# Rajalakshmi Engineering College

Name: Rayvan Sanjai
Email: 240701425@rajalakshmi.edu.in
Roll no: 2116240701425
Phone: 9380572043
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

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 coef;
    int exp;
    struct Node* next;
} Node;

Node* createNode(int coef, int exp) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->coef = coef;
    newNode->exp = exp;
    newNode->next = NULL;
    return newNode;
}
void insertSorted(Node** head, int coef, int exp) {
    if (coef == 0) return;
```

```
    Node* newNode = createNode(coef, exp);
    Node* current = *head, *prev = NULL;
    while (current && current->exp > exp) {
        prev = current;
        current = current->next;
    }
    if (current && current->exp == exp) {
        current->coef += coef;
        if (current->coef == 0) {
            if (prev) prev->next = current->next;
            else *head = current->next;
            free(current);
        }
        free(newNode);
    } else {
        if (prev)
            prev->next = newNode;
        else
            *head = newNode;
        newNode->next = current;
    }
}

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) {
            insertSorted(&result, p1->coef * p2->coef, p1->exp + p2->exp);
        }
    }
    return result;
}
void printPolynomial(Node* poly) {
    if (!poly) {
        printf("0\n");
        return;
    }
    while (poly) {
        if (poly->exp > 1)
            printf("%dx^%d", poly->coef, poly->exp);
        else if (poly->exp == 1)
            printf("%dx", poly->coef);
```

```
        else
            printf("%d", poly->coef);

        if (poly->next)
            printf(" + ");

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

Node* readPolynomial() {
    Node* poly = NULL;
    int coef, exp;
    char cont;
    do {
        scanf("%d %d", &coef, &exp);
        insertSorted(&poly, coef, exp);
        scanf(" %c", &cont);
    } while (cont == 'y' || cont == 'Y');
    return poly;
}
int main() {
    Node* poly1 = readPolynomial();
    Node* poly2 = readPolynomial();
    Node* resultPoly = multiplyPolynomials(poly1, poly2);
    printPolynomial(resultPoly);
    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***

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int coefficient;
    int exponent;
    struct Node* next;
};
struct Node* createNode(int coefficient, int exponent) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->coefficient = coefficient;
    newNode->exponent = exponent;
    newNode->next = NULL;
    return newNode;
}
void insertTerm(struct Node** head, int coefficient, int exponent) {
    if (coefficient == 0) {
        return;
    }

    if (*head == NULL) {
        *head = createNode(coefficient, exponent);
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = createNode(coefficient, exponent);
    }
}
void displayPolynomial(struct Node* head) {
    struct Node* temp = head;
    int isFirst = 1;
    while (temp != NULL) {
        if (temp->coefficient != 0) {
            if (!isFirst && temp->coefficient > 0) {
                printf(" + ");
            }
            if (temp->exponent == 0) {
```

```
            printf("%d", temp->coefficient);
        } else if (temp->exponent == 1) {
            printf("%dx", temp->coefficient);
        } else {
            printf("%dx^%d", temp->coefficient, temp->exponent);
        }
        isFirst = 0;
    }
    temp = temp->next;
  }
  printf("\n");
}
void removeDuplicates(struct Node* head) {
  struct Node* ptr1, *ptr2, *dup;
  ptr1 = head;

  while (ptr1 != NULL && ptr1->next != NULL) {
    ptr2 = ptr1;

    while (ptr2->next != NULL) {
      if (ptr1->exponent == ptr2->next->exponent) {
        ptr1->coefficient = ptr1->coefficient + ptr2->next->coefficient;
        dup = ptr2->next;
        ptr2->next = ptr2->next->next;
        free(dup);
      } else {
        ptr2 = ptr2->next;
      }
    }
    ptr1 = ptr1->next;
  }
}
struct Node* multiplyPolynomials(struct Node* poly1, struct Node* poly2) {
  struct Node* result = NULL;
  struct Node* temp1 = poly1;
  while (temp1 != NULL) {
    struct Node* temp2 = poly2;
    while (temp2 != NULL) {
      int coeff = temp1->coefficient * temp2->coefficient;
      int exp = temp1->exponent + temp2->exponent;
      insertTerm(&result, coeff, exp);
      temp2 = temp2->next;
```

```
        }
        temp1 = temp1->next;
    }
    removeDuplicates(result);
    return result;
}
void deleteTerm(struct Node** head, int exponent) {
    struct Node* prev = NULL;
    struct Node* curr = *head;
    while (curr != NULL) {
        if (curr->exponent == exponent) {
            if (prev == NULL) {
                *head = curr->next;
            } else {
                prev->next = curr->next;
            }
            free(curr);
            return;
        }
        prev = curr;
        curr = curr->next;
    }
}
void freePolynomial(struct Node* head) {
    while (head != NULL) {
        struct Node* temp = head;
        head = head->next;
        free(temp);
    }
}
void addSamePowerCoefficients(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        struct Node* current = temp->next;
        while (current != NULL) {
            if (temp->exponent == current->exponent) {
                temp->coefficient += current->coefficient;
                deleteTerm(&head, current->exponent);
                current = temp->next;
            } else {
                current = current->next;
            }
```

```
        }
        temp = temp->next;
    }
}
int main() {
    struct Node* poly1 = NULL;
    struct Node* poly2 = NULL;
    struct Node* result = NULL;
    int coefficient, exponent;
    int numTerms1, numTerms2;
    scanf("%d", &numTerms1);
    for (int i = 0; i < numTerms1; i++) {
        scanf("%d", &coefficient);
        scanf("%d", &exponent);
        insertTerm(&poly1, coefficient, exponent);
    }
    scanf("%d", &numTerms2);
    for (int i = 0; i < numTerms2; i++) {
        scanf("%d", &coefficient);
        scanf("%d", &exponent);
        insertTerm(&poly2, coefficient, exponent);
    }
    result = multiplyPolynomials(poly1, poly2);
    printf("Result of the multiplication: ");
    displayPolynomial(result);
    scanf("%d", &exponent);
    deleteTerm(&result, exponent);
    printf("Result after deleting the term: ");
    displayPolynomial(result);
    freePolynomial(poly1);
    freePolynomial(poly2);
    freePolynomial(result);
    return 0;
}
```

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

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>
struct Node {
    int coefficient;
    int exponent;
    struct Node* next;
};
struct Node* createNode(int coefficient, int exponent) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->coefficient = coefficient;
    newNode->exponent = exponent;
    newNode->next = NULL;
    return newNode;
}
void insertNode(struct Node** head, int coefficient, int exponent) {
    struct Node* newNode = createNode(coefficient, exponent);
    if (*head == NULL || exponent > (*head)->exponent) {
        newNode->next = *head;
        *head = newNode;
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL && temp->next->exponent >= exponent) {
            temp = temp->next;
        }
        newNode->next = temp->next;
        temp->next = newNode;
    }
}
void addSamePowerCoefficients(struct Node* head) {
    struct Node* current = head;
    while (current != NULL && current->next != NULL) {
        if (current->exponent == current->next->exponent) {
            current->coefficient += current->next->coefficient;
```

```
            struct Node* temp = current->next;
            current->next = temp->next;
            free(temp);
        } else {
            current = current->next;
        }
    }
}
struct Node* multiplyPolynomials(struct Node* poly1, struct Node* poly2) {
    struct Node* result = NULL;
    struct Node* tempPoly1 = poly1;
    while (tempPoly1 != NULL) {
        struct Node* tempPoly2 = poly2;
        while (tempPoly2 != NULL) {
            int coeff = tempPoly1->coefficient * tempPoly2->coefficient;
            int exp = tempPoly1->exponent + tempPoly2->exponent;
            insertNode(&result, coeff, exp);
            tempPoly2 = tempPoly2->next;
        }
        tempPoly1 = tempPoly1->next;
    }
    addSamePowerCoefficients(result);
    return result;
}
void displayPolynomial(struct Node* head) {
    struct Node* temp = head;
    int isFirst = 1;
    while (temp != NULL) {
        if (temp->coefficient != 0) {
            if (!isFirst && temp->coefficient > 0) {
                printf(" + ");
            }
            if (temp->exponent == 0) {
                printf("%d", temp->coefficient);
            } else if (temp->exponent == 1) {
                printf("%dx", temp->coefficient);
            } else {
                printf("%dx^%d", temp->coefficient, temp->exponent);
            }
            isFirst = 0;
        }
        temp = temp->next;
```

```
    }
    printf("\n");
}
void freePolynomial(struct Node* head) {
    while (head != NULL) {
        struct Node* temp = head;
        head = head->next;
        free(temp);
    }
}
int main() {
    struct Node* poly1 = NULL;
    struct Node* poly2 = NULL;
    struct Node* product = NULL;
    int coefficient, exponent;
    int numTerms1;
    scanf("%d", &numTerms1);
    for (int i = 0; i < numTerms1; i++) {
        scanf("%d %d", &coefficient, &exponent);
        insertNode(&poly1, coefficient, exponent);
    }
    int numTerms2;
    scanf("%d", &numTerms2);
    for (int i = 0; i < numTerms2; i++) {
        scanf("%d %d", &coefficient, &exponent);
        insertNode(&poly2, coefficient, exponent);
    }
    product = multiplyPolynomials(poly1, poly2);
    displayPolynomial(poly1);
    displayPolynomial(poly2);
    displayPolynomial(product);
    freePolynomial(poly1);
    freePolynomial(poly2);
    freePolynomial(product);
    return 0;
}
```

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