解題說明:

這次的 homework3 利用到了 homework2 的多項式類別。然後將多項式類別的數值存入圓形鏈結中，然後在將圓形鏈結中的值做運算。

程式:

```
struct Node {
    int coef; // 係數
    int exp;  // 次方
    Node* link;

    Node(int c = 0, int e = 0) : coef(c), exp(e), link(nullptr) {}
};
```

```
class Polynomial {
private:
    Node* first; //圓形鏈結的第一個節點

    //刪除鏈結
    void clear() {
        if (!first) return;
        Node* current = first->link;
        while (current != first) {
            Node* temp = current;
            current = current->link;
            delete temp;
        }
        delete first;
        first = nullptr;
    }
```

```
// 複製
void copy(const Polynomial& other) {
    if (!other.first) {
        first = nullptr;
        return;
    }

    first = new Node();
    Node* current = other.first->link;
    Node* tail = first;
    while (current != other.first) {
        tail->link = new Node(current->coef, current->exp);
        tail = tail->link;
        current = current->link;
    }
    tail->link = first;
}
```

```
public:
    // 建構函數
    Polynomial() : first(nullptr) {}

    // 複製建構函數
    Polynomial(const Polynomial& other) {
        copy(other);
    }

    //解構函數
    ~Polynomial() {
        clear();
    }

    Polynomial& operator=(const Polynomial& other) {
        if (this != &other) {
            clear();
            copy(other);
        }
        return *this;
    }
```

```cpp
// 多載 >>，用於輸入
friend istream& operator>>(istream& is, Polynomial& x) {
    int n;
    is >> n; // 項數
    x.clear();
    x.first = new Node();
    Node* tail = x.first;

    for (int i = 0; i < n; ++i) {
        int coef, exp;
        is >> coef >> exp;
        tail->link = new Node(coef, exp);
        tail = tail->link;
    }
    tail->link = x.first;
    return is;
}

// 多載 <<，用於輸出
friend ostream& operator<<(ostream& os, const Polynomial& x) {
    if (!x.first) return os;

    Node* current = x.first->link;
    while (current != x.first) {
        os << current->coef << "x^" << current->exp;
        if (current->link != x.first) os << " + ";
        current = current->link;
    }
    return os;
}
```

```cpp
// 加法運算
Polynomial operator+(const Polynomial& b) const {
    if (!first) return b;  // 如果當前多項式為空，返回 b
    if (!b.first) return *this;  // 如果 b 為空，返回當前多項式

    Polynomial result;
    Node* aPtr = first->link;
    Node* bPtr = b.first->link;
    Node* tail = result.first = new Node();

    while (aPtr != first || bPtr != b.first) {
        if (aPtr != first && (bPtr == b.first || aPtr->exp > bPtr->exp)) {
            tail->link = new Node(aPtr->coef, aPtr->exp);
            aPtr = aPtr->link;
        }
        else if (bPtr != b.first && (aPtr == first || aPtr->exp < bPtr->exp)) {
            tail->link = new Node(bPtr->coef, bPtr->exp);
            bPtr = bPtr->link;
        }
        else {
            int sum = aPtr->coef + bPtr->coef;
            if (sum != 0) {
                tail->link = new Node(sum, aPtr->exp);
            }
            aPtr = aPtr->link;
            bPtr = bPtr->link;
        }
        tail = tail->link;
    }
    tail->link = result.first;
    return result;
}
```

```cpp
// 減法運算
Polynomial operator-(const Polynomial& b) const {
    if (!first) return b;
    if (!b.first) return *this;

    Polynomial result;
    Node* aPtr = first->link;
    Node* bPtr = b.first->link;
    Node* tail = result.first = new Node();

    while (aPtr != first || bPtr != b.first) {
        if (aPtr->exp > bPtr->exp) {
            tail->link = new Node(aPtr->coef, aPtr->exp);
            aPtr = aPtr->link;
        }
        else if (aPtr->exp < bPtr->exp) {
            tail->link = new Node(-bPtr->coef, bPtr->exp);
            bPtr = bPtr->link;
        }
        else {
            int diff = aPtr->coef - bPtr->coef;
            if (diff != 0) {
                tail->link = new Node(diff, aPtr->exp);
            }
            aPtr = aPtr->link;
            bPtr = bPtr->link;
        }
        tail = tail->link;
    }
    tail->link = result.first;
    return result;
}
```

```cpp
// 乘法運算
Polynomial operator*(const Polynomial& b) const {
    Polynomial result;
    if (!first) return b;
    if (!b.first) return *this;

    if (!first || !b.first) return result;

    Node* aPtr = first->link;
    while (aPtr != first) {
        Node* bPtr = b.first->link;
        Polynomial temp;
        Node* tail = temp.first = new Node();

        while (bPtr != b.first) {
            tail->link = new Node(aPtr->coef * bPtr->coef, aPtr->exp + bPtr->exp);
            tail = tail->link;
            bPtr = bPtr->link;
        }
        tail->link = temp.first;
        result = result + temp;
        aPtr = aPtr->link;
    }
    return result;
}
```

```
// 給 x 值計算多項式的值
int Evaluate(int x) const {
    int result = 0;
    if (!first) return result;
    Node* current = first->link;
    while (current != first) {
        result += current->coef * pow(x, current->exp);
        current = current->link;
    }
    return result;
}
```

```
int main() {
    Polynomial p1, p2;
    cout << "輸入第一個多項式（輸入項數、每項的係數和次方）：\n";
    cin >> p1;
    cout << "輸入第二個多項式（輸入項數、每項的係數和次方）：\n";
    cin >> p2;

    Polynomial sum = p1 + p2;
    cout << "加法結果：" << sum << endl;

    Polynomial diff = p1 - p2;
    cout << "減法結果：" << diff << endl;

    Polynomial prod = p1 * p2;
    cout << "乘法結果：" << prod << endl;

    int x;
    cout << "輸入 x 的值：";
    cin >> x;
    cout << "計算結果：" << p1.Evaluate(x) << endl;

    return 0;
}
```

```
輸入第一個多項式（輸入項數、每項的係數和次方）：
3
4 3
2 2
-1 0
輸入第二個多項式（輸入項數、每項的係數和次方）：
2
3 3
-2 1
加法結果：7x^3 + 2x^2 + -2x^1 + -1x^0
減法結果：1x^3 + 2x^2 + 2x^1 + -1x^0 + -3x^3 + 2x^1
乘法結果：12x^6 + 6x^5 + -8x^4 + -7x^3 + 2x^1
輸入 x 的值：2
計算結果(加法)：59
計算結果(減法)：-1
計算結果(乘法)：780
```

效能分析:

時間複雜度:

輸入/輸出:O(n)

加法/減法:O(n + m)

乘法:O(n * m)

帶入 x 求值:O(n)

刪除:O(n)

複製:O(n)

空間複雜度:

輸入:O(n)

輸出:O(1)

加法/減法:O(n + m)

乘法:O(n * m)

帶入 x 求值:O(n)

刪除:O(1)

複製:O(1)