

# 資料結構報告

李柏勳

August 6, 2024

# CONTENT

CHAPTER 1	解題說明.....	3
CHAPTER 2	演算法設計與實作.....	8
CHAPTER 3	效能分析.....	16
CHAPTER 4	測試與驗證.....	17
CHAPTER 5	效能量測.....	18
<u>CHAPTER 6</u>	心得討論.....	19

## CHAPTER 1 解題說明


(一)做出一個多項式加法跟乘法跟 Eval 驗證

想法陳述：

1. **加法的敘述**：為了確保敘述流暢，加法、乘法、以及 Eval 的敘述及虛擬碼是使用 ChatGPT 所撰寫。

加法的核心是將兩個多項式相加。首先，初始化結果多項式的儲存空間，並使用指針遍歷兩個多項式的非零項。比較兩個多項式的指數，若相等則將係數相加，否則將較大指數的項目加入結果。當其中一個多項式遍歷完畢後，將另一個多項式剩餘的項目加入結果。最後，將結果多項式按指數降序排列並返回。這個方法確保多項式相加後的結果正確且有序。

```
函數 Add(poly):  
    初始化 result 為大小為 max(terms, poly.terms) 的空間  
    i, j, k 設為 0  
  
    當 i < terms 且 j < poly.terms 時:  
        如果 termArray[i].exp == poly.termArray[j].exp:  
            將 termArray[i].coef 與 poly.termArray[j].coef 相加為 coefSum  
            如果 coefSum 不等於 0:  
                將 coefSum 和 termArray[i].exp 設為新項目 result[k]  
                k 增加 1  
            i 增加 1  
            j 增加 1  
        否則如果 termArray[i].exp > poly.termArray[j].exp:  
            將 termArray[i] 放入 result[k]  
            i 和 k 增加 1  
        否則:  
            將 poly.termArray[j] 放入 result[k]  
            j 和 k 增加 1  
  
    當 i < terms 時:  
        將 termArray[i] 放入 result[k]  
        i 和 k 增加 1  
  
    當 j < poly.terms 時:  
        將 poly.termArray[j] 放入 result[k]  
        j 和 k 增加 1  
  
    設 result.terms 為 k  
  
    將 result 按項次降序排列  
  
    返回 result
```



2. **乘法的敘述：** Mult 函數的核心在於逐項相乘兩個多項式。透過雙重迴圈，將第一個多項式的每一項與第二個多項式的每一項進行相乘，計算出新的係數和次方。若結果多項式中已經存在相同次方的項目，則將係數相加；若無則新增該項。這樣的設計有效地避免了次方重複計算，並且在必要時擴充儲存空間，確保結果多項式的正確性和完整性。

```
初始化結果多項式
對每一項 termArray[i]:
    對每一項 poly.termArray[j]:
        計算係數相乘
        計算次方相加
        如果結果中有相同次方:
            將結果的係數相加
        如果沒有相同次方:
            如果結果空間不足:
                擴充結果空間
            將新的項目加入結果
返回結果多項式
```

3. **Eval 的敘述**：Eval 函數的核心在於計算多項式在特定  $x$  值下的結果。它透過單一迴圈，對每一項進行計算，即將該項的係數乘上  $x$  的次方，並將這些結果累加，最終得到多項式在該點的數值。這樣的逐項計算方法確保了結果的精確度，適合用來評估任意  $x$  值下的多項式結果。

```
初始化結果為0
對每一項 termArray[i]:
    計算此項對應x的值
    將此項值累加到結果
返回結果
```

實作參見檔案 poly.cpp :

```
7  class Polynomial;
8
9  class Term {
10     friend Polynomial;
11     friend ostream& operator<<(ostream& os, const Polynomial& poly);
12     friend istream& operator>>(istream& is, Polynomial& poly);
13 public:
14     Term(float c = 0, int e = 0) : coef(c), exp(e) {}
15 private:
16     float coef;
17     int exp;
18 };
19
20 class Polynomial {
21 public:
22     Polynomial();
23     ~Polynomial();
24     Polynomial Add(const Polynomial& poly) const;
25     Polynomial Mult(const Polynomial& poly) const;
26     float Eval(float f) const;
27     friend ostream& operator<<(ostream& os, const Polynomial& poly);
28     friend istream& operator>>(istream& is, Polynomial& poly);
29 private:
30     Term *termArray;
31     int capacity;
32     int terms;
33     void resize(int newCapacity);
34 };
```

## CHAPTER 2 演算法設計與實作

1. 由於對 C++ 不夠熟悉，因此使用 ChatGPT 得出答案，一開始的答案為

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

class Polynomial;

class Term {
    friend class Polynomial;
private:
    float coef; // Coefficient
    int exp; // Exponent
};

class Polynomial {
public:
    Polynomial(); // Construct the polynomial p(x) = 0
    Polynomial Add(const Polynomial& poly) const; // Return the sum of the polynomials *this and poly
    Polynomial Mult(const Polynomial& poly) const; // Return the product of the polynomials *this and poly
    float Eval(float f) const; // Evaluate the polynomial *this at f and return the result
    friend ostream& operator<<(ostream& os, const Polynomial& poly); // Output polynomial
    friend istream& operator>>(istream& is, Polynomial& poly); // Input polynomial
private:
    vector<Term> termArray; // Vector of nonzero terms
    int capacity; // Size of termArray
    int terms; // Number of nonzero terms
};

Polynomial::Polynomial() {
    capacity = 1;
    terms = 0;
}

Polynomial Polynomial::Add(const Polynomial& poly) const {
    Polynomial result;
    int i = 0, j = 0;

    while (i < terms && j < poly.terms) {
        if (termArray[i].exp == poly.termArray[j].exp) {
            float coefSum = termArray[i].coef + poly.termArray[j].coef;
            if (coefSum != 0) {
                result.termArray.push_back(Term{coefSum, termArray[i].exp});
            }
        }
        else if (termArray[i].exp > poly.termArray[j].exp) {
            result.termArray.push_back(termArray[i]);
            i++;
        }
        else {
            result.termArray.push_back(poly.termArray[j]);
            j++;
        }
    }

    while (i < terms) {
        result.termArray.push_back(termArray[i]);
        i++;
    }

    while (j < poly.terms) {
        result.termArray.push_back(poly.termArray[j]);
        j++;
    }

    result.terms = result.termArray.size();
    return result;
}
```



```
if (termArray[i].exp == poly.termArray[j].exp) {
    float coefSum = termArray[i].coef + poly.termArray[j].coef;
    if (coefSum != 0) {
        Term term = { coefSum, termArray[i].exp };
        result.termArray.push_back(term);
    }
    i++;
    j++;
} else if (termArray[i].exp > poly.termArray[j].exp) {
    result.termArray.push_back(termArray[i]);
    i++;
} else {
    result.termArray.push_back(poly.termArray[j]);
    j++;
}
}

while (i < terms) {
    result.termArray.push_back(termArray[i]);
    i++;
}

while (j < poly.terms) {
    result.termArray.push_back(poly.termArray[j]);
    j++;
}

result.terms = result.termArray.size();
return result;
}

Polynomial Polynomial::Mult(const Polynomial& poly) const {
    Polynomial result;
    for (int i = 0; i < terms; i++) {
        for (int j = 0; j < poly.terms; j++) {
            float coefProduct = termArray[i].coef * poly.termArray[j].coef;
            int expSum = termArray[i].exp + poly.termArray[j].exp;
            bool termFound = false;

            for (int k = 0; k < result.terms; k++) {
                if (result.termArray[k].exp == expSum) {
                    result.termArray[k].coef += coefProduct;
                }
            }
        }
    }
    result.terms = result.termArray.size();
    return result;
}
```

```
        for (int k = 0; k < result.terms; k++) {
            if (result.termArray[k].exp == expSum) {
                result.termArray[k].coef += coefProduct;
                termFound = true;
                break;
            }
        }

        if (!termFound) {
            Term term = { coefProduct, expSum };
            result.termArray.push_back(term);
        }
    }
}

result.terms = result.termArray.size();
return result;
}

float Polynomial::Eval(float f) const {
    float result = 0.0;
    for (int i = 0; i < terms; i++) {
        result += termArray[i].coef * pow(f, termArray[i].exp);
    }
    return result;
}

ostream& operator<<(ostream& os, const Polynomial& poly) {
    for (int i = 0; i < poly.terms; i++) {
        os << poly.termArray[i].coef << "x^" << poly.termArray[i].exp;
        if (i != poly.terms - 1) {
            os << " + ";
        }
    }
    return os;
}

istream& operator>>(istream& is, Polynomial& poly) {
    cout << "Enter number of terms: ";
    is >> poly.terms;
    poly.termArray.resize(poly.terms);
```

```
poly.termArray.resize(poly.terms);
for (int i = 0; i < poly.terms; i++) {
    cout << "Enter coefficient and exponent: ";
    is >> poly.termArray[i].coef >> poly.termArray[i].exp;
}
return is;
}

int main() {
    Polynomial p1, p2;
    cout << "Enter polynomial 1:\n";
    cin >> p1;
    cout << "Enter polynomial 2:\n";
    cin >> p2;
    Polynomial sum = p1.Add(p2);
    Polynomial product = p1.Mult(p2);
    cout << "Sum: " << sum << endl;
    cout << "Product: " << product << endl;
    return 0;
}
```

再來他所寫的 ostream 跟 istream 都無法通過編譯，因為裡面有使用到類別 term 的 coef 跟 exp，所以重新在 term 裡面把他們加入 friend，讓他們可以使用 coef 跟 exp。並且同時有使用到 pow 函數，故在開頭加上 #include <cmath>。

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

```
class Term {
    friend class Polynomial;
    friend ostream& operator<<(ostream& os, const Polynomial& poly);
    friend istream& operator>>(istream& is, Polynomial& poly);
```

接著計算出來的結果次方數並沒有照順序排，因此我讓 ChatGPT 分別在 Add 方法跟 Mult 方法裡加上了排序。

```
73 // 74到76行是讓加完的多項式照項次排列
74 sort(result.termArray.begin(), result.termArray.end(), [](const Term& a, const Term& b) {
75     return a.exp > b.exp;
76 });

105 // 105到108行是讓乘完的多項式照項次排列
106 sort(result.termArray.begin(), result.termArray.end(), [](const Term& a, const Term& b) {
107     return a.exp > b.exp;
108 });
```

再來我發現到他並沒有使用 capacity 去做空間配置，所以重新加上後的最終答案為：

```
2  #include <iostream>
3  #include <cmath>
4  #include <algorithm>
5  using namespace std;
6
7  class Polynomial;
8
9  class Term {
10     friend Polynomial;
11     friend ostream& operator<<(ostream& os, const Polynomial& poly);
12     friend istream& operator>>(istream& is, Polynomial& poly);
13 public:
14     Term(float c = 0, int e = 0) : coef(c), exp(e) {}
15 private:
16     float coef;
17     int exp;
18 };
19
20 class Polynomial {
21 public:
22     Polynomial();
23     ~Polynomial();
24     Polynomial Add(const Polynomial& poly) const;
25     Polynomial Mult(const Polynomial& poly) const;
26     float Eval(float f) const;
27     friend ostream& operator<<(ostream& os, const Polynomial& poly);
28     friend istream& operator>>(istream& is, Polynomial& poly);
29 private:
30     Term *termArray;
```

```
30     Term *termArray;
31     int capacity;
32     int terms;
33     void resize(int newCapacity);
34 };
35
36 Polynomial::Polynomial() {
37     capacity = 1;
38     terms = 0;
39     termArray = new Term[capacity];
40 }
41
42 Polynomial::~~Polynomial() {
43     delete[] termArray;
44 }
45
46 void Polynomial::resize(int newCapacity) {
47     if (newCapacity <= capacity) return;
48     Term* newArray = new Term[newCapacity];
49     for (int i = 0; i < terms; i++) {
50         newArray[i] = termArray[i];
51     }
52     delete[] termArray;
53     termArray = newArray;
54     capacity = newCapacity;
55 }
56
57 Polynomial Polynomial::Add(const Polynomial& poly) const {
58     Polynomial result;
59     result.resize(max(terms, poly.terms));
60
61     while (i < terms && j < poly.terms) {
62         if (termArray[i].exp == poly.termArray[j].exp) {
63             float coefSum = termArray[i].coef + poly.termArray[j].coef;
64             if (coefSum != 0) {
65                 result.termArray[k++] = Term(coefSum, termArray[i].exp);
66             }
67             i++;
68             j++;
69         } else if (termArray[i].exp > poly.termArray[j].exp) {
70             result.termArray[k++] = termArray[i++];
71         } else {
72             result.termArray[k++] = poly.termArray[j++];
73         }
74     }
75
76     // 將剩下的termArray[i]跟poly.termArray[j]放進termArray[k]
77     while (i < terms) {
78         result.termArray[k++] = termArray[i++];
79     }
80
81     while (j < poly.terms) {
82         result.termArray[k++] = poly.termArray[j++];
83     }
84 }
85
```

```
86     result.terms = k; // 將termArray[k]放進result.terms
87     // 88到90行是讓加完的多項式照項次排列
88     sort(result.termArray, result.termArray + result.terms, [](const Term& a, const Term& b) {
89         return a.exp > b.exp;
90     });
91
92     return result;
93 }
94
95 Polynomial Polynomial::Mult(const Polynomial& poly) const {
96     Polynomial result;
97     result.resize(terms * poly.terms); // 將儲存空間resize成terms * poly.terms
98     for (int i = 0; i < terms; i++) {
99         for (int j = 0; j < poly.terms; j++) {
100             float coefProduct = termArray[i].coef * poly.termArray[j].coef;
101             int expSum = termArray[i].exp + poly.termArray[j].exp;
102             bool termFound = false;
103
104             for (int k = 0; k < result.terms; k++) {
105                 if (result.termArray[k].exp == expSum) {
106                     result.termArray[k].coef += coefProduct;
107                     termFound = true;
108                     break;
109                 }
110             }
111
112             if (!termFound) {
113                 if (result.terms == result.capacity) {
114                     result.resize(result.capacity * 2);
115                 }
116                 result.termArray[result.terms++] = Term(coefProduct, expSum);
117             }
118         }
119     }
120     return result;
121 }
122
123 // 輸入值給x，將答案算出來
124 float Polynomial::Eval(float f) const {
125     float result = 0.0;
126     for (int i = 0; i < terms; i++) {
127         result += termArray[i].coef * pow(f, termArray[i].exp);
128     }
129     return result;
130 }
131
132 // 將poly.termArray的值回傳給os
133 ostream& operator<<(ostream& os, const Polynomial& poly) {
134     for (int i = 0; i < poly.terms; i++) {
135         os << poly.termArray[i].coef << "x^" << poly.termArray[i].exp;
136         if (i != poly.terms - 1) {
137             os << " + ";
138         }
139     }
140 }
```

```
137         os << " + ";
138     }
139 }
140 return os;
141 }
142
143 // 接受input並將值傳給ostream
144 istream& operator>>(istream& is, Polynomial& poly) {
145     cout << "Enter number of terms: ";
146     is >> poly.terms;
147     if (poly.terms > poly.capacity) {
148         poly.resize(poly.terms); // 如果儲存空間不足
149     }
150     for (int i = 0; i < poly.terms; i++) {
151         cout << "Enter coefficient and exponent: ";
152         is >> poly.termArray[i].coef >> poly.termArray[i].exp; // 將輸入的值依序
153     }
154
155     // 156到158行是將乘完的多項式照項次排列
156     sort(poly.termArray, poly.termArray + poly.terms, [](const Term& a, const Term& b) {
157         return a.exp > b.exp;
158     });
159
160     return is;
161 }
```

```
163 int main() {
164     Polynomial p1, p2, p3; // 在類別Polynomial創建p1跟p2,
165     cout << "Enter polynomial 1:\n";
166     cin >> p1;
167     cout << "Enter polynomial 2:\n";
168     cin >> p2;
169     Polynomial sum = p1.Add(p2); // 呼叫類別Polynomial的Add方法，讓p
170     Polynomial product = p1.Mult(p2); // 呼叫類別Polynomial的Mult方法，讓
171     cout << "Sum: " << sum << endl;
172     cout << "Product: " << product << endl;
173     float evalPoint;
174     cout << "Enter the value to evaluate the polynomials: ";
175     cin >> evalPoint;
176     cout << "p1(" << evalPoint << ") = " << p1.Eval(evalPoint) << endl;
177     cout << "p2(" << evalPoint << ") = " << p2.Eval(evalPoint) << endl;
178     cout << "Sum(" << evalPoint << ") = " << sum.Eval(evalPoint) << endl;
179     cout << "Product(" << evalPoint << ") = " << product.Eval(evalPoint) << endl;
180     return 0;
181 }
```

## CHAPTER 3 效能分析

### (一)

#### 時間複雜度

1. Add 的時間複雜度為  $\text{bigO}(m+n+k \log k)$ ， $m$  是 terms 的長度， $n$  是 poly. terms 的長度， $k$  是多項式的項數。
2. Mult 的時間複雜度為  $\text{bigO}(m*n)$ ， $m$  是 terms 的長度， $n$  是 poly. terms 的長度。
3. Eval 的時間複雜度為  $\text{bigO}(m)$ ， $m$  是 terms 的長度。

### (二)空間複雜度

1. Add 的空間複雜度為  $\text{bigO}(m+n)$ ， $m$  是 terms 的長度， $n$  是 poly. terms 的長度。
2. Mult 的空間複雜度為  $\text{bigO}(m*n)$ ， $m$  是 terms 的長度， $n$  是 poly. terms 的長度。
3. Eval 的空間複雜度為  $\text{bigO}(1)$ ， $f$  一個變數



## CHAPTER 4 測試與驗證

### 驗證

1. 驗證加法(Add) :  $(2x^4 + 4x^2) + (3x^5 + 5x^3)$  經排序  
過後 =  $3x^5 + 2x^4 + 5x^3 + 4x^2$ 。

2. 驗證乘法(Mult) :  $(2x^4 + 4x^2) * (3x^5 + 5x^3) = (6x^9 + 10x^7 + 12x^7 + 20x^5)$  經排序過後 =  $(6x^9 + 22x^7 + 20x^5)$ 。

3. 驗證數值(Eval) : 數值皆代 1 ->

$$p1 = (2x^4 + 4x^2) = 6$$

$$p2 = (3x^5 + 5x^3) = 8$$

$$\text{sum} = (3x^5 + 2x^4 + 5x^3 + 4x^2) = 14$$

$$\text{product} = (6x^9 + 22x^7 + 20x^5) = 48$$

```
PS C:\Users\user\Desktop\school\c++> .\3.exe
Enter polynomial 1:
Enter number of terms: 2
Enter coefficient and exponent: 2 4
Enter coefficient and exponent: 4 2
Enter polynomial 2:
Enter number of terms: 2
Enter coefficient and exponent: 3 5
Enter coefficient and exponent: 5 3
Sum: 3x^5 + 2x^4 + 5x^3 + 4x^2
Product: 6x^9 + 22x^7 + 20x^5
Enter the value to evaluate the polynomials: 1
p1(1) = 6
p2(1) = 8
Sum(1) = 14
Product(1) = 48
PS C:\Users\user\Desktop\school\c++> 
```

## CHAPTER 5 效能量測

### 1. Add 的時間複雜度：

```
Enter coefficient and exponent: 2 4
Enter coefficient and exponent: 4 2
Enter polynomial 2:
Enter number of terms: 2
Enter coefficient and exponent: 3 5
Enter coefficient and exponent: 5 3
Add Time: 0.004 ms
```

### 2. Mult 的時間複雜度：

```
Enter number of terms: 2
Enter coefficient and exponent: 3 5
Enter coefficient and exponent: 5 3
Mult Time: 0.002 ms
```

### 3. Eval 的時間複雜度：

```
Enter the value to evaluate the polynomials: 1
Eval Time: 0.019 ms
```

## CHAPTER 6 心得討論

此多項式作業的理論大致上都能明白，但由於對 C++ 不熟的緣故，實作時常常遇到程式碼編譯不了的問題，因此只好使用 ChatGPT 來實作出來。實作完後也透過詳細的註解讓自己更了解 C++。在撰寫這份作業的時候，深深感受到自己的不足，包括虛擬碼的撰寫，效能該如何量測等等，也詢問了熟悉此方面的同學，跟不間斷查詢網路上的資料，並且透過影片從 C++ 的基礎開始學習。