資料結構作業

姓名 ：莊笙禾

第一題 ：解題說明

實作一個使用「循環連結串列」來儲存並運算多項式的 C++ 類別

功能：

- 讀取多項式
- 多項式的加法、減法、乘法運算
- 針對任意 x 值進行 Evaluate( )
- 輸出多項式

抽象資料型態 (ADT)

多項式 Polynomial：讀取多項式

- 係數 (coefficients)
- 指數 (exponents)
- 以「循環串列 + 標頭節點」做儲存

多項式表示方式(舉例)

$$F(x) = 3x^2 + 2x + 1$$

- 項 1：係數 3，次方 2
- 項 2：係數 2，次方 1
- 項 3：係數 1，次方 0

此時 n c1 e1 c2 e2 分別為：3 3 2 2 1 1 0

第一題 ：演算法設計與實作

```
#include <iostream>
using namespace std;

struct Node {
    int coef;
    int exp;
    Node* link;
};

class Polynomial {
private:
    Node* head;
    Node* createNode(int c, int e, Node* nxt = nullptr) {
        Node* newNode = new Node;
        newNode->coef = c;
        newNode->exp = e;
        newNode->link = nxt;
        return newNode;
    }
    void clearList() {
        Node* curr = head->link;
        while (curr != head) {
            Node* tmp = curr;
            curr = curr->link;
            delete tmp;
        }
        head->link = head;
    }
public:
    Polynomial() {
        head = new Node;
        head->coef = 0; head->exp = 0; head->link = head;
    }
    Polynomial(const Polynomial& other) {
        head = new Node;
        head->coef = 0; head->exp = 0; head->link = head;
        Node* p = other.head->link;
        while (p != other.head) {
            insertTerm(p->coef, p->exp);
            p = p->link;
        }
    }
    ~Polynomial() {
        clearList();
        delete head;
    }
    Polynomial& operator=(const Polynomial& rhs) {
        if (this == &rhs) return *this;
        clearList();
        Node* p = rhs.head->link;
        while (p != rhs.head) {
            insertTerm(p->coef, p->exp);
            p = p->link;
        }
        return *this;
    }
    void insertTerm(int c, int e) {
        if (c == 0) return;
        Node* prev = head;
        Node* curr = head->link;
        while (curr != head && curr->exp > e) {
            prev = curr; curr = curr->link;
        }
        if (curr != head && curr->exp == e) {
            curr->coef += c;
            if (curr->coef == 0) {
                prev->link = curr->link;
                delete curr;
            }
        }
        else {
            Node* newNode = createNode(c, e);
            prev->link = newNode;
            newNode->link = curr;
        }
    }
    Polynomial operator+(const Polynomial& rhs) const {
        Polynomial result;
        Node* p = head->link;
        while (p != head) {
            result.insertTerm(p->coef, p->exp);
            p = p->link;
        }
        p = rhs.head->link;
        while (p != rhs.head) {
            result.insertTerm(p->coef, p->exp);
            p = p->link;
        }
        return result;
    }
    Polynomial operator-(const Polynomial& rhs) const {
        Polynomial result;
        Node* p = head->link;
        while (p != head) {
            result.insertTerm(p->coef, p->exp);
            p = p->link;
        }
        p = rhs.head->link;
        while (p != rhs.head) {
            result.insertTerm(-p->coef, p->exp);
            p = p->link;
        }
        return result;
    }
    Polynomial operator*(const Polynomial& rhs) const {
        Polynomial result;
        Node* p = head->link;
        while (p != head) {
            Node* q = rhs.head->link;
            while (q != rhs.head) {
                result.insertTerm(p->coef * q->coef, p->exp + q->exp);
                q = q->link;
            }
            p = p->link;
        }
        return result;
    }
    double Evaluate(double x) const {
        double sum = 0.0;
        Node* p = head->link;
        while (p != head) {
            double termVal = p->coef;
            for (int i = 0; i < p->exp; i++)
                termVal *= x;
            sum += termVal;
            p = p->link;
        }
        return sum;
    }
    friend ostream& operator<<(ostream& os, const Polynomial& poly) {
        int count = 0;
        Node* p = poly.head->link;
        while (p != poly.head) {
            count++;
            p = p->link;
        }
        os << count;
        p = poly.head->link;
        while (p != poly.head) {
            os << " " << p->coef << " " << p->exp;
            p = p->link;
        }
        return os;
    }
    friend istream& operator>>(istream& is, Polynomial& poly) {
        poly.clearList();
        int n; is >> n;
        for (int i = 0; i < n; i++) {
            int c, e; is >> c >> e;
            poly.insertTerm(c, e);
        }
        return is;
    }
};

int main() {
    Polynomial p1, p2;
    cout << "請輸入第一個多項式 (格式: n c1 e1 c2 e2 ...): ";
    cin >> p1;
    cout << "請輸入第二個多項式 (格式: n c1 e1 c2 e2 ...): ";
    cin >> p2;

    cout << "p1 = " << p1 << endl;
    cout << "p2 = " << p2 << endl;

    Polynomial p3 = p1 + p2;
    Polynomial p4 = p1 - p2;
    Polynomial p5 = p1 * p2;

    cout << "p1 + p2 = " << p3 << endl;
    cout << "p1 - p2 = " << p4 << endl;
    cout << "p1 * p2 = " << p5 << endl;

    double xValue;
    cout << "請輸入一個 x 的值: ";
    cin >> xValue;
    cout << "p1 在 x=" << xValue << " 時的值為: " << p1.Evaluate(xValue) << endl;
    return 0;
}
```

第一題 ：效能分析

時間複雜度分析

插入 (insertTerm)：

1. 需要在串列中尋找插入位置，最久需要走訪所有節點，
   O(m)，其中 m為多項式的項目數

加法 / 減法：

1. 若直接把每個項目依序插入結果多項式，最久可達 O((m+n)^2)；
   若已按指數順序合併，可降至 O(m+n)

乘法 ：

1. 需對兩多項式各節點兩兩相乘 (m×n次)，每次結果插入結果多項式；最壞可到 O(mn(m+n))

Evaluate(x)：

1. 只要線性走訪一遍做計算，為 O(m)

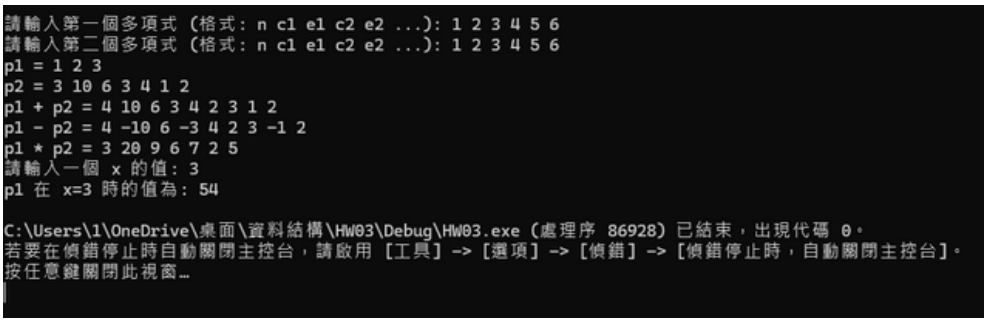
空間複雜度分析

1. 每個非零項目對應一個節點，若有 m個項目即需 O(m)空間；
2. 加減乘後的結果最多 m+n或 m+n−1 項，
3. 依題目定義空間上限也在 O(m+n)量級

第一題 ：測試與驗證

執行畫面

先讀兩個多項式 (p1, p2)->依序顯示 p1 與 p2 的內容->顯示它們加、減、乘的結果->輸入一個 x 值將 x 代入 p1 計算



第一題 ：效能量測

1. 設定計時器：
   使用計時功能（例如 time 模組）測量程序執行時間

2. 計算時間：
   測量從運算開始到結束所花費的時間

3. 輸出結果：
   顯示計算結果以及所需的總運算時間。