Rajalakshmi Engineering College

Name: NITHYASHREE K
Email: 240701369@rajalakshmi.edu.in
Roll no: 240701369
Phone: 9043544115
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

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

```
// You are using GCC
#include <stdio.h>
#include <math.h>

int main() {
    int degree, coeff2, coeff1, coeff0, x;
    int result;
    scanf("%d", &degree);
    scanf("%d", &coeff2);
    scanf("%d", &coeff1);
    scanf("%d", &coeff0);
    scanf("%d", &x);
    result = coeff2 * x * x + coeff1 * x + coeff0;
    printf("%d\n", result);
    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>
```

```
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 addTerm(struct Term** poly, int coeff, int exp) {
    if (coeff == 0) return;

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

    while (current != NULL && current->exp > exp) {
        prev = current;
        current = current->next;
    }

    if (current != NULL && current->exp == exp) {
        current->coeff += coeff;
        if (current->coeff == 0) {
            if (prev == NULL) *poly = current->next;
            else prev->next = current->next;
            free(current);
        }
    } else {
        struct Term* newTerm = createTerm(coeff, exp);
        if (prev == NULL) {
            newTerm->next = *poly;
            *poly = newTerm;
        } else {
            newTerm->next = prev->next;
            prev->next = newTerm;
        }
    }
```

```
}

void readPolynomial(struct Term** poly, int n) {
    for (int i = 0; i < n; i++) {
        int coeff, exp;
        scanf("%d %d", &coeff, &exp);
        addTerm(poly, coeff, exp);
    }
}

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;
            addTerm(&result, coeff, exp);
        }
    }
    return result;
}

void printPolynomial(struct Term* poly) {
    struct Term* current = poly;
    int first = 1;
    while (current != NULL) {
        if (current->coeff != 0) {
            if (!first) printf(" + ");
            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);
            }
            first = 0;
        }
        current = current->next;
    }
    printf("\n");
}
```

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

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

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

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

    result = multiplyPolynomials(poly1, poly2);

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

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

    return 0;
}
```

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

3. 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>

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 appendTerm(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;
    }
}

void printPolynomial(struct Term* poly) {
    struct Term* current = poly;
    int first = 1;
    while (current != NULL) {
```

```
        if (!first) printf(" + ");
        printf("(%dx^%d)", current->coeff, current->exp);
        current = current->next;
        first = 0;
    }
    printf("\n");
}

int arePolynomialsEqual(struct Term* poly1, struct Term* poly2) {
    while (poly1 != NULL && poly2 != NULL) {
        if (poly1->coeff != poly2->coeff || poly1->exp != poly2->exp)
            return 0;
        poly1 = poly1->next;
        poly2 = poly2->next;
    }
    return (poly1 == NULL && poly2 == NULL);
}

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

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

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

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

```
    printf("Polynomial 1: ");
    printPolynomial(poly1);

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

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

    freePolynomial(poly1);
    freePolynomial(poly2);

    return 0;
}
```

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