# Code ภูมิไทย พรมโกฏิ 65090500451

```cpp
#include <bits/stdc++.h>
using namespace std;
struct Node {
    int coefficient;
    int exponent;
    Node *next;
};
void insert(Node *&head, int coefficient, int exponent) {
    Node *newNode = new Node;
    newNode->coefficient = coefficient;
    newNode->exponent = exponent;
    newNode->next = head;
    head = newNode;
}
void print(Node *poly) {
    if (poly == nullptr) {
        cout << "0" << endl;
        return;
    }
    bool isFirstTerm = true;
    while (poly != nullptr) {
        if (poly->coefficient != 0) {
            if (isFirstTerm) {
                cout << poly->coefficient;
                isFirstTerm = false;
            } else {
                if (poly->coefficient > 0) {
                    cout << " + " << poly->coefficient;
                } else {
                    cout << " - " << abs(poly->coefficient);
                }
            }

            if (poly->exponent == 1) {
                cout << "x";
            } else if (poly->exponent > 1) {
                cout << "x^" << poly->exponent;
            }
        }
        poly = poly->next;
    }
```

```
    cout << endl;
}
//add
Node* add(Node *poly1, Node *poly2) {
    Node *result = nullptr;
    Node **lastPtrRef = &result;

    while (poly1 != nullptr && poly2 != nullptr) {
        if (poly1->exponent > poly2->exponent) {
            insert(*lastPtrRef, poly1->coefficient, poly1->exponent);
            poly1 = poly1->next;
        } else if (poly1->exponent < poly2->exponent) {
            insert(*lastPtrRef, poly2->coefficient, poly2->exponent);
            poly2 = poly2->next;
        } else {
            int sum = poly1->coefficient + poly2->coefficient;
            if (sum != 0) {
                insert(*lastPtrRef, sum, poly1->exponent);
            }
            poly1 = poly1->next;
            poly2 = poly2->next;
        }
        lastPtrRef = &((*lastPtrRef)->next);
    }


    while (poly1 != nullptr) {
        insert(*lastPtrRef, poly1->coefficient, poly1->exponent);
        poly1 = poly1->next;
        lastPtrRef = &((*lastPtrRef)->next);
    }

    while (poly2 != nullptr) {
        insert(*lastPtrRef, poly2->coefficient, poly2->exponent);
        poly2 = poly2->next;
        lastPtrRef = &((*lastPtrRef)->next);
    }

    // Reverse the linked list
    Node *prev = nullptr;
    Node *curr = result;
    Node *next = nullptr;
```

```
    while (curr != nullptr) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    result = prev;

    return result;
}

//multiply
Node* multiply(Node *poly1, Node *poly2) {
    Node *result = nullptr;

    while (poly1 != nullptr) {
        Node *temp = poly2;
        while (temp != nullptr) {
            int coefficient = poly1->coefficient * temp->coefficient;
            int exponent = poly1->exponent + temp->exponent;
            insert(result, coefficient, exponent);
            temp = temp->next;
        }
        poly1 = poly1->next;
    }

    // Simplify
    Node *prev = result;
    Node *curr = result->next;
    while (curr != nullptr) {
        if (prev->exponent == curr->exponent) {
            prev->coefficient += curr->coefficient;
            prev->next = curr->next;
            delete curr;
            curr = prev->next;
        } else {
            prev = curr;
            curr = curr->next;
        }
        }
    return result;
}
int main() {
```

```
    // Create the first polynomial
    Node *poly1 = nullptr;
    insert(poly1, 5, 2);
    insert(poly1, 4, 1);
    insert(poly1, 2, 0);
    cout << "Poly1: ";
    print(poly1);
    // Create the second polynomial
    Node *poly2 = nullptr;
    insert(poly2, 3, 1);
    insert(poly2, 2, 0);
    cout << "Poly2: ";
    print(poly2);

    // Add the polynomials
    Node *sum = add(poly1, poly2);
    cout << "Addition : ";
    print(sum);

    // Multiply the polynomials
    Node *product = multiply(poly1, poly2);
    cout << "Multiplication : ";
    print(product);

    return 0;
}
```

# OUTPUT

```
PS D:\C Projet\output> & .\'poly.exe'
Poly1: 2 + 4x + 5x^2
Poly2: 2 + 3x
Addition : 5x^2 + 7x + 4
Multiplication : 15x^3 + 22x^2 + 14x + 4
```

# Time Complexity

Time Complexityของฟังก์ชัน ***insert()*** คือ ***O(1)*** เนื่องจากเกี่ยวข้องกับการกำหนดอย่างง่ายและการดำเนินการพอยน์เตอร์เท่านั้น

Time Complexityของฟังก์ชัน ***print()*** คือ ***O(n)*** โดยที่ $n$ คือจำนวนของโหนดในรายการที่เชื่อมโยง เนื่องจากจำเป็นต้องสำรวจรายการที่เชื่อมโยงและพิมพ์แต่ละโหนด

Time Complexityของฟังก์ชัน ***add()*** คือ ***O(m+n)*** โดยที่ ***m*** และ ***n*** คือความยาวของรายการที่เชื่อมโยงอินพุต เนื่องจากจำเป็นต้องข้ามรายการที่เชื่อมโยงทั้งสองอย่างน้อยหนึ่งครั้ง

Time Complexityของฟังก์ชัน ***multiply()*** คือ ***O(m*n)*** โดยที่ ***m*** และ ***n*** คือความยาวของรายการที่เชื่อมโยงอินพุต เนื่องจากจำเป็นต้องทำการวนซ้ำแบบซ้อนเพื่อคูณแต่ละเทอมในรายการที่เชื่อมโยงแรกกับแต่ละเทอมใน รายการเชื่อมโยงที่สอง

โดยรวมแล้ว Time ComplexityของโปรแกรมถูกควบคุมโดยTime Complexityของฟังก์ชัน ***multiply()*** ซึ่งก็คือ

***O(m*n)***