Rajalakshmi Engineering College

Name: Pranav Narayanan
Email: 240701393@rajalakshmi.edu.in
Roll no:
Phone: 9500579427
Branch: REC
Department: I CSE FD
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 : 25

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>
```

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

Term* createTerm(int coeff, int exp) {
    Term* newTerm = (Term*)malloc(sizeof(Term));
    newTerm->coeff = coeff;
    newTerm->exp = exp;
    newTerm->next = NULL;
    return newTerm;
}

void insertTerm(Term** head, int coeff, int exp) {
    Term* newTerm = createTerm(coeff, exp);
    if (*head == NULL || (*head)->exp < exp) {
        newTerm->next = *head;
        *head = newTerm;
    } else {
        Term* temp = *head;
        while (temp->next != NULL && temp->next->exp >= exp)
            temp = temp->next;
        newTerm->next = temp->next;
        temp->next = newTerm;
    }
}

void readPolynomial(Term** poly, int count) {
    int coeff, exp;
    for (int i = 0; i < count; ++i) {
        scanf("%d %d", &coeff, &exp);
        insertTerm(poly, coeff, exp);
    }
}

void printPolynomial(const char* label, Term* poly) {
    printf("%s", label);
    while (poly) {
        printf("(%dx^%d)", poly->coeff, poly->exp);
        if (poly->next)
```

```
            printf(" + ");
        poly = poly->next;
    }
    printf("\n");
}

int comparePolynomials(Term* p1, Term* p2) {
    while (p1 && p2) {
        if (p1->coeff != p2->coeff || p1->exp != p2->exp)
            return 0;
        p1 = p1->next;
        p2 = p2->next;
    }
    return p1 == NULL && p2 == NULL;
}

void freePolynomial(Term* poly) {
    Term* temp;
    while (poly) {
        temp = poly;
        poly = poly->next;
        free(temp);
    }
}

int main() {
    Term *poly1 = NULL, *poly2 = NULL;
    int n, m;

    scanf("%d", &n);
    readPolynomial(&poly1, n);

    scanf("%d", &m);
    readPolynomial(&poly2, m);

    printPolynomial("Polynomial 1: ", poly1);
    printPolynomial("Polynomial 2: ", poly2);

    if (comparePolynomials(poly1, poly2))
        printf("Polynomials are Equal.\n");
    else
        printf("Polynomials are Not Equal.\n");
    freePolynomial(poly1);
    freePolynomial(poly2);
    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>

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

Term* createTerm(int coeff, int exp) {
    Term* newTerm = (Term*)malloc(sizeof(Term));
    newTerm->coeff = coeff;
    newTerm->exp = exp;
    newTerm->next = NULL;
    return newTerm;
}

void insertOrUpdateTerm(Term** head, int coeff, int exp) {
    if (coeff == 0) return;
    Term* prev = NULL;
    Term* current = *head;

    // Insert in descending order and combine like terms
    while (current && current->exp > exp) {
        prev = current;
        current = current->next;
    }

    if (current && current->exp == exp) {
        current->coeff += coeff;
        if (current->coeff == 0) { // remove zero coefficient term
            if (prev)
                prev->next = current->next;
            else
                *head = current->next;
            free(current);
        }
    } else {
        Term* newTerm = createTerm(coeff, exp);
        if (prev == NULL) {
            newTerm->next = *head;
            *head = newTerm;
        } else {
            newTerm->next = prev->next;
            prev->next = newTerm;
        }
    }
}

void readPolynomial(Term** poly, int count) {
    int coeff, exp;
    for (int i = 0; i < count; i++) {
        scanf("%d %d", &coeff, &exp);
        insertOrUpdateTerm(poly, coeff, exp);
    }
}

Term* multiplyPolynomials(Term* poly1, Term* poly2) {
    Term* result = NULL;
    for (Term* p1 = poly1; p1 != NULL; p1 = p1->next) {
        for (Term* p2 = poly2; p2 != NULL; p2 = p2->next) {
            int coeff = p1->coeff * p2->coeff;
            int exp = p1->exp + p2->exp;
            insertOrUpdateTerm(&result, coeff, exp);
        }
    }
    return result;
}

void deleteTermByExponent(Term** head, int exp) {
    Term* temp = *head, *prev = NULL;

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

    if (temp == NULL) return;

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

    free(temp);
}

void printPolynomial(const char* label, Term* poly) {
    printf("%s", label);
    Term* temp = poly;
    if (!temp) {
        printf("0\n");
        return;
    }

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

```
void freePolynomial(Term* poly) {
   while (poly) {
      Term* temp = poly;
      poly = poly->next;
      free(temp);
   }
}

int main() {
   int n, m, expToDelete;
   Term *poly1 = NULL, *poly2 = NULL;

   scanf("%d", &n);
   readPolynomial(&poly1, n);

   scanf("%d", &m);
   readPolynomial(&poly2, m);

   scanf("%d", &expToDelete);

   Term* result = multiplyPolynomials(poly1, poly2);

   printPolynomial("Result of the multiplication: ", result);

   deleteTermByExponent(&result, expToDelete);

   printPolynomial("Result after deleting the term: ", result);

   freePolynomial(poly1);
   freePolynomial(poly2);
   freePolynomial(result);
   return 0;
}
```

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

3. Problem Statement

Lisa is studying polynomials in her class. She is learning about the multiplication of polynomials.

To practice her understanding, she wants to write a program that multiplies two polynomials and displays the result. Each polynomial is represented as a linked list, where each node contains the coefficient and exponent of a term.

Example

Input:

```
4 3
y
3 1
y
1 0
n
2 2
y
3 1
y
2 0
n
```

Output:

```
8x^5 + 12x^4 + 14x^3 + 11x^2 + 9x + 2
```

Explanation

1. Poly1: 4x^3 + 3x + 1
2. Poly2: 2x^2 + 3x + 2

Multiplication Steps:

1. Multiply 4x^3 by Poly2:
   -> 4x^3 * 2x^2 = 8x^5
   -> 4x^3 * 3x = 12x^4
   -> 4x^3 * 2 = 8x^3
2. Multiply 3x by Poly2:
   -> 3x * 2x^2 = 6x^3
   -> 3x * 3x = 9x^2
   -> 3x * 2 = 6x
3. Multiply 1 by Poly2:
   -> 1 * 2x^2 = 2x^2
   -> 1 * 3x = 3x
   -> 1 * 2 = 2

Combine the results: 8x^5 + 12x^4 + (8x^3 + 6x^3) + (9x^2 + 2x^2) + (6x + 3x) + 2

The combined polynomial is: 8x^5 + 12x^4 + 14x^3 + 11x^2 + 9x + 2

Input Format

The input consists of two sets of polynomial terms.

Each polynomial term is represented by two integers separated by a space:

- The first integer represents the coefficient of the term.
- The second integer represents the exponent of the term.

After entering a polynomial term, the user is prompted to input a character indicating whether to continue adding more terms to the polynomial.

If the user inputs 'y' or 'Y', the program continues to accept more terms.

If the user inputs 'n' or 'N', the program moves on to the next polynomial.

Output Format

The output consists of a single line representing the resulting polynomial after multiplying the two input polynomials.

Each term of the resulting polynomial is formatted as follows:

- The coefficient and exponent are separated by 'x^' if the exponent is greater than 1.
- If the exponent is 1, only 'x' is displayed without the exponent.
- If the exponent is 0, only the coefficient is displayed.

Refer to the sample output for the formatting specifications.

Sample Test Case

```
Input: 4 3
y
3 1
y
1 0
n
2 2
y
3 1
y
2 0
n
Output: 8x^5 + 12x^4 + 14x^3 + 11x^2 + 9x + 2
```

Answer

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

typedef struct Node {
```

```c
    int coeff;
    int expo;
    struct Node* next;
} Node;

Node* createNode(int coeff, int expo) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->coeff = coeff;
    newNode->expo = expo;
    newNode->next = NULL;
    return newNode;
}

Node* insertTerm(Node* head, int coeff, int expo) {
    Node* newNode = createNode(coeff, expo);
    if (!head) return newNode;

    Node* temp = head;
    while (temp->next) temp = temp->next;
    temp->next = newNode;
    return head;
}

Node* multiply(Node* poly1, Node* poly2) {
    Node* result = NULL;
    for (Node* p1 = poly1; p1 != NULL; p1 = p1->next) {
        for (Node* p2 = poly2; p2 != NULL; p2 = p2->next) {
            int coeff = p1->coeff * p2->coeff;
            int expo = p1->expo + p2->expo;

            Node* temp = result;
            int found = 0;

            while (temp) {
                if (temp->expo == expo) {
                    temp->coeff += coeff;
                    found = 1;
                    break;
                }
                temp = temp->next;
            }
            if (!found) {
                result = insertTerm(result, coeff, expo);
            }
        }
    }
    return result;
}

void printPolynomial(Node* head) {
    Node* temp = head;
    while (temp) {
        if (temp->coeff == 0) {
            temp = temp->next;
            continue;
        }
        if (temp != head && temp->coeff > 0) printf(" + ");

        if (temp->expo == 0)
            printf("%d", temp->coeff);
        else if (temp->expo == 1)
            printf("%dx", temp->coeff);
        else
            printf("%dx^%d", temp->coeff, temp->expo);
        temp = temp->next;
    }
    printf("\n");
}

Node* inputPolynomial() {
    int coeff, expo;
    char ch;
    Node* poly = NULL;
    do {
        scanf("%d%d", &coeff, &expo);
        poly = insertTerm(poly, coeff, expo);
        scanf(" %c", &ch);
    } while (ch == 'y' || ch == 'Y');
    return poly;
}

int main() {
    Node *poly1, *poly2, *result;
    poly1 = inputPolynomial();
    poly2 = inputPolynomial();

    result = multiply(poly1, poly2);
    printPolynomial(result);

    return 0;
}
```

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