# Rajalakshmi Engineering College

Name: Naren S
Email: 240701346@rajalakshmi.edu.in
Roll no: 240701346
Phone: 7695937740
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 : 21.5

## 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>

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

void multiply_polynomials(Term poly1[], int n, Term poly2[], int m, Term result[],
int* result_size) {
    *result_size = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            int new_coefficient = poly1[i].coefficient * poly2[j].coefficient;
            int new_exponent = poly1[i].exponent + poly2[j].exponent;

            int found = 0;
            for (int k = 0; k < *result_size; k++) {
                if (result[k].exponent == new_exponent) {
                    result[k].coefficient += new_coefficient;
                    found = 1;
                    break;
                }
            }

            if (!found) {
                result[*result_size].coefficient = new_coefficient;
                result[*result_size].exponent = new_exponent;
                (*result_size)++;
            }
        }
    }
}

void delete_term(Term result[], int* result_size, int exponent_to_delete) {
    for (int i = 0; i < *result_size; i++) {
        if (result[i].exponent == exponent_to_delete) {
            for (int j = i; j < *result_size - 1; j++) {
                result[j] = result[j + 1];
            }
            (*result_size)--;
            break;
        }
    }
}
```

```
void print_polynomial(Term result[], int result_size) {
    if (result_size == 0) {
        printf("0");
        return;
    }

    for (int i = 0; i < result_size; i++) {
        if (i > 0) {
            if (result[i].coefficient > 0) {
                printf(" + ");
            } else {
                printf(" - ");
                result[i].coefficient = -result[i].coefficient;
            }
        }

        if (result[i].exponent == 0) {
            printf("%d", result[i].coefficient);
        } else if (result[i].exponent == 1) {
            printf("%dx", result[i].coefficient);
        } else {
            printf("%dx^%d", result[i].coefficient, result[i].exponent);
        }
    }
}

int main() {
    int n, m, delete_exponent;

    scanf("%d", &n);
    Term poly1[n];
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &poly1[i].coefficient, &poly1[i].exponent);
    }

    scanf("%d", &m);
    Term poly2[m];
    for (int i = 0; i < m; i++) {
        scanf("%d %d", &poly2[i].coefficient, &poly2[i].exponent);
    }

    scanf("%d", &delete_exponent);
    Term result[25];
    int result_size = 0;
    multiply_polynomials(poly1, n, poly2, m, result, &result_size);

    printf("Result of the multiplication: ");
    print_polynomial(result, result_size);
    printf("\n");

    delete_term(result, &result_size, delete_exponent);
    printf("Result after deleting the term: ");
    print_polynomial(result, result_size);
    printf("\n");

    return 0;
}
```

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

2. 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>

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

```
Term* createTerm(int coefficient, int exponent) {
   Term* newTerm = (Term*)malloc(sizeof(Term));
   newTerm->coefficient = coefficient;
   newTerm->exponent = exponent;
   newTerm->next = NULL;
   return newTerm;
}

// Append at end to preserve input order
void appendTerm(Term** head, int coefficient, int exponent) {
   Term* newTerm = createTerm(coefficient, exponent);
   if (*head == NULL) {
      *head = newTerm;
   } else {
      Term* current = *head;
      while (current->next != NULL) {
         current = current->next;
      }
      current->next = newTerm;
   }
}

void printPolynomial(Term* head) {
   Term* current = head;
   int first = 1;
   while (current != NULL) {
      if (!first) {
         printf(" + ");
      }
      printf("(%dx^%d)", current->coefficient, current->exponent);
      first = 0;
      current = current->next;
   }
}

int areEqual(Term* p1, Term* p2) {
   while (p1 && p2) {
      if (p1->coefficient != p2->coefficient || p1->exponent != p2->exponent) {
         return 0;
      }
      p1 = p1->next;
      p2 = p2->next;
```

```
    }
    return p1 == NULL && p2 == NULL;
}

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

    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        int coef, exp;
        scanf("%d %d", &coef, &exp);
        appendTerm(&poly1, coef, exp);
    }

    scanf("%d", &m);
    for (int i = 0; i < m; i++) {
        int coef, exp;
        scanf("%d %d", &coef, &exp);
        appendTerm(&poly2, coef, exp);
    }

    printf("Polynomial 1: ");
    printPolynomial(poly1);
    printf("\n");

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

    if (areEqual(poly1, poly2)) {
        printf("Polynomials are Equal.\n");
    } else {
        printf("Polynomials are Not Equal.\n");
    }

    return 0;
}
```

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

3. 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>

struct Term {
   int coeff;
   int exp;
   struct Term *next;
};

struct Term* createTerm(int coeff, int exp) {
   struct Term* newTerm =(struct Term*)malloc(sizeof(struct Term));
   newTerm->coeff = coeff;
   newTerm->exp = exp;
   newTerm->next = NULL;
   return newTerm;
}

void insertTerm(struct Term** poly, int coeff, int exp) {
   struct Term* newTerm = createTerm(coeff, exp);
   if (*poly == NULL) {
      *poly = newTerm;
   } else {
      struct Term* current = *poly;
      while (current->next != NULL) {
         current = current->next;
      }
      current->next = newTerm;
   }
}

struct Term* multiplyPolynomials(struct Term* poly1, struct Term* poly2) {
   struct Term* result = NULL;
   for (struct Term* p1 = poly1; p1 != NULL; p1 = p1->next) {
      for (struct Term* p2 = poly2; p2 != NULL; p2 = p2->next) {
         int coeff = p1->coeff * p2->coeff;
         int exp = p1->exp + p2->exp;
         insertTerm(&result, coeff, exp);
      }
   }
   return result;
}
```

```
struct Term* combineLikeTerms(struct Term* poly) {
   if (poly == NULL) return NULL;

   struct Term* result = NULL;
   struct Term* current = poly;

   while (current != NULL) {
      struct Term* temp = result;
      struct Term* prev = NULL;
      while (temp != NULL && temp->exp > current->exp) {
         prev = temp;
         temp = temp->next;
      }

      if (temp != NULL && temp->exp == current->exp) {
         temp->coeff += current->coeff;
      } else {
         struct Term* newTerm = createTerm(current->coeff, current->exp);
         if (prev == NULL) {
            newTerm->next = result;
            result = newTerm;
         } else {
            newTerm->next = temp;
            prev->next = newTerm;
         }
      }
      current = current->next;
   }
   return result;
}

void printPolynomial(struct Term* poly) {
   if (poly == NULL) {
      printf("0");
      return;
   }

   struct Term* current = poly;
   int firstTerm = 1;
   while (current != NULL) {
      if (!firstTerm) {
         printf(" + ");
```

```
        }
        firstTerm = 0;

        if (current->exp == 0) {
            printf("%d", current->coeff);
        } else if (current->exp == 1) {
            printf("%dx", current->coeff);
        } else {
            printf("%dx^%d", current->coeff, current->exp);
        }
        current = current->next;
    }
    printf("\n");
}

void freePolynomial(struct Term* poly) {
    while (poly != NULL) {
        struct Term* temp = poly;
        poly = poly->next;
        free(temp);
    }
}

int main() {
    struct Term* poly1 = NULL;
    struct Term* poly2 = NULL;
    char choice;

    do {
        int coeff, exp;
        scanf("%d %d", &coeff, &exp);
        insertTerm(&poly1, coeff, exp);
        scanf(" %c", &choice);
    } while (choice == 'y' || choice == 'Y');

    do {
        int coeff, exp;
        scanf("%d %d", &coeff, &exp);
        insertTerm(&poly2, coeff, exp);
        scanf(" %c", &choice);
    } while (choice == 'y' || choice == 'Y');
    struct Term* product = multiplyPolynomials(poly1, poly2);
    struct Term* result = combineLikeTerms(product);

    printPolynomial(result);

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

    return 0;
}*/

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

struct Term {
    int coeff;
    int exp;
    struct Term *next;
};

struct Term* createTerm(int coeff, int exp) {
    struct Term* newTerm = (struct Term*)malloc(sizeof(struct Term));
    newTerm->coeff = coeff;
    newTerm->exp = exp;
    newTerm->next = NULL;
    return newTerm;
}

void insertTerm(struct Term** poly, int coeff, int exp) {
    struct Term* newTerm = createTerm(coeff, exp);
    if (*poly == NULL) {
        *poly = newTerm;
    } else {
        struct Term* current = *poly;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newTerm;
    }
}
```

```
struct Term* multiplyPolynomials(struct Term* poly1, struct Term* poly2) {
   struct Term* result = NULL;
   for (struct Term* p1 = poly1; p1 != NULL; p1 = p1->next) {
      for (struct Term* p2 = poly2; p2 != NULL; p2 = p2->next) {
         insertTerm(&result, p1->coeff * p2->coeff, p1->exp + p2->exp);
      }
   }
   return result;
}

struct Term* combineLikeTerms(struct Term* poly) {
   struct Term* result = NULL;
   while (poly != NULL) {
      struct Term* current = poly;
      poly = poly->next;

      struct Term** tracer = &result;
      while (*tracer != NULL && (*tracer)->exp > current->exp) {
         tracer = &(*tracer)->next;
      }

      if (*tracer != NULL && (*tracer)->exp == current->exp) {
         (*tracer)->coeff += current->coeff;
         free(current);
      } else {
         current->next = *tracer;
         *tracer = current;
      }
   }
   return result;
}

void printPolynomial(struct Term* poly) {
   if (poly == NULL) {
      printf("0");
      return;
   }

   int first = 1;
   while (poly != NULL) {
      if (!first) {
         printf(poly->coeff > 0 ? " + " : " - ");
      } else if (poly->coeff < 0) {
         printf("-");
      }

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

      first = 0;
      poly = poly->next;
   }
}

void freePolynomial(struct Term* poly) {
   while (poly != NULL) {
      struct Term* temp = poly;
      poly = poly->next;
      free(temp);
   }
}

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

   while (1) {
      if (scanf("%d %d", &coeff, &exp) != 2) {
         freePolynomial(poly1);
         freePolynomial(poly2);
         return 1;
      }
      insertTerm(&poly1, coeff, exp);
      if (scanf(" %c", &choice) != 1 || (choice != 'y' && choice != 'Y' && choice != 'n'
&& choice != 'N')) {
```

```
            freePolynomial(poly1);
            freePolynomial(poly2);
            return 1;
        }
        if (choice == 'n' || choice == 'N') break;
    }

    while (1) {
        if (scanf("%d %d", &coeff, &exp) != 2) {
            freePolynomial(poly1);
            freePolynomial(poly2);
            return 1;
        }
        insertTerm(&poly2, coeff, exp);
        if (scanf(" %c", &choice) != 1 || (choice != 'y' && choice != 'Y' && choice != 'n'
&& choice != 'N')) {
            freePolynomial(poly1);
            freePolynomial(poly2);
            return 1;
        }
        if (choice == 'n' || choice == 'N') break;
    }

    struct Term* product = multiplyPolynomials(poly1, poly2);
    struct Term* result = combineLikeTerms(product);

    printPolynomial(result);

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

    return 0;
}
```

***Status :*** Partially correct ***Marks : 1.5/10***