: ";  l1.display();  // Add elements at the beginning and end  int a = 10;  int b = 100;  l1.addFirst(a);  l1.addLast(b);  // Display list after adding elements  cout << "List after adding " << a << " at the beginning and " << b << " at the end: ";  l1.display();  return 0;  } | **Initial List:**  Original List: 2 -> 8 -> 9 -> 1 -> 5 -> 4 -> 3  **🔄 Dry Run Table for oddEven() Method**  We’ll track how elements are moved to either the **odd** or **even** list.   | **Step** | **Current Node (val)** | **Is Even?** | **Action** | **Odd List** | **Even List** | | --- | --- | --- | --- | --- | --- | | 1 | 2 | ✅ Yes | Add to Even |  | 2 | | 2 | 8 | ✅ Yes | Add to Even |  | 2 -> 8 | | 3 | 9 | ❌ No | Add to Odd | 9 | 2 -> 8 | | 4 | 1 | ❌ No | Add to Odd | 9 -> 1 | 2 -> 8 | | 5 | 5 | ❌ No | Add to Odd | 9 -> 1 -> 5 | 2 -> 8 | | 6 | 4 | ✅ Yes | Add to Even | 9 -> 1 -> 5 | 2 -> 8 -> 4 | | 7 | 3 | ❌ No | Add to Odd | 9 -> 1 -> 5 -> 3 | 2 -> 8 -> 4 |   **🧩 Reconnecting Lists**   * Since **both odd and even lists exist**, we connect:   + odd.tail->next = even.head   + New head = odd.head   + New tail = even.tail   + New size = odd.size + even.size = 4 + 3 = 7   **🟢 Result after oddEven():**  List after Odd-Even Segregation: 9 -> 1 -> 5 -> 3 -> 2 -> 8 -> 4  **➕ Add 10 at beginning, 100 at end:**   * After addFirst(10): 10 -> 9 -> 1 -> 5 -> 3 -> 2 -> 8 -> 4 * After addLast(100): 10 -> 9 -> 1 -> 5 -> 3 -> 2 -> 8 -> 4 -> 100   **✅ Final Output:**  List after adding 10 at the beginning and 100 at the end: 10 -> 9 -> 1 -> 5 -> 3 -> 2 -> 8 -> 4 -> 100 |
| Output:- List after adding 10 at the beginning and 100 at the end: 10 -> 9 -> 1 -> 5 -> 3 -> 2 -> 8 -> 4 -> 100 | |