Rajalakshmi Engineering College

Name: Rakesh H
Email: 240701415@rajalakshmi.edu.in
Roll no: 240701415
Phone: 7305737702
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 : 30

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** poly, int coeff, int exp) {
    if (coeff == 0) return;

    Node* newNode = createNode(coeff, exp);
    if (*poly == NULL || (*poly)->exp > exp) {
        newNode->next = *poly;
        *poly = newNode;
        return;
    }

    Node* current = *poly;
    while (current->next && current->next->exp < exp) {
        current = current->next;
    }

    if (current->next && current->next->exp == exp) {
        current->next->coeff += coeff;
        if (current->next->coeff == 0) {
            Node* temp = current->next;
            current->next = temp->next;
            free(temp);
        }
        free(newNode);
    } else {
        newNode->next = current->next;
        current->next = newNode;
    }
}

void readPolynomial(Node** poly) {
    int coeff, exp;
    while (1) {
        scanf("%d %d", &coeff, &exp);
        if (coeff == 0 && exp == 0) break;
        insertSorted(poly, coeff, exp);
    }
}

Node* addPolynomials(Node* poly1, Node* poly2) {
    Node* result = NULL;

    while (poly1 || poly2) {
        int coeff = 0, exp = 0;

        if (poly1 && (!poly2 || poly1->exp < poly2->exp)) {
            coeff = poly1->coeff;
            exp = poly1->exp;
            poly1 = poly1->next;
        } else if (poly2 && (!poly1 || poly2->exp < poly1->exp)) {
            coeff = poly2->coeff;
            exp = poly2->exp;
            poly2 = poly2->next;
        } else {
            coeff = poly1->coeff + poly2->coeff;
            exp = poly1->exp;
            poly1 = poly1->next;
            poly2 = poly2->next;
        }

        insertSorted(&result, coeff, exp);
    }

    return result;
}

void printPolynomial(Node* poly) {
    if (!poly) {
        printf("0\n");
        return;
    }

    while (poly) {
        printf("%dx^%d", poly->coeff, poly->exp);
        if (poly->next) printf(" + ");
        poly = poly->next;
    }
    printf("\n");
}


int main() {
    Node *poly1 = NULL, *poly2 = NULL, *result = NULL;

    readPolynomial(&poly1);

    readPolynomial(&poly2);

    result = addPolynomials(poly1, poly2);

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

    return 0;
}
```

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

2. Problem Statement

Hasini is studying polynomials in her class. Her teacher has introduced a new concept of two polynomials using linked lists.

The teacher provides Hasini with a program that takes two polynomials as input, represented as linked lists, and then displays them together. The polynomials are simplified and should be displayed in the format ax^b, where a is the coefficient and b is the exponent.

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 polynomials should be displayed in the format ax^b, where a is the coefficient and b is the exponent.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 3
1 2
2 1
3 0
3
2 2
1 1
4 0
Output: 1x^2 + 2x + 3
2x^2 + 1x + 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 insertTerm(Node** head, int coeff, int exp) {
    if (coeff == 0) return;

    if (*head == NULL || (*head)->exp < exp) {
        Node* newNode = createNode(coeff, exp);
        newNode->next = *head;
        *head = newNode;
        return;
    }

    Node* current = *head;
    Node* 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) {
                *head = current->next;
            } else {
                prev->next = current->next;
            }
            free(current);
        }
    } else {
        Node* newNode = createNode(coeff, exp);
        newNode->next = current;
        if (prev == NULL) {
            *head = newNode;
        } else {
            prev->next = newNode;
        }
    }
}

void printPolynomial(Node* head) {
    if (head == NULL) {
        printf("0");
        return;
    }

    Node* current = head;
    int firstTerm = 1;

    while (current != NULL) {
        int c = current->coeff;
        int e = current->exp;

        if (c < 0) {
            if (firstTerm)
                printf("-");
            else
                printf(" - ");
            c = -c;
        } else {
            if (!firstTerm)
                printf(" + ");
        }

        if (e == 0) {
            printf("%d", c);
        } else if (e == 1) {
            printf("%dx", c);
        } else {
            printf("%dx^%d", c, e);
        }
        firstTerm = 0;
        current = current->next;
    }
    printf("\n");
}
```

```
void freeList(Node* head) {
    while (head) {
        Node* tmp = head;
        head = head->next;
        free(tmp);
    }
}

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

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

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

    printPolynomial(poly1);
    printf(" ");
    printPolynomial(poly2);
    printf("\n");

    freeList(poly1);
    freeList(poly2);

    return 0;
}
```

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

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>

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;
}

// Insert node in descending order by exponent; add coefficients if exponents
match
void insertTerm(Node** head, int coeff, int exp) {
   if (coeff == 0) return;  // skip zero coefficient terms

   Node* newNode = createNode(coeff, exp);

   if (*head == NULL || (*head)->exp < exp) {
      newNode->next = *head;
      *head = newNode;
      return;
   }

   Node* current = *head;
```

```
    Node* prev = NULL;

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

    if (current != NULL && current->exp == exp) {
        current->coeff += coeff;
        free(newNode);
        if (current->coeff == 0) {
            // Remove the node
            if (prev == NULL) {
                *head = current->next;
            } else {
                prev->next = current->next;
            }
            free(current);
        }
    } else {
        newNode->next = current;
        if (prev == NULL) {
            *head = newNode;
        } else {
            prev->next = newNode;
        }
    }
}

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

```
int comparePolynomials(Node* head1, Node* head2) {
   Node* c1 = head1;
   Node* c2 = head2;
   while (c1 != NULL && c2 != NULL) {
      if (c1->exp != c2->exp || c1->coeff != c2->coeff)
         return 0; // not equal
      c1 = c1->next;
      c2 = c2->next;
   }
   if (c1 == NULL && c2 == NULL)
      return 1; // equal
   return 0;
}

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 n, m;
   int coeff, exp;

   // Read first polynomial
   scanf("%d", &n);
   for (int i = 0; i < n; i++) {
      scanf("%d %d", &coeff, &exp);
      insertTerm(&poly1, coeff, exp);
   }

   // Read second polynomial
   scanf("%d", &m);
   for (int i = 0; i < m; i++) {
      scanf("%d %d", &coeff, &exp);
      insertTerm(&poly2, coeff, exp);
```

```
    }

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

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

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

    freeList(poly1);
    freeList(poly2);

    return 0;
}
```

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