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

// Define the node structure for a polynomial term
typedef struct Term {
    int coefficient;
    int exponent;
    struct Term* next;
} Term;

// Function to create a new term
Term* createTerm(int coefficient, int exponent) {
    Term* newTerm = (Term*)malloc(sizeof(Term));
    newTerm->coefficient = coefficient;
    newTerm->exponent = exponent;
    newTerm->next = NULL;
    return newTerm;
}

// Function to display a polynomial
void displayPolynomial(Term* poly) {
```

```
    while (poly != NULL) {
        if (poly->coefficient != 0) {
            if (poly->exponent == 0) {
                printf("%d", poly->coefficient);
            } else if (poly->exponent == 1) {
                printf("%dx", poly->coefficient);
            } else {
                printf("%dx^%d", poly->coefficient, poly->exponent);
            }
            if (poly->next != NULL && poly->next->coefficient > 0) {
                printf(" + ");
            }
        }
        poly = poly->next;
    }
    printf("\n");
}
```

```
// Function to add two polynomials
Term* addPolynomials(Term* poly1, Term* poly2) {
    Term* result = NULL;
    Term* last = NULL;

    while (poly1 != NULL && poly2 != NULL) {
        Term* newTerm;

        if (poly1->exponent > poly2->exponent) {
            newTerm = createTerm(poly1->coefficient, poly1->exponent);
            poly1 = poly1->next;
        } else if (poly1->exponent < poly2->exponent) {
            newTerm = createTerm(poly2->coefficient, poly2->exponent);
            poly2 = poly2->next;
        } else {
            newTerm = createTerm(poly1->coefficient + poly2->coefficient, poly1->exponent);
            poly1 = poly1->next;
            poly2 = poly2->next;
        }
```

```
        if (result == NULL) {
            result = newTerm;
        } else {
            last->next = newTerm;
        }
        last = newTerm;
    }

    while (poly1 != NULL) {
        Term* newTerm = createTerm(poly1->coefficient, poly1->exponent);
        if (result == NULL) {
            result = newTerm;
        } else {
            last->next = newTerm;
        }
        last = newTerm;
        poly1 = poly1->next;
    }
```

```
    while (poly2 != NULL) {
        Term* newTerm = createTerm(poly2->coefficient, poly2->exponent);
        if (result == NULL) {
            result = newTerm;
        } else {
            last->next = newTerm;
        }
        last = newTerm;
        poly2 = poly2->next;
    }

    return result;
}
```

```
// Function to multiply two polynomials
Term* multiplyPolynomials(Term* poly1, Term* poly2) {
    Term* result = NULL;

    for (Term* p1 = poly1; p1 != NULL; p1 = p1->next) {
        Term* temp = NULL;
        Term* last = NULL;

        for (Term* p2 = poly2; p2 != NULL; p2 = p2->next) {
            Term* newTerm = createTerm(p1->coefficient * p2->coefficient, p1->exponent + p2->exponent);

            if (temp == NULL) {
                temp = newTerm;
            } else {
                last->next = newTerm;
            }
            last = newTerm;
        }

        result = addPolynomials(result, temp);
```

```
    }

    return result;
}

// Main function to demonstrate polynomial operations
int main() {
    // Creating first polynomial: 5x^3 + 3x^2 + 2x
    Term* poly1 = createTerm(5, 3);
    poly1->next = createTerm(3, 2);
    poly1->next->next = createTerm(2, 1);

    // Creating second polynomial: 4x^3 + x^2 + 7
    Term* poly2 = createTerm(4, 3);
    poly2->next = createTerm(1, 2);
    poly2->next->next = createTerm(7, 0);

    // Displaying the polynomials
    printf("Polynomial 1: ");
    displayPolynomial(poly1);
    printf("Polynomial 2: ");
```

```
    displayPolynomial(poly2);

    // Adding polynomials
    Term* sum = addPolynomials(poly1, poly2);
    printf("Sum: ");
    displayPolynomial(sum);

    // Multiplying polynomials
    Term* product = multiplyPolynomials(poly1, poly2);
    printf("Product: ");
    displayPolynomial(product);

    // Freeing allocated memory (not shown here, but should be included in practice)
    // ...

    return 0;
}
```