壹、解題說明

一、加法功能 (Add)

透過兩個指標移動，如果兩個指標指到的項次方數相同，則將係數相加，加到結果多項式中；如果 poly1 的該項指標次方數小於 poly2 的該項指標次方數，則將 poly2 該項加入到結果多項式中；如果 poly1 的該項指標次方數大於 poly2 的該項指標次方數，則將 poly1 該項加入到結果多項式中。

二、乘法功能 (Mult)

透過兩個巢狀迴圈，將第一個多項式的每一項與第二個多項式的每一項相乘，並將結果加入到新的多項式中。接著，合併相同次方數的項，並按次方數降冪排列。

三、求值 (Eval)

使用輸入的值，將其代入多項式的每項中，計算每項的數值，並將其全部加起來。

四、輸入

使用重載輸入運算子 ">>"，每次先忽略開頭的 "("，接著讀入每項係數，忽略後面的 "X" 和 "^"，再讀入次方數，直到讀入 ")" 時，停止輸入。

五、輸出

依序將多項式的係數輸出，並加上 X 和 "^"。如果指數次方為 0，則只輸出係數，並判斷是否為最後一項。如果是，則不輸出"＋"；反之，則在每項後輸出"＋"。

## 貳、程式實作

```
// 預先宣告 Polynomial 類別，以供 Term 類別的 friend 使用
class Polynomial;

// 定義 Term 類別，表示多項式中的單一項目
class Term {
    friend Polynomial; // 讓 Polynomial 類別可以直接存取 Term 類別的私有成員
    friend istream& operator>>(istream& in, Polynomial& p); // 多載輸入運算子 >>
    friend ostream& operator<<(ostream& out, const Polynomial& p); // 多載輸出運算子 <<
private:
    double coef; // 項目的係數
    int exp; // 項目的次方數
};
```

圖 1 定義 Term 物件

```
// 定義 Polynomial 類別，表示多項式的集合
class Polynomial {
    friend istream& operator>>(istream& in, Polynomial& p); // 多載輸入運算子 >>
    friend ostream& operator<<(ostream& out, const Polynomial& p); // 多載輸出運算子 <<
private:
    Term* termArray; // 用來儲存多項式的非零項（以陣列形式儲存）
    int capacity; // termArray的容量
    int terms; // 多項式中非零項目的數量
public:
    Polynomial(); // 預設建構子，初始化多項式
    Polynomial(const Polynomial& p); // 拷貝建構子
    Polynomial& operator=(const Polynomial& p); // 賦值運算子
    ~Polynomial(); // 解構子，釋放記憶體

    Polynomial Add(Polynomial poly); // 多項式加法運算
    Polynomial Mult(Polynomial poly); // 多項式乘法運算
    double Eval(double f); // 計算多項式在 x = f 時的值
    void NewTerm(const double newCoef, const int newExp); // 新增項目至多項式
};
```

圖 2 定義 Polynomial 物件

```
// 預設建構子：初始化多項式
Polynomial::Polynomial() : capacity(2), terms(0) {
    termArray = new Term[capacity]; // 配置初始陣列空間
}
```

圖3 Polynomial預設建構子

```
// 拷貝建構子
Polynomial::Polynomial(const Polynomial& p) : capacity(p.capacity), terms(p.terms) {
    termArray = new Term[capacity]; // 分配新的記憶體
    copy(p.termArray, p.termArray + terms, termArray); // 複製內容
}
```

圖 4 拷貝建構子

```
// 賦值運算子
Polynomial& Polynomial::operator=(const Polynomial& p) {
    if (this != &p) { // 防止自我賦值
        delete[] termArray; // 釋放原有的記憶體
        capacity = p.capacity;
        terms = p.terms;
        termArray = new Term[capacity]; // 分配新的記憶體
        copy(p.termArray, p.termArray + terms, termArray); // 複製內容
    }
    return *this;
}
```

圖 5 賦值建構子

```
// 解構子：釋放記憶體
Polynomial::~Polynomial() {
    delete[] termArray; // 釋放動態分配的陣列記憶體
}
```

圖 6 Polynomial 解構子

```
// 新增一項至多項式中，若容量不足則擴展容量
void Polynomial::NewTerm(const double newCoef, const int newExp) {
    // 檢查是否需要擴展容量
    if (terms == capacity) {
        capacity *= 2; // 容量翻倍
        Term* tmp = new Term[capacity]; // 新的陣列
        copy(termArray, termArray + terms, tmp); // 複製現有的項到新陣列
        delete[] termArray; // 釋放舊的陣列記憶體
        termArray = tmp; // 更新 termArray 指標
    }
    termArray[terms].coef = newCoef; // 設定新的係數
    termArray[terms++].exp = newExp; // 設定新的次方數，並更新項數
}
```

圖 7 加入新項

```
// 多項式加法運算
Polynomial Polynomial::Add(Polynomial b) {
    Polynomial poly3; // 用來儲存加法結果的多項式
    int c1 = terms; // poly1 的項數
    int c2 = b.terms; // poly2 的項數
    int i1 = 0; // poly1 的當前項索引
    int i2 = 0; // poly2 的當前項索引
    // 進行加法運算，直到兩個多項式的所有項都處理完畢
    while (c1 > 0 || c2 > 0) {
        if (termArray[i1].exp > b.termArray[i2].exp) {
            poly3.NewTerm(termArray[i1].coef, termArray[i1].exp); // poly1 的項較大，加入 poly3
            i1++; c1--;
        }
        else if (b.termArray[i2].exp > termArray[i1].exp) {
            poly3.NewTerm(b.termArray[i2].coef, b.termArray[i2].exp); // poly2 的項較大，加入 poly3
            i2++; c2--;
        }
        else if (b.termArray[i2].exp == termArray[i1].exp) {
            // 若兩個多項式的項次相同，則將其係數相加
            poly3.NewTerm((b.termArray[i2].coef + termArray[i1].coef), b.termArray[i2].exp);
            i1++; i2++; c1--; c2--;
        }
    }
    return poly3;
}
```

圖 8 多項式加法運算

```
// 多項式乘法運算
Polynomial Polynomial::Mult(Polynomial b) {
    Polynomial poly3; // 用來儲存乘法結果的多項式
    for (int i = 0; i < terms; i++) { // 遍歷 poly1 的每個項
        for (int j = 0; j < b.terms; j++) { // 遍歷 poly2 的每個項
            double newCoef = termArray[i].coef * b.termArray[j].coef; // 係數相乘
            int newExp = termArray[i].exp + b.termArray[j].exp; // 次方數相加
            poly3.NewTerm(newCoef, newExp); // 新項加入 poly3
        }
    }

    // 合併同次方的項
    for (int i = 0; i < poly3.terms; i++) {
        for (int j = i + 1; j < poly3.terms; j++) {
            if (poly3.termArray[i].exp == poly3.termArray[j].exp) {
                poly3.termArray[i].coef += poly3.termArray[j].coef; // 合併係數
                // 移除第 j 項，將後續項向前移動
                for (int k = j; k < poly3.terms - 1; k++) {
                    poly3.termArray[k] = poly3.termArray[k + 1];
                }
                poly3.terms--; // 減少項數
                j--; // 調整索引以重新檢查合併後的位置
            }
        }
    }

    // 依次方數降冪排序
    for (int i = 0; i < poly3.terms - 1; i++) {
        for (int j = i + 1; j < poly3.terms; j++) {
            if (poly3.termArray[i].exp < poly3.termArray[j].exp) {
                Term temp = poly3.termArray[i];
                poly3.termArray[i] = poly3.termArray[j];
                poly3.termArray[j] = temp;
            }
        }
    }

    return poly3;
}
```

圖 9 多項式乘法運算

```
// 計算多項式在某個 x 值的值
double Polynomial::Eval(double x) {
    double result = 0;
    for (int i = 0; i < terms; i++) {
        result += termArray[i].coef * pow(x, termArray[i].exp); // 計算每項的值並累加
    }
    return result; // 回傳結果
}
```

圖 10 多項式求值

```
// 多載 << 運算子，用來輸出多項式
ostream& operator<<(ostream& out, const Polynomial& p) {
    out << "(";
    out << p.termArray[0].coef << "X^" << p.termArray[0].exp; // 輸出第一項
    for (int i = 1; i < p.terms; i++) {
        if (p.termArray[i].coef > 0)
            out << " + " << p.termArray[i].coef << "X^" << p.termArray[i].exp; // 若係數為正數，顯示加號
        else
            out << " - " << p.termArray[i].coef * -1 << "X^" << p.termArray[i].exp; // 若係數為負數，顯示減號
    }
    out << ")";
    return out;
}
```

圖 11 多載<<

```
istream& operator>>(istream& in, Polynomial& p) { // 多載 >>
    double coeftemp = 0; // 暫存係數
    int exptemp = 0; // 暫存次方數
    char ch;

    while (1) {
        coeftemp = 0;
        exptemp = 0;
        ch = in.peek(); // 檢查cin流上的第一個位元
        while (ch < '0' || ch > '9') { // 直到下一個位元是數字就代表是系數了就停下來準備輸入到coeftemp
            if (ch == '-') {
                in.get();
                ch = in.peek();
                while (ch < '0' && ch > '9') {
                    in.get();
                    ch = in.peek();
                }
                in >> coeftemp; // 讀取係數
                coeftemp *= -1;
                break;
            }
            in.get();
            ch = in.peek();
            if (ch >= '0' && ch <= '9') {
                in >> coeftemp; // 讀取係數
                break;
            }
        }
        bool check = true;

        ch = in.peek(); // 檢查cin流上的第一個位元
        while (ch < '0' || ch > '9') { // 反覆確認直到下一個位元是數字就代表是次方數了就停下來準備輸入到exptemp
            in.get();
            ch = in.peek();
        }
        in >> exptemp; // 讀取次方數
        p.NewTerm(coeftemp, exptemp); // 新增項目
        ch = in.peek(); // 檢查cin流上的第一個位元
        if (ch == ')') break;
        while (1) { // 反覆確認直到下一個位元，如果是 + or - 就代表後面還有至少一組要輸入，如果再過程中找到')'就代表已經結束輸入了
            if (ch == '+' || ch == '-') {
                break;
            }
            if (ch == '+' || ch == '-') {
                break;
            }
            in.get();
            ch = in.peek();

            if (ch == ')') {
                check = false;
                break;
            }
        }
        if (!check)
            break;
    }
    in.get(); // 忽略換行符號（如果有的話）
    return in;
}
```

圖 12 多載>>

```
// 主程式
int main() {
    Polynomial poly1, poly2, poly3, poly4;
    cout << "輸入範例:(5X^3+2X^2+1X^0) <== 括號一定要加( )" << endl;
    cout << "請輸入第一個多項式:" << endl;
    cin >> poly1;
    cout << "請輸入第二個多項式:" << endl;
    cin >> poly2;
    poly3 = poly1.Add(poly2);
    cout << "相加結果為：" << poly3 << endl;
    poly4 = poly1.Mult(poly2);
    cout << "相乘結果為：" << poly4 << endl;
    double x = 0; cout << "輸入一個值來計算多項式的值: ";
    cin >> x;
    cout << poly3 << ": x=" << "(" << x << ") = " << poly3.Eval(x) << endl;
    cout << poly4 << ": x=" << "(" << x << ") = " << poly4.Eval(x) << endl;
    return 0;
}
```

圖 13 主程式

參、效能分析

| | 時間複雜度 | 空間複雜度 |
|---|---|---|
| 加入新項 | O(terms) | O(capacity) |
| 多項式加法 | O(terms + b.terms) | O(terms + b.terms) |
| 多項式乘法 | O(terms * b.terms) | O(terms * b.terms) |
| 求值 | O(terms) | O(1) |
| 多載<< | O(terms) | O(1) |
| 多載>> | O(terms) | O(1) |

表 1 效能分析

肆、測試與驗證

輸入:

Poly1:(3X^2-2X^1+3X^0)

Poly2:(3X^2-2X^1+3X^0)

代入數字:2

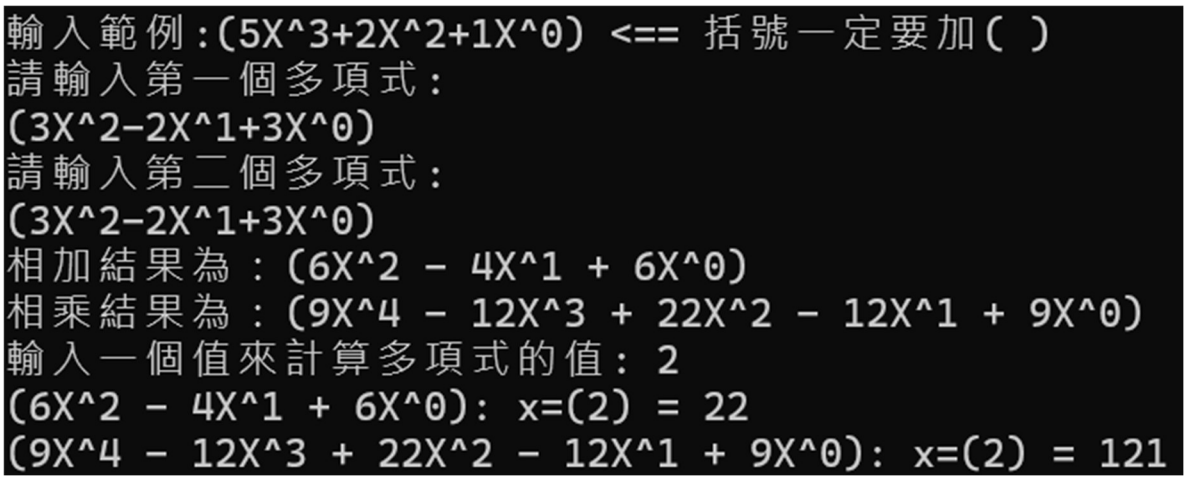

圖 14 測試與驗證

伍、申論及開發報告

在開發過程中，我遇到了一個因為解構子所引起的程序崩潰問題，這讓我意識到在某些情況下，如果沒有正確地實現拷貝建構子和賦值建構子，會導致程序出現未預期的錯誤。

具體來說，在處理多項式類別 Polynomial 的加法運算時，我寫了一個返回 Polynomial 類型資料的函數 Add()。該函數返回一個新的 Polynomial 物件。當我未實現拷貝建構子時，發現返回的物件在函式執行完畢後會崩潰。這是因為 Add() 函數返回的是一個物件的位址，然而當函式返回時，該物件的記憶體會被解構，導致位址失效，從而引發崩潰。

為了解決這個問題，我加入了拷貝建構子和賦值建構子。拷貝建構子可以正確地創建新物件的副本，避免了原始物件記憶體被釋放後新物件的位址無效問題。

賦值建構子則解決了當多項式加法運算後，將返回結果賦值給 poly3 時可能出現的問題。若沒有賦值建構子，poly3 = poly1.Add(poly2); 這樣的賦值語句會將物件的位址複製給 poly3，使得 poly3 在函式結束後指向空空間，最終引發崩潰。經過這些調整後，問題得以解決，並且使我更加明白在 C++ 中如何正確處理物件的記憶體管理，特別是在使用返回物件的函數時。

此外，在開發過程中，我也學習了許多關於 operator 的相關知識，這讓我更加深入理解如何自定義運算符來實現多項式類型的加法運算。通過正確地實現運算符重載，我能夠使代碼更加簡潔且符合直覺，從而提高程式的可讀性與維護性。

我也加強了輸入流與輸出流的操作，特別是使用者的輸入部分。在處理使用者輸入時，我下了相當大的功夫，以提高容錯率。這樣不僅讓程式能夠更靈活地處理各種錯誤輸入，還能改善使用者體驗。例如，對於多項式的係數和指數，我設計了更為寬容的錯誤處理機制，讓使用者可以輸入不完全符合預期格式的數據，並給出清晰的提示信息，避免程式崩潰或運行錯誤。

這些經驗讓我更深入了解 C++ 程式設計中的各個方面，特別是運算符重載、輸入輸出流處理以及如何設計更好的使用者界面。這些技巧不僅幫助我解決了開發中的技術問題，也提升了我對程式設計更深的理解。