# Rajalakshmi Engineering College

Name: SUBASH R
Email: 240701538@rajalakshmi.edu.in
Roll no: 240701538
Phone: 9150202710
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 : 30

## Section 1 : Coding

1. 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>

struct Term {
```

```
    int coefficient;
    int exponent;
};

struct Polynomial {
    struct Term *terms;
    int count;
    int capacity;
};

struct Polynomial createPolynomial(int capacity) {
    struct Polynomial poly;
    poly.terms = (struct Term *)malloc(capacity * sizeof(struct Term));
    poly.count = 0;
    poly.capacity = capacity;
    return poly;
}

void addTerm(struct Polynomial *poly, int coefficient, int exponent) {
    for (int i = 0; i < poly->count; i++) {
        if (poly->terms[i].exponent == exponent) {
            poly->terms[i].coefficient += coefficient;
            return;
        }
    }

    if (poly->count >= poly->capacity) {
        poly->capacity *= 2;
        poly->terms = (struct Term *)realloc(poly->terms, poly->capacity *
sizeof(struct Term));
    }

    poly->terms[poly->count].coefficient = coefficient;
    poly->terms[poly->count].exponent = exponent;
    poly->count++;
}

void deleteTerm(struct Polynomial *poly, int exponent) {
    for (int i = 0; i < poly->count; i++) {
        if (poly->terms[i].exponent == exponent) {
            for (int j = i; j < poly->count - 1; j++) {
                poly->terms[j] = poly->terms[j + 1];
```

```
            }
            poly->count--;
            return;
        }
    }
}

struct Polynomial multiplyPolynomials(struct Polynomial *poly1, struct
Polynomial *poly2) {
    struct Polynomial result = createPolynomial(poly1->count * poly2->count);

    for (int i = 0; i < poly1->count; i++) {
        for (int j = 0; j < poly2->count; j++) {
            int new_coef = poly1->terms[i].coefficient * poly2->terms[j].coefficient;
            int new_exp = poly1->terms[i].exponent + poly2->terms[j].exponent;
            addTerm(&result, new_coef, new_exp);
        }
    }

    return result;
}

void printPolynomial(struct Polynomial *poly) {
    if (poly->count == 0) {
        printf("0");
        return;
    }

    int firstTerm = 1;
    for (int i = 0; i < poly->count; i++) {
        if (poly->terms[i].coefficient == 0)
            continue;

        if (!firstTerm)
            printf(" + ");

        if (poly->terms[i].exponent == 0) {
            printf("%d", poly->terms[i].coefficient);
        } else if (poly->terms[i].exponent == 1) {
            printf("%dx", poly->terms[i].coefficient);
        } else {
            printf("%dx^%d", poly->terms[i].coefficient, poly->terms[i].exponent);
```

```
        }

        firstTerm = 0;
    }
}

int main() {
    int n, m, coefficient, exponent, delete_exp;

    scanf("%d", &n);
    struct Polynomial poly1 = createPolynomial(n);
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &coefficient, &exponent);
        addTerm(&poly1, coefficient, exponent);
    }

    scanf("%d", &m);
    struct Polynomial poly2 = createPolynomial(m);
    for (int i = 0; i < m; i++) {
        scanf("%d %d", &coefficient, &exponent);
        addTerm(&poly2, coefficient, exponent);
    }

    scanf("%d", &delete_exp);

    struct Polynomial result = multiplyPolynomials(&poly1, &poly2);


    printf("Result of the multiplication: ");
    printPolynomial(&result);
    printf("\n");

    deleteTerm(&result, delete_exp);

    printf("Result after deleting the term: ");

    printPolynomial(&result);
    printf("\n");

    free(poly1.terms);
    free(poly2.terms);
    free(result.terms);
```

```
    return 0;
}
```

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

2. Problem Statement

Timothy wants to evaluate polynomial expressions for his mathematics homework. He needs a program that allows him to input the coefficients of a polynomial based on its degree and compute the polynomial's value for a given input of x. Implement a function that takes the degree, coefficients, and the value of x, and returns the evaluated result of the polynomial.

Example

Input:

degree of the polynomial = 2

coefficient of x2 = 13

coefficient of x1 = 12

coefficient of x0 = 11

x = 1

Output:

36

Explanation:

Calculate the value of 13x2: 13 * 12 = 13.

Calculate the value of 12x1: 12 * 11 = 12.

Calculate the value of 11x0: 11 * 10 = 11.

Add the values of x2, x1, and x0 together: 13 + 12 + 11 = 36.

***Input Format***

The first line of input consists of an integer representing the degree of the polynomial.

The second line consists of an integer representing the coefficient of x2.

The third line consists of an integer representing the coefficient of x1.

The fourth line consists of an integer representing the coefficient of x0.

The fifth line consists of an integer representing the value of x, at which the polynomial should be evaluated.

***Output Format***

The output is an integer value obtained by evaluating the polynomial at the given value of x.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 2
13
12
11
1

Output: 36

***Answer***

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

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

Node* createNode(int coefficient, int exponent) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->coefficient = coefficient;
    newNode->exponent = exponent;
```

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

void insertNode(Node** head, int coefficient, int exponent) {
    Node* newNode = createNode(coefficient, exponent);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

int evaluatePolynomial(Node* head, int x) {
    int result = 0;
    Node* temp = head;

    while (temp != NULL) {
        result += temp->coefficient * pow(x, temp->exponent);
        temp = temp->next;
    }

    return result;
}

int main() {
    int degree, coefficient, x;
    Node* head = NULL;
    scanf("%d", &degree);
    for (int i = degree; i >= 0; i--) {
        scanf("%d", &coefficient);
        insertNode(&head, coefficient, i);
    }
    scanf("%d", &x);
    int result = evaluatePolynomial(head, x);
    printf("%d\n", result);
    return 0;
```

}

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

3. 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>

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

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

void insertNode(Node** head, int coefficient, int exponent) {
    Node* newNode = createNode(coefficient, exponent);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

void printPolynomial(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        printf("%dx^%d", temp->coefficient, temp->exponent);
        if (temp->next != NULL) {
            printf(" + ");
```

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

void simplifyPolynomialInOrder(Node** head) {
    Node* current = *head;
    Node* outer = current;

    while (outer != NULL) {
        Node* inner = outer->next;
        Node* prev = outer;

        while (inner != NULL) {
            if (inner->exponent == outer->exponent) {
                outer->coefficient += inner->coefficient;
                prev->next = inner->next;
                free(inner);
                inner = prev->next;
            } else {
                prev = inner;
                inner = inner->next;
            }
        }
        outer = outer->next;
    }
}

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

int main() {
    int n, coefficient, exponent;
    Node* head = NULL;
```

```
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &coefficient, &exponent);
        insertNode(&head, coefficient, exponent);
    }

    printf("Original polynomial: ");
    printPolynomial(head);

    simplifyPolynomialInOrder(&head);

    printf("Simplified polynomial: ");
    if (head == NULL) {
        printf("0\n");
    } else {
        printPolynomial(head);
    }

    freeList(head);
    return 0;
}
```

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