

Rajalakshmi Engineering College

Name: Vikashini M
Email: 241801314@rajalakshmi.edu.in
Roll no: 241801314
Phone: 9345747519
Branch: REC
Department: I AI & DS AF
Batch: 2028
Degree: B.E - AI & DS

Scan to verify results



NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_week 1_CY

Attempt : 1
Total Mark : 30
Marks Obtained : 27.5

Section 1 : Coding

1. 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

```
#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;  
}
```

```
void insertSorted(Node**head,int coeff,int exp){  
    Node* newNode=createNode(coeff,exp);  
    if(*head==NULL||(*head)->exp > exp){  
        newNode->next=*head;  
        *head=newNode;  
    }else{  
        Node* current=*head;  
        while(current->next != NULL && current->next->exp<=exp) {  
            current=current->next;  
        }  
        newNode->next=current->next;  
        current->next=newNode;  
    }  
}
```

```
Node*addPolynomials(Node* poly1,Node* poly2){  
    Node* result=NULL;  
    Node* temp1=poly1;  
    Node* temp2=poly2;  
  
    while(temp1 != NULL || temp2 != NULL){  
        if(temp1==NULL){  
            insertSorted(&result,temp2->coeff,temp2->exp);  
            temp2=temp2->next;  
        }else if(temp2==NULL){  
            insertSorted(&result,temp1->coeff,temp1->exp);  
            temp1=temp1->next;  
        }else if(temp1->exp<temp2->exp){  
            insertSorted(&result,temp1->coeff,temp1->exp);  
            temp1=temp1->next;  
        }else if(temp2->exp<temp1->exp){  
            insertSorted(&result,temp2->coeff,temp2->exp);  
            temp2=temp2->next;  
        }
```

```

    }else{
        int sum=temp1->coeff+temp2->coeff;
        if(sum != 0){
            insertSorted(&result,sum,temp1->exp);
        }
        temp1=temp1->next;
        temp2=temp2->next;
    }
}
return result;
}

void printPolynomial(Node* head){
    if(head==NULL){
        printf("0");
        return;
    }
    Node* temp=head;
    while(temp != NULL){
        printf("%dx^%d",temp->coeff,temp->exp);
        if(temp->next != NULL){
            printf("+");
        }
        temp=temp->next;
    }
}

void freeList(Node* head) {
    Node* temp;
    while(head != NULL){
        temp=head;
        head=head->next;
        free(temp);
    }
}

int main(){
    Node* poly1 = NULL;
    Node* poly2 = NULL;
    int coeff,exp;
    while(scanf("%d %d",&coeff,&exp)==2 && (coeff != 0 || exp != 0)){
        insertSorted(&poly1,coeff,exp);
    }
    while(scanf("%d %d",&coeff,&exp)==2 && (coeff != 0 || exp != 0)){
        insertSorted(&poly2,coeff,exp);
    }
}

```

```
}  
    printPolynomial(poly1);  
    printf("\n");  
    printPolynomial(poly2);  
    printf("\n");  
    Node* result = addPolynomials(poly1,poly2);  
    printPolynomial(result);  
    printf("\n");  
    freeList(poly1);  
    freeList(poly2);  
    freeList(result);  
    return 0;  
}
```

Status : Correct

Marks : 10/10

2. 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 x^2 = 13

coefficient of x^1 = 12

coefficient of x^0 = 11

$x = 1$

Output:

36

Explanation:

Calculate the value of $13x^2$: $13 * 12 = 13$.

Calculate the value of $12x^1$: $12 * 11 = 12$.

Calculate the value of $11x^0$: $11 * 10 = 11$.

Add the values of x^2 , x^1 , and x^0 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 x^2 .

The third line consists of an integer representing the coefficient of x^1 .

The fourth line consists of an integer representing the coefficient of x^0 .

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

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

```
#include<math.h>
```

```

int main(){
    int degere;
    scanf("%d",&degere);
    int coeff[degere + 1];
    for(int i=degere; i>=0;i--){
        scanf("%d",&coeff[i]);
    }
    int x;
    scanf("%d",&x);
    int result = 0;
    for(int i= degere;i>=0;i--){
        result += coeff[i] * pow(x, i);
    }
    printf("%d",result);
    return 0;
}

```

Status : Correct

Marks : 10/10

3. 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 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;
}

```

```

void insertNode(Node**head,int coeff, int exp)
{
    Node*newNode = createNode(coeff,exp);
    if(*head == NULL){
        *head = newNode;
    }else{
        Node*temp = *head;
        while(temp->next != NULL){
            temp = temp->next;
        }
        temp->next=newNode;
    }
}

```

```

void printPolynomial(Node*head, char*prefix)
{
    printf("%s",prefix);
    Node*temp=head;
    int firstTerm = 1;
    while(temp != NULL){
        if(temp->coeff != 0){
            if(!firstTerm){
                printf("+");
            }else{
                firstTerm = 0;
            }
            if(temp->exp == 0){
                printf("%d",temp->coeff);
            }else if(temp->exp==1){
                printf("%dx",temp->coeff);
            }else{
                printf("%dx^%d",temp->coeff,temp->exp);
            }
        }
        temp = temp->next;
    }
}

```

```

    printf("\n");
}

Node*multiplyPolynomials(Node*poly1, Node*poly2){
    Node*result = NULL;
    Node*temp1 = poly1;
    Node*temp2 = poly2;

    while(temp1 != NULL){
        temp2 = poly2;
        while(temp2 != NULL){
            int coeff = temp1->coeff*temp2->coeff;
            int exp=temp1->exp+temp2->exp;
            Node*current=result;
            Node*prev = NULL;
            int found=0;

            while(current != NULL){
                if(current->exp==exp){
                    current->coeff+=coeff;
                    found = 1;
                    break;
                }
                prev=current;
                current=current->next;
            }

            if(!found){
                Node*newNode = createNode(coeff,exp);
                if(result == NULL){
                    result = newNode;
                }else if(result->exp<exp){
                    newNode->next=result;
                    result = newNode;
                }else{
                    current=result;
                    prev=NULL;
                    while(current != NULL&&current->exp>exp){
                        prev=current;
                        current=current->next;
                    }
                    if(prev== NULL){

```

```

        newNode->next=result;
        result = newNode;
    }else{
        newNode->next=current;
        prev->next=newNode;
    }
}
temp2 = temp2->next;
temp1 = temp1->next;
}
return result;
}

```

```

Node*deleteTerm(Node*head,int expToDelete){
    Node*current=head;
    Node*prev=NULL;

    while(current != NULL){
        if(current->exp==expToDelete){
            if(prev==NULL){
                head=current->next;
            }else{
                prev->next=current->next;
            }
            Node*temp=current;
            current=current->next;
            free(temp);
        }else{
            prev=current;
            current=current->next;
        }
    }
    return head;
}

```

```

void freeList(Node*head){
    Node*temp;
    while(head != NULL){
        temp=head;
        head=head->next;
    }
}

```

```

    free(temp);
}
}

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

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

    scanf("%d",&m);
    for(int i=0;i<m;i++){
        scanf("%d%d",&coeff,&exp);
        insertNode(&poly2,coeff,exp);
    }
    Node*result=multiplyPolynomials(poly1,poly2);
    printPolynomial(result,"Result of the multiplication: ");

    scanf("%d",&expToDelete);
    result=deleteTerm(result,expToDelete);
    printPolynomial(result, "Result after deleting the term: ");

    freeList(poly1);
    freeList(poly2);
    freeList(result);

    return 0;

}

```

Status : Partially correct

Marks : 7.5/10