Rajalakshmi Engineering College

Name: Sidharth P
Email: 240701514@rajalakshmi.edu.in
Roll no: 240701514
Phone: 7695968308
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 : 17.5

Section 1 : Coding

1. 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, exp;
   struct Node *next;
} Node;

// Function to create a new node
Node* createNode(int coeff, int exp) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   if (newNode == NULL) {
      printf("Memory allocation failed\n");
      exit(1);
   }
   newNode->coeff = coeff;
   newNode->exp = exp;
   newNode->next = NULL;
   return newNode;
}

// Function to insert a term into the polynomial in sorted order (descending
exponent)
Node* insertSorted(Node* head, int coeff, int exp) {
   if (coeff == 0) {
      return head; // Ignore zero coefficient terms
   }

   Node* newNode = createNode(coeff, exp);

   if (head == NULL || head->exp < exp) {
      newNode->next = head;
      return newNode;
   }

   Node* current = head;
   Node* prev = NULL;

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

   if (current != NULL && current->exp == exp) {
      current->coeff += coeff;
```

```
        free(newNode);
        if (current->coeff == 0) {
            if (prev == NULL) {
                head = current->next;
            } else {
                prev->next = current->next;
            }
            free(current);
        }
    } else {
        newNode->next = current;
        if (prev == NULL) {
            head = newNode;
        } else {
            prev->next = newNode;
        }
    }

    return head;
}

// Function to multiply two polynomials
Node* multiplyPolynomials(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 newCoeff = p1->coeff * p2->coeff;
            int newExp = p1->exp + p2->exp;
            result = insertSorted(result, newCoeff, newExp);
        }
    }

    return result;
}

// Function to print the polynomial with proper formatting
void printPolynomial(Node* poly) {
    if (poly == NULL) {
        printf("0\n");
        return;
    }
    int firstTerm = 1;
    while (poly != NULL) {
        if (!firstTerm) {
            printf(" + ");
        }
        firstTerm = 0;

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

// Function to free the memory allocated for the polynomial
void freePolynomial(Node* poly) {
    while (poly != NULL) {
        Node* temp = poly;
        poly = poly->next;
        free(temp);
    }
}

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

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

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

    // Print the input polynomials
    printPolynomial(poly1);
    printPolynomial(poly2);

    // Multiply the polynomials and print the result
    Node* result = multiplyPolynomials(poly1, poly2);
    printPolynomial(result);

    // Free the allocated memory
    freePolynomial(poly1);
    freePolynomial(poly2);
    freePolynomial(result);

    return 0;
}
```

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

2. Problem Statement

Akila is a tech enthusiast and wants to write a program to add two polynomials. Each polynomial is represented as a linked list, where each node in the list represents a term in the polynomial.

A term in the polynomial is represented in the format ax^b, where a is the coefficient and b is the exponent.

Akila needs your help to implement a program that takes two polynomials as input, adds them, and stores the result in ascending order in a new polynomial-linked list. Write a program to help her.

Input Format

The input consists of lines containing pairs of integers representing the coefficients and exponents of polynomial terms.

Each line represents a single term, with the coefficient and exponent separated by a space.

The input for each polynomial ends with a line containing "0 0".

Output Format

The output consists of three lines representing the first, second, and resulting polynomial after the addition operation, with terms sorted in ascending order of exponents.

Each line contains terms of the polynomial in the format "coefficientx^exponent", separated by " + ".

If the resulting polynomial is zero, the output is "0".

Refer to the sample output for the formatting specifications.

Sample Test Case

```
Input: 3 4
2 3
1 2
0 0
1 2
2 3
3 4
0 0
Output: 1x^2 + 2x^3 + 3x^4
1x^2 + 2x^3 + 3x^4
2x^2 + 4x^3 + 6x^4
```

Answer

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

typedef struct Node {
   int coeff, exp;
   struct Node *next;
```

```
} Node;

// Insert term into sorted linked list, merging like terms
Node* insertSorted(Node* head, int coeff, int exp) {
    if (coeff == 0) return head; // Ignore zero coefficient terms

    Node* temp = (Node*)malloc(sizeof(Node));
    temp->coeff = coeff;
    temp->exp = exp;
    temp->next = NULL;

    if (!head || head->exp > exp) { // Insert at head if exponent is smallest
        temp->next = head;
        return temp;
    }

    Node *prev = NULL, *curr = head;
    while (curr && curr->exp < exp) { // Find correct position
        prev = curr;
        curr = curr->next;
    }

    if (curr && curr->exp == exp) { // Merge like terms
        curr->coeff += coeff;
        if (curr->coeff == 0) { // Remove zero term
            if (prev) prev->next = curr->next;
            else head = curr->next;
            free(curr);
        }
        free(temp);
    } else { // Insert new term in sorted order
        temp->next = curr;
        if (prev) prev->next = temp;
        else head = temp;
    }
    return head;
}

// Add two polynomials and return the result in ascending order
Node* addPolynomials(Node* poly1, Node* poly2) {
    Node* res = NULL;
    while (poly1 || poly2) {
        if (poly2 == NULL || (poly1 && poly1->exp < poly2->exp)) {
            res = insertSorted(res, poly1->coeff, poly1->exp);
            poly1 = poly1->next;
        } else if (poly1 == NULL || (poly2 && poly2->exp < poly1->exp)) {
            res = insertSorted(res, poly2->coeff, poly2->exp);
            poly2 = poly2->next;
        } else { // Same exponent, add coefficients
            res = insertSorted(res, poly1->coeff + poly2->coeff, poly1->exp);
            poly1 = poly1->next;
            poly2 = poly2->next;
        }
    }
    return res;
}

// Print polynomial in required format
void printPoly(Node* poly) {
    if (!poly) {
        printf("0\n");
        return;
    }
    while (poly) {
        printf("%d%s%d%s", poly->coeff, "x^", poly->exp, poly->next ? " + " : "\n");
        poly = poly->next;
    }
}

// Free memory allocated for polynomial linked list
void freePoly(Node* poly) {
    while (poly) {
        Node* temp = poly;
        poly = poly->next;
        free(temp);
    }
}

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

    // Read first polynomial
    while (scanf("%d %d", &coeff, &exp) && (coeff || exp))
        poly1 = insertSorted(poly1, coeff, exp);

    // Read second polynomial
    while (scanf("%d %d", &coeff, &exp) && (coeff || exp))
        poly2 = insertSorted(poly2, coeff, exp);

    // Print input polynomials
    printPoly(poly1);
    printPoly(poly2);

    // Add polynomials and print result
    Node* result = addPolynomials(poly1, poly2);
    printPoly(result);

    // Free allocated memory
    freePoly(poly1);
    freePoly(poly2);
    freePoly(result);

    return 0;
}
```

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

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

// Insert a term into the polynomial in descending order
Node* insertSorted(Node* head, int coeff, int exp) {
   if (coeff == 0) return head; // Ignore zero coefficient terms

   Node* temp = (Node*)malloc(sizeof(Node));
   temp->coeff = coeff;
   temp->exp = exp;
   temp->next = NULL;

   if (!head || head->exp < exp) { // Insert at head if highest exponent
```

```
        temp->next = head;
        return temp;
    }

    Node *prev = NULL, *curr = head;
    while (curr && curr->exp > exp) { // Find correct position
        prev = curr;
        curr = curr->next;
    }

    if (curr && curr->exp == exp) { // Merge like terms
        curr->coeff += coeff;
        if (curr->coeff == 0) { // Remove zero coefficient term
            if (prev) prev->next = curr->next;
            else head = curr->next;
            free(curr);
        }
        free(temp);
    } else { // Insert new term
        temp->next = curr;
        if (prev) prev->next = temp;
        else head = temp;
    }
    return head;
}

// Multiply two polynomials
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) {
            result = insertSorted(result, p1->coeff * p2->coeff, p1->exp + p2->exp);
        }
    }
    return result;
}

// Print polynomial in correct format
void printPoly(Node* poly) {
    if (!poly) {
        printf("0\n");
        return;
    }

    int first = 1; // Flag to avoid extra "+"
    while (poly) {
        if (poly->coeff < 0) {
            if (!first) printf(" - ");
            else printf("-");
        } else if (!first) {
            printf(" + ");
        }

        if (poly->exp > 1)
            printf("%dx^%d", abs(poly->coeff), poly->exp);
        else if (poly->exp == 1)
            printf("%dx", abs(poly->coeff));
        else
            printf("%d", abs(poly->coeff));

        first = 0;
        poly = poly->next;
    }
    printf("\n");
}

// Free allocated memory
void freePoly(Node* poly) {
    while (poly) {
        Node* temp = poly;
        poly = poly->next;
        free(temp);
    }
}

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

    // Read first polynomial
    do {
        scanf("%d %d %c", &coeff, &exp, &choice);
        poly1 = insertSorted(poly1, coeff, exp);
    } while (choice == 'y' || choice == 'Y');

    // Read second polynomial
    do {
        scanf("%d %d %c", &coeff, &exp, &choice);
        poly2 = insertSorted(poly2, coeff, exp);
    } while (choice == 'y' || choice == 'Y');

    // Multiply polynomials and print result
    Node* result = multiplyPolynomials(poly1, poly2);
    printPoly(result);

    // Free allocated memory
    freePoly(poly1);
    freePoly(poly2);
    freePoly(result);

    return 0;
}
```

Status : Partially correct ***Marks : 2.5/10***