|  |  |
| --- | --- |
| **K Reverse in C++** | |
| #include <iostream>  using namespace std;  // Node class definition  class Node {  public:  int data;  Node\* next;  // Constructor  Node(int d) {  data = d;  next = nullptr;  }  };  // LinkedList class definition  class LinkedList {  private:  Node\* head;  Node\* tail;  int size;  public:  // Constructor  LinkedList() {  head = nullptr;  tail = nullptr;  size = 0;  }  // Method to add a node at the beginning of the list  void addFirst(int val) {  Node\* temp = new Node(val);  temp->next = head;  head = temp;  if (size == 0) {  tail = temp;  }  size++;  }  // Method to add a node at the end of the list  void addLast(int val) {  Node\* temp = new Node(val);  if (size == 0) {  head = tail = temp;  } else {  tail->next = temp;  tail = temp;  }  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 {  Node\* temp = head;  head = head->next;  delete temp;  size--;  if (size == 0) {  tail = nullptr;  }  }  }  // Method to get the first element of the list  int getFirst() {  if (size == 0) {  cout << "List is empty" << endl;  return -1;  } else {  return head->data;  }  }  // Method to reverse every k nodes in the list  void kReverse(int k) {  LinkedList prev;    while (size > 0) {  LinkedList curr;  if (size >= k) {  for (int i = 0; i < k; i++) {  int val = getFirst();  removeFirst();  curr.addFirst(val);  }  } else {  int sz = size;  for (int i = 0; i < sz; i++) {  int val = getFirst();  removeFirst();  curr.addLast(val);  }  }  if (prev.size == 0) {  prev = curr;  } else {  tail->next = curr.head;  tail = curr.tail;  size += curr.size;  }  }  head = prev.head;  tail = prev.tail;  size = prev.size;  }  // Destructor to free memory  ~LinkedList() {  Node\* curr = head;  while (curr != nullptr) {  Node\* temp = curr;  curr = curr->next;  delete temp;  }  }  };  // Main function to demonstrate LinkedList operations  int main() {  LinkedList l1;  l1.addLast(1);  l1.addLast(2);  l1.addLast(3);  l1.addLast(4);  l1.addLast(5);  l1.addLast(6);  l1.addLast(7);  l1.addLast(8);  l1.addLast(9);  l1.addLast(10);  l1.addLast(11);  int k = 3;  int a = 100;  int b = 200;  l1.display(); // Original list: 1 2 3 4 5 6 7 8 9 10 11  l1.kReverse(k); // Reverse every k nodes  l1.display(); // After kReverse: 3 2 1 6 5 4 9 8 7 10 11  l1.addFirst(a); // Add element at the beginning: 100 3 2 1 6 5 4 9 8 7 10 11  l1.addLast(b); // Add element at the end: 100 3 2 1 6 5 4 9 8 7 10 11 200  l1.display(); // Final list  return 0;  } | Initial Input: List: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 11 k = 3 🔁 kReverse Logic Dry Run: We reverse **groups of 3 elements**. Let's track the changes in a **table** as each k-group is processed:   | **Group #** | **Extracted Nodes** | **Reversed Order** | **prev List After Merge** | | --- | --- | --- | --- | | 1 | 1 2 3 | 3 2 1 | 3 → 2 → 1 | | 2 | 4 5 6 | 6 5 4 | 3 → 2 → 1 → 6 → 5 → 4 | | 3 | 7 8 9 | 9 8 7 | 3 → 2 → 1 → 6 → 5 → 4 → 9 → 8 → 7 | | 4 | 10 11 | (unchanged) | ... → 9 → 8 → 7 → 10 → 11 |  🔃 After kReverse: **List:** 3 → 2 → 1 → 6 → 5 → 4 → 9 → 8 → 7 → 10 → 11 |
| 1 2 3 4 5 6 7 8 9 10 11 | |