資料結構報告

王語晨

DEC 28, 2024

CONTENTS

CHAPTER 1

解題說明

以建構函式、解構函式以及多項式實作，已知多項式加法、減法、乘法計算方式如下：

在定義類別時，您可以使用建構函式（Constructor）來進行物件的初始化，而在物件釋放資源之前，您也可以使用「解構函式」（Destructor）來進行一些善後的工作，例如清除動態配置的記憶體，或像是檔案的儲存、記錄檔的撰寫等等。

在C++中建構函式是與類別名稱相同的方法成員，且沒有傳回值，而解構函式則是在與類別名稱同的方法成員前加上~，不具有參數，也不具有傳回值，例如：

- SafeArray.h

```
class SafeArray {
public:
    // 建構函式
    SafeArray(int);
    // 解構函式
    ~SafeArray();

    int get(int);
    void set(int, int);

    int length;
private:
    int *_array;

    bool isSafe(int i);
};
```

多項式的加法 ［編輯］

兩個多項式相加可以看作是對兩組單項式的和進行重組與合併同類項。通過加法結合律，可以將同類項放在一起，合併之後就得到了兩個多項式的和[1][2]。例如以下的兩個多項式：

$$P = 3X^2 - 2X + 5XY - 2$$
$$Q = -3X^2 + 3X + 4Y^2 + 8$$

它們的和是：

$$P + Q = (3X^2 - 2X + 5XY - 2) + (-3X^2 + 3X + 4Y^2 + 8)$$

化簡之後得到：

$$P + Q = X + 5XY + 4Y^2 + 6$$

多項式的減法 ［編輯］

例：$P = 36x^5 + 7x^4 + 66x^3 + 36x^2 + 66x + 6$ 、 $Q = 5x^5 - 73x^4 - 11x^3 - 11x^2 + 5x + 3$ 則

$$P - Q = (36 - 5)x^5 + (7 + 73)x^4 + (66 + 11)x^3 + (36 + 11)x^2 + (66 - 5)x + (6 - 3) = 31x^5 + 80x^4 + 77x^3 + 47x^2 + 61x + 3$$

多項式乘法 ［編輯］

例如以下的兩個多項式：

$$P = 2X + 3Y + 5$$
$$Q = 2X + 5Y + XY + 1$$

計算它們的乘積，步驟如下：

$$\begin{aligned} PQ = {} & (2X \cdot 2X) + (2X \cdot 5Y) + (2X \cdot XY) + (2X \cdot 1) \\ & + (3Y \cdot 2X) + (3Y \cdot 5Y) + (3Y \cdot XY) + (3Y \cdot 1) \\ & + (5 \cdot 2X) + (5 \cdot 5Y) + (5 \cdot XY) + (5 \cdot 1) \end{aligned}$$

化簡之後得到：

$$PQ = 4X^2 + 21XY + 2X^2Y + 12X + 15Y^2 + 3XY^2 + 28Y + 5$$

實作參見檔案 polynomial.cpp，其 polynomial 函式：

```
// Polynomial 類別：用來表示與操作多項式
class Polynomial {
private:
    // 節點結構：表示多項式中的一個項
    struct Node {
        int coef;   // 項的係數 (coefficient)
        int exp;    // 項的指數 (exponent)
        Node* link; // 指向下一個節點的指標
    };
    Node* head; // 頭節點，用於形成循環鏈結串列
public:
    // 預設建構函式：初始化空的循環鏈結串列
    Polynomial() {
        head = new Node{0, 0, nullptr}; // 建立一個虛擬節點
        head->link = head; // 將虛擬節點連結回自身，形成循環
    }
    // 解構函式：釋放動態記憶體，避免記憶體洩漏
    ~Polynomial() {
        clear();      // 清空鏈結串列
        delete head;  // 刪除頭節點
    }
    // 清空多項式：刪除所有節點，保留頭節點
    void clear() {
        Node* current = head->link;
        while (current != head) { // 遍歷所有節點
            Node* temp = current; // 暫存當前節點
            current = current->link; // 移動到下一個節點
            delete temp; // 釋放記憶體
        }
        head->link = head; // 恢復循環鏈結串列
    }

    // 複製建構函式：用於複製另一個 Polynomial 的內容
    Polynomial(const Polynomial& other) {
        head = new Node{0, 0, nullptr};
        head->link = head;
        Node* current = other.head->link;
        while (current != other.head) { // 遍歷另一個多項式的節點
            addTerm(current->coef, current->exp); // 添加到當前多項式
            current = current->link;
        }
    }
```

Figure 1.1: polynomial.c

```
// Polynomial 類別：用來表示與操作多項式
class Polynomial {
private:
    // 節點結構：表示多項式中的一個項
    struct Node {
        int coef;   // 項的係數 (coefficient)
        int exp;    // 項的指數 (exponent)
        Node* link; // 指向下一個節點的指標
    };
    Node* head; // 頭節點，用於形成循環鏈結串列

public:
    // 預設建構函式：初始化空的循環鏈結串列
    Polynomial() {
        head = new Node{0, 0, nullptr}; // 建立一個虛擬節點
        head->link = head; // 將虛擬節點連結回自身，形成循環
    }

    // 解構函式：釋放動態記憶體，避免記憶體洩漏
    ~Polynomial() {
        clear();      // 清空鏈結串列
        delete head;  // 刪除頭節點
    }

    // 清空多項式：刪除所有節點，保留頭節點
    void clear() {
        Node* current = head->link;
        while (current != head) { // 遍歷所有節點
            Node* temp = current; // 暫存當前節點
            current = current->link; // 移動到下一個節點
            delete temp; // 釋放記憶體
        }
        head->link = head; // 恢復循環鏈結串列
    }

    // 複製建構函式：用於複製另一個 Polynomial 的內容
    Polynomial(const Polynomial& other) {
        head = new Node{0, 0, nullptr};
        head->link = head;
        Node* current = other.head->link;
        while (current != other.head) { // 遍歷另一個多項式的節點
            addTerm(current->coef, current->exp); // 添加到當前多項式
            current = current->link;
        }
    }
```

CHAPTER 2

程式實作

```
// 主程式：執行多項式操作
int main() {
    Polynomial p1, p2;
    // 輸入多項式
    cout << "請輸入第一個多項式 (格式: n coef1 exp1 coef2 exp2 ...): ";
    cin >> p1;
    cout << "請輸入第二個多項式 (格式: n coef1 exp1 coef2 exp2 ...): ";
    cin >> p2;
    // 輸出多項式
    cout << "P1: " << p1 << endl;
    cout << "P2: " << p2 << endl;
    // 多項式加法
    Polynomial sum = p1 + p2;
    cout << "P1 + P2: " << sum << endl;
    // 多項式減法
    Polynomial diff = p1 - p2;
    cout << "P1 - P2: " << diff << endl;
    // 多項式乘法
    Polynomial prod = p1 * p2;
    cout << "P1 * P2: " << prod << endl;
    // 評估多項式值
    float x;
    cout << "請輸入 x 的值來評估 P1(x): ";
    cin >> x;
    cout << "P1(" << x << "): " << p1.Evaluate(x) << endl;
    cout << "請輸入 x 的值來評估 P2(x): ";
    cin >> x;
    cout << "P2(" << x << "): " << p2.Evaluate(x) << endl;
    return 0;
}
```

Figure 2.1: main.cpp

CHAPTER 3

效能分析

$$f(n) = O(n)$$

時間複雜度

1. 插入多項式項 (addTerm)：最壞情況，鏈結串列的長度為 n，需遍歷 n 個節點，因此時間複雜度為 O(n)。 而最佳情況，插入位置為第一個節點，時間複雜度為 O(1)。
2. 多項式加法 (operator+)與減法 (operator-)：假設多項式 p1 和 p2 分別有 n 和 m 項，則需要合併 n + m 項。時間複雜度為 O(n + m)。
3. 多項式乘法(operator*)： 乘法需進行所有項的交叉相乘。對於 p1 的每一項，需遍歷 p2 的每一項。時間複雜度 O(n * m)，其中 n 和 m 分別是 p1 和 p2 的項數。
4. 多項式評估 (Evaluate): 遍歷多項式的所有項，對每一項計算 coef * x^{exp}。時間複雜度為 O(n)，其中 n 是多項式的項數。
5. 輸入 (operator>>)與輸出(operator<<)：輸入需插入 n 項，多項式的每一項需要 O(n)的插入時間，因此總時間複雜度為 O(n^2)（最壞情況）。 輸出需遍歷所有項並格式化輸出，時間複雜度為 O(n)。

總結：

操作	**時間複雜度**
插入多項式項	**O(n)**
加法	**O(n + m)**
減法	**O(n + m)**
乘法	**O(n * m)**
評估多項式	**O(n)**
輸入	**O(n^2)**
輸出	**O(n)**

空間複雜度

1. 插入多項式項 (addTerm)：每個多項式的項用一個節點表示，每個節點佔用固定的空間：節點大小為：coef (int) + exp (int) + link (Node*)，因此每個節點佔用 O(1)。
2. 多項式加法 (operator+)與減法 (operator-)：產生的新多項式需要額外存儲 n + m 項，因此空間複雜度為 O(n + m)。
3. 多項式乘法(operator*)：假設 p1 有 n 項， p2 有 m 項，則結果多項式最多有 n * m 項。空間複雜度為 O(n * m)。
4. 多項式評估 (Evaluate): 僅使用固定的局部變數和指標，無需新增記憶體。空間複雜度為 O(1)。
5. 輸入 (operator>>)與輸出(operator<<)：輸入每次新增項目都需要為該項分配一個節點，因此空間複雜度與多項式的項數 n 成正比。空間複雜度為 O(n)。輸出僅使用固定的指標來遍歷多項式，無需額外的空間。空間複雜度為 O(1)。

總結：

操作	**空間複雜度**
插入多項式項	**O(1)**
加法	**O(n + m)**
減法	**O(n + m)**
乘法	**O(n * m)**
評估多項式	**O(1)**
輸入	**O(n)**
輸出	**O(1)**

CHAPTER 4 測試與驗證

```
// Polynomial 類別：用來表示與操作多項式
class Polynomial {
private:
    // 節點結構：表示多項式中的一個項
    struct Node {
        int coef;   // 項的係數 (coefficient)
        int exp;    // 項的指數 (exponent)
        Node* link; // 指向下一個節點的指標
    };
    Node* head; // 頭節點，用於形成循環鏈結串列
public:
    // 預設建構函式：初始化空的循環鏈結串列
    Polynomial() {
        head = new Node{0, 0, nullptr}; // 建立一個虛擬節點
        head->link = head; // 將虛擬節點連結回自身，形成循環
    }
    // 解構函式：釋放動態記憶體，避免記憶體洩漏
    ~Polynomial() {
        clear();      // 清空鏈結串列
        delete head;  // 刪除頭節點
    }
// 清空多項式：刪除所有節點，保留頭節點
    void clear() {
        Node* current = head->link;
        while (current != head) { // 遍歷所有節點
            Node* temp = current; // 暫存當前節點
            current = current->link; // 移動到下一個節點
            delete temp; // 釋放記憶體
        }
        head->link = head; // 恢復循環鏈結串列
    }
// 複製建構函式：用於複製另一個 Polynomial 的內容
    Polynomial(const Polynomial& other) {
        head = new Node{0, 0, nullptr};
        head->link = head;
        Node* current = other.head->link;
        while (current != other.head) { // 遍歷另一個多項式的節點
            addTerm(current->coef, current->exp); // 添加到當前多項式
            current = current->link;
        }
    }

```

Figure4.1: shellcommand

CHAPTER 4

測試與驗證

```
// 賦值運算子：將另一個多項式的內容複製到當前多項式
    Polynomial& operator=(const Polynomial& other) {
        if (this != &other) { // 避免自我賦值
            clear(); // 清空當前多項式
            Node* current = other.head->link;
            while (current != other.head) { // 遍歷另一個多項式的節點
                addTerm(current->coef, current->exp); // 添加到當前多項式
                current = current->link;
            }
        }
        return *this;
    }
    // 添加一個項到多項式
    void addTerm(int coef, int exp) {
        if (coef == 0) return; // 係數為 0 的項忽略
        Node* prev = head;
        Node* current = head->link;
// 找到插入位置：確保指數按降序排列
        while (current != head && current->exp > exp) {
            prev = current;
            current = current->link;
        }

        if (current != head && current->exp == exp) {
            // 若已有相同指數的項，則合併係數
            current->coef += coef;
            if (current->coef == 0) { // 若係數加總後為 0，刪除該項
                prev->link = current->link;
                delete current;
            }
        } else {
            // 否則，插入新項
            Node* newNode = new Node{coef, exp, current};
            prev->link = newNode;
        }
    }
```

Figure4.1: shellcommand

CHAPTER 4

測試與驗證

```
// 評估多項式值：計算 P(x)
    float Evaluate(float x) const {
        float result = 0;
        Node* current = head->link;
        while (current != head) { // 遍歷所有節點
            result += current->coef * pow(x, current->exp); // coef * x^exp
            current = current->link;
        }
        return result;
    }
// 重載輸入運算子：從輸入流讀取多項式
    friend istream& operator>>(istream& is, Polynomial& x) {
        int n;
        is >> n; // 讀取多項式項數
        for (int i = 0; i < n; ++i) {
            int coef, exp;
            is >> coef >> exp; // 讀取每一項的係數與指數
            x.addTerm(coef, exp); // 添加該項
        }
        return is;
    }
// 重載輸出運算子：將多項式格式化輸出
    friend ostream& operator<<(ostream& os, const Polynomial& x) {
        Node* current = x.head->link;
        bool first = true; // 控制是否添加 "+" 號
        while (current != x.head) {
            if (!first && current->coef > 0) os << " + ";
            if (current->coef < 0) os << " - ";
            os << abs(current->coef);
            if (current->exp != 0) os << "x^" << current->exp;
            first = false;
            current = current->link;
        }
        return os;
    }
```

Figure 4.1: shell command

```
// 重載加法運算子
    Polynomial operator+(const Polynomial& b) const {
        Polynomial result;
        Node* p1 = head->link;
        Node* p2 = b.head->link;
        // 合併兩個多項式
        while (p1 != head || p2 != b.head) {
            if (p2 == b.head || (p1 != head && p1->exp > p2->exp)) {
                result.addTerm(p1->coef, p1->exp);
                p1 = p1->link;
            } else if (p1 == head || (p2 != b.head && p2->exp > p1->exp)) {
                result.addTerm(p2->coef, p2->exp);
                p2 = p2->link;
            } else {
                result.addTerm(p1->coef + p2->coef, p1->exp);
                p1 = p1->link;
                p2 = p2->link;
            }
        }
        return result;
    }
    // 重載減法運算子
    Polynomial operator-(const Polynomial& b) const {
        Polynomial result;
        Node* p2 = b.head->link;
        while (p2 != b.head) {
            result.addTerm(-p2->coef, p2->exp); // 取反後添加
            p2 = p2->link;
        }
        return *this + result;
    }
// 重載乘法運算子
    Polynomial operator*(const Polynomial& b) const {
        Polynomial result;
        Node* p1 = head->link;
        while (p1 != head) {
            Node* p2 = b.head->link;
            while (p2 != b.head) {
                result.addTerm(p1->coef * p2->coef, p1->exp + p2->exp);
                p2 = p2->link;
            }
            p1 = p1->link;
        }
```

Figure 4.1: shell command

請輸入第一個多項式 (格式: n coef1 exp1 coef2 exp2 ...): 3 4 3-2 2 5 0

請輸入第二個多項式 (格式: n coef1 exp1 coef2 exp2 ...): 2 3 2 -1 0

P1: 4x^3 - 2x^2 + 5

P2: 3x^2 - 1

P1 + P2: 4x^3 + 1x^2 + 4

P1 - P2: 4x^3 - 5x^2 + 6

P1 * P2: 12x^5 - 6x^4 - 4x^3 + 17x^2 - 5

請輸入 x 的值來評估 P1(x): 2

P1(2): 29

請輸入 x 的值來評估 P2(x): 2

P2(2): 11

...Program finished with exit code 0

Press ENTER to exit console.

驗證

在主函式 main（）中，設計了多項式的輸入、運算與評估功能。多項式輸入部分，先輸入多項式的項數，再輸入第一個多項式 P1，最後輸入第二個多項式 P2，程式將會執行多項式加法、減法以及乘法的運算，然後會輸出多項式加法、減法及乘法結果，而最後的評估部分需要輸入 x 值，以計算多項式的值。根據多項式函數的定義，指數相同則係數做相加或相減，因此輸入第一個多項式 P1 為 3 項，係數 4 指數 3 和係數-2 指數 2 以及 係數 5，再輸入第二個多項式 P2 為 2 項，係數 3 指數 2 和係數-1，輸出相加結果為係數 4 指數 3 加係數-2 指數 2 加係數 5，相減結果為係數 4 指數減係數 5 指數 2 加 6，而相乘則是每個項相乘，並合併相同指數項，結果為係數 12 指數 5 減係數 8 指數 4 加係數 13 指數 3 減係數-2 指數 2 減係數 5，最後多項式評估值輸入 2 則第一個多項式 P1 值為 29，第二個多項式 P2 值為 11。

CHAPTER 5

申論與開發報告

申論

這題要求實現一個以循環鏈結串列為基礎的多項式類別 Polynomial 對多項式進行基本的數學運算（加法、減法、乘法）與評估值的計算。程式中將多項式以節點構成的循環鏈結串列儲存，每個節點包含三個欄位（係數（coef）、指數（exp）、以及指向下一個節點的指標（link））。

主要功能的設計，運算符重載部分，實現＋、-、*等操作符，支援多項式間的加法、減法和乘法運算，通過遍歷鏈結串列合併同次項，而輸入與輸出部分，使用重載的>>與<<運算符，提供用戶多項式輸入與輸出格式，方便進行測試與調試，最後是評估部分，通過 Evaluate 函數計算多項式在特定點的值，利用鏈結串列逐項計算並累加結果。

開發報告

根據題目了解到本題主要在設計一個用於處理多項式運算的程式，實現對多項式的基本操作，包括輸入、輸出、加法、減法、乘法以及在特定值下的評估功能。程式的核心以鏈結串列實現，每個節點表示多項式的一個項，能動態處理多項式的插入與操作。基本操作包含了多項式的輸入與輸出，支援多項式項次與係數的動態輸入，還支援多項式的加法、減法、乘法運算，最後還提供在特定值 x 下計算多項式值的功能。然而在結構上，選擇使用單向鏈結串列表示多項式，其中每個節點包含「係數」和「指數」兩個欄位，並將節點將指數以降序排列，以便於計算時快速合併同次方項。