#include <iostream>

using namespace std;

struct node {

struct node\* prev;

int data;

struct node\* next;

};

void addtofront(struct node\*\* head, int data) {

struct node\* temp = new struct node;

temp->prev = NULL;

temp->data = data;

temp->next = \*head;

if (\*head != NULL) {

(\*head)->prev = temp;

}

\*head = temp;

}

void addtoend(struct node\*\* head, int data) {

struct node\* temp = new struct node;

struct node\* ptr = \*head;

temp->next = NULL;

temp->data = data;

if (\*head == NULL) {

temp->prev = NULL;

\*head = temp;

return;

}

while (ptr->next != NULL) {

ptr = ptr->next;

}

ptr->next = temp;

temp->prev = ptr;

}

void addtopos(struct node\*\* head, int data, int pos)

{

struct node\* newnode = new struct node;

newnode->data = data;

newnode->prev = nullptr;

newnode->next = nullptr;

struct node\* temp1 = \*head;

struct node\* temp2 = nullptr;

int count = 0;

if (pos == 0) {

// Insert at the beginning

newnode->next = temp1;

if (temp1 != nullptr) {

temp1->prev = newnode;

}

\*head = newnode;

}

while (count < pos && temp1 != nullptr) {

temp2 = temp1;

temp1 = temp1->next;

count++;

}

if (temp1 == nullptr && count < pos) {

// Position is beyond the end of the list.

// Handle this case based on your requirements.

}

temp2->next = newnode;

newnode->prev = temp2;

newnode->next = temp1;

if (temp1 != nullptr) {

temp1->prev = newnode;

}

}

void deletefromfirst(struct node\*\* head)

{

if (\*head == nullptr) {

// Empty list, nothing to delete

return;

}

struct node\* temp = \*head;

if (temp->next != nullptr) {

// Update the new head's prev pointer

temp->next->prev = nullptr;

}

\*head = temp->next;

free(temp);

}

void deletefromlast(struct node\*\* head)

{

if (\*head == nullptr) {

// Empty list, nothing to delete

return;

}

struct node\* temp = \*head;

// Move to the last node

while (temp->next != nullptr) {

temp = temp->next;

}

if (temp->prev != nullptr) {

// If there's a previous node, update its next pointer

temp->prev->next = nullptr;

} else {

// If there is no previous node, it means temp is the only node in the list

\*head = nullptr;

}

free(temp);

}

void deletefrompos(struct node\*\* head, int pos)

{

if (\*head == nullptr || pos < 0) {

// Empty list or invalid position, nothing to delete

return;

}

struct node\* temp = \*head;

int count = 0;

while (count != pos && temp != nullptr) {

count++;

temp = temp->next;

}

if (count < pos || temp == nullptr) {

// Position is beyond the end of the list or invalid position

return;

}

if (temp->prev != nullptr) {

// If there is a previous node, update its next pointer

temp->prev->next = temp->next;

} else {

// If there is no previous node, it means temp is the first node

\*head = temp->next;

if (\*head != nullptr) {

// If there is a new head, update its prev pointer

(\*head)->prev = nullptr;

}

}

if (temp->next != nullptr) {

// If there is a next node, update its prev pointer

temp->next->prev = temp->prev;

}

free(temp);

}

void reverseList(struct node\*\* head)

{

if (\*head == nullptr || (\*head)->next == nullptr) {

// Empty list or a list with only one node, nothing to reverse

return;

}

struct node\* current = \*head;

struct node\* nextNode = nullptr;

struct node\* prevNode = nullptr;

while (current != nullptr) {

nextNode = current->next;

// Reverse the links

current->next = prevNode;

current->prev = nextNode;

// Move to the next nodes

prevNode = current;

current = nextNode;

}

// Update the head to the last node

\*head = prevNode;

}

int main() {

struct node\* head = NULL;

struct node\* ptr;

addtofront(&head, 12);

addtofront(&head, 30);

addtofront(&head, 40);

addtoend(&head, 50);

addtoend(&head, 10);

addtoend(&head, 90);

addtopos(&head, 45, 2);

deletefromfirst(&head);

deletefromlast(&head);

deletefrompos(&head,2);

reverseList(&head);

ptr = head;

while (ptr != NULL) {

cout << ptr->data << " ";

ptr = ptr->next;

}

// Don't forget to free the memory when done.

return 0;

}