# C++ Linked List Operations

#include <iostream>  
using namespace std;  
  
struct Node {  
 int data;  
 Node\* next;  
 Node\* prev;  
 Node(int val) : data(val), next(nullptr), prev(nullptr) {}  
};  
  
void makenull(Node\*& head) {  
 while (head) {  
 Node\* temp = head;  
 head = head->next;  
 delete temp;  
 }  
}  
  
Node\* insertSortedSingly(Node\* head, int val) {  
 Node\* newNode = new Node(val);  
 if (!head || head->data >= val) {  
 newNode->next = head;  
 return newNode;  
 }  
 Node\* cur = head;  
 while (cur->next && cur->next->data < val) {  
 cur = cur->next;  
 }  
 newNode->next = cur->next;  
 cur->next = newNode;  
 return head;  
}  
  
Node\* insertSortedDoubly(Node\* head, int val) {  
 Node\* newNode = new Node(val);  
 if (!head || head->data >= val) {  
 newNode->next = head;  
 if (head) head->prev = newNode;  
 return newNode;  
 }  
 Node\* cur = head;  
 while (cur->next && cur->next->data < val) {  
 cur = cur->next;  
 }  
 newNode->next = cur->next;  
 newNode->prev = cur;  
 if (cur->next) cur->next->prev = newNode;  
 cur->next = newNode;  
 return head;  
}  
  
Node\* mergeLists(Node\* L1, Node\* L2) {  
 if (!L1) return L2;  
 if (!L2) return L1;  
 if (L1->data > L2->data) swap(L1, L2);  
 Node\* head = L1, \*tail = L1;  
 L1 = L1->next;  
   
 while (L1 && L2) {  
 if (L1->data <= L2->data) {  
 tail->next = L1;  
 tail = L1;  
 L1 = L1->next;  
 } else {  
 tail->next = L2;  
 tail = L2;  
 L2 = L2->next;  
 }  
 }  
 if (L1) tail->next = L1;  
 if (L2) tail->next = L2;  
 return head;  
}  
  
Node\* swapAdjacentSingly(Node\* head) {  
 if (!head || !head->next) return head;  
 Node\* prev = nullptr, \*cur = head, \*newHead = head->next;  
   
 while (cur && cur->next) {  
 Node\* next = cur->next;  
 cur->next = next->next;  
 next->next = cur;  
 if (prev) prev->next = next;  
 prev = cur;  
 cur = cur->next;  
 }  
 return newHead;  
}  
  
Node\* swapAdjacentDoubly(Node\* head) {  
 if (!head || !head->next) return head;  
 Node\* cur = head, \*newHead = head->next;  
  
 while (cur && cur->next) {  
 Node\* next = cur->next;  
 cur->next = next->next;  
 if (next->next) next->next->prev = cur;  
 next->prev = cur->prev;  
 if (cur->prev) cur->prev->next = next;  
 cur->prev = next;  
 next->next = cur;  
 cur = cur->next;  
 }  
 return newHead;  
}  
  
void printList(Node\* head) {  
 while (head) {  
 cout << head->data << " ";  
 head = head->next;  
 }  
 cout << endl;  
}  
  
int main() {  
 /\*  
 Node\* singlyList = nullptr;  
 singlyList = insertSortedSingly(singlyList, 3);  
 singlyList = insertSortedSingly(singlyList, 1);  
 singlyList = insertSortedSingly(singlyList, 5);  
 singlyList = insertSortedSingly(singlyList, 2);  
 cout << "Sorted Singly List: ";  
 printList(singlyList);  
 makenull(singlyList);  
  
 Node\* doublyList = nullptr;  
 doublyList = insertSortedDoubly(doublyList, 4);  
 doublyList = insertSortedDoubly(doublyList, 2);  
 doublyList = insertSortedDoubly(doublyList, 5);  
 doublyList = insertSortedDoubly(doublyList, 1);  
 cout << "Sorted Doubly List: ";  
 printList(doublyList);  
 makenull(doublyList);  
  
 Node\* L1 = new Node(1);  
 L1->next = new Node(3);  
 L1->next->next = new Node(5);  
   
 Node\* L2 = new Node(2);  
 L2->next = new Node(4);  
 L2->next->next = new Node(6);  
  
 Node\* merged = mergeLists(L1, L2);  
 cout << "Merged List: ";  
 printList(merged);  
 makenull(merged);  
  
 Node\* singlySwapList = new Node(1);  
 singlySwapList->next = new Node(2);  
 singlySwapList->next->next = new Node(3);  
 singlySwapList->next->next->next = new Node(4);  
  
 singlySwapList = swapAdjacentSingly(singlySwapList);  
 cout << "Swapped Adjacent Singly List: ";  
 printList(singlySwapList);  
 makenull(singlySwapList);  
  
 Node\* doublySwapList = new Node(1);  
 doublySwapList->next = new Node(2);  
 doublySwapList->next->prev = doublySwapList;  
 doublySwapList->next->next = new Node(3);  
 doublySwapList->next->next->prev = doublySwapList->next;  
 doublySwapList->next->next->next = new Node(4);  
 doublySwapList->next->next->next->prev = doublySwapList->next->next;  
  
 doublySwapList = swapAdjacentDoubly(doublySwapList);  
 cout << "Swapped Adjacent Doubly List: ";  
 printList(doublySwapList);  
 makenull(doublySwapList);  
 \*/  
  
 return 0;  
}