#include <string>

#include <iostream>

#include <cstdio>

using namespace std;

int total = 0;

class Node

{

public:

Node(){

this->next = NULL;

this->pre = NULL;

}

Node(const int element , Node \*next, Node \*pre){

this->data = element;

this->next = next;

this->pre = pre;

}

~Node(){

this->next = NULL;

this->pre = NULL;

}

friend class Chain;

Node \*next;

int data;

Node \*pre;

};

class Chain

{

public:

Chain(){ // constructor

head = NULL;

tail = NULL;

}

Chain(Node \*head){ // copy constructor

this->head = head;

this->tail = head;

}

void InsertBack(int data);

void InsertFront(int data);

void InsertAfter(int data,int data\_ref);

void InsertBefore(int data,int data\_ref);

void Delete(int data);

void DeleteFront();

void DeleteBack();

void Reverse();

void Rotate(int k);

void Swap(int k, int j);

bool IsEmpty(){

this->tail = NULL;

return this->head == NULL;

}

std::string PrintChain(){

Node \*cur = this->head;

std::string result = "";

if(cur == NULL){

result = "Empty";

return result;

}

while(cur != NULL){

int num = cur->data;

std::string num\_str = std::to\_string(num);

result.append(num\_str);

if(cur -> next){

result.append("->");

}

cur = cur->next;

}

return result;

}

Node \*head;

Node \*tail;

};

void Chain::InsertBack(int data)

{

total = total+1;

if(head==NULL){

head = new Node();

head->data = data;

head->next = NULL;

head->pre = NULL;

tail = head;

}else{

Node \*newptr = new Node();

newptr->data = data;

newptr->next = NULL;

newptr->pre = tail;

tail->next = newptr;

tail = newptr;

}

}

void Chain::InsertFront(int data)

{

total = total+1;

if(head==NULL){

head = new Node();

head->data = data;

head->next = NULL;

head->pre = NULL;

tail = head;

}else{

Node \*newptr = new Node();

newptr->data = data;

newptr->next = head;

newptr->pre = NULL;

head->pre = newptr;

head = newptr;

}

}

void Chain::InsertAfter(int data,int data\_ref)

{

total = total+1;

if(head==NULL) return;

else{

Node \*current = head;

Node \*tmp = new Node();

Node \*p = NULL;

int find = 0;

tmp->data = data;

tmp->pre = NULL;

tmp->next = NULL;

// while(current->data!=data\_ref && current!=NULL){

// current = current->next;

// }

//

// if(data\_ref == current->data) find = 1;

while(current!=NULL){

if(current->data == data\_ref){

find = 1;

break;

}

current = current->next;

}

if(find == 1){

if(current == tail){

tail->next = tmp;

tmp->pre = tail;

tail = tmp;

}else{

p = current->next;

current->next = tmp;

tmp->pre = current;

tmp->next = p;

p->pre = tmp;

}

}else return;

}

}

void Chain::InsertBefore(int data,int data\_ref)

{

total = total+1;

if(head==NULL) return;

else{

Node \*current = head;

Node \*tmp = new Node();

Node \*p = NULL;

int find = 0;

tmp->data = data;

tmp->next = NULL;

tmp->pre = NULL;

while(current->data!=data\_ref && current!=NULL){

current = current->next;

}

if(data\_ref == current->data) find = 1;

if(find == 1){

if(current==head){

tmp->next = head;

head->pre = tmp;

head = tmp;

}else{

p = current->pre;

p->next = tmp;

tmp->pre = p;

tmp->next = current;

current->pre = tmp;

}

}else return;

}

}

void Chain::Delete(int data)

{

Node \*current = head;

Node \*tmp = NULL;

Node \*p = NULL;

Node \*n = NULL;

int find = 0;

if(head==NULL) return;

else{

current = head;

while(current!=NULL){

if(data == current->data){

find = 1;

break;

}

current = current->next;

}

if(find==1){

if(total==1){

tmp = head;

head = NULL;

tail = NULL;

delete tmp;

}else{

if(current == head){

head = current->next;

head->pre = NULL;

delete current;

}else if(current==tail){

tail = current->pre;

tail->next = NULL;

delete current;

}else{

p = current->pre;

n = current->next;

delete current;

p->next = n;

n->pre = p;

}

}

}else return;

}

// while(current->data!=data && current!=NULL){

// current = current->next;

// }

// if(data == current->data) find = 1;

// }

total = total-1;

}

void Chain::DeleteFront()

{

if(head==NULL) return;

else{

if(total==1){

Node \*tmp = head;

head = NULL;

tail = NULL;

delete tmp;

}else if(total==2){

Node \*tmp = head;

head = tail;

head->pre = NULL;

head->next = NULL;

tail->pre = NULL;

delete tmp;

}else{

Node \*tmp = head;

head = head->next;

head->pre = NULL;

delete tmp;

}

}

total = total-1;

}

void Chain::DeleteBack()

{

if(head==NULL) return;

else{

if(total==1){

Node \*tmp = head;

head = NULL;

tail = NULL;

delete tmp;

}

else if(total==2){

Node \*tmp = tail;

tail = head;

tail->next = NULL;

tail->pre = NULL;

head->next = NULL;

head->pre = NULL;

delete tmp;

}

else{

Node \*tmp = tail;

tail = tail->pre;

tail->next = NULL;

delete tmp;

}

}

total = total-1;

}

void Chain::Reverse()

{

Node \*current = head;

Node \*tmp = NULL;

if(head==NULL) return;

else{

if(total==1) return;

else if(total==2){

tmp = head;

head = tail;

tail = tmp;

head->pre = NULL;

head->next = tail;

tail->next = NULL;

tail->pre = head;

}else{

current = head;

while(current->next!=NULL){

tmp = current->pre;

current->pre = current->next;

current->next = tmp;

current = current->pre;

}

tmp = current->pre;

current->next = tmp;

current->pre = NULL;

tmp = head;

head = tail;

tail = tmp;

head->pre = NULL;

tail->next = NULL;

}

}

}

void Chain::Rotate(int k)

{

//int num = total - (k%total);

int count = 0;

Node \*current = head;

Node \*tmp = NULL;

if(head==NULL) return;

else{

if(k==1){

tmp = tail;

tail->next = head;

head->pre = tail;

tail = tail->pre;

tail->next = NULL;

head = tmp;

head->pre = NULL;

}else if(k==total){

head = head;

tail = tail;

}else{

count = 0;

current = tail;

while(count < (k%total)-1){

current = current->pre;

count++;

}

tail->next = head;

head->pre = tail;

head = current;

tail = current->pre;

head->pre = NULL;

tail->next = NULL;

}

}

}

void Chain::Swap(int k, int j)

{

Node \*current = head;

Node \*tmp = NULL;

int data1;

int data2;

int find1 = 0;

int find2 = 0;

if(head==NULL) return;

else{

find1 = 0;

find2 = 0;

current = head;

while(current!=NULL){

if(current->data == k){

find1 = 1;

}else if(current->data == j){

find2 = 1;

}

current = current->next;

}

if(find1==1 && find2==1){

current = head;

while(current!=NULL){

if(current->data == k){

current->data = j;

}else if(current->data == j){

current->data = k;

}

current = current->next;

}

}

}

}

int main()

{

Chain inst = \*(new Chain());

string command;

int data , data\_ref;

while(cin>>command){

if(command == "InsertBack"){

cin>>data;

inst.InsertBack(data);

}else if(command == "InsertFront"){

cin>>data;

inst.InsertFront(data);

}else if(command == "InsertAfter"){

cin>>data>>data\_ref;

inst.InsertAfter(data , data\_ref);

}else if(command == "InsertBefore"){

cin>>data>>data\_ref;

inst.InsertBefore(data , data\_ref);

}else if(command == "Delete"){

cin>>data;

inst.Delete(data);

}else if(command == "DeleteFront"){

inst.DeleteFront();

}else if(command == "DeleteBack"){

inst.DeleteBack();

}else if(command == "Reverse"){

inst.Reverse();

}else if(command == "Rotate"){

int k;

cin>>k;

inst.Rotate(k);

}else if(command == "Swap"){

int j,k;

cin>>j>>k;

inst.Swap(j,k);

}

else if(command == "PrintChain"){

cout<<inst.PrintChain()<<endl;

}

}

return 0;

}

#include <stdio.h>

#include <stdlib.h>

typedef struct \_Node{

long data;

int power;

struct \_Node \*next;

struct \_Node \*prev;

}Node;

void printNode(Node\* head);

Node\* create();

void destroy(Node \*node);

Node\* multiple(Node\* p1, Node\* p2);

Node \*makeNode(int data,int power);

Node\* create()

{

Node \*head = NULL;

int data,power;

scanf("%d%d",&data,&power);

head = (Node \*)malloc(sizeof(Node));

head->data = data;

head->power = power;

head->prev = NULL;

head->next = NULL;

if(power==0) return head;

head->next = create(); // point

head->next->prev = head;

return head;

}

void printNode(Node\* head)

{

if(head==NULL) return;

if(head->data!=0) printf(" %d %d",head->data,head->power);

printNode(head->next);

}

void destroy(Node \*node)

{

if(node != NULL){

destroy(node->next);

free(node);

}

}

Node \*makeNode(int data,int power)

{

Node \*newptr;

newptr = (Node \*)malloc(sizeof(Node));

newptr->data = data;

newptr->power = power;

newptr->next = NULL;

newptr->prev = NULL;

return newptr;

}

Node\* multiple(Node\* p1, Node\* p2)

{

Node \*t1, \*t2, \*t3, \*t4;

Node \*newptr;

Node \*p3 = NULL; // multiplied polynomial

int data, power;

for(t1=p1;t1!=NULL;t1 = t1->next){

for(t2=p2;t2!=NULL;t2 = t2->next){

data = t1->data \* t2->data;

power = t1->power + t2->power;

// insert the result to p3

if(p3==NULL){

p3 = makeNode(data,power);

}else{

for(t3=p3;t3!=NULL;t3 = t3->next){

if(power == t3->power){

t3->data += data;

break;

}else if(power > t3->power){ t4 = makeNode(data,power);

t4->next = t3;

t4->prev = t3->prev;

t3->prev->next = t4;

t3->prev = t4;

break;

}else if(power < t3->power){

if(t3->next==NULL){

t3->next = makeNode(data,power);

t3->next->prev = t3;

break;

}

}

}

}

}

}

return p3;

}

int main(void){

Node \*p1=create(); //polynomial linked list1

Node \*p2=create(); //polynomial linked list2

Node \*mul=multiple(p1,p2);

printNode(mul);

destroy(mul);

destroy(p2);

destroy(p1);

return 0;

}