# Rajalakshmi Engineering College

Name: Pavithra J
Email: 240701381@rajalakshmi.edu.in
Roll no: 240701381
Phone: 9363364978
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 : 22.5

## Section 1 : Coding

1. Problem Statement

John is working on a math processing application, and his task is to simplify polynomials entered by users. The polynomial is represented as a linked list, where each node contains two properties:

Coefficient of the term.

Exponent of the term.

John's goal is to combine all the terms that have the same exponent, effectively simplifying the polynomial.

***Input Format***

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

The next n lines of input consist of two integers, representing the coefficient and exponent of the polynomial in each line separated by space.

***Output Format***

The first line of output prints the original polynomial in the format 'cx^e + cx^e + ...' (where c is the coefficient and e is the exponent of each term).

The second line of output displays the simplified polynomial in the same format as the original polynomial.

If the polynomial is 0, then only '0' will be printed.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 3
5 2
3 1
6 2
Output: Original polynomial: 5x^2 + 3x^1 + 6x^2
Simplified polynomial: 11x^2 + 3x^1

***Answer***

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

// Define the structure for a polynomial term
typedef struct Node {
    int coefficient;
    int exponent;
    struct Node* next;
} Node;

// Function to create a new node for the polynomial
Node* createNode(int coefficient, int exponent) {
    Node* newNode = (Node*) malloc(sizeof(Node));
    newNode->coefficient = coefficient;
    newNode->exponent = exponent;
    newNode->next = NULL;
```

```
    return newNode;
}

// Function to insert a node in the polynomial list
void insertNode(Node** head, int coefficient, int exponent) {
    Node* newNode = createNode(coefficient, exponent);
    if (*head == NULL) {
        *head = newNode;
    } else {
        Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

// Function to print the polynomial in the required format
void printPolynomial(Node* head) {
    if (head == NULL) {
        printf("0");
        return;
    }

    Node* temp = head;
    int first = 1; // To handle '+' sign formatting
    while (temp != NULL) {
        if (first) {
            first = 0;
        } else {
            printf(" + ");
        }
        printf("%dx^%d", temp->coefficient, temp->exponent);
        temp = temp->next;
    }
}

// Function to simplify the polynomial by combining like terms
Node* simplifyPolynomial(Node* head) {
    Node* simplified = NULL;
    Node* current = head;
```

```
    while (current != NULL) {
        Node* temp = simplified;
        int found = 0;

        // Check if the exponent already exists in the simplified list
        while (temp != NULL) {
            if (temp->exponent == current->exponent) {
                // If found, combine the coefficients
                temp->coefficient += current->coefficient;
                found = 1;
                break;
            }
            temp = temp->next;
        }

        if (!found) {
            // If not found, insert the term in the simplified list
            insertNode(&simplified, current->coefficient, current->exponent);
        }

        current = current->next;
    }

    return simplified;
}

int main() {
    int n;
    scanf("%d", &n);

    Node* head = NULL;

    // Input the polynomial terms
    for (int i = 0; i < n; i++) {
        int coefficient, exponent;
        scanf("%d %d", &coefficient, &exponent);
        insertNode(&head, coefficient, exponent);
    }

    // Print the original polynomial
    printf("Original polynomial: ");
    printPolynomial(head);
```

```
    printf("\n");

    // Simplify the polynomial
    Node* simplified = simplifyPolynomial(head);

    // Print the simplified polynomial
    printf("Simplified polynomial: ");
    printPolynomial(simplified);
    printf("\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>

typedef struct Term {
    int coefficient;
    int exponent;
    struct Term* next;
} Term;

// Function to create a new term node
Term* createTerm(int coefficient, int exponent) {
    Term* newTerm = (Term*)malloc(sizeof(Term));
    newTerm->coefficient = coefficient;
    newTerm->exponent = exponent;
```

```
    newTerm->next = NULL;
    return newTerm;
}

// Function to insert a term in the polynomial in descending order of exponent
void insertTerm(Term** poly, int coefficient, int exponent) {
    Term* newTerm = createTerm(coefficient, exponent);

    if (*poly == NULL || (*poly)->exponent < exponent) {
        newTerm->next = *poly;
        *poly = newTerm;
    } else {
        Term* temp = *poly;
        while (temp->next != NULL && temp->next->exponent > exponent) {
            temp = temp->next;
        }

        if (temp->next != NULL && temp->next->exponent == exponent) {
            temp->next->coefficient += coefficient;
            free(newTerm);  // No need to insert if the exponent already exists
        } else {
            newTerm->next = temp->next;
            temp->next = newTerm;
        }
    }
}

// Function to multiply two polynomials
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->coefficient * p2->coefficient;
            int exp = p1->exponent + p2->exponent;
            insertTerm(&result, coeff, exp);
        }
    }

    return result;
}
```

```
// Function to delete a term with the given exponent from the polynomial
void deleteTerm(Term** poly, int exponent) {
    Term* temp = *poly;
    Term* prev = NULL;

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

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

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

    free(temp);
}

// Function to print the polynomial in the required format
void printPolynomial(Term* poly) {
    if (poly == NULL) {
        printf("0");
        return;
    }

    Term* temp = poly;
    int first = 1;

    while (temp != NULL) {
        if (!first) {
            printf(" + ");
        }
        if (temp->exponent == 1) {
            printf("%dx", temp->coefficient);
        } else if (temp->exponent == 0) {
            printf("%d", temp->coefficient);
        } else {
            printf("%dx^%d", temp->coefficient, temp->exponent);
        }
```

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

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

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

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

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

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

    // Print the resulting polynomial
    printf("Result of the multiplication: ");
    printPolynomial(result);

    // Delete the term with the specified exponent
    deleteTerm(&result, exponentToDelete);

    // Print the resulting polynomial 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>

typedef struct Node {
    int coefficient;
    int exponent;
    struct Node* next;
} Node;

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

// Function to insert a node into the polynomial in descending order of exponents
void insertNode(Node** head, int coefficient, int exponent) {
    Node* newNode = createNode(coefficient, exponent);
```

```
    if (*head == NULL || (*head)->exponent < exponent) {
        newNode->next = *head;
        *head = newNode;
    } else {
        Node* temp = *head;
        while (temp->next != NULL && temp->next->exponent > exponent) {
            temp = temp->next;
        }
        if (temp->next != NULL && temp->next->exponent == exponent) {
            temp->next->coefficient += coefficient;
            free(newNode);
        } else {
            newNode->next = temp->next;
            temp->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->coefficient * p2->coefficient;
            int exp = p1->exponent + p2->exponent;
            insertNode(&result, coeff, exp);
        }
    }
    return result;
}

// Function to print the polynomial in the required format
void printPolynomial(Node* head) {
    if (head == NULL) {
        printf("0");
        return;
    }

    Node* temp = head;
    int first = 1;
    while (temp != NULL) {
        if (!first) {
```

```
            printf(" + ");
        }
        if (temp->exponent == 1) {
            printf("%dx", temp->coefficient);
        } else if (temp->exponent == 0) {
            printf("%d", temp->coefficient);
        } else {
            printf("%dx^%d", temp->coefficient, temp->exponent);
        }
        temp = temp->next;
        first = 0;
    }
    printf("\n");
}

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

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

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

    // Print the first polynomial
    printPolynomial(poly1);

    // Print the second polynomial
    printPolynomial(poly2);
```

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

    // Print the resulting polynomial
    printPolynomial(result);

    return 0;
}
```

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