資料結構報告

梁詠琳

January 6, 2025

CONTENTS

內容

CHAPTER 1 解題說明

問題 1：開發一個 C++ 類別 Polynomial，用來表示和操作具有整數係數的單變數多項式（使用帶有頭節點的圓形鏈結串列）。多項式的每一項將以一個節點表示。因此，系統中的每個節點將包含以下三個資料成員：

Polynomial 類別代表多項式，它需要有適當的成員來儲存多項式的係數與指數。

每個多項式將被表示為一個帶有頭節點的圓形鏈結串列。為了高效地刪除多項式，我們需要使用一個可用空間列表及其相關的功能（如第4.5節所述）。單變數多項式的外部（例如輸入或輸出）表示形式假設為以下整數序列的格式：

$$n, c_1, e_1, c_2, e_2, c_3, e_3, \ldots, c_n, e_n$$

en —表示指數

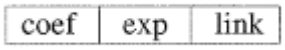
cn —表示係數

n —表示多項式的項數

$$e_1 > e_2 > \cdots > e_n$$

設計目標

1. 多項式的表示：
   a. 使用帶有頭節點的圓形鏈結串列來表示多項式，每個節點包含以下資料成員：



      i. 係數(coef):每項的係數
      ii. 指數(exp):每項的指數
      iii. 鏈接指標（link）： 指向下一個節點。
2. 支持高效操作：
   a. 多項式的刪除操作應高效，並使用可用空間列表來管理內存。
3. 輸入和輸出的數據格式：
   a. 整數序列的格式
   b. 指數e1>e2>…>en 需按降序排列

需要設計:

資料結構設計：

節點結構（Node）：包含係數、指數、和指向下一節點的指標。

多項式類別（Polynomial）：包含操作多項式的方法及鏈結串列的管理。

(e) *Polynomial*::~*Polynomial*() [Destructor]: Return all nodes of the polynomial ***this** to the available-space list.

必需功能的實現：

插入多項式項（AddTerm）：

按指數降序插入新項。

如果存在相同指數的項，合併其係數。

如果合併後係數為 0，則刪除此項。

刪除多項式（Clear）：

刪除多項式中的所有項，釋放鏈結串列所佔內存。

輸入與輸出：

輸入多項式： 重載 >> 運算子，將輸入格式轉換為鏈結串列。

(a) *istream*& **operator>>**(*istream*& *is*, *Polynomial*& *x*): Read in an input polynomial and convert it to its circular list representation using a header node.

輸出多項式： 重載 << 運算子，將鏈結串列轉換為外部表示形式。

(b) *ostream*& **operator<<**(*ostream*& *os*, *Polynomial*& *x*): Convert *x* from its linked list representation to its external representation and output it.

多項式運算：

加法（+）： 計算兩個多項式的和。

(f) *Polynomial* **operator+** (**const** *Polynomial*& *b*) **const** [Addition]: Create and return the polynomial ***this** + *b*.

減法（-）： 計算兩個多項式的差。

(g) *Polynomial* **operator**− (**const** *Polynomial*& *b*) **const** [Subtraction] : Create and return the polynomial ***this** − *b*.

乘法（*）： 計算兩個多項式的積。

(h) *Polynomial* **operator***(**const** *Polynomial*& *b*) **const** [Multiplication]: Create and return the polynomial ***this** * *b*.

多項式評估：

Evaluate(x)： 輸入變數 x 的值，計算多項式在 x 處的值。

(i) **float** *Polynomial*::*Evaluate*(**float** *x*) **const**: Evaluate the polynomial ***this** at *x* and return the result.

內存管理：

複製建構子： 複製多項式物件的內容。

解構子： 釋放多項式的內存。

賦值運算子重載： 支援多項式的賦值操作。

要完成這份作業，我們需要實現一個 Polynomial **類別**，並提供多項式的輸入、輸出和基本運算功能（如加法、減法等）。以下是詳細步驟和思路：

CHAPTER 2 演算法設計與實作

問題：表示和操作具有整數係數的單變數多項式（使用帶有頭節點的圓形鏈結串列）問題 的程式應包含：

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

// 節點結構
struct Node {
    int coef; // 系數
    int exp;  // 指數
    Node* link; // 指向下一個節點的指標
};

// 多項式類別
class Polynomial {
private:
    Node* head; // 頭節點

    // 插入多項式項（保持指數降序）
    void addTerm(int coef, int exp) {
        Node* prev = head, * curr = head->link;
        while (curr != head && curr->exp > exp) {
            prev = curr;
            curr = curr->link;
        }
        if (curr != head && curr->exp == exp) {
            curr->coef += coef; // 合併同類項
            if (curr->coef == 0) { // 刪除係數為0的項
                prev->link = curr->link;
                delete curr;
            }
```

Figure1.1 HW3_1.cpp

```
            delete curr;
        }
    }
    else { // 新增節點
        prev->link = new Node{ coef, exp, curr };
    }
}

public:
    // 預設建構子
    Polynomial() : head(new Node{ 0, 0, nullptr }) { head->link = head; }

    // 複製建構子
    Polynomial(const Polynomial& a) : Polynomial() {
        for (Node* temp = a.head->link; temp != a.head; temp = temp->link)
            addTerm(temp->coef, temp->exp);
    }

    // 解構子
    ~Polynomial() { clear(); delete head; }

    // 清除鏈結串列
    void clear() {
        Node* temp = head->link;
        while (temp != head) {
            Node* del = temp;
            temp = temp->link;
            delete del;
        }
```

Figure1.1 HW3_1.cpp

```
57          head->link = head;
58      }
59
60      // 賦值運算子重載
61      Polynomial& operator=(const Polynomial& a) {
62          if (this != &a) {
63              clear();
64              for (Node* temp = a.head->link; temp != a.head; temp = temp->link)
65                  addTerm(temp->coef, temp->exp);
66          }
67          return *this;
68      }
69
70      // 輸入運算子重載
71      friend istream& operator>>(istream& is, Polynomial& x) {
72          int n, coef, exp;
73          is >> n;
74          while (n--) {
75              is >> coef >> exp;
76              x.addTerm(coef, exp);
77          }
78          return is;
79      }
80
81      // 輸出運算子重載
82      friend ostream& operator<<(ostream& os, const Polynomial& x) {
83          for (Node* temp = x.head->link; temp != x.head; temp = temp->link) {
84              if (temp != x.head->link && temp->coef > 0) os << " + ";
85              os << temp->coef << "x^" << temp->exp;
86          }
87          return os;
88      }
89
90      // 加法運算子重載
91      Polynomial operator+(const Polynomial& b) const {
92          Polynomial result;
93          Node* p1 = head->link, * p2 = b.head->link;
94          while (p1 != head || p2 != b.head) {
95              if (p1 == head || (p2 != b.head && p2->exp > p1->exp)) {
96                  result.addTerm(p2->coef, p2->exp);
97                  p2 = p2->link;
98              }
99              else if (p2 == b.head || (p1 != head && p1->exp > p2->exp)) {
100                 result.addTerm(p1->coef, p1->exp);
101                 p1 = p1->link;
102             }
103             else {
104                 result.addTerm(p1->coef + p2->coef, p1->exp);
105                 p1 = p1->link;
106                 p2 = p2->link;
107             }
108         }
109         return result;
110     }
111
```

梁詠琳

```cpp
    // 減法運算子重載
    Polynomial operator-(const Polynomial& b) const {
        Polynomial result = *this;
        for (Node* temp = b.head->link; temp != b.head; temp = temp->link)
            result.addTerm(-temp->coef, temp->exp);
        return result;
    }

    // 乘法運算子重載
    Polynomial operator*(const Polynomial& b) const {
        Polynomial result;
        for (Node* p1 = head->link; p1 != head; p1 = p1->link)
            for (Node* p2 = b.head->link; p2 != b.head; p2 = p2->link)
                result.addTerm(p1->coef * p2->coef, p1->exp + p2->exp);
        return result;
    }

    // 評估多項式
    float Evaluate(float x) const {
        float result = 0;
        for (Node* temp = head->link; temp != head; temp = temp->link)
            result += temp->coef * pow(x, temp->exp);
        return result;
    }
};

// 主程式：測試功能
```

```cpp
int main() {
    Polynomial p1, p2;
    cout << "Enter first polynomial (n c1 e1 c2 e2 ...): ";
    cin >> p1;
    cout << "Enter second polynomial (n c1 e1 c2 e2 ...): ";
    cin >> p2;

    cout << "P1: " << p1 << endl;
    cout << "P2: " << p2 << endl;

    cout << "P1 + P2: " << (p1 + p2) << endl;
    cout << "P1 - P2: " << (p1 - p2) << endl;
    cout << "P1 * P2: " << (p1 * p2) << endl;

    float x;
    cout << "Enter a value for x to evaluate P1: ";
    cin >> x;
    cout << "P1(" << x << ") = " << p1.Evaluate(x) << endl;

    return 0;
}
```

Figure1.2 HW3_1.cpp

CHAPTER 3 效能分析

Polynomial 類別

時間複雜度我以不同功能分類直接計算。

```
// 輸入運算子重載
friend istream& operator>>(istream& is, Polynomial& x) {
    int n, coef, exp;
    is >> n;
    while (n--) {
        is >> coef >> exp;
        x.addTerm(coef, exp);
    }
    return is;
}

// 輸出運算子重載
friend ostream& operator<<(ostream& os, const Polynomial& x) {
    for (Node* temp = x.head->link; temp != x.head; temp = temp->link) {
        if (temp != x.head->link && temp->coef > 0) os << " + ";
        os << temp->coef << "x^" << temp->exp;
    }
    return os;
}
```

時間複雜度(輸入/出運算子重載):

輸入/出運算子重載：O(n)

空間複雜度(加法):

輸入/出運算子重載:O(1)

```
// 加法運算子重載
Polynomial operator+(const Polynomial& b) const {
    Polynomial result;
    Node* p1 = head->link, * p2 = b.head->link;
    while (p1 != head || p2 != b.head) {
        if (p1 == head || (p2 != b.head && p2->exp > p1->exp)) {
            result.addTerm(p2->coef, p2->exp);
            p2 = p2->link;
        }
        else if (p2 == b.head || (p1 != head && p1->exp > p2->exp)) {
            result.addTerm(p1->coef, p1->exp);
            p1 = p1->link;
        }
        else {
            result.addTerm(p1->coef + p2->coef, p1->exp);
            p1 = p1->link;
            p2 = p2->link;
        }
    }
    return result;
}
```

時間複雜度(加法):

加法：O(n+m)，其中 n 與 m 是兩個多項式的項數。

空間複雜度(加法):

加法:O(k)

```
// 減法運算子重載
Polynomial operator-(const Polynomial& b) const {
    Polynomial result = *this;
    for (Node* temp = b.head->link; temp != b.head; temp = temp->link)
        result.addTerm(-temp->coef, temp->exp);
    return result;
}
```

時間複雜度(減法):

減法：O(m(n+m))

```
// 乘法運算子重載
Polynomial operator*(const Polynomial& b) const {
    Polynomial result;
    for (Node* p1 = head->link; p1 != head; p1 = p1->link)
        for (Node* p2 = b.head->link; p2 != b.head; p2 = p2->link)
            result.addTerm(p1->coef * p2->coef, p1->exp + p2->exp);
    return result;
}
```

時間複雜度(乘法):

乘法：O(n^2*m^2)

CHAPTER 4 測試與驗證

問題 1 測試：

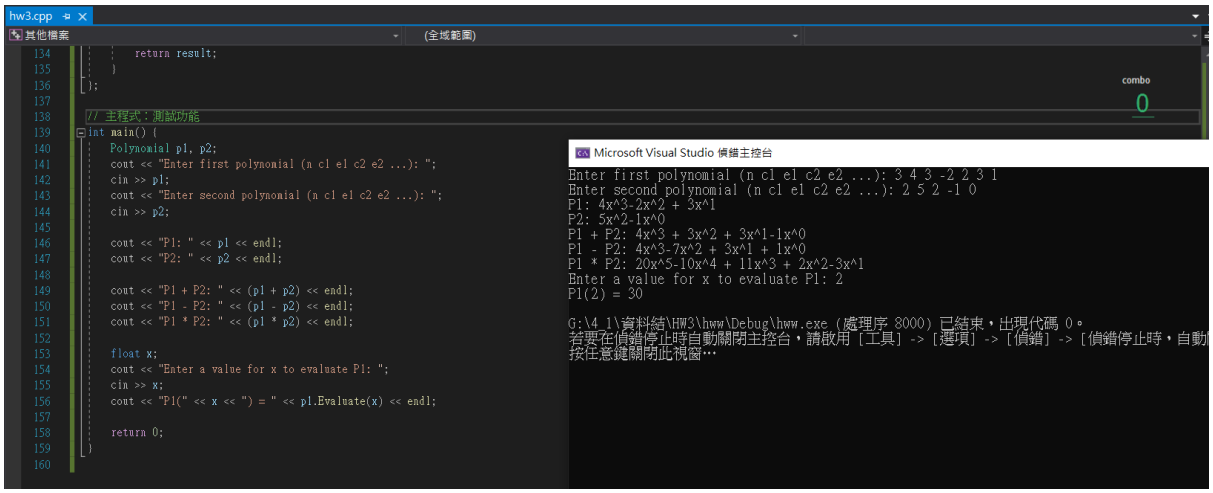


Figure4.1HW1.cpp

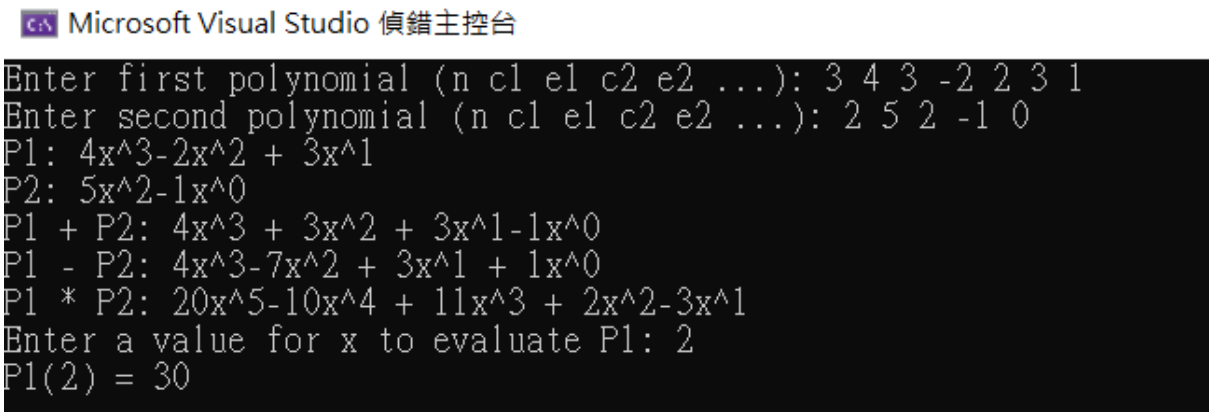


Figure4.3HW1.cpp

n=是多項式有n項

P1我的輸入是 $4x^3 - 2x^2 + 3x^1$

P2我的輸入是 $5x^2 - 1x^0$

梁詠琳

加法: $4x^3 + 3x^2 + 3x^1 - 1x^0$

$$4x^3 - 7x^2 + 3x^1 + 1x^0$$

減法:

$$20x^5 - 10x^4 + 11x^3 + 2x^2 - 3x^1$$

乘法:

最後把x，我們可以輸入一個整數代為x，我輸入的是2
=(4*8)-(2*4)+(3*2)
=32-8+6
P1(2)=30

CHAPTER 5 申論

多項式（Polynomial）作為數學中的重要概念，經常出現在計算與分析中。而在程式設計中，將多項式抽象化並加以實作，可以幫助我們理解資料結構的核心概念。本次作業以多項式的 ADT（抽象資料型態）與運算符多載為核心，實現了多項式的基本運算與評估功能，讓我在實作中收穫良多。

多項式 ADT：資料抽象的實現

ADT（抽象資料型態）是一種程式設計的思維方式，強調操作的行為而非內部細節。在這次作業中，多項式被設計為由「係數」與「指數」構成的一組節點集合，並使用 鏈結串列 進行實現。透過這種結構，我們實現了多項式的插入（addTerm）、刪除（clear）與顯示內容（operator<<）等基本操作。

鏈結串列的使用，讓多項式的項目管理更加靈活。同時，透過指數的降序排列，我們可以有效避免不必要的遍歷與冗餘操作，進一步提升程式的效能。

運算符多載：

C++ 的運算符多載是一項強大的功能，它允許我們重新定義運算符的行為，使程式操作更直觀。在這次作業中，我們通過運算符多載實現了輸入（>>）、輸出（<<）、加法（+）、減法（-）與乘法（*）的功能。

例如，輸入多項式時，使用 cin >> p1，讓程式自動處理係數與指數的輸入；而輸出時，透過 cout << p1，以數學表達形式顯示多項式。這種方式不僅貼近數學表達，還讓多項式運算的程式邏輯更符合使用者的直覺需求。加上加減乘的運算符多載，程式中的多項式操作幾乎與數學公式無異，大大提升了程式的易用性和可讀性。

實作中的挑戰與收穫

這次作業的實作過程中，我面臨了多項挑戰，但同時也有不少收穫。

挑戰：

多項式運算設計較為複雜，尤其是加法與乘法中，如何有效合併相同指數的項目並保證結果正確，花費了不少時間。

收穫： 透過助教與同學的幫助，我學會了如何設計更清晰的邏輯來實現運算功能，並進一步理解了鏈結串列在動態資料管理中的優勢。 多項式的操作不僅讓我熟悉了運算符多載的用法，也讓我更加深刻地體會到程式結構規劃的重要性。

程式設計的價值

這次作業讓我認識到程式設計的核心價值在於，將抽象的數學概念轉化為具體的程式實現，並透過靈活的資料結構設計，將複雜的運算簡化為簡單直觀的操作。多項式 ADT 和運算符多載的結合，不僅幫助我加深了對資料結構的理解。 這次作業雖然挑戰不小，但完成後收穫頗豐！

梁詠琳

CHAPTER 6心得

這次作業讓我對 **多項式 ADT** 和 **運算符多載** 的用法有了更多了解。
以前總覺得這些東西離自己很遠，但透過實作，我發現用程式把多項式的運算寫出來，其實比想像中有趣多了。把多項式拆成一項一項的節點，再用程式實現加法、乘法，還能計算值，真的蠻新奇的。

這次作業雖然跟上次的功課差不多，但蠻有成就感的。能自己寫出多項式的運算程式，甚至用運算符多載讓操作看起來像數學公式一樣，真的挺酷的。雖然過程中有點波折，但最後收穫不少，也讓我對未來的程式設計更有信心。

梁詠琳

CHAPTER 7開發報告

設計目標

實現 多項式的抽象資料型態（ADT），以便對多項式進行高效的數據管理。
支援多項式的基本運算（加法、減法、乘法）與評估功能（Eval）。
使用 C++ 的運算符多載，讓多項式的操作更加直觀，類似數學表達式。
確保程式結構清晰、邏輯完整，並便於擴展。

開發過程

1. 程式架構設計

我們設計了一個 Polynomial 類別，使用 鏈結串列結構 儲存多項式的每一項。
每一個節點包含三個屬性：
係數（coef）： 儲存多項式的係數。
指數（exp）： 儲存多項式的指數。
指標（link）： 指向下一個節點。 此外，實現了以下基礎功能：
新增項目（addTerm）： 支援按指數降序插入，並合併相同指數的項目。
顯示多項式（operator<<）： 支援多項式內容的友好輸出。

2. 運算符多載

為了讓多項式操作更加符合數學習慣，我們透過運算符多載實現了以下功能：
加法（operator+）： 計算兩個多項式的和。
減法（operator-）： 計算兩個多項式的差。
乘法（operator）：* 計算兩個多項式的積。
輸入（operator>>）： 讓多項式的輸入更加簡潔直觀。
輸出（operator<<）： 將多項式輸出為數學格式。

3. 功能擴展

在完成多項式的基本運算後，我們進一步實現了 多項式評估功能：
Eval(float x)：
計算多項式在指定變數x值下的結果。
使用迴圈計算公式result+=coef×x exp ，確保準確性與效能。

4. 問題與挑戰

-需求理解偏差：

起初，我將題目的需求拆分為「顯示功能」和「運算功能」，認為它們是分開的兩部分。 最後重新調整程式設計，將顯示、輸入與運算功能有機結合，提升了程式的可讀性與整體性。

-運算符多載的挑戰：
在實現加法與乘法時，處理鏈結串列的結構與合併邏輯存在一定複雜度。為此，我分別設計了「插入節點」與「合併相同指數項」的邏輯，確保了程式的正確性。

程式結構優化
多項式資料結構： 使用 鏈結串列結構 表示多項式，支援高效的插入與刪除操作。
多項式的操作： 透過運算符多載實現了輸入、輸出、加減乘運算，以及x的點值計算。
程式設計風格： 每個函數的功能劃分清晰，便於維護與擴展。

學習與收穫
C++ 運算符多載的實踐：
我深入學習了 C++ 運算符多載的用法，並運用於多項式運算的實現，使程式更接近數學表達式。

抽象資料型態（ADT）的理解： 多項式的操作是一個典型的 ADT 實現案例，我學會了如何在程式中將資料結構與操作相結合，提升程式的抽象性與可用性。

程式邏輯的整體性： 在助教的指導下，我重新規劃了程式結構，將多項式的顯示、運算與評估功能結合在一起，最終實現了一個邏輯完整的多項式運算系統。