Rajalakshmi Engineering College

Name: Prithika S
Email: 240701400@rajalakshmi.edu.in
Roll no: 240701400
Phone: 9790212894
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 : 2
Total Mark : 30
Marks Obtained : 25

Section 1 : Coding

1. 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 {
    int coeff, exp;
    struct Node* next;
} Node;

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

void insert(Node** poly, int coeff, int exp) {
    Node* newNode = createNode(coeff, exp);
```

```
    if (*poly == NULL) {
        *poly = newNode;
        return;
    }
    Node* temp = *poly;
    while (temp->next)
        temp = temp->next;
    temp->next = newNode;
}

Node* multiply(Node* poly1, Node* poly2) {
    Node* result = NULL;
    for (Node* ptr1 = poly1; ptr1; ptr1 = ptr1->next) {
        for (Node* ptr2 = poly2; ptr2; ptr2 = ptr2->next) {
            insert(&result, ptr1->coeff * ptr2->coeff, ptr1->exp + ptr2->exp);
        }
    }
    return result;
}

void simplify(Node** poly) {
    Node* temp = *poly;
    while (temp) {
        Node* runner = temp->next, *prev = temp;
        while (runner) {
            if (runner->exp == temp->exp) {
                temp->coeff += runner->coeff;
                prev->next = runner->next;
                free(runner);
                runner = prev->next;
            } else {
                prev = runner;
                runner = runner->next;
            }
        }
        temp = temp->next;
    }
}

void display(Node* poly) {
    while (poly) {
        printf("%d", poly->coeff);
```

```
        if (poly->exp > 1)
            printf("x^%d", poly->exp);
        else if (poly->exp == 1)
            printf("x");
        poly = poly->next;
        if (poly)
            printf(" + ");
    }
    printf("\n");
}

int main() {
    Node* poly1 = NULL, *poly2 = NULL;
    int coeff, exp;
    char choice;

    while (scanf("%d %d", &coeff, &exp) == 2) {
        insert(&poly1, coeff, exp);
        scanf(" %c", &choice);
        if (choice == 'n' || choice == 'N')
            break;
    }

    while (scanf("%d %d", &coeff, &exp) == 2) {
        insert(&poly2, coeff, exp);
        scanf(" %c", &choice);
        if (choice == 'n' || choice == 'N')
            break;
    }

    Node* result = multiply(poly1, poly2);
    simplify(&result);
    display(result);

    return 0;
}
```

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

2. 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>

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

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

void appendTerm(Node** head, int coeff, int exp) {
    Node* newNode = createNode(coeff, exp);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* temp = *head;
    while (temp->next)
        temp = temp->next;
    temp->next = newNode;
}
```

```
void insertTerm(Node** head, int coeff, int exp) {
    if (coeff == 0) return;
    Node* temp = *head;
    Node* prev = NULL;
    while (temp && temp->exp > exp) {
        prev = temp;
        temp = temp->next;
    }
    if (temp && temp->exp == exp) {
        temp->coeff += coeff;
        if (temp->coeff == 0) {
            if (prev)
                prev->next = temp->next;
            else
                *head = temp->next;
            free(temp);
        }
    } else {
        Node* newNode = createNode(coeff, exp);
        newNode->next = temp;
        if (prev)
            prev->next = newNode;
        else
            *head = newNode;
    }
}

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

void printPolynomial(Node* head) {
    Node* temp = head;
    int first = 1;
    while (temp) {
        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");
}

int main() {
    int n, m, coeff, exp;
    Node* poly1 = NULL;
    Node* poly2 = NULL;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &coeff, &exp);
        appendTerm(&poly1, coeff, exp);
    }
    scanf("%d", &m);
    for (int i = 0; i < m; i++) {
        scanf("%d %d", &coeff, &exp);
        appendTerm(&poly2, coeff, exp);
    }
    Node* result = multiplyPolynomials(poly1, poly2);
    printPolynomial(poly1);
    printPolynomial(poly2);
    printPolynomial(result);
    return 0;
}
```

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

3. 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>

// Define structure for polynomial term
typedef struct Term {
    int coeff;
    int expo;
    struct Term* next;
} Term;

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

// Append term to the polynomial linked list
void appendTerm(Term** head, int coeff, int expo) {
    Term* newNode = createTerm(coeff, expo);
    if (*head == NULL) {
        *head = newNode;
    } else {
        Term* temp = *head;
        while (temp->next)
            temp = temp->next;
        temp->next = newNode;
    }
}

// Display the polynomial
```

```
void printPolynomial(const char* label, Term* head) {
    printf("%s", label);
    Term* current = head;
    while (current) {
        printf("(%dx^%d)", current->coeff, current->expo);
        if (current->next)
            printf(" + ");
        current = current->next;
    }
    printf(" ");
}

// Compare two polynomials for equality
int arePolynomialsEqual(Term* poly1, Term* poly2) {
    while (poly1 && poly2) {
        if (poly1->coeff != poly2->coeff || poly1->expo != poly2->expo)
            return 0;
        poly1 = poly1->next;
        poly2 = poly2->next;
    }
    return (poly1 == NULL && poly2 == NULL);
}

// Free memory used by the polynomial
void freePolynomial(Term* head) {
    while (head) {
        Term* temp = head;
        head = head->next;
        free(temp);
    }
}

// Main function
int main() {
    int n, m, coeff, expo;

    // Read first polynomial
    scanf("%d", &n);
    Term* poly1 = NULL;
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &coeff, &expo);
        appendTerm(&poly1, coeff, expo);
```

```
    }

    // Read second polynomial
    scanf("%d", &m);
    Term* poly2 = NULL;
    for (int i = 0; i < m; i++) {
        scanf("%d %d", &coeff, &expo);
        appendTerm(&poly2, coeff, expo);
    }

    // Output
    printPolynomial("Polynomial 1: ", poly1);
    printf("\n");
    printPolynomial("Polynomial 2: ", poly2);
    printf("\n");

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

    // Clean up
    freePolynomial(poly1);
    freePolynomial(poly2);

    return 0;
}
```

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