Polynomial Representation Using Linked List

A polynomial can be represented using a linked list where each node stores the coefficient and exponent of a term in the polynomial. Here's how you can think about it:

Structure of Each Node

- **Coefficient**: The coefficient of the term (e.g., in 3x23x^23x2, 3 is the coefficient).
- **Exponent**: The exponent of the term (e.g., in 3x23x^23x2, 2 is the exponent).
- **Next Pointer**: A pointer to the next term in the polynomial.

Example

Consider the polynomial:

Example

Consider the polynomial:

$5x^3 + 4x^2 + 2x + 1$

This can be represented as:

Coefficient	Exponent	Next Pointer
5	3	Pointer to next term ($4x^2$)
4	2	Pointer to next term (2x)
2	1	Pointer to next term (1)
1	0	NULL (end of the list)

Class Representation in C/C++ Style Pseudocode

```
struct Node {
    int coefficient;  // Coefficient of the term
    int exponent;     // Exponent of the term
    Node* next;       // Pointer to the next node
(term)

    Node(int coeff, int exp) {
        coefficient = coeff;
        exponent = exp;
        next = nullptr;
    }
```

```
};
```

Operations

1. **Addition**: Traverse two linked lists (polynomials) and combine terms with the same exponents.
2. **Multiplication**: Multiply each term of the first polynomial with every term of the second polynomial, and accumulate terms with the same exponents.
3. **Display**: Traverse the list and print each term in the form $coefficient \times x^{exponent}$.

Example of Polynomial Addition

Let's add two polynomials:

- $5x^3 + 4x^2 + 2x + 1$
- $3x^3 + 2x + 6$

The result will be:

$(5x^3 + 3x^3) + (4x^2) + (2x + 2x) + (1 + 6)\ 8x^3 + 4x^2 + 4x + 7$

In C, a polynomial can be efficiently represented using a linked list where each node contains information about a single term of the polynomial. Each term has a coefficient and an exponent. A node in the linked list contains these values along with a pointer to the next node (term). Here's how you can structure a linked list for polynomial representation and perform basic operations like addition.

Structure of a Node for Polynomial Representation

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

// Structure to represent a node in the linked list
struct Node {
    int coeff; // Coefficient of the term
    int exp;   // Exponent of the term
    struct Node* next; // Pointer to the next node
};

// Function to create a new node
struct Node* createNode(int coeff, int exp) {
```

```
    struct Node* newNode = (struct
Node*)malloc(sizeof(struct Node));
    newNode->coeff = coeff;
    newNode->exp = exp;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a new term in the polynomial
void insertTerm(struct Node** poly, int coeff, int exp)
{
    struct Node* newNode = createNode(coeff, exp);
    newNode->next = *poly;
    *poly = newNode;
}

// Function to display the polynomial
void displayPolynomial(struct Node* poly) {
    struct Node* temp = poly;
    while (temp != NULL) {
        printf("%dx^%d", temp->coeff, temp->exp);
        temp = temp->next;
        if (temp != NULL) {
            if (temp->coeff >= 0)
                printf(" + ");
        }
    }
    printf("\n");
}
```

Example of Creating and Displaying a Polynomial

```
int main() {
    struct Node* poly1 = NULL;

    // Polynomial: 3x^3 + 5x^2 + 2
    insertTerm(&poly1, 2, 0);  // 2 (constant term)
    insertTerm(&poly1, 5, 2);  // 5x^2
    insertTerm(&poly1, 3, 3);  // 3x^3
```

```
    printf("Polynomial 1: ");
    displayPolynomial(poly1);

    return 0;
}
```

Adding Two Polynomials

To add two polynomials, traverse both linked lists simultaneously. If the exponents are equal, add the coefficients; otherwise, append the term with the higher exponent to the result.

```
struct Node* addPolynomials(struct Node* poly1, struct
Node* poly2) {
    struct Node* result = NULL;
    struct Node** lastPtrRef = &result;

    while (poly1 != NULL && poly2 != NULL) {
        if (poly1->exp == poly2->exp) {
            int sumCoeff = poly1->coeff + poly2->coeff;
            if (sumCoeff != 0) {
                insertTerm(lastPtrRef, sumCoeff, poly1-
>exp);
                lastPtrRef = &(*lastPtrRef)->next;
            }
            poly1 = poly1->next;
            poly2 = poly2->next;
        } else if (poly1->exp > poly2->exp) {
            insertTerm(lastPtrRef, poly1->coeff, poly1-
>exp);
            lastPtrRef = &(*lastPtrRef)->next;
            poly1 = poly1->next;
        } else {
            insertTerm(lastPtrRef, poly2->coeff, poly2-
>exp);
            lastPtrRef = &(*lastPtrRef)->next;
            poly2 = poly2->next;
        }
    }
```

```
    // Append remaining terms of poly1 or poly2
    while (poly1 != NULL) {
        insertTerm(lastPtrRef, poly1->coeff, poly1->exp);
        lastPtrRef = &(*lastPtrRef)->next;
        poly1 = poly1->next;
    }
    while (poly2 != NULL) {
        insertTerm(lastPtrRef, poly2->coeff, poly2->exp);
        lastPtrRef = &(*lastPtrRef)->next;
        poly2 = poly2->next;
    }

    return result;
}
```

Example of Adding Polynomials

```
int main() {
    struct Node* poly1 = NULL;
    struct Node* poly2 = NULL;

    // Polynomial 1: 3x^3 + 5x^2 + 2
    insertTerm(&poly1, 2, 0);
    insertTerm(&poly1, 5, 2);
    insertTerm(&poly1, 3, 3);

    // Polynomial 2: 4x^2 + 3x + 1
    insertTerm(&poly2, 1, 0);
    insertTerm(&poly2, 3, 1);
    insertTerm(&poly2, 4, 2);

    printf("Polynomial 1: ");
    displayPolynomial(poly1);

    printf("Polynomial 2: ");
    displayPolynomial(poly2);

    struct Node* result = addPolynomials(poly1, poly2);
    printf("Sum of polynomials: ");
    displayPolynomial(result);

    return 0;
}
```

Output

```
Polynomial 1: 3x^3 + 5x^2 + 2
Polynomial 2: 4x^2 + 3x + 1
Sum of polynomials: 3x^3 + 9x^2 + 3x + 3
```