**🧮 PART 1: Polynomial Operations using Linked List**

**✅ Supported Operations:**

* Creation of Polynomial
* Display
* Addition of Two Polynomials
* Multiplication of Two Polynomials

**🔧 C++ Code:**

#include <iostream>

using namespace std;

struct PolyNode {

int coeff, exp;

PolyNode\* next;

};

PolyNode\* createNode(int coeff, int exp) {

PolyNode\* node = new PolyNode;

node->coeff = coeff;

node->exp = exp;

node->next = nullptr;

return node;

}

void insertTerm(PolyNode\*& poly, int coeff, int exp) {

PolyNode\* newNode = createNode(coeff, exp);

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

newNode->next = poly;

poly = newNode;

} else {

PolyNode\* temp = poly;

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

temp = temp->next;

}

if (temp->exp == exp) {

temp->coeff += coeff;

} else if (temp->next && temp->next->exp == exp) {

temp->next->coeff += coeff;

} else {

newNode->next = temp->next;

temp->next = newNode;

}

}

}

void displayPoly(PolyNode\* poly) {

while (poly) {

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

if (poly->next) cout << " + ";

poly = poly->next;

}

cout << endl;

}

PolyNode\* addPoly(PolyNode\* p1, PolyNode\* p2) {

PolyNode\* result = nullptr;

while (p1 || p2) {

if (p1 && (!p2 || p1->exp > p2->exp)) {

insertTerm(result, p1->coeff, p1->exp);

p1 = p1->next;

} else if (p2 && (!p1 || p2->exp > p1->exp)) {

insertTerm(result, p2->coeff, p2->exp);

p2 = p2->next;

} else {

insertTerm(result, p1->coeff + p2->coeff, p1->exp);

p1 = p1->next;

p2 = p2->next;

}

}

return result;

}

PolyNode\* multiplyPoly(PolyNode\* p1, PolyNode\* p2) {

PolyNode\* result = nullptr;

for (PolyNode\* i = p1; i != nullptr; i = i->next) {

for (PolyNode\* j = p2; j != nullptr; j = j->next) {

insertTerm(result, i->coeff \* j->coeff, i->exp + j->exp);

}

}

return result;

}

int main() {

PolyNode\* p1 = nullptr;

PolyNode\* p2 = nullptr;

// First Polynomial: 5x^3 + 4x^2 + 2x^0

insertTerm(p1, 5, 3);

insertTerm(p1, 4, 2);

insertTerm(p1, 2, 0);

// Second Polynomial: 5x^1 + 5x^0

insertTerm(p2, 5, 1);

insertTerm(p2, 5, 0);

cout << "Polynomial 1: ";

displayPoly(p1);

cout << "Polynomial 2: ";

displayPoly(p2);

PolyNode\* sum = addPoly(p1, p2);

cout << "Sum: ";

displayPoly(sum);

PolyNode\* product = multiplyPoly(p1, p2);

cout << "Product: ";

displayPoly(product);

return 0;

}

**🔷 PART 2: Set Operations using Linked List**

**✅ Supported Operations:**

* Union
* Intersection
* Difference

**🔧 C++ Code:**

#include <iostream>

using namespace std;

struct SetNode {

int data;

SetNode\* next;

};

SetNode\* createSetNode(int data) {

SetNode\* node = new SetNode;

node->data = data;

node->next = nullptr;

return node;

}

void insertUnique(SetNode\*& head, int data) {

SetNode\* temp = head;

while (temp) {

if (temp->data == data) return;

temp = temp->next;

}

SetNode\* newNode = createSetNode(data);

newNode->next = head;

head = newNode;

}

void displaySet(SetNode\* head) {

cout << "{ ";

while (head) {

cout << head->data << " ";

head = head->next;

}

cout << "}" << endl;

}

SetNode\* unionSet(SetNode\* a, SetNode\* b) {

SetNode\* result = nullptr;

while (a) {

insertUnique(result, a->data);

a = a->next;

}

while (b) {

insertUnique(result, b->data);

b = b->next;

}

return result;

}

SetNode\* intersectionSet(SetNode\* a, SetNode\* b) {

SetNode\* result = nullptr;

for (SetNode\* i = a; i != nullptr; i = i->next) {

for (SetNode\* j = b; j != nullptr; j = j->next) {

if (i->data == j->data) {

insertUnique(result, i->data);

break;

}

}

}

return result;

}

SetNode\* differenceSet(SetNode\* a, SetNode\* b) {

SetNode\* result = nullptr;

for (SetNode\* i = a; i != nullptr; i = i->next) {

bool found = false;

for (SetNode\* j = b; j != nullptr; j = j->next) {

if (i->data == j->data) {

found = true;

break;

}

}

if (!found) insertUnique(result, i->data);

}

return result;

}

int main() {

SetNode\* A = nullptr;

SetNode\* B = nullptr;

insertUnique(A, 1);

insertUnique(A, 2);

insertUnique(A, 3);

insertUnique(B, 3);

insertUnique(B, 4);

insertUnique(B, 5);

cout << "Set A: ";

displaySet(A);

cout << "Set B: ";

displaySet(B);

SetNode\* u = unionSet(A, B);

cout << "Union: ";

displaySet(u);

SetNode\* i = intersectionSet(A, B);

cout << "Intersection: ";

displaySet(i);

SetNode\* d = differenceSet(A, B);

cout << "Difference (A - B): ";

displaySet(d);

return 0;}

**🧮 PART 1: Polynomial Operations using Arrays**

**✅ Supported Operations:**

* Create Polynomial
* Display Polynomial
* Add Two Polynomials
* Multiply Two Polynomials

**🔧 C++ Code for Polynomial Operations using Arrays:**

#include <iostream>

using namespace std;

const int MAX = 100;

struct Term {

int coeff;

int exp;

};

// Read a polynomial

int readPolynomial(Term poly[]) {

int n;

cout << "Enter number of terms: ";

cin >> n;

for (int i = 0; i < n; i++) {

cout << "Enter coefficient and exponent for term " << i + 1 << ": ";

cin >> poly[i].coeff >> poly[i].exp;

}

return n;

}

// Display polynomial

void displayPolynomial(Term poly[], int n) {

for (int i = 0; i < n; i++) {

cout << poly[i].coeff << "x^" << poly[i].exp;

if (i != n - 1)

cout << " + ";

}

cout << endl;

}

// Add two polynomials

int addPolynomials(Term p1[], int n1, Term p2[], int n2, Term result[]) {

int i = 0, j = 0, k = 0;

while (i < n1 && j < n2) {

if (p1[i].exp > p2[j].exp)

result[k++] = p1[i++];

else if (p1[i].exp < p2[j].exp)

result[k++] = p2[j++];

else {

result[k].exp = p1[i].exp;

result[k++].coeff = p1[i++].coeff + p2[j++].coeff;

}

}

while (i < n1) result[k++] = p1[i++];

while (j < n2) result[k++] = p2[j++];

return k;

}

// Multiply two polynomials

int multiplyPolynomials(Term p1[], int n1, Term p2[], int n2, Term result[]) {

int k = 0;

for (int i = 0; i < n1; i++) {

for (int j = 0; j < n2; j++) {

int coeff = p1[i].coeff \* p2[j].coeff;

int exp = p1[i].exp + p2[j].exp;

// Combine like terms

bool found = false;

for (int m = 0; m < k; m++) {

if (result[m].exp == exp) {

result[m].coeff += coeff;

found = true;

break;

}

}

if (!found) {

result[k].coeff = coeff;

result[k].exp = exp;

k++;

}

}

}

return k;

}

int main() {

Term p1[MAX], p2[MAX], sum[MAX], product[MAX];

int n1 = readPolynomial(p1);

int n2 = readPolynomial(p2);

cout << "Polynomial 1: ";

displayPolynomial(p1, n1);

cout << "Polynomial 2: ";

displayPolynomial(p2, n2);

int sumSize = addPolynomials(p1, n1, p2, n2, sum);

cout << "Sum: ";

displayPolynomial(sum, sumSize);

int prodSize = multiplyPolynomials(p1, n1, p2, n2, product);

cout << "Product: ";

displayPolynomial(product, prodSize);

return 0;

}

**🔷 PART 2: Set Operations using Arrays**

**✅ Supported Operations:**

* Union
* Intersection
* Difference

**🔧 C++ Code for Set Operations using Arrays:**

#include <iostream>

using namespace std;

const int MAX = 100;

void readSet(int set[], int& size) {

cout << "Enter number of elements: ";

cin >> size;

cout << "Enter elements: ";

for (int i = 0; i < size; i++) {

cin >> set[i];

}

}

void displaySet(int set[], int size) {

cout << "{ ";

for (int i = 0; i < size; i++) {

cout << set[i] << " ";

}

cout << "}" << endl;

}

bool exists(int set[], int size, int value) {

for (int i = 0; i < size; i++) {

if (set[i] == value)

return true;

}

return false;

}

int unionSet(int a[], int na, int b[], int nb, int result[]) {

int k = 0;

for (int i = 0; i < na; i++)

result[k++] = a[i];

for (int i = 0; i < nb; i++) {

if (!exists(a, na, b[i]))

result[k++] = b[i];

}

return k;

}

int intersectionSet(int a[], int na, int b[], int nb, int result[]) {

int k = 0;

for (int i = 0; i < na; i++) {

if (exists(b, nb, a[i]))

result[k++] = a[i];

}

return k;

}

int differenceSet(int a[], int na, int b[], int nb, int result[]) {

int k = 0;

for (int i = 0; i < na; i++) {

if (!exists(b, nb, a[i]))

result[k++] = a[i];

}

return k;

}

int main() {

int A[MAX], B[MAX], U[MAX], I[MAX], D[MAX];

int na, nb, nu, ni, nd;

cout << "Enter Set A:\n";

readSet(A, na);

cout << "Enter Set B:\n";

readSet(B, nb);

cout << "Set A: ";

displaySet(A, na);

cout << "Set B: ";

displaySet(B, nb);

nu = unionSet(A, na, B, nb, U);

cout << "Union: ";

displaySet(U, nu);

ni = intersectionSet(A, na, B, nb, I);

cout << "Intersection: ";

displaySet(I, ni);

nd = differenceSet(A, na, B, nb, D);

cout << "Difference (A - B): ";

displaySet(D, nd);

return 0;

}