# Rajalakshmi Engineering College

Name: Sherwin G M
Email: 240701496@rajalakshmi.edu.in
Roll no: 240701496
Phone: 7708605966
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 : 25

## Section 1 : Coding

1. Problem Statement

Hayley loves studying polynomials, and she wants to write a program to compare two polynomials represented as linked lists and display whether they are equal or not.

The polynomials are expressed as a series of terms, where each term consists of a coefficient and an exponent. The program should read the polynomials from the user, compare them, and then display whether they are equal or not.

*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 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 representing the coefficient and the exponent of the term in the second polynomial.

***Output Format***

The first line of output prints "Polynomial 1: " followed by the first polynomial.

The second line prints "Polynomial 2: " followed by the second polynomial.

The polynomials should be displayed in the format ax^b, where a is the coefficient and b is the exponent.

If the two polynomials are equal, the third line prints "Polynomials are Equal."

If the two polynomials are not equal, the third line prints "Polynomials are Not Equal."

Refer to the sample output for the formatting specifications.

***Sample Test Case***

Input: 2
1 2
2 1
2
1 2
2 1

Output: Polynomial 1: (1x^2) + (2x^1)
Polynomial 2: (1x^2) + (2x^1)
Polynomials are Equal.

***Answer***

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

```
// Node structure for polynomial term
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 term into the polynomial in sorted order (descending
exponent)
Node* insertTerm(Node* head, int coeff, int exp) {
   if (coeff == 0) return head; // Ignore zero coefficient terms

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

   Node* curr = head;
   Node* prev = NULL;

   while (curr && curr->exp > exp) {
      prev = curr;
      curr = curr->next;
   }

   if (curr && curr->exp == exp) { // Merge terms with the same exponent
      curr->coeff += coeff;
      if (curr->coeff == 0) { // Remove if coefficient becomes 0
         if (prev) prev->next = curr->next;
         else head = curr->next;
         free(curr);
      }
```

```
        free(newNode);
    } else {
        newNode->next = curr;
        if (prev) prev->next = newNode;
        else head = newNode;
    }
    return head;
}

// Function to read polynomial input
Node* readPolynomial(int n) {
    Node* head = NULL;
    int coeff, exp;
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &coeff, &exp);
        head = insertTerm(head, coeff, exp);
    }
    return head;
}

// Function to print a polynomial in the required format
void printPolynomial(Node* poly) {
    if (!poly) {
        printf("0\n");
        return;
    }

    while (poly) {
        printf("(%d", poly->coeff);
        printf("x^%d)", poly->exp);  // Always show x^exponent
        if (poly->next)
            printf(" + ");  // Add '+' only if another term exists

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

// Function to check if two polynomials are equal
int arePolynomialsEqual(Node* poly1, Node* poly2) {
    while (poly1 && poly2) {
        if (poly1->coeff != poly2->coeff || poly1->exp != poly2->exp)
```

```
            return 0; // Not equal

        poly1 = poly1->next;
        poly2 = poly2->next;
    }
    return (poly1 == NULL && poly2 == NULL);
}

// Driver function
int main() {
    int n, m;

    // Read first polynomial
    scanf("%d", &n);
    Node* poly1 = readPolynomial(n);

    // Read second polynomial
    scanf("%d", &m);
    Node* poly2 = readPolynomial(m);

    // Print polynomials in required format
    printf("Polynomial 1: ");
    printPolynomial(poly1);

    printf("Polynomial 2: ");
    printPolynomial(poly2);

    // Compare and print the result
    if (arePolynomialsEqual(poly1, poly2)) {
        printf("Polynomials are Equal.\n");
    } else {
        printf("Polynomials are Not Equal.\n");
    }

    return 0;
}
```

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

2. 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;
   int exp;
   struct Node* next;
} Node;


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

Node* insertTerm(Node* head, int coeff, int exp) {
   if (coeff == 0) return head;

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

   Node* curr = head;
   while (curr->next && curr->next->exp >= exp) {
        if (curr->next->exp == exp) {
            curr->next->coeff += coeff;
            if (curr->next->coeff == 0) {
                Node* temp = curr->next;
                curr->next = temp->next;
                free(temp);
            }
            free(newNode);
            return head;
        }
        curr = curr->next;
    }

    newNode->next = curr->next;
    curr->next = newNode;
    return head;
}


Node* readPolynomial(int n) {
    Node* head = NULL;
    int coeff, exp;
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &coeff, &exp);
        head = insertTerm(head, coeff, exp);
    }
    return head;
}


Node* multiplyPolynomials(Node* poly1, Node* poly2) {
    Node* result = NULL;
    for (Node* ptr1 = poly1; ptr1 != NULL; ptr1 = ptr1->next) {
        for (Node* ptr2 = poly2; ptr2 != NULL; ptr2 = ptr2->next) {
            result = insertTerm(result, ptr1->coeff * ptr2->coeff, ptr1->exp + ptr2-
>exp);
        }
    }
    return result;
}
```

```
void printPolynomial(Node* poly) {
   if (!poly) {
      printf("0\n");
      return;
   }

   while (poly) {
      printf("%d", poly->coeff);
      if (poly->exp > 1)
         printf("x^%d", poly->exp);
      else if (poly->exp == 1)
         printf("x");

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

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

int main() {
   int n, m;


   scanf("%d", &n);
   Node* poly1 = readPolynomial(n);


   scanf("%d", &m);
   Node* poly2 = readPolynomial(m);


   Node* result = multiplyPolynomials(poly1, poly2);


   printPolynomial(poly1);
   printPolynomial(poly2);
   printPolynomial(result);

   return 0;
}
```

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

3. 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;
   int expo;
   struct Node* next;
} Node;

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

void insertTerm(Node** poly, int coeff, int expo) {
   if (coeff == 0) return;

   Node* newNode = createNode(coeff, expo);
   if (*poly == NULL || (*poly)->expo > expo) {
      newNode->next = *poly;
      *poly = newNode;
   } else {
      Node* temp = *poly;
      while (temp->next != NULL && temp->next->expo < expo)
         temp = temp->next;
```

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

void printPolynomial(Node* poly) {
    if (poly == NULL) {
        printf("0\n");
        return;
    }
    while (poly) {
        printf("%dx^%d", poly->coeff, poly->expo);
        if (poly->next) printf(" + ");
        poly = poly->next;
    }
    printf("\n");
}

Node* addPolynomials(Node* poly1, Node* poly2) {
    Node* result = NULL;

    while (poly1 && poly2) {
        if (poly1->expo < poly2->expo) {
            insertTerm(&result, poly1->coeff, poly1->expo);
            poly1 = poly1->next;
        } else if (poly1->expo > poly2->expo) {
            insertTerm(&result, poly2->coeff, poly2->expo);
            poly2 = poly2->next;
        } else {
            insertTerm(&result, poly1->coeff + poly2->coeff, poly1->expo);
            poly1 = poly1->next;
            poly2 = poly2->next;
        }
    }

    while (poly1) {
        insertTerm(&result, poly1->coeff, poly1->expo);
        poly1 = poly1->next;
    }

    while (poly2) {
        insertTerm(&result, poly2->coeff, poly2->expo);
        poly2 = poly2->next;
    }

    return result;
}

void freePolynomial(Node* poly) {
    Node* temp;
    while (poly) {
        temp = poly;
        poly = poly->next;
        free(temp);
    }
}

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

    while (1) {
        scanf("%d %d", &coeff, &expo);
        if (coeff == 0 && expo == 0) break;
        insertTerm(&poly1, coeff, expo);
    }

    while (1) {
        scanf("%d %d", &coeff, &expo);
        if (coeff == 0 && expo == 0) break;
        insertTerm(&poly2, coeff, expo);
    }
    printPolynomial(poly1);
    printPolynomial(poly2);

    Node* result = addPolynomials(poly1, poly2);

    printPolynomial(result);

    freePolynomial(poly1);
    freePolynomial(poly2);
    freePolynomial(result);

    return 0;
}
```

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