資料結構 HW2

Chapter 1 解題說明

本次作業目標是實作一個多項式類別（Polynomial），其主要功能包括：

1. 動態新增多項式的項目，並自動合併同類項（即指數相同的項目）。
2. 使用運算子重載的方式，提供直觀的多項式輸入（>>）與輸出（<<）。
3. 遵循物件導向設計，確保程式結構清晰且具有擴展性。

關鍵想法：

- **數據結構**：使用結構（struct Term）來表示多項式的每一項，並使用向量（std::vector）儲存所有項目。
- **合併同類項**：新增項目時，檢查是否已存在相同指數的項目，若有則合併，否則直接加入向量中。

範例輸入與輸出：

輸入項目如下：

(3 2)

(-4 1)

(5 0)

(0 0)

表示（3x^2 - 4x +5)的內容。

Chapter 2 程式實作

核心程式碼：

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

// 定義 Polynomial 類別
class Polynomial {
private:
    struct Term {
        int coefficient; // 係數
        int exponent;    // 指數
        Term(int c, int e) : coefficient(c), exponent(e) {}
    };
    vector<Term> terms; // 儲存多項式的項目

public:
    // 新增一項到多項式
    void addTerm(int coefficient, int exponent) {
        for (auto &term : terms) {
            if (term.exponent == exponent) {
                term.coefficient += coefficient; // 合併同類項
```

```
                return;
            }
        }
        terms.emplace_back(coefficient, exponent); //  新增新項目
}

//  重載輸入運算子  >>
friend istream &operator>>(istream &in, Polynomial &poly) {
        int coefficient, exponent;
        cout << "輸入係數與指數（輸入  0 0  結束）: \n";
        while (true) {
            in >> coefficient >> exponent;
            if (coefficient == 0 && exponent == 0) break;
            poly.addTerm(coefficient, exponent);
        }
        return in;
}

//  重載輸出運算子  <<
friend ostream &operator<<(ostream &out, const Polynomial &poly) {
        if (poly.terms.empty()) {
            out << "0";
            return out;
        }
```

```
        bool firstTerm = true;
        for (const auto &term : poly.terms) {
            if (term.coefficient == 0) continue;
            if (!firstTerm && term.coefficient > 0) out << " + ";
            if (term.coefficient < 0) out << " - ";
            if (abs(term.coefficient) != 1 || term.exponent == 0)
                out << abs(term.coefficient);
            if (term.exponent > 0) out << "x";
            if (term.exponent > 1) out << "^" << term.exponent;
            firstTerm = false;
        }
        return out;
    }
};

int main() {
    Polynomial p1;
    cin >> p1; //  輸入多項式
    cout << "您輸入的多項式為：  " << p1 << endl; //  輸出多項式
    return 0;
}
```

Chapter 3 效能分析

時間複雜度

1. **addTerm** 方法：
   每次新增項目需要遍歷目前的多項式，時間複雜度為 **O(n)**，其中 nnn 是多項式中目前的項目數。

2. **>> 輸入運算子重載**：
   若輸入 mmm 項，則每次輸入都需要調用 addTerm，時間複雜度為 **O(m × n)**。

3. **<< 輸出運算子重載**：
   輸出需要遍歷所有項目，時間複雜度為 **O(n)**。

空間複雜度

多項式的項目存於向量中，空間複雜度為 **O(n)**，其中 nnn 為多項式的項目數。

Chapter 4 測試與驗證

測試案例

案例 1：一般輸入

- 輸入：3 2 -4 1 5 0 0 0
- 預期輸出：3x^2 - 4x + 5

案例 2：含重複指數

- 輸入：2 2 3 2 -1 1 0 0
- 預期輸出：5x^2 - x

案例 3：所有係數為 0

- 輸入：0 0
- 預期輸出：0

Chapter 5 申論及開發報告

在本次作業中，通過設計多項式類別，熟悉了如何運用運算子重載來簡化輸入與輸出操作，並加深了對物件導向程式設計的理解。同時，透過 AI 協助完成基礎結構與效能分析，顯著提高了效率。未來可考慮增加多項式運算（如加法、乘法等）以提升實用性。

AI 協助部分：

1. 協助撰寫初始程式碼框架。
2. 協助進行時間與空間複雜度分析。
3. 提供結構化測試案例範例。