Rajalakshmi Engineering College

Name: Sowmyalakshmi N
Email: 240701524@rajalakshmi.edu.in
Roll no: 240701524
Phone: 9003767185
Branch: REC
Department: I CSE FE
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_week 1_CY

Attempt : 1
Total Mark : 30
Marks Obtained : 22.5

Section 1 : Coding

1. Problem Statement

Hayley loves studying polynomials, and she wants to write a program to compare two polynomials represented as linked lists and display whether they are equal or not.

The polynomials are expressed as a series of terms, where each term consists of a coefficient and an exponent. The program should read the polynomials from the user, compare them, and then display whether they are equal or not.

Input Format

The first line of input consists of an integer n, representing the number of terms in the first polynomial.

The following n lines of input consist of two integers, each representing the coefficient and the exponent of the term in the first polynomial.

The next line of input consists of an integer m, representing the number of terms in the second polynomial.

The following m lines of input consist of two integers, each representing the coefficient and the exponent of the term in the second polynomial.

Output Format

The first line of output prints "Polynomial 1: " followed by the first polynomial.

The second line prints "Polynomial 2: " followed by the second polynomial.

The polynomials should be displayed in the format ax^b, where a is the coefficient and b is the exponent.

If the two polynomials are equal, the third line prints "Polynomials are Equal."

If the two polynomials are not equal, the third line prints "Polynomials are Not Equal."

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 2
1 2
2 1
2
1 2
2 1

Output: Polynomial 1: (1x^2) + (2x^1)
Polynomial 2: (1x^2) + (2x^1)
Polynomials are Equal.

Answer

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>
```

```
// Structure to represent a term in the polynomial
struct Term {
    int coeff;
    int exp;
    struct Term* next;
};

// Create a new term node
struct Term* createTerm(int coeff, int exp) {
    struct Term* newTerm = (struct Term*)malloc(sizeof(struct Term));
    newTerm->coeff = coeff;
    newTerm->exp = exp;
    newTerm->next = NULL;
    return newTerm;
}

// Insert term at the end of the linked list
void insertTerm(struct Term** head, int coeff, int exp) {
    struct Term* newTerm = createTerm(coeff, exp);
    if (*head == NULL) {
        *head = newTerm;
        return;
    }
    struct Term* temp = *head;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = newTerm;
}

// Print a polynomial in the specified format
void printPolynomial(struct Term* poly) {
    struct Term* temp = poly;
    while (temp != NULL) {
        printf("(%dx^%d)", temp->coeff, temp->exp);
        if (temp->next != NULL)
            printf(" + ");
        temp = temp->next;
    }
    printf("\n");
}
```

```
// Compare two polynomials term by term
int arePolynomialsEqual(struct Term* p1, struct Term* p2) {
    while (p1 != NULL && p2 != NULL) {
        if (p1->coeff != p2->coeff || p1->exp != p2->exp)
            return 0;
        p1 = p1->next;
        p2 = p2->next;
    }
    // If both are NULL, then equal
    return (p1 == NULL && p2 == NULL);
}

// ---- Main Function ----
int main() {
    int n, m, coeff, exp;
    struct Term* poly1 = NULL;
    struct Term* poly2 = NULL;

    // Read first polynomial
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &coeff, &exp);
        insertTerm(&poly1, coeff, exp);
    }

    // Read second polynomial
    scanf("%d", &m);
    for (int i = 0; i < m; i++) {
        scanf("%d %d", &coeff, &exp);
        insertTerm(&poly2, coeff, exp);
    }

    // Print polynomials
    printf("Polynomial 1: ");
    printPolynomial(poly1);

    printf("Polynomial 2: ");
    printPolynomial(poly2);

    // Compare polynomials
    if (arePolynomialsEqual(poly1, poly2))
        printf("Polynomials are Equal.\n");
```

```
    else
        printf("Polynomials are Not Equal.\n");

    return 0;
}
```

Status : Correct **Marks : 10/10**

2. Problem Statement

Keerthi is a tech enthusiast and is fascinated by polynomial expressions. She loves to perform various operations on polynomials.

Today, she is working on a program to multiply two polynomials and delete a specific term from the result.

Keerthi needs your help to implement this program. She wants to take the coefficients and exponents of the terms of the two polynomials as input, perform the multiplication, and then allow the user to specify an exponent for deletion from the resulting polynomial, and display the result.

Input Format

The first line of input consists of an integer n, representing the number of terms in the first polynomial.

The following n lines of input consist of two integers, each representing the coefficient and the exponent of the term in the first polynomial.

The next line consists of an integer m, representing the number of terms in the second polynomial.

The following m lines of input consist of two integers, each representing the coefficient and the exponent of the term in the second polynomial.

The last line consists of an integer, representing the exponent of the term that Keerthi wants to delete from the multiplied polynomial.

Output Format

The first line of output displays the resulting polynomial after multiplication.

The second line displays the resulting polynomial after deleting the specified term.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 3
2 2
3 1
4 0
2
1 2
2 1
2
Output: Result of the multiplication: 2x^4 + 7x^3 + 10x^2 + 8x
Result after deleting the term: 2x^4 + 7x^3 + 8x

Answer

```
#include <stdio.h>
#include <stdlib.h>

// Structure for each term in the polynomial
struct Term {
    int coeff;
    int exp;
    struct Term* next;
};

// Create a new term
struct Term* createTerm(int coeff, int exp) {
    struct Term* newTerm = (struct Term*)malloc(sizeof(struct Term));
    newTerm->coeff = coeff;
    newTerm->exp = exp;
    newTerm->next = NULL;
    return newTerm;
}

// Insert term at the end of the polynomial
```

```
void insertTerm(struct Term** head, int coeff, int exp) {
    struct Term* newTerm = createTerm(coeff, exp);
    if (*head == NULL) {
        *head = newTerm;
        return;
    }
    struct Term* temp = *head;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = newTerm;
}

// Add a term into result, combining like terms
void addTerm(struct Term** result, int coeff, int exp) {
    if (coeff == 0) return;

    struct Term* temp = *result;
    struct Term* prev = NULL;

    while (temp && temp->exp > exp) {
        prev = temp;
        temp = temp->next;
    }

    if (temp && temp->exp == exp) {
        temp->coeff += coeff;
    } else {
        struct Term* newTerm = createTerm(coeff, exp);
        if (prev == NULL) {
            newTerm->next = *result;
            *result = newTerm;
        } else {
            newTerm->next = temp;
            prev->next = newTerm;
        }
    }
}

// Multiply two polynomials
struct Term* multiplyPolynomials(struct Term* poly1, struct Term* poly2) {
    struct Term* result = NULL;
    for (struct Term* ptr1 = poly1; ptr1 != NULL; ptr1 = ptr1->next) {
```

```
        for (struct Term* ptr2 = poly2; ptr2 != NULL; ptr2 = ptr2->next) {
            int newCoeff = ptr1->coeff * ptr2->coeff;
            int newExp = ptr1->exp + ptr2->exp;
            addTerm(&result, newCoeff, newExp);
        }
    }
    return result;
}

// Delete a term with a specific exponent
void deleteTerm(struct Term** head, int exp) {
    struct Term* temp = *head;
    struct Term* prev = NULL;

    while (temp != NULL && temp->exp != exp) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) return; // Term not found

    if (prev == NULL) {
        *head = temp->next;
    } else {
        prev->next = temp->next;
    }

    free(temp);
}

// Print the polynomial
void printPolynomial(struct Term* poly) {
    struct Term* temp = poly;
    int first = 1;
    while (temp != NULL) {
        if (temp->coeff != 0) {
            if (!first) printf(" + ");
            if (temp->exp == 0)
                printf("%d", temp->coeff);
            else if (temp->exp == 1)
                printf("%dx", temp->coeff);
            else
```

```
            printf("%dx^%d", temp->coeff, temp->exp);
            first = 0;
        }
        temp = temp->next;
    }
    if (first) {
        // If no terms were printed
        printf("0");
    }
    printf("\n");
}

// ---- Main Function ----
int main() {
    int n, m, coeff, exp, deleteExp;
    struct Term* poly1 = NULL;
    struct Term* poly2 = NULL;

    // Read polynomial 1
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &coeff, &exp);
        insertTerm(&poly1, coeff, exp);
    }

    // Read polynomial 2
    scanf("%d", &m);
    for (int i = 0; i < m; i++) {
        scanf("%d %d", &coeff, &exp);
        insertTerm(&poly2, coeff, exp);
    }

    // Read exponent to delete
    scanf("%d", &deleteExp);

    // Multiply the polynomials
    struct Term* result = multiplyPolynomials(poly1, poly2);

    // Print the result before deletion
    printf("Result of the multiplication: ");
    printPolynomial(result);
```

```
    // Delete the specified exponent term
    deleteTerm(&result, deleteExp);

    // Print the result after deletion
    printf("Result after deleting the term: ");
    printPolynomial(result);

    return 0;
}
```

Status : Partially correct ***Marks : 7.5/10***

3. Problem Statement

Rani is studying polynomials in her class. She has learned about polynomial multiplication and is eager to try it out on her own. However, she finds the process of manually multiplying polynomials quite tedious. To make her task easier, she decides to write a program to multiply two polynomials represented as linked lists.

Help Rani by designing a program that takes two polynomials as input and outputs their product polynomial. Each polynomial is represented by a linked list of terms, where each term has a coefficient and an exponent. The terms are entered in descending order of exponents.

Input Format

The first line of input consists of an integer n, representing the number of terms in the first polynomial.

The following n lines of input consist of two integers each: the coefficient and the exponent of the term in the first polynomial.

The next line of input consists of an integer m, representing the number of terms in the second polynomial.

The following m lines of input consist of two integers each: the coefficient and the exponent of the term in the second polynomial.

Output Format

The first line of output prints the first polynomial.

The second line of output prints the second polynomial.

The third line of output prints the resulting polynomial after multiplying the given polynomials.

The polynomials should be displayed in the format, where each term is represented as ax^b, where a is the coefficient and b is the exponent.

Refer to the sample output for the exact format.

Sample Test Case

Input: 2
2 3
3 2
2
3 2
2 1

Output: 2x^3 + 3x^2
3x^2 + 2x
6x^5 + 13x^4 + 6x^3

Answer

```
#include <stdio.h>
#include <stdlib.h>

struct Term {
    int coeff;
    int exp;
    struct Term* next;
};

// Function to create a new term
struct Term* createTerm(int coeff, int exp) {
    struct Term* newTerm = (struct Term*)malloc(sizeof(struct Term));
    newTerm->coeff = coeff;
    newTerm->exp = exp;
    newTerm->next = NULL;
```

```
    return newTerm;
}

// Function to insert a term at the end
void insertTerm(struct Term** head, int coeff, int exp) {
    struct Term* newTerm = createTerm(coeff, exp);
    if (*head == NULL) {
        *head = newTerm;
        return;
    }
    struct Term* temp = *head;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = newTerm;
}

// Function to add a term to result (combine like terms)
void addTerm(struct Term** result, int coeff, int exp) {
    if (coeff == 0) return; // Skip zero coefficient terms

    struct Term* temp = *result;
    struct Term* prev = NULL;

    // Insert in descending order of exponents
    while (temp && temp->exp > exp) {
        prev = temp;
        temp = temp->next;
    }

    if (temp && temp->exp == exp) {
        temp->coeff += coeff;
    } else {
        struct Term* newTerm = createTerm(coeff, exp);
        if (prev == NULL) {
            newTerm->next = *result;
            *result = newTerm;
        } else {
            newTerm->next = temp;
            prev->next = newTerm;
        }
    }
}
```

```
// Function to multiply two polynomials
struct Term* multiplyPolynomials(struct Term* poly1, struct Term* poly2) {
    struct Term* result = NULL;
    struct Term* ptr1 = poly1;

    while (ptr1 != NULL) {
        struct Term* ptr2 = poly2;
        while (ptr2 != NULL) {
            int newCoeff = ptr1->coeff * ptr2->coeff;
            int newExp = ptr1->exp + ptr2->exp;
            addTerm(&result, newCoeff, newExp);
            ptr2 = ptr2->next;
        }
        ptr1 = ptr1->next;
    }
    return result;
}

// Function to print a polynomial with proper formatting
void printPolynomial(struct Term* poly) {
    struct Term* temp = poly;
    int first = 1;

    while (temp != NULL) {
        if (temp->coeff != 0) {
            if (!first)
                printf(" + ");
            if (temp->exp == 0)
                printf("%d", temp->coeff);
            else if (temp->exp == 1)
                printf("%dx", temp->coeff);
            else
                printf("%dx^%d", temp->coeff, temp->exp);
            first = 0;
        }
        temp = temp->next;
    }
    printf("\n");
}

// ---- Main ----
```

```
int main() {
    int n, m, coeff, exp;
    struct Term* poly1 = NULL;
    struct Term* poly2 = NULL;

    // Read first polynomial
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &coeff, &exp);
        insertTerm(&poly1, coeff, exp);
    }

    // Read second polynomial
    scanf("%d", &m);
    for (int i = 0; i < m; i++) {
        scanf("%d %d", &coeff, &exp);
        insertTerm(&poly2, coeff, exp);
    }

    // Multiply polynomials
    struct Term* result = multiplyPolynomials(poly1, poly2);

    // Print polynomials and result
    printPolynomial(poly1);
    printPolynomial(poly2);
    printPolynomial(result);

    return 0;
}
```

Status : Partially correct ***Marks : 5/10***