Rajalakshmi Engineering College

Name: Siddesh Kumar L
Email: 240701512@rajalakshmi.edu.in
Roll no: 240701512
Phone: null
Branch: REC
Department: I CSE FE
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 : 27

Section 1 : Coding

1. Problem Statement

John is working on a math processing application, and his task is to simplify polynomials entered by users. The polynomial is represented as a linked list, where each node contains two properties:

Coefficient of the term.

Exponent of the term.

John's goal is to combine all the terms that have the same exponent, effectively simplifying the polynomial.

Input Format

The first line of input consists of an integer representing the number of terms in the polynomial.

The next n lines of input consist of two integers, representing the coefficient and exponent of the polynomial in each line separated by space.

Output Format

The first line of output prints the original polynomial in the format 'cx^e + cx^e + ...' (where c is the coefficient and e is the exponent of each term).

The second line of output displays the simplified polynomial in the same format as the original polynomial.

If the polynomial is 0, then only '0' will be printed.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 3
5 2
3 1
6 2

Output: Original polynomial: 5x^2 + 3x^1 + 6x^2
Simplified polynomial: 11x^2 + 3x^1

Answer

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

typedef struct Node {
   int coefficient;
   int exponent;
   struct Node* next;
} Node;

Node* createNode(int coefficient, int exponent) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   newNode->coefficient = coefficient;
   newNode->exponent = exponent;
   newNode->next = NULL;
   return newNode;
}
```

```
void insertNode(Node** head, int coefficient, int exponent) {
    Node* newNode = createNode(coefficient, exponent);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

void printPolynomial(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        printf("%dx^%d", temp->coefficient, temp->exponent);
        if (temp->next != NULL) {
            printf(" + ");
        }
        temp = temp->next;
    }
    printf("\n");
}

void simplifyPolynomialInOrder(Node** head) {
    Node* current = *head;
    Node* outer = current;

    while (outer != NULL) {
        Node* inner = outer->next;
        Node* prev = outer;

        while (inner != NULL) {
            if (inner->exponent == outer->exponent) {
                outer->coefficient += inner->coefficient;
                prev->next = inner->next;
                free(inner);
                inner = prev->next;
            } else {
                prev = inner;
            inner = inner->next;
        }
      }
      outer = outer->next;
    }
}

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

int main() {
    int n, coefficient, exponent;
    Node* head = NULL;

    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
      scanf("%d %d", &coefficient, &exponent);
      insertNode(&head, coefficient, exponent);
    }

    printf("Original polynomial: ");
    printPolynomial(head);

    simplifyPolynomialInOrder(&head);

    printf("Simplified polynomial: ");
    if (head == NULL) {
      printf("0\n");
    } else {
      printPolynomial(head);
    }

    freeList(head);
    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 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

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

typedef struct Node {
   int coefficient;
   int exponent;
   struct Node* next;
} Node;

Node* createNode(int coefficient, int exponent) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   newNode->coefficient = coefficient;
   newNode->exponent = exponent;
   newNode->next = NULL;
   return newNode;
}
```

```
void insertNode(Node** head, int coefficient, int exponent) {
    Node* newNode = createNode(coefficient, exponent);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

int evaluatePolynomial(Node* head, int x) {
    int result = 0;
    Node* temp = head;

    while (temp != NULL) {
        result += temp->coefficient * pow(x, temp->exponent);
        temp = temp->next;
    }

    return result;
}

int main() {
    int degree, coefficient, x;
    Node* head = NULL;
    scanf("%d", &degree);
    for (int i = degree; i >= 0; i--) {
        scanf("%d", &coefficient);
        insertNode(&head, coefficient, i);
    }
    scanf("%d", &x);
    int result = evaluatePolynomial(head, x);
    printf("%d\n", result);
    return 0;
}
```

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

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

Term *create_polynomial() {
   int n, coeff, exp;
   scanf("%d", &n);
   Term *head = NULL, *tail = NULL, *new_term;
   for (int i = 0; i < n; i++) {
      scanf("%d %d", &coeff, &exp);
      new_term = (Term *)malloc(sizeof(Term));
      new_term->coeff = coeff;
      new_term->exp = exp;
      new_term->next = NULL;
      if (!head) {
         head = new_term;
         tail = new_term;
      } else {
         tail->next = new_term;
         tail = new_term;
      }
   }
   return head;
```

```
}

Term *multiply_polynomials(Term *poly1, Term *poly2) {
   Term *result_head = NULL, *result_tail = NULL, *new_term;
   Term *curr1 = poly1, *curr2;
   while (curr1) {
      curr2 = poly2;
      while (curr2) {
         new_term = (Term *)malloc(sizeof(Term));
         new_term->coeff = curr1->coeff * curr2->coeff;
         new_term->exp = curr1->exp + curr2->exp;
         new_term->next = NULL;
         if (!result_head) {
            result_head = new_term;
            result_tail = new_term;
         } else {
            result_tail->next = new_term;
            result_tail = new_term;
         }
         curr2 = curr2->next;
      }
      curr1 = curr1->next;
   }
   return result_head;
}

Term *simplify_polynomial(Term *poly) {
   if (!poly) {
      return NULL;
   }
   int terms[2001] = {0};
   Term *curr = poly, *simplified_head = NULL, *simplified_tail = NULL,
*new_term;
   while (curr) {
      terms[curr->exp] += curr->coeff;
      curr = curr->next;
   }
   for (int exp = 2000; exp >= 0; exp--) {
      if (terms[exp] != 0) {
         new_term = (Term *)malloc(sizeof(Term));
         new_term->coeff = terms[exp];
         new_term->exp = exp;
```

```
            new_term->next = NULL;
            if (!simplified_head) {
                simplified_head = new_term;
                simplified_tail = new_term;
            } else {
                simplified_tail->next = new_term;
                simplified_tail = new_term;
            }
        }
    }
    return simplified_head;
}

void print_polynomial(Term *poly) {
    if (!poly) {
        printf("\n");
        return;
    }
    Term *curr = poly;
    int first_term = 1;
    while (curr) {
        if (curr->coeff != 0) {
            if (!first_term && curr->coeff > 0) {
                printf(" + ");
            }
            printf("%d", curr->coeff);
            if (curr->exp != 0) {
                printf("x");
                if (curr->exp != 1) {
                    printf("^%d", curr->exp);
                }
            }
            first_term = 0;
        }
        curr = curr->next;
    }
    printf("\n");
}

int main() {
    Term *poly1 = create_polynomial();
    Term *poly2 = create_polynomial();
    print_polynomial(poly1);
    print_polynomial(poly2);
    Term *result_poly = multiply_polynomials(poly1, poly2);
    Term *simplified_poly = simplify_polynomial(result_poly);
    print_polynomial(simplified_poly);

    Term *curr, *temp;
    curr = poly1;
    while(curr){
        temp = curr;
        curr = curr->next;
        free(temp);
    }
    curr = poly2;
    while(curr){
        temp = curr;
        curr = curr->next;
        free(temp);
    }
    curr = result_poly;
    while(curr){
        temp = curr;
        curr = curr->next;
        free(temp);
    }
    curr = simplified_poly;
    while(curr){
        temp = curr;
        curr = curr->next;
        free(temp);
    }

    return 0;
}
```

Status : Partially correct ***Marks : 7/10***