#include <iostream>

using namespace std;

struct node {

int data;

struct node\* next;

};

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

struct node\* temp = new node;

temp->data = data;

temp->next = nullptr;

if (\*head == nullptr)

\*head = temp;

else {

temp->next = \*head;

\*head = temp;

}

}

void addatend(struct node\*\*head, int data)

{

struct node\*temp=new struct node;

struct node\* ptr;

temp->data=data;

temp->next=NULL;

if(\*head==NULL)

{

\*head=temp;

}

else

{

ptr=\*head;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=temp;

}

}

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

{

struct node\* newnode = new struct node;

struct node\* temp;

int count = 0;

newnode->data = data;

newnode->next = nullptr;

if (pos == 0)

{

newnode->next = \*head;

\*head = newnode;

return;

}

temp = \*head;

while (count < pos - 1 && temp != nullptr)

{

count++;

temp = temp->next;

}

if (count <= pos - 1 && temp != nullptr)

{

newnode->next = temp->next;

temp->next = newnode;

}

else

{

// Handle invalid position, for example, if pos is greater than the length of the list.

cout << "Invalid position." << endl;

delete newnode; // Free the allocated memory for the new node.

}

}

void deletefrombegin(struct node\*\* head)

{

if (\*head == nullptr)

return;

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

delete \*head;

\*head = nullptr;

} else {

struct node\* temp = \*head;

\*head = (\*head)->next;

delete temp;

}

}

void deletefromend(struct node\*\* head)

{

if(\*head==nullptr)

return;

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

{

delete \*head;

\*head=nullptr;

return;

}

struct node\* ptr;

struct node\* prev;

ptr=\*head;

while(ptr->next!=nullptr)

{

prev=ptr;

ptr=ptr->next;

}

prev->next=nullptr;

delete ptr;

}

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

{

int count = 0;

struct node\* temp = \*head;

struct node\* prev = nullptr;

if (\*head == nullptr)

return;

// Traverse to the node at position pos or the last node if pos is greater than the list length

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

{

prev = temp;

temp = temp->next;

count++;

}

// If pos is greater than the list length, return

if (count < pos)

return;

// If the node to be deleted is not the first node

if (prev != nullptr)

{

prev->next = temp->next;

delete temp;

}

else // If the node to be deleted is the first node

{

\*head = temp->next;

delete temp;

}

}

int main() {

struct node\* head = nullptr;

struct node\* ptr;

addAtBegin(&head, 12);

addAtBegin(&head, 23);

addAtBegin(&head, 11);

addAtBegin(&head, 10);

addatend(&head, 100);

addatend(&head, 101);

addatend(&head, 102);

addatpos(&head, 200,4);

addatpos(&head, 201,5);

deletefrombegin(&head);

deletefrombegin(&head);

deletefromend(&head);

deletefromend(&head);

deletefrompos(&head,2);

ptr = head;

while (ptr != nullptr) {

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

ptr = ptr->next;

}

return 0;

}