# **Assignment 4** Dated Dec 16th, 2024

## **Problem Statement**

A program in C to perform addition and multiplication for polynomials, where the no. of terms is given by the user.

## **Algorithm**

### Input

read\_pol() function fills up the array passed into it up to terms t.

### Output

display\_pol() function displays the polynomial in order.

We start by defining a structure ‘Poly’, that contains ‘coef’, and ‘expo’. It’s a term.

**Step 1:** Start.  
**Step 2:** Input the polynomial array p[] and the total number of terms t.  
**Step 3:** For i from 0 to t - 1, perform the following:

**Step 3.1:** Print the coefficient p[i].coef and exponent p[i].expo in the format coef x^expo.

**Step 3.2:** If i < t - 1, print " + " to separate terms.  
**Step 4:** Print a newline after all terms are displayed.  
**Step 5:** Stop.  
**Step 6:** [End of function display\_pol defined at Step 1.]

**Algorithm for read\_poly()**

**Step 7:** Start.  
**Step 8:** Input the polynomial array p[].  
**Step 9:** Declare an integer variable terms.  
**Step 10:** Prompt the user to input the total number of terms in the polynomial and store it in terms.  
**Step 11:** If terms >= TERMS, display an error message and terminate the program.  
**Step 12:** Display a message instructing the user to input coefficients and exponents in descending order.  
**Step 13:** For i from 0 to terms - 1, perform the following:

**Step 13.1:** Prompt the user to input the coefficient and exponent of the i + 1th term.

**Step 13.2:** Store the values in p[i].coef and p[i].expo.

**Step 13.3:** If input is invalid, display an error message and terminate the program.  
**Step 14:** Return terms.  
**Step 15:** Stop.  
**Step 16:** [End of function read\_poly defined at Step 7.]

**Algorithm for add\_poly()**

**Step 17:** Start.  
**Step 18:** Input arrays p1[] and p2[], integers t1 and t2, and the result array p3[].  
**Step 19:** Declare three integers: i, j, and k, all initialized to 0.  
**Step 20:** While i < t1 and j < t2, perform the following:

**Step 20.1:** If p1[i].expo == p2[j].expo, perform the following:

**Step 20.1.1:** Set p3[k].coef = p1[i].coef + p2[j].coef.

**Step 20.1.2:** Set p3[k].expo = p1[i].expo.

**Step 20.1.3:** Increment i, j, and k.

**Step 20.2:** Else if p1[i].expo > p2[j].expo, perform the following:

**Step 20.2.1:** Set p3[k].coef = p1[i].coef.

**Step 20.2.2:** Set p3[k].expo = p1[i].expo.

**Step 20.2.3:** Increment i and k.

**Step 20.3:** Else, perform the following:

**Step 20.3.1:** Set p3[k].coef = p2[j].coef.

**Step 20.3.2:** Set p3[k].expo = p2[j].expo.

**Step 20.3.3:** Increment j and k.  
**Step 21:** While i < t1, copy remaining terms of p1[] into p3[] and increment i and k.  
**Step 22:** While j < t2, copy remaining terms of p2[] into p3[] and increment j and k.  
**Step 23:** Return k (the total number of terms in p3[]).  
**Step 24:** Stop.  
**Step 25:** [End of function add\_poly defined at Step 17.]

**Algorithm for mul\_poly()**

**Step 26:** Start.  
**Step 27:** Input arrays p1[] and p2[], integers t1 and t2, and the result array p4[].  
**Step 28:** Declare an integer k and initialize it to 0.  
**Step 29:** For i from 0 to t1 - 1, perform the following:

**Step 29.1:** For j from 0 to t2 - 1, perform the following:

**Step 29.1.1:** Compute the product of coefficients: p4[k].coef = p1[i].coef \* p2[j].coef.

**Step 29.1.2:** Compute the sum of exponents: p4[k].expo = p1[i].expo + p2[j].expo.

**Step 29.1.3:** Increment k.  
**Step 30:** For i from 0 to k - 1, perform the following:

**Step 30.1:** For j from i + 1 to k - 1, perform the following:

**Step 30.1.1:** If p4[i].expo == p4[j].expo, perform the following:

**Step 30.1.1.1:** Add coefficients: p4[i].coef += p4[j].coef.

**Step 30.1.1.2:** Shift terms of p4[] left from index j to k - 1.

**Step 30.1.1.3:** Decrement k and j.  
**Step 31:** Return k (the total number of terms in p4[]).  
**Step 32:** Stop.  
**Step 33:** [End of function mul\_poly defined at Step 26.]

**Algorithm for main()**

**Step 34:** Start.  
**Step 35:** Declare integers t1, t2, and t3, all initialized to 0.  
**Step 36:** Call read\_poly() with p1[] and store the result in t1.  
**Step 37:** Display the first polynomial using display\_pol(p1, t1).  
**Step 38:** Call read\_poly() with p2[] and store the result in t2.  
**Step 39:** Display the second polynomial using display\_pol(p2, t2).  
**Step 40:** Call add\_poly(p1, p2, t1, t2, p3) and store the result in t3.  
**Step 41:** Display the addition result using display\_pol(p3, t3).  
**Step 42:** Call mul\_poly(p1, p2, t1, t2, p4) and store the result in t3.  
**Step 43:** Display the multiplication result using display\_pol(p4, t3).  
**Step 44:** Stop.  
[End of function main defined at Step 34.]

## **Source Code**

#include <stdio.h>

#include <stdlib.h>

#define TERMS 512

typedef struct {

    int coef;

    int expo;

} Poly;

Poly p1[TERMS];

Poly p2[TERMS];

Poly p3[TERMS];

Poly p4[TERMS];

void display\_pol(Poly p[], int t)

{

    for (int i = 0; i < t; i++) {

        printf("%dx^%d", p[i].coef, p[i].expo);

        if (i < t - 1) {

            printf(" + ");

        }

    }

    puts("\n");

}

int read\_poly(Poly p[])

{

    int terms = 0;

    printf("Input total no. of terms in the polynomial: ");

    scanf("%d", &terms);

    if (terms >= TERMS) {

        printf("error: Unsupported no. of terms. %d is limit.\n", TERMS);

        exit(1);

    }

    printf("Input coefficient and exponent in descending order:\n");

    for (int i = 0; i < terms; i++) {

        printf("Coef <space> exponent of %dth term: ", i + 1);

        if (scanf("%d%d", &p[i].coef, &p[i].expo) != 2) {

            printf("error: Invalid input.\n");

            exit(1);

        }

    }

    return terms;

}

int add\_poly(Poly p1[], Poly p2[], int t1, int t2, Poly p3[])

{

    int i = 0;

    int j = 0;

    int k = 0;

    while (i < t1 && j < t2) {

        if (p1[i].expo == p2[j].expo) {

            p3[k].coef = p1[i].coef + p2[j].coef;

            p3[k].expo = p1[i].expo;

            i++;

            j++;

            k++;

        } else if (p1[i].expo > p2[j].expo) {

            p3[k].coef = p1[i].coef;

            p3[k].expo = p1[i].expo;

            i++;

            k++;

        } else {

            p3[k].coef = p2[j].coef;

            p3[k].expo = p2[j].expo;

            j++;

            k++;

        }

    }

    // Leftover terms are now added to the array

    while (i < t1) {

        p3[k].coef = p1[i].coef;

        p3[k].expo = p1[i].expo;

        i++;

        k++;

    }

    while (j < t2) {

        p3[k].coef = p2[j].coef;

        p3[k].expo = p2[j].expo;

        j++;

        k++;

    }

    return k;

}

int mul\_poly(Poly p1[], Poly p2[], int t1, int t2, Poly p4[])

{

    int k = 0;

    // Initialize result polynomial

    for (int i = 0; i < t1; i++) {

        for (int j = 0; j < t2; j++) {

            p4[k].coef = p1[i].coef \* p2[j].coef;

            p4[k].expo = p1[i].expo + p2[j].expo;

            k++;

        }

    }

    // Combine terms with the same exponent

    for (int i = 0; i < k; i++) {

        for (int j = i + 1; j < k; j++) {

            if (p4[i].expo == p4[j].expo) {

                p4[i].coef += p4[j].coef;

                for (int m = j; m < k - 1; m++) {

                    p4[m] = p4[m + 1];

                }

                k--;

                j--;

            }

        }

    }

    return k;

}

int main(void)

{

    int t1 = 0;

    int t2 = 0;

    int t3 = 0;

    t1 = read\_poly(p1);

    printf("1st polynomial: ");

    display\_pol(p1, t1);

    t2 = read\_poly(p2);

    printf("2nd polynomial: ");

    display\_pol(p2, t2);

    t3 = add\_poly(p1, p2, t1, t2, p3);

    printf("Resultant polynomial addition: ");

    display\_pol(p3, t3);

    t3 = mul\_poly(p1, p2, t1, t2, p4);

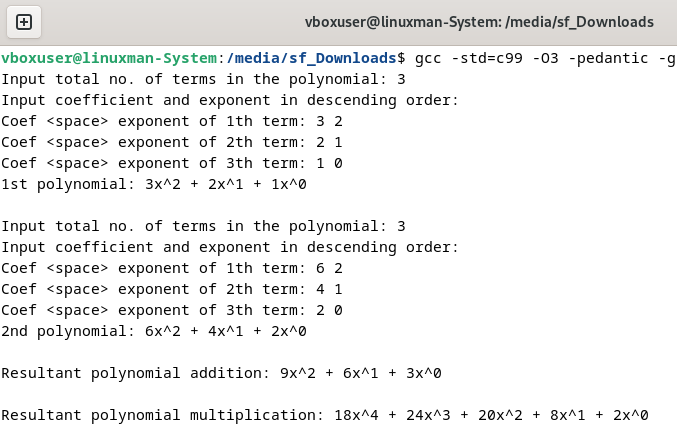
    printf("Resultant polynomial multiplication: ");

    display\_pol(p4, t3);

    return 0;

}

## **Output**



A screenshot of a computer

Description automatically generated

### Discussion

Global variables should be used to the least. However, it has been applied here to reduce the complexity of using pointers and tricky lines.

**Teacher’s signature**