//nasirin, abdul m.

//ccc121A, machine problem 4

// adding polynomials

#include <iostream>

#include <queue>

using namespace std;

// Node structure for linked list

struct Node {

int coeff; // Coefficient

int exp; // Exponent

Node\* next;

// Constructor without member initializer list

Node(int c, int e) {

coeff = c; // Initialize coefficient

exp = e; // Initialize exponent

next = nullptr; // Initialize next pointer

}

};

// LinkedList class to manage polynomial operations

class LinkedList {

private:

Node\* head;

public:

// Constructor

LinkedList() {

head = nullptr; // Initialize head to nullptr

}

// Insert a new term in sorted order (descending by exponent)

void insert(int coeff, int exp) {

Node\* newNode = new Node(coeff, exp);

// If the list is empty or the new node has higher exponent than the head

if (!head || head->exp < exp) {

newNode->next = head;

head = newNode;

return;

}

Node\* temp = head;

// Traverse the list to find the correct position for the new node

while (temp->next && temp->next->exp >= exp) {

temp = temp->next;

}

// Insert the new node in the correct position

newNode->next = temp->next;

temp->next = newNode;

}

// Add two polynomials using queues

static LinkedList addPolynomials(LinkedList & poly1, LinkedList & poly2) {

LinkedList result;

queue<Node\*> q1, q2;

// Add terms from poly1 to queue q1

Node\* temp = poly1.head;

while (temp) {

q1.push(temp);

temp = temp->next;

}

// Add terms from poly2 to queue q2

temp = poly2.head;

while (temp) {

q2.push(temp);

temp = temp->next;

}

// Process both queues and add the terms

while (!q1.empty() && !q2.empty()) {

Node\* term1 = q1.front();

Node\* term2 = q2.front();

if (term1->exp == term2->exp) {

result.insert(term1->coeff + term2->coeff, term1->exp);

q1.pop();

q2.pop();

}

else if (term1->exp > term2->exp) {

result.insert(term1->coeff, term1->exp);

q1.pop();

}

else {

result.insert(term2->coeff, term2->exp);

q2.pop();

}

}

// Add remaining terms from queue 1 (if any)

while (!q1.empty()) {

Node\* term = q1.front();

result.insert(term->coeff, term->exp);

q1.pop();

}

// Add remaining terms from queue 2 (if any)

while (!q2.empty()) {

Node\* term = q2.front();

result.insert(term->coeff, term->exp);

q2.pop();

}

return result;

}

// Display the polynomial

void display() {

Node\* temp = head;

bool firstTerm = true; // To handle the first term without a "+" sign

while (temp) {

if (temp->coeff != 0) { // Skip terms with a zero coefficient

// Handling the sign

if (firstTerm) {

firstTerm = false;

}

else if (temp->coeff > 0) {

cout << " + ";

}

// Display the term

if (temp->exp == 0) {

// Constant term, no x

cout << temp->coeff<<" ";

}

else if (temp->exp == 1) {

// Linear term, only x

if (temp->coeff == 1) {

cout << " x"<<" ";

}

else if (temp->coeff == -1) {

cout << "-x ";

}

else {

cout<<" " << temp->coeff << "x"<<" ";

}

}

else {

// General case, x with exponent

if (temp->coeff == 1) {

cout << "x^" << temp->exp<<" ";

}

else if (temp->coeff == -1) {

cout <<" "<< "-x^" << temp->exp<<" ";

}

else {

cout << temp->coeff << "x^" << temp->exp<<" ";

}

}

}

temp = temp->next;

}

cout << endl;

}

};

// Main function

int main() {

// Polynomial 1: 2x^2 + x + 32

LinkedList poly1;

//poly1.insert(Coefficient, Exponent);

poly1.insert(2, 2);

poly1.insert(-1, 1);

poly1.insert(99, 0);

poly1.insert(7, 3);

poly1.insert(3, 5);

// Polynomial 2: 7x^4 + 4x^2 + 2x + 7

LinkedList poly2;

//poly2.insert(Coefficient, Exponent);

poly2.insert(7, 4);

poly2.insert(8, 2);

poly2.insert(2, 1);

poly2.insert(7, 0);

poly2.insert(7, 3);

poly2.insert(4, 5);

// Add the two polynomials

LinkedList result = LinkedList::addPolynomials(poly1, poly2);

// Display the result

cout << "Result Polynomial: ";

result.display();

return 0;

}