Rajalakshmi Engineering College

Name: shobbika T
Email: 240701502@rajalakshmi.edu.in
Roll no: 240701502
Phone: 7305423247
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

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

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

// Node structure for singly linked list
typedef struct Node {
    int coeff;
    int exp;
    struct Node* next;
} Node;

// Function to create a new node
Node* createNode(int coeff, int exp) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->coeff = coeff;
    newNode->exp = exp;
    newNode->next = NULL;
    return newNode;
```

```
}

// Insert node at the end
void insertNode(Node** head, int coeff, int exp) {
    Node* newNode = createNode(coeff, exp);
    if (*head == NULL) {
        *head = newNode;
    } else {
        Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

// Add terms with the same exponent
void addTerm(Node** head, int coeff, int exp) {
    if (coeff == 0) return;
    Node* temp = *head;
    Node* prev = NULL;
    while (temp != NULL && temp->exp > exp) {
        prev = temp;
        temp = temp->next;
    }

    if (temp != NULL && temp->exp == exp) {
        temp->coeff += coeff;
        if (temp->coeff == 0) {
            // Remove node if coefficient becomes 0
            if (prev == NULL) {
                *head = temp->next;
            } else {
                prev->next = temp->next;
            }
            free(temp);
        }
    } else {
        Node* newNode = createNode(coeff, exp);
        if (prev == NULL) {
            newNode->next = *head;
            *head = newNode;
```

```
        } else {
            newNode->next = prev->next;
            prev->next = newNode;
        }
    }
}

// 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 coeff = p1->coeff * p2->coeff;
            int exp = p1->exp + p2->exp;
            addTerm(&result, coeff, exp);
        }
    }
    return result;
}

// Print polynomial
void printPolynomial(Node* head) {
    Node* temp = head;
    int first = 1;

    while (temp != NULL) {
        if (temp->coeff > 0 && !first)
            printf(" + ");
        else if (temp->coeff < 0)
            printf(" - ");

        int coeff = abs(temp->coeff);

        if (temp->exp == 0)
            printf("%d", coeff);
        else if (temp->exp == 1)
            printf("%dx", coeff);
        else
            printf("%dx^%d", coeff, temp->exp);

        temp = temp->next;
        first = 0;
```

```
    }
    printf("\n");
}

// Free memory
void freeList(Node* head) {
    Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}

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

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

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

    // Multiply and display result
    Node* result = multiplyPolynomials(poly1, poly2);
    printPolynomial(result);

    // Free memory
    freeList(poly1);
    freeList(poly2);
    freeList(result);
```

```
    return 0;
}
```

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

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

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

// Node structure for the linked list
typedef struct Node {
    int coeff;
    int exp;
    struct Node* next;
} Node;

// Function to create a new node
Node* createNode(int coeff, int exp) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->coeff = coeff;
    newNode->exp = exp;
    newNode->next = NULL;
    return newNode;
}

// Insert node in sorted order (by exponent descending)
void insertNode(Node** head, int coeff, int exp) {
    if (coeff == 0) return; // Skip zero coefficients
```

```
    Node* newNode = createNode(coeff, exp);
    if (*head == NULL || (*head)->exp < exp) {
        newNode->next = *head;
        *head = newNode;
        return;
    }

    Node* current = *head;
    while (current->next != NULL && current->next->exp > exp) {
        current = current->next;
    }

    if (current->next != NULL && current->next->exp == exp) {
        current->next->coeff += coeff;
        if (current->next->coeff == 0) {
            Node* temp = current->next;
            current->next = temp->next;
            free(temp);
        }
        free(newNode);
    } else {
        newNode->next = current->next;
        current->next = newNode;
    }
}

// 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 coeff = p1->coeff * p2->coeff;
            int exp = p1->exp + p2->exp;
            insertNode(&result, coeff, exp);
        }
    }
    return result;
}

// Delete a term by exponent
void deleteTerm(Node** head, int exp) {
```

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

    if (current != NULL) {
        if (prev == NULL) {
            *head = current->next;
        } else {
            prev->next = current->next;
        }
        free(current);
    }
}

// Print polynomial
void printPolynomial(Node* head) {
    Node* temp = head;
    int first = 1;

    while (temp != NULL) {
        if (temp->coeff > 0 && !first)
            printf(" + ");
        else if (temp->coeff < 0)
            printf(" - ");

        int coeff = abs(temp->coeff);
        if (temp->exp == 0)
            printf("%d", coeff);
        else if (temp->exp == 1)
            printf("%dx", coeff);
        else
            printf("%dx^%d", coeff, temp->exp);

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

```
// Free memory
void freeList(Node* head) {
    Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}

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

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

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

    // Input exponent to delete
    scanf("%d", &delExp);

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

    // Display multiplication result
    printf("Result of the multiplication: ");
    printPolynomial(result);

    // Delete specified term
    deleteTerm(&result, delExp);

    // Display result after deletion
```

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

    // Free memory
    freeList(poly1);
    freeList(poly2);
    freeList(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

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

// Node structure for the linked list
typedef struct Node {
    int coeff;
    int exp;
    struct Node* next;
} Node;

// Function to create a new node
Node* createNode(int coeff, int exp) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->coeff = coeff;
```

```
    newNode->exp = exp;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a node at the end of the list
void insertNode(Node** head, int coeff, int exp) {
    Node* newNode = createNode(coeff, exp);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* current = *head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = newNode;
}

// 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 coeff = p1->coeff * p2->coeff;
            int exp = p1->exp + p2->exp;

            // Insert or update the term in the result
            Node* temp = result;
            Node* prev = NULL;
            while (temp != NULL && temp->exp > exp) {
                prev = temp;
                temp = temp->next;
            }

            if (temp != NULL && temp->exp == exp) {
                temp->coeff += coeff;
                if (temp->coeff == 0) {  // Remove zero coefficient terms
                    if (prev == NULL) {
                        result = temp->next;
                    } else {
                        prev->next = temp->next;
```

```
            }
            free(temp);
        }
    } else {
        Node* newNode = createNode(coeff, exp);
        if (prev == NULL) {
            newNode->next = result;
            result = newNode;
        } else {
            newNode->next = prev->next;
            prev->next = newNode;
        }
    }
  }
  return result;
}

// Function to print a polynomial
void printPolynomial(Node* head) {
  Node* current = head;
  int first = 1;
  while (current != NULL) {
    if (!first && current->coeff > 0)
      printf(" + ");
    else if (current->coeff < 0)
      printf(" - ");
    else if (!first)
      printf(" + ");

    int coeff = abs(current->coeff);
    if (current->exp == 0)
      printf("%d", coeff);
    else if (current->exp == 1)
      printf("%dx", coeff);
    else
      printf("%dx^%d", coeff, current->exp);

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

```
}

// Free memory allocated for the linked list
void freeList(Node* head) {
    Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}

int main() {
    int n, m, coeff, exp;

    // Input for the first polynomial
    scanf("%d", &n);
    Node* poly1 = NULL;
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &coeff, &exp);
        insertNode(&poly1, coeff, exp);
    }

    // Input for the second polynomial
    scanf("%d", &m);
    Node* poly2 = NULL;
    for (int i = 0; i < m; i++) {
        scanf("%d %d", &coeff, &exp);
        insertNode(&poly2, coeff, exp);
    }

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

    // Outpu
    printPolynomial(poly1);
    printPolynomial(poly2);
    printPolynomial(result);

    // Free allocated memory
    freeList(poly1);
    freeList(poly2);
```

```
    freeList(result);

    return 0;
}
```

Status : Partially correct ***Marks : 7/10***