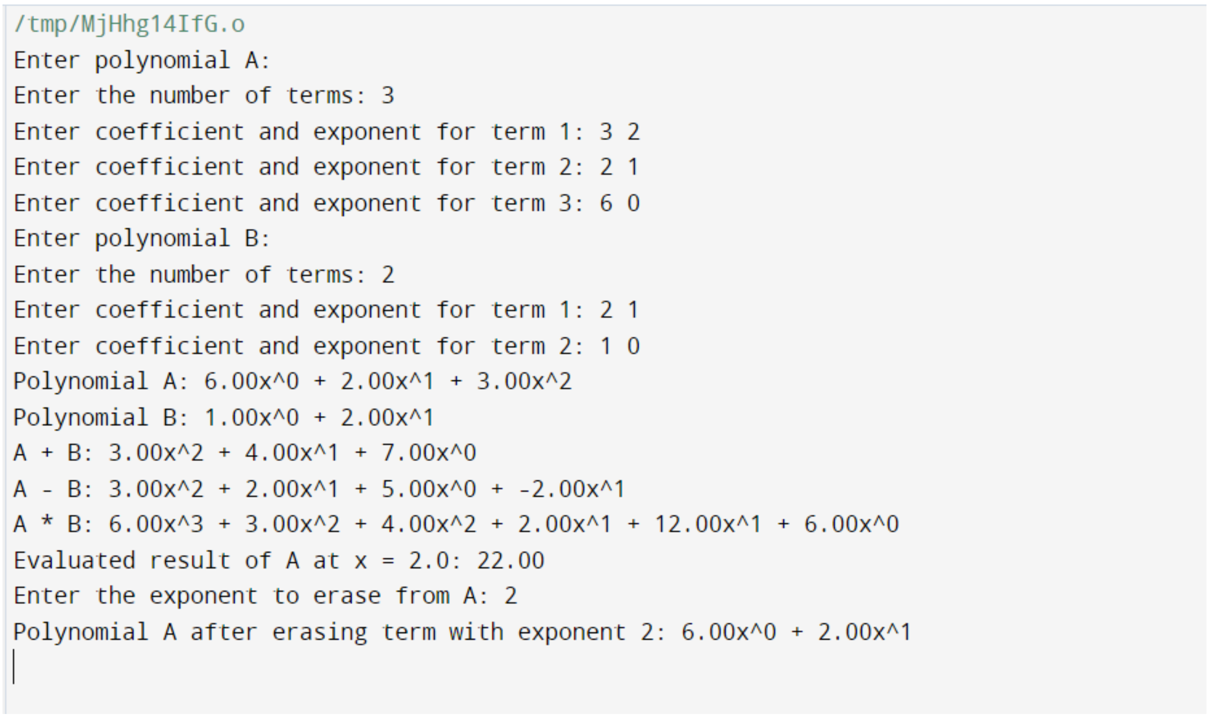
NAME-SAUMYA KUMAR

BT22CSH001



```
/tmp/MjHhg14IfG.o
Enter polynomial A:
Enter the number of terms: 3
Enter coefficient and exponent for term 1: 3 2
Enter coefficient and exponent for term 2: 2 1
Enter coefficient and exponent for term 3: 6 0
Enter polynomial B:
Enter the number of terms: 2
Enter coefficient and exponent for term 1: 2 1
Enter coefficient and exponent for term 2: 1 0
Polynomial A: 6.00x^0 + 2.00x^1 + 3.00x^2
Polynomial B: 1.00x^0 + 2.00x^1
A + B: 3.00x^2 + 4.00x^1 + 7.00x^0
A - B: 3.00x^2 + 2.00x^1 + 5.00x^0 + -2.00x^1
A * B: 6.00x^3 + 3.00x^2 + 4.00x^2 + 2.00x^1 + 12.00x^1 + 6.00x^0
Evaluated result of A at x = 2.0: 22.00
Enter the exponent to erase from A: 2
Polynomial A after erasing term with exponent 2: 6.00x^0 + 2.00x^1
```

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>


struct Node {

   float coefficient;

   int exponent;

   struct Node* link;

};


struct Polynomial {

   struct Node* header;

};
```

```
void insertTerm(struct Polynomial* poly, float coefficient, int exponent) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->coefficient = coefficient;
    newNode->exponent = exponent;

    newNode->link = poly->header->link;
    poly->header->link = newNode;
}

void initPolynomial(struct Polynomial* poly) {
    poly->header = (struct Node*)malloc(sizeof(struct Node));
    poly->header->link = poly->header;
}

void Pread(struct Polynomial* poly) {
    int n;
    printf("Enter the number of terms: ");
    scanf("%d", &n);

    for (int i = 0; i < n; ++i) {
        float coefficient;
        int exponent;
        printf("Enter coefficient and exponent for term %d: ", i + 1);
        scanf("%f %d", &coefficient, &exponent);
        insertTerm(poly, coefficient, exponent);
    }
}

void Pwrite(const struct Polynomial* poly) {
    struct Node* current = poly->header->link;
    while (current != poly->header) {
        printf("%.2fx^%d", current->coefficient, current->exponent);
        current = current->link;
        if (current != poly->header) {
            printf(" + ");
        }
    }
    printf("\n");
}

struct Polynomial Padd(const struct Polynomial* a, const struct Polynomial* b) {
    struct Polynomial c;
    initPolynomial(&c);
    struct Node* aPtr = a->header->link;
    struct Node* bPtr = b->header->link;

    while (aPtr != a->header && bPtr != b->header) {
        if (aPtr->exponent > bPtr->exponent) {
            insertTerm(&c, aPtr->coefficient, aPtr->exponent);
            aPtr = aPtr->link;
        } else if (aPtr->exponent < bPtr->exponent) {
            insertTerm(&c, bPtr->coefficient, bPtr->exponent);
            bPtr = bPtr->link;
        } else {
            float sumCoeff = aPtr->coefficient + bPtr->coefficient;
            if (sumCoeff != 0) {
                insertTerm(&c, sumCoeff, aPtr->exponent);
            }
            aPtr = aPtr->link;
            bPtr = bPtr->link;
        }
    }

    while (aPtr != a->header) {
        insertTerm(&c, aPtr->coefficient, aPtr->exponent);
        aPtr = aPtr->link;
    }

    while (bPtr != b->header) {
        insertTerm(&c, bPtr->coefficient, bPtr->exponent);
        bPtr = bPtr->link;
    }

    return c;
}

struct Polynomial Psub(const struct Polynomial* a, const struct Polynomial* b) {
    struct Polynomial negB;
    initPolynomial(&negB);

    struct Node* bPtr = b->header->link;
    while (bPtr != b->header) {
        insertTerm(&negB, -bPtr->coefficient, bPtr->exponent);
        bPtr = bPtr->link;
    }

    struct Polynomial result = Padd(a, &negB);
    freePolynomial(&negB);
    return result;
}

struct Polynomial Pmult(const struct Polynomial* a, const struct Polynomial* b) {
    struct Polynomial c;
    initPolynomial(&c);
    struct Node* aPtr = a->header->link;

    while (aPtr != a->header) {
        struct Node* bPtr = b->header->link;
        while (bPtr != b->header) {
            insertTerm(&c, aPtr->coefficient * bPtr->coefficient, aPtr->exponent + bPtr->exponent);
            bPtr = bPtr->link;
        }
        aPtr = aPtr->link;
    }

    return c;
}

float Peval(const struct Polynomial* poly, float a) {
    float result = 0;
    struct Node* current = poly->header->link;

    while (current != poly->header) {
        result += current->coefficient * pow(a, current->exponent);
        current = current->link;
    }

    return result;
}

void Pearse(struct Polynomial* poly, int targetExponent) {
    struct Node* current = poly->header;
    while (current->link != poly->header) {
        if (current->link->exponent == targetExponent) {
            struct Node* temp = current->link;
            current->link = temp->link;
            free(temp);
            return;
        }
        current = current->link;
    }
}

void freePolynomial(struct Polynomial* poly) {
    struct Node* current = poly->header->link;
    while (current != poly->header) {
        struct Node* temp = current;
        current = current->link;
        free(temp);
    }
    free(poly->header);
}

int main() {
    struct Polynomial a, b;
    initPolynomial(&a);
    initPolynomial(&b);

    printf("Enter polynomial A:\n");
    Pread(&a);

    printf("Enter polynomial B:\n");
    Pread(&b);
```

```
printf("Polynomial A: ");
Pwrite(&a);

printf("Polynomial B: ");
Pwrite(&b);

// Add test cases for other functions
struct Polynomial sum = Padd(&a, &b);
printf("A + B: ");
Pwrite(&sum);

struct Polynomial difference = Psub(&a, &b);
printf("A - B: ");
Pwrite(&difference);

struct Polynomial product = Pmult(&a, &b);
printf("A * B: ");
Pwrite(&product);

float evalResult = Peval(&a, 2.0);  // Evaluate polynomial a at x = 2.0
printf("Evaluated result of A at x = 2.0: %.2f\n", evalResult);

int targetExponent;
printf("Enter the exponent to erase from A: ");
scanf("%d", &targetExponent);
Pearse(&a, targetExponent);
printf("Polynomial A after erasing term with exponent %d: ", targetExponent);
Pwrite(&a);



freePolynomial(&a);
```

```
    freePolynomial(&b);

    freePolynomial(&sum);

    freePolynomial(&difference);

    freePolynomial(&product);


    return 0;

}
```