# Linked List Labs — Solutions A through L

Author: ChatGPT (prepared for Nuri)  
  
Purpose: consolidated explanations, code and notes for problems A..L.

## A. Nearest number

Summary:  
Find index (0-based) of element closest to k by absolute difference. If many, pick earliest.

Approach & key points:  
- Build a singly linked list from input values (head/tail pattern).  
- Traverse once, keep the minimal absolute difference and its first position.  
- Complexity: O(n) time, O(n) nodes memory.

#include <bits/stdc++.h>  
  
using namespace std;  
  
struct Node {  
 int data;  
 Node\* next;  
 Node(int x) : data(x), next(NULL){}  
};  
   
int main() {  
   
 int n, k, pos, mn = INT\_MAX, cnt = 0, x;  
   
 cin >> n;  
   
 Node \*head = NULL, \*tail = NULL;  
   
 for(int i = 1; i <= n; i++){  
 cin >> x;  
 Node\* node = new Node(x);  
 if(head == NULL) head = tail = node;  
 else {  
 tail->next = node;  
 tail = node;  
 }  
 }  
   
 Node\* cur = head;  
   
 cin >> k;  
   
 while(cur != NULL){  
 if(abs(cur->data - k) < mn){  
 mn = abs(cur->data - k);  
 pos = cnt;  
 }  
 cur = cur->next;  
 cnt++;  
 }  
 cout << pos;  
   
 return 0;  
}

## B. Jonathan the Poet

Summary:  
Perform cyclic left shift of a list of strings by k positions and print the result.

Approach & key points:  
- Build list, then output nodes from index k onward, followed by first k nodes.  
- Caveat: if k >= n, do k %= n for robustness.  
- Complexity: O(n).

#include <bits/stdc++.h>  
  
using namespace std;  
  
struct Node {  
 string data;  
 Node\* next;  
 Node(string x) : data(x), next(NULL){}  
};  
   
int main() {  
   
 int n, k, cnt = 0;  
 string s;  
   
 cin >> n >> k;  
   
 Node \*head = NULL, \*tail = NULL;  
   
 for(int i = 1; i <= n; i++){  
 cin >> s;  
 Node\* node = new Node(s);  
 if(head == NULL) head = tail = node;  
 else {  
 tail->next = node;  
 tail = node;  
 }  
 }  
 Node\* cur = head;  
 while(cur != NULL){  
 if(cnt >= k) cout << cur->data << ' ';  
 cur = cur->next;  
 cnt++;  
 }  
 cnt = 0; cur = head;  
 while(cnt < k) {  
 cout << cur->data << ' ';  
 cur = cur->next;  
 cnt++;  
 }  
   
 return 0;  
}

## C. Kuanyshbek

Summary:  
Given N elements produced by two threads interleaved, remove every second element (the second thread's writes) and print the rest.

Approach & key points:  
- Build a list containing only elements at odd positions (1-based) while reading.  
- This avoids explicit removal and is simple and efficient for constraints.  
- Complexity: O(n).

#include <bits/stdc++.h>  
  
using namespace std;  
  
struct Node {  
 int data;  
 Node\* next;  
 Node(int x) : data(x), next(NULL){}  
};  
   
int main() {  
   
 int n, x;  
   
 cin >> n;  
   
 Node \*head = NULL, \*tail = NULL;  
   
 for(int i = 1; i <= n; i++){  
 cin >> x;  
 Node\* node = new Node(x);  
 if(head == NULL) head = tail = node;  
 else {  
 if(i % 2 == 1) {  
 tail->next = node;  
 tail = node;  
 }  
 }  
 }  
   
 Node\* cur = head;  
   
 while(cur != NULL){  
 cout << cur->data << ' ';  
 cur = cur->next;  
 }  
   
 return 0;  
}

## D. List modes

Summary:  
Print the number(s) that occur most frequently (mode). If multiple modes, print them in descending order.

Approach & key points:  
- Use a frequency map while reading input and a linked list to store values.  
- Track maximal frequency mx during input.  
- Traverse the list and insert values with frequency == mx into a set to deduplicate and then print set in reverse order.  
- Complexity: O(n log n) overall (map + set operations).

#include <bits/stdc++.h>  
  
using namespace std;  
  
struct Node {  
 int val;  
 Node\* next;  
 Node(int x) : val(x), next(NULL){}  
};  
   
int main() {  
   
 int n, x, mx = 0;  
 map<int, int> mp;  
   
 cin >> n;  
   
 Node \*head = NULL, \*tail = NULL;  
   
 for(int i = 1; i <= n; i++){  
 cin >> x;  
 mp[x]++, mx = max(mx, mp[x]);  
 Node\* node = new Node(x);  
 if(head == NULL) head = tail = node;  
 else {  
 tail->next = node;  
 tail = node;  
 }  
 }  
   
 Node\* cur = head;  
 set<int> st;  
   
 while(cur != NULL){  
 if(mp[cur->val] == mx) st.insert(cur->val);  
 cur = cur->next;  
 }  
 for(auto it = st.rbegin(); it != st.rend(); it++) cout << \*it << ' ';  
   
   
 return 0;  
}

## E. Database (remove consecutive duplicates)

Summary:  
Names are given; duplicates of the same name occupy consecutive positions. Remove consecutive duplicates and print the count and the names. Implementation prints unique names in reverse order (tail→head).

Approach & key points:  
- Build a doubly linked list, increment count only when name differs from previous tail.  
- Print in reverse from tail, skipping neighboring equal names.  
- Complexity: O(n).

#include <bits/stdc++.h>  
  
using namespace std;  
  
struct Node {  
 string val;  
 Node\* next;  
 Node\* prev;  
 Node(string x) : val(x), next(NULL), prev(NULL){}  
};  
   
int main() {  
   
 int n, cnt = 1;  
 string s;  
   
 cin >> n;  
   
 Node \*head = NULL, \*tail = NULL;  
   
 for(int i = 1; i <= n; i++){  
 cin >> s;  
 Node\* node = new Node(s);  
 if(head == NULL) head = tail = node;  
 else {  
 if(s != tail->val) cnt++;  
 tail->next = node;  
 node->prev = tail;  
 tail = node;  
   
 }  
 }  
 cout << "All in all: " << cnt << '\n' << "Students:\n";  
 Node\* cur = tail;  
 while(cur != NULL){  
 cout << cur->val << '\n';  
 while(cur->prev != NULL && cur->val == cur->prev->val){  
 cur = cur->prev;  
 }  
 cur = cur->prev;  
 }  
   
 return 0;  
}

## F. Insertion of Node

Summary:  
Insert a new node with given value at a given position (0-based). Print resulting list.

Approach & key points:  
- If pos==0 set new node as head; otherwise traverse to pos-1 and insert.  
- Complexity O(n).

#include <bits/stdc++.h>  
  
using namespace std;  
  
struct Node {  
 int val;  
 Node\* next;  
 Node(int x) : val(x), next(NULL){}  
};  
   
int main() {  
   
 int n, x, pos, var;  
   
 cin >> n;  
   
 Node \*head = NULL, \*tail = NULL;  
   
 for(int i = 1; i <= n; i++){  
 cin >> x;  
 Node\* node = new Node(x);  
 if(head == NULL) head = tail = node;  
 else {  
 tail->next = node;  
 tail = node;  
 }  
 }  
 cin >> var >> pos;  
  
 Node\* new\_node = new Node(var);  
   
 if(pos == 0){  
 new\_node->next = head;  
 head = new\_node;  
 } else {  
 int cnt = 0;  
 Node\* cur = head;  
 while(++cnt != pos)  
 cur = cur->next;  
 new\_node->next = cur->next;  
 cur->next = new\_node;  
 }  
 Node\* cur = head;  
 while(cur != NULL){  
 cout << cur->val << ' ';  
 cur = cur->next;  
 }  
   
 return 0;  
}

## H. Zoro and Seven Sword Style

Summary:  
Interactive processor for many linked-list operations (insert/remove/print/replace/reverse/cyclic\_left/cyclic\_right).

Approach & key points:  
- Implement helper functions that receive head and return new head where necessary.  
- Carefully manipulate pointers and edge cases.  
- Complexity: O(n) for most operations.

#include <bits/stdc++.h>  
   
using namespace std;  
   
struct Node{  
 int val;  
 Node\* next;  
 Node(): val(0), next(nullptr) {}  
 Node(int x): val(x), next(nullptr) {}  
 Node(Node\* next): val(0), next(next) {}  
 Node(int x, Node\* next): val(x), next(next) {}  
};  
   
Node\* insert(Node\* head, Node\* node, int p) {  
 if (p == 0) {  
 node->next = head;  
 return node;  
 }  
 Node\* cur = head;  
 for(int i = 1; i <= p - 1 && cur != NULL; i++) cur = cur->next;  
 node->next = cur->next;  
 cur->next = node;  
 return head;  
}  
  
Node\* remove(Node\* head, int p) {  
 if (!head) return NULL;  
 if (p == 0) {  
 Node\* tmp = head->next;  
 delete head;  
 return tmp;  
 }  
 Node\* cur = head;  
 for (int i = 1; i <= p - 1 && cur; i++) cur = cur->next;  
 Node\* tmp = cur->next;  
 cur->next = tmp->next;  
 delete tmp;  
 return head;  
}  
  
Node\* replace(Node\* head, int p1, int p2) {  
 if (!head || p1 == p2) return head;  
  
 Node\* prev = NULL;  
 Node\* cur1 = head;  
 for (int i = 1; i <= p1; i++) {  
 prev = cur1;  
 cur1 = cur1->next;  
 }  
  
 if(prev != NULL) prev->next = cur1->next;  
 else head = cur1->next;  
  
 Node\* node = cur1;  
  
 if(p2 == 0){  
 node->next = head;  
 return node;  
 }  
  
 Node\* cur2 = head;  
 for (int i = 0; i < p2 - 1; i++) cur2 = cur2->next;  
 node->next = cur2->next;  
 cur2->next = node;  
  
 return head;  
}  
  
Node\* reverse(Node\* head) {  
 Node\* prev = NULL;  
 Node\* cur = head;  
 while(cur){  
 Node\* nxt = cur->next;  
 cur->next = prev;  
 prev = cur;  
 cur = nxt;  
 }  
 return prev;  
}  
  
void print(Node\* head) {  
 if (!head) {  
 cout << -1 << endl;  
 return;  
 }  
 while (head) {  
 cout << head->val;  
 if (head->next) cout << " ";  
 head = head->next;  
 }  
 cout << endl;  
}  
  
Node\* cyclic\_left(Node\* head, int x) {  
 if (!head || !head->next) return head;  
  
 int len = 0;  
 Node\* tail = head;  
 while (tail->next) {  
 tail = tail->next;  
 len++;  
 }  
 len++;  
  
 x %= len;  
 if (x == 0) return head;  
  
 tail->next = head;  
 Node\* newTail = head;  
 for (int i = 0; i < x - 1; i++) newTail = newTail->next;  
 Node\* newHead = newTail->next;  
 newTail->next = NULL;  
  
 return newHead;  
}  
  
Node\* cyclic\_right(Node\* head, int x) {  
 if (!head || !head->next) return head;  
  
 int len = 0;  
 Node\* tail = head;  
 while (tail->next) {  
 tail = tail->next;  
 len++;  
 }  
 len++;  
  
 x %= len;  
 if (x == 0) return head;  
  
 return cyclic\_left(head, len - x);  
}  
  
   
int main(){  
 Node\* head = NULL;  
 while (true){  
 int command; cin >> command;  
 if (command == 0){  
 break;  
 }else if(command == 1){  
 int x, p; cin >> x >> p;  
 head = insert(head, new Node(x), p);  
 }else if (command == 2){  
 int p; cin >> p;  
 head = remove(head, p);  
 }else if (command == 3){  
 print(head);  
 }else if (command == 4){  
 int p1, p2; cin >> p1 >> p2;  
 head = replace(head, p1, p2);  
 }else if (command == 5){  
 head = reverse(head);  
 }else if (command == 6){  
 int x; cin >> x;  
 head = cyclic\_left(head, x);  
 }else if (command == 7){  
 int x; cin >> x;  
 head = cyclic\_right(head, x);  
 }   
 }  
 return 0;  
}

## I. Doubly linked list

Summary:  
Implement a deque-like doubly linked list storing book names supporting add\_front/add\_back/erase\_front/erase\_back/front/back/clear/exit.

Approach & key points:  
- Keep head/tail and count of elements for quick empty checks and O(1) operations at both ends.  
- On clear, deallocate nodes by repeated erase\_front.

#include <bits/stdc++.h>  
using namespace std;  
  
struct Node {  
 Node \*prev;  
 Node \*next;  
 string val;  
 Node(string \_val) {  
 prev = NULL;  
 next = NULL;  
 val = \_val;  
 }  
};  
  
Node \*head = NULL;  
Node \*tail = NULL;  
int cnt = 0;  
  
void add\_back(string s) {  
 Node \*node = new Node(s);  
 if (!head) {  
 head = tail = node;  
 } else {  
 tail->next = node;  
 node->prev = tail;  
 tail = node;  
 }  
 cnt++;  
}  
  
void add\_front(string s) {  
 Node \*node = new Node(s);  
 if (!head) {  
 head = tail = node;  
 } else {  
 node->next = head;  
 head->prev = node;  
 head = node;  
 }  
 cnt++;  
}  
  
bool empty() {  
 return cnt == 0;  
}  
  
void erase\_front() {  
 if (empty()) return;  
 Node \*tmp = head;  
 if (head == tail) {  
 head = tail = NULL;  
 } else {  
 head = head->next;  
 head->prev = NULL;  
 }  
 delete tmp;  
 cnt--;  
}  
  
void erase\_back() {  
 if (empty()) return;  
 Node \*tmp = tail;  
 if (head == tail) {  
 head = tail = NULL;  
 } else {  
 tail = tail->prev;  
 tail->next = NULL;  
 }  
 delete tmp;  
 cnt--;  
}  
  
string front() {  
 return head->val;  
}  
  
string back() {  
 return tail->val;  
}  
  
void clear() {  
 while (!empty()) {  
 erase\_front();  
 }  
}  
  
int main() {  
 string s;  
 while (cin >> s) {  
 if (s == "add\_front") {  
 string t;  
 cin >> t;  
 add\_front(t);  
 cout << "ok" << endl;  
 }  
 if (s == "add\_back") {  
 string t;  
 cin >> t;  
 add\_back(t);  
 cout << "ok" << endl;  
 }  
 if (s == "erase\_front") {  
 if (empty()) {  
 cout << "error" << endl;  
 } else {  
 cout << front() << endl;  
 erase\_front();  
 }  
 }  
 if (s == "erase\_back") {  
 if (empty()) {  
 cout << "error" << endl;  
 } else {  
 cout << back() << endl;  
 erase\_back();  
 }  
 }  
 if (s == "front") {  
 if (empty()) {  
 cout << "error" << endl;  
 } else {  
 cout << front() << endl;  
 }  
 }  
 if (s == "back") {  
 if (empty()) {  
 cout << "error" << endl;  
 } else {  
 cout << back() << endl;  
 }  
 }  
 if (s == "clear") {  
 clear();  
 cout << "ok" << endl;  
 }  
 if (s == "exit") {  
 cout << "goodbye" << endl;  
 break;  
 }  
 }  
 return 0;  
}

## K. One-time guests

Summary:  
After each char insertion, print the leftmost non-repeating char among all seen so far, or -1 if none.

Approach & key points:  
- The submitted solution keeps the stream in a linked list and uses a map of frequencies; after each insertion it scans from head to first char with freq==1.  
- Complexity O(n^2) worst-case; acceptable for small input sizes. Optimize using a queue for O(1) amortized.

#include <bits/stdc++.h>  
using namespace std;  
  
struct Node {  
 char val;  
 Node\* next;  
 Node(char x) : val(x), next(NULL){}   
};  
  
int cnt = 0;  
  
int main() {  
   
 int tt;  
   
 cin >> tt;  
   
 while(tt--){  
 int n; char c;  
 map<char, int> mp;  
 cin >> n;  
   
 Node \*head = NULL, \*tail = NULL;  
   
 for(int i = 1; i <= n; i++){  
 cin >> c;  
 Node\* node = new Node(c);  
 if(head == NULL) head = tail = node;  
 else {  
 tail->next = node;  
 tail = node;  
   
 }  
 Node\* cur = head;  
 mp[c]++;  
 while(cur != NULL && mp[cur->val] > 1) {  
 cur = cur->next;  
 }  
 if(cur == NULL) cout << -1 << ' ';  
 else cout << cur->val << ' ';  
 }  
   
 cout << endl;  
 }  
   
 return 0;  
}

## L. Ragnarok (max subarray sum)

Summary:  
Find the maximal sum of a contiguous subarray in a linked list. The provided solution uses nested loops (O(n^2)), but Kadane's algorithm gives O(n) on a single pass.

Approach & key points:  
- Provided nested approach computes all possible contiguous sums.  
- Recommended: implement Kadane on a linked list: iterate once maintaining current\_sum and max\_sum.

#include <bits/stdc++.h>  
  
using namespace std;  
  
struct Node {  
 int val;  
 Node \*next;  
  
 Node() {  
 val = 0;  
 next = NULL;  
 }  
};  
  
int mx = INT\_MIN, sum = 0;  
  
int findMaxSum(int n, Node \*head) {  
   
 for(int i = 0; i < n; i++){  
 Node\* cur = head;  
 head = head->next;  
 sum = 0;  
 while(cur != NULL){  
 sum += cur->val;  
 cur = cur->next;  
 mx = max(mx, sum);  
 }  
 }  
 return mx;  
   
}  
  
  
int main() {  
 int n;  
 cin >> n;  
  
 Node \*head, \*tail;  
 for (int i = 1; i <= n; ++i) {  
 int x;  
 cin >> x;  
 Node \*cur = new Node();  
 cur -> val = x;  
   
 if (i == 1) {  
 head = tail = cur;  
 } else {  
 tail -> next = cur;  
 tail = cur;  
 }  
 }  
   
 cout << findMaxSum(n, head) << "\n";  
 return 0;  
}