

22,2024

資料結構報告

梁詠琳

November 19,2024

CONTENTS

CHAPTER 1 解題說明	3
Figure1.1HW1.cpp	3
CHAPTER 1 解題說明	4
問題 2: 集合的冪集	4
Figure1.2HW2.cpp	4
CHAPTER 2 演算法設計與實作	5
CHAPTER 3 效能分析	6
CHAPTER 4 測試與過程	7
CHAPTER 5 申論及心得	
CHAPTER 6 開發報告	

22,2024

-CHAPTER 1 解題說明

問題 1: 實作 **Polynomial** 類別

根據 **Figure1** 和 **Figure2** 提供的抽象資料型別 (ADT) 和私有數據成員, 實現一個 **Polynomial** 類別。

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。

舉例: $3x^2 + 2x + 1$

它的指數: [2, 1, 0]

而係數 : [3, 2, 1]

設計目標

1. 使用私有成員儲存係數和指數。
2. 提供接口來添加、刪除或操作多項式項目。
3. 支援多項式的輸入與輸出。

需要設計:

多項式的基本結構 (Polynomial Class)

包含係數和指數的存儲。提供對多項式的輸入和輸出功能。

多項式的加法功能 (Polynomial Add)

實現兩個多項式的加法, 返回一個新多項式作為加法結果。

多項式的乘法功能 (Polynomial Mult)

實現兩個多項式的逐項相乘, 並將結果整理為一個新多項式。

Eval 函數 在給定的點 x , 計算多項式的值。

這是多項式評估的功能, 通常用於計算 $p(x)$ 的結果。

22,2024

```
class Polynomial {  
    //  $p(x) = a_0x^{e_0} + \dots + a_nx^{e_n}$ ; a set of ordered pairs of  $\langle e_i, a_i \rangle$ ,  
    // where  $a_i$  is a nonzero float coefficient and  $e_i$  is a non-negative integer exponent.  
public:  
    Polynomial();  
    // Construct the polynomial  $p(x) = 0$ .  
    Polynomial Add(Polynomial poly);  
    // Return the sum of the polynomials *this and poly.  
    Polynomial Mult(Polynomial poly);  
    // Return the product of the polynomials *this and poly.  
    float Eval(float f);  
    // Evaluate the polynomial *this at f and return the result.  
};
```

Figure 1. Abstract data type of *Polynomial* class

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 1 解題說明

問題 2: 撰寫 **C++** 函數來輸入和輸出多項式

這些函數需要支持多項式的輸入與輸出。

函數需要重載 **<<** 和 **>>** 運算符。

需要設計兩個重載運算符：

>> 運算符: 用於從輸入(例如鍵盤或檔案)讀取多項式資料。

<< 運算符: 用於輸出多項式的格式化表示。

22,2024

```
class Polynomial ; // forward declaration
```

```
class Term {  
friend Polynomial;  
private:  
    float coef; // coefficient  
    int exp;    // exponent  
};
```

The private data members of *Polynomial* are defined as follows:

```
private:  
    Term *termArray; // array of nonzero terms  
    int capacity;     // size of termArray  
    int terms;        // number of nonzero terms
```

Figure 2. The private data members of *Polynomial* class

22,2024

-CHAPTER 2 演算法設計與實作

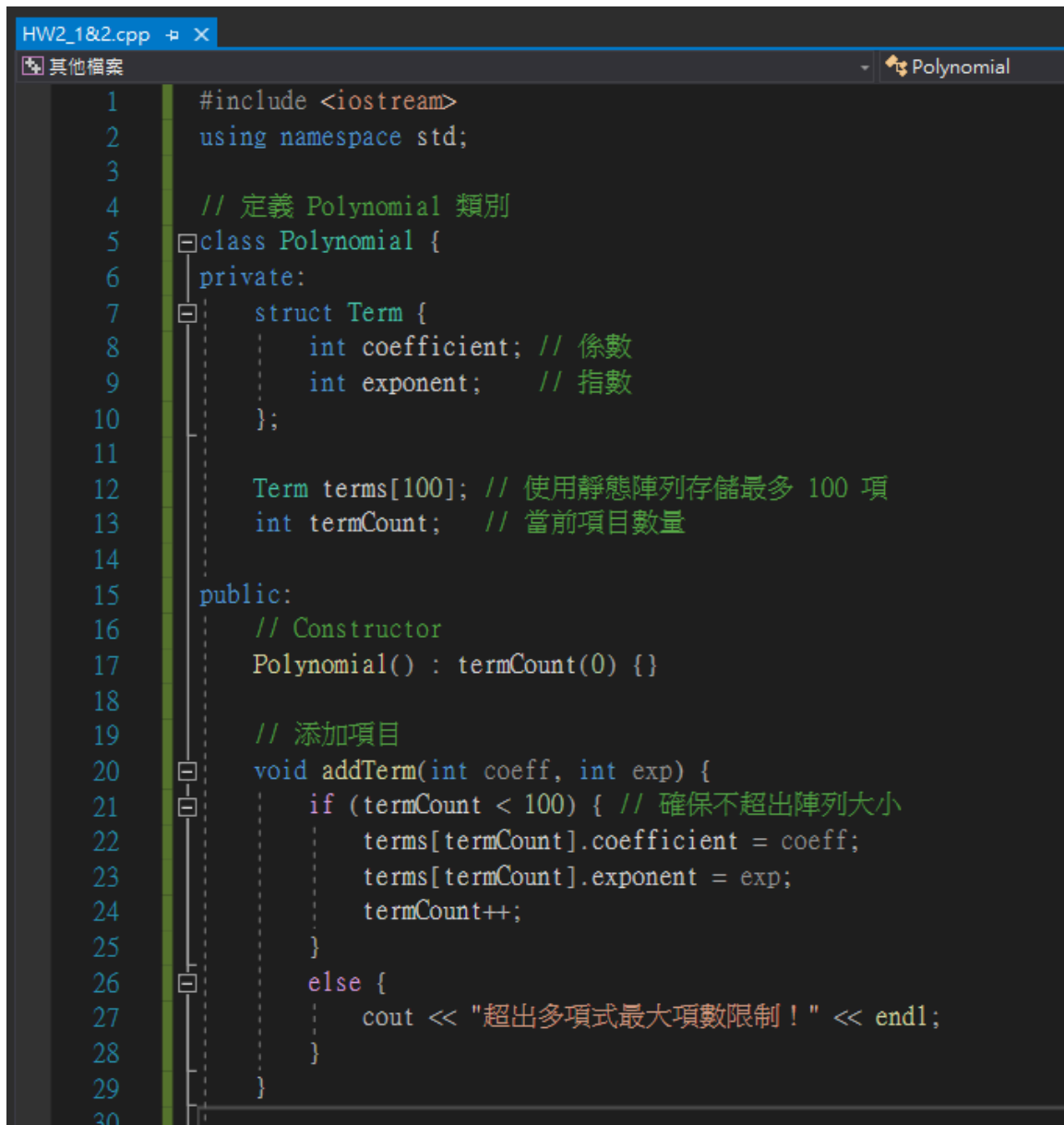
問題1: 設計了基本的多項式類別, 實現了添加和顯示功能。

問題1 的程式應包含:

Polynomial 類別的基本結構。

```
Polynomial();  
// Construct the polynomial  $p(x) = 0$ .
```

添加和顯示多項式項目的方法(如 addTerm 和 display)。

22,2024

```
1  #include <iostream>
2  using namespace std;
3
4  // 定義 Polynomial 類別
5  class Polynomial {
6  private:
7      struct Term {
8          int coefficient; // 係數
9          int exponent;    // 指數
10     };
11
12     Term terms[100]; // 使用靜態陣列存儲最多 100 項
13     int termCount;   // 當前項目數量
14
15 public:
16     // Constructor
17     Polynomial() : termCount(0) {}
18
19     // 添加項目
20     void addTerm(int coeff, int exp) {
21         if (termCount < 100) { // 確保不超出陣列大小
22             terms[termCount].coefficient = coeff;
23             terms[termCount].exponent = exp;
24             termCount++;
25         }
26         else {
27             cout << "超出多項式最大項數限制！" << endl;
28         }
29     }
30 }
```

Figure1.1 HW2_1&2.cpp

22,2024

```
100 private:
101     // 共用加法和減法的處理函式
102     Polynomial addOrSubtract(const Polynomial& other, bool isAddition) const {
103         Polynomial result;
104         int i = 0, j = 0;
105
106         while (i < termCount && j < other.termCount) {
107             if (terms[i].exponent == other.terms[j].exponent) {
108                 int coeff = isAddition ? terms[i].coefficient + other.terms[j].coefficient
109                     : terms[i].coefficient - other.terms[j].coefficient;
110                 if (coeff != 0) result.addTerm(coeff, terms[i].exponent);
111                 i++;
112                 j++;
113             }
114             else if (terms[i].exponent > other.terms[j].exponent) {
115                 result.addTerm(terms[i].coefficient, terms[i].exponent);
116                 i++;
117             }
118             else {
119                 int coeff = isAddition ? other.terms[j].coefficient : -other.terms[j].coefficient;
120                 result.addTerm(coeff, other.terms[j].exponent);
121                 j++;
122             }
123         }
124
125         while (i < termCount) result.addTerm(terms[i].coefficient, terms[i].exponent), i++;
126         while (j < other.termCount) {
127             int coeff = isAddition ? other.terms[j].coefficient : -other.terms[j].coefficient;
128             result.addTerm(coeff, other.terms[j].exponent), j++;
129         }
130
131         return result;
132     }
```

Figure1.2 HW2_1&2.cpp

多項式加法：

Polynomial Add(Polynomial poly);
*// Return the sum of the polynomials *this and poly.*

將兩個多項式相加，合併同類項。

多項式減法：(這是我另外加的)

將一個多項式從另一個多項式中減去。

多項式乘法：

Polynomial Mult(Polynomial poly);
*// Return the product of the polynomials *this and poly.*

計算兩個多項式的乘積。(這個是額外寫的，我想說chatgpt有教)

Eval 函數：

梁詠琳

22,2024**float Eval(float f);***// Evaluate the polynomial *this at f and return the result.*

簡單來說就是把x的部份換成我們的輸入。

```
90
91 // 評估多項式在點 f 的值
92 float Eval(float f) const {
93     float result = 0.0;
94     for (int i = 0; i < termCount; ++i) {
95         result += terms[i].coefficient * pow(f, terms[i].exponent); // a_i * f^e_i
96     }
97     return result;
98 }
```

Figure1.3 HW2_1&2.cpp

22,2024

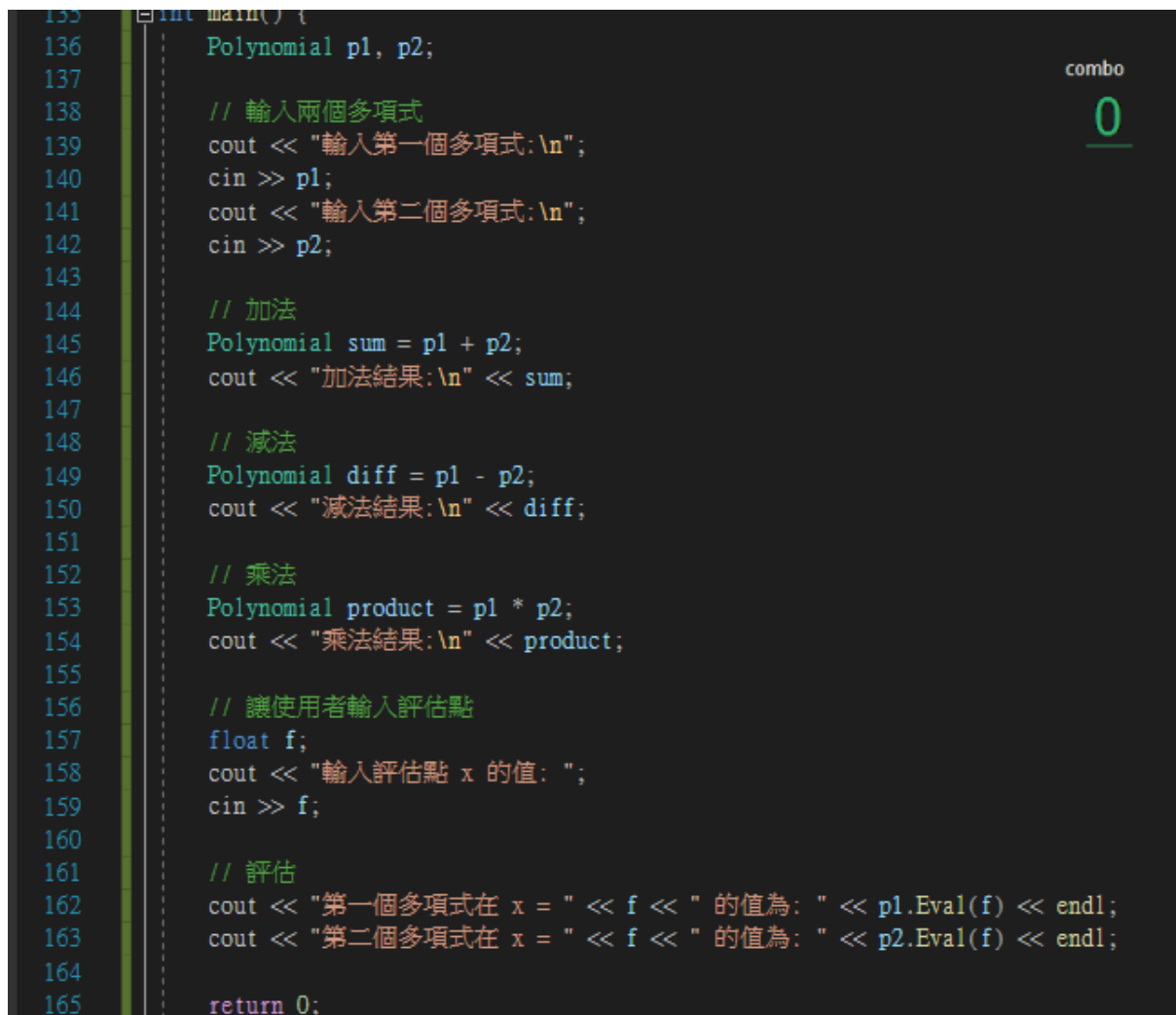
CHAPTER 2 演算法設計與實作

問題2: 實現多項式的輸入和輸出, 並重載運算符。

以下是程式碼的設計:

輸入與輸出:

使用重載的 \gg 和 \ll 運算符。



```
135 int main() {
136     Polynomial p1, p2;
137
138     // 輸入兩個多項式
139     cout << "輸入第一個多項式:\n";
140     cin >> p1;
141     cout << "輸入第二個多項式:\n";
142     cin >> p2;
143
144     // 加法
145     Polynomial sum = p1 + p2;
146     cout << "加法結果:\n" << sum;
147
148     // 減法
149     Polynomial diff = p1 - p2;
150     cout << "減法結果:\n" << diff;
151
152     // 乘法
153     Polynomial product = p1 * p2;
154     cout << "乘法結果:\n" << product;
155
156     // 讓使用者輸入評估點
157     float f;
158     cout << "輸入評估點 x 的值: ";
159     cin >> f;
160
161     // 評估
162     cout << "第一個多項式在 x = " << f << " 的值為: " << p1.Eval(f) << endl;
163     cout << "第二個多項式在 x = " << f << " 的值為: " << p2.Eval(f) << endl;
164
165     return 0;
}
```

combo
0

Figure2.1HW2_1&2.cpp

22,2024

```
31
32 // 重載輸入運算符 >>
33 friend istream& operator>>(istream& in, Polynomial& poly) {
34     int coeff, exp;
35     cout << "輸入一個項目 (係數 指數) , 輸入 -1 -1 結束: \n";
36     while (true) {
37         in >> coeff >> exp;
38         if (coeff == -1 && exp == -1) break;
39         poly.addTerm(coeff, exp);
40     }
41     return in;
42 }
43
44 // 重載輸出運算符 <<
45 friend ostream& operator<<(ostream& out, const Polynomial& poly) {
46     out << "Polynomial Details:\n";
47     for (int i = 0; i < poly.termCount; ++i) {
48         out << "    Term " << i + 1 << ":\n";
49         out << "        Coefficient: " << poly.terms[i].coefficient << "\n";
50         out << "        Exponent: " << poly.terms[i].exponent << "\n";
51     }
52     return out;
53 }
54
55 // 加法運算符重載
56 Polynomial operator+(const Polynomial& other) const {
57     return addOrSubtract(other, true); // 呼叫共用函式
58 }
59
60 // 減法運算符重載
61 Polynomial operator-(const Polynomial& other) const {
62     return addOrSubtract(other, false); // 呼叫共用函式
63 }
64
65 // 乘法運算符重載
66 Polynomial operator*(const Polynomial& other) const {
67     Polynomial result;
```

Figure2.2HW2_1&2.cpp

22,2024

CHAPTER 3 效能分析

Polynomial 類別(2題都使用同一個程式碼)

時間複雜度我以不同功能分類直接計算。

```
106 while (i < termCount && j < other.termCount) {
107     if (terms[i].exponent == other.terms[j].exponent) {
108         int coeff = isAddition ? terms[i].coefficient + other.terms[j].coefficient
109                     : terms[i].coefficient - other.terms[j].coefficient;
110         if (coeff != 0) result.addTerm(coeff, terms[i].exponent);
111         i++;
112         j++;
113     }
114     else {
115         int coeff = isAddition ? other.terms[j].coefficient : -other.terms[j].coefficient;
116         result.addTerm(coeff, other.terms[j].exponent);
117         j++;
118     }
119 }
```

時間複雜度(加/減法):

$$O(n + m)$$

加法與減法: $O(n+m)$ $O(n + m)$ $O(n+m)$

, 其中 n 與 m 是兩個多項式的項數。

```
68 // 逐項相乘
69 for (int i = 0; i < termCount; ++i) {
70     for (int j = 0; j < other.termCount; ++j) {
71         int newCoeff = terms[i].coefficient * other.terms[j].coefficient;
72         int newExp = terms[i].exponent + other.terms[j].exponent;
73     }
74 }
```

時間複雜度(乘法):

$$O(n \times m)$$

22,2024

```
90
91 // 評估多項式在點 f 的值
92 float Eval(float f) const {
93     float result = 0.0;
94     for (int i = 0; i < termCount; ++i) {
95         result += terms[i].coefficient * pow(f, terms[i].exponent); // a_i * f^e_i
96     }
97     return result;
98 }
```

時間複雜度(評估):

$O(n)$

評估: $O(n)$ 。

空間複雜度:

靜態陣列存儲多項式, 固定空間需求為 $O(100)$ 。

22,2024

CHAPTER 4 測試與驗證

問題 1 測試：

以下是我在針對第一題中Polynomial的產生跟輸入型式。

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  // 定義 Polynomial 類別
6  class Polynomial {
7  private:
8      vector<int> coefficients; // 儲存係數
9      vector<int> exponents;   // 儲存指數
10
11 public:
12     Polynomial() {}
13
14     // 添加新項目
15     void addTerm(int coeff, int exp) {
16         coefficients.push_back(coeff);
17         exponents.push_back(exp);
18     }
19
20     // 輸出多項式
21     void display() const {
22         for (size_t i = 0; i < coefficients.size(); ++i) {
23             cout << coefficients[i] << "x^" << exponents[i];
24             if (i < coefficients.size() - 1) cout << " + ";
25         }
26         cout << endl;
27     }
28 };
29
30 int main() {
31     Polynomial p;
32     p.addTerm(3, 2); // 添加 3x^2
33     p.addTerm(2, 1); // 添加 2x
34     p.addTerm(1, 0); // 添加常數 1
35
36     cout << "多項式為：";
37     p.display();
38     return 0;
39 }
40
```

Microsoft Visual Studio 偵錯主控台

多項式為：3x^2 + 2x^1 + 1x^0

C:\Users\kitty\Downloads\WH2\H

若要在偵錯停止時自動關閉主控台
按任意鍵關閉此視窗...

Figure4.1HW1.cpp

以下我拿了 $3x^2 + 2x^1 + 1x^0$ 去測試。有成功顯示答案。

```
int main() {
    Polynomial p;
    p.addTerm(3, 2); // 添加 3x^2
    p.addTerm(2, 1); // 添加 2x^1
    p.addTerm(1, 0); // 添加常數 1
}
```

Figure4.2HW1.cpp

梁詠琳

22,2024

CHAPTER 4 測試與驗證

問題 2 測試：

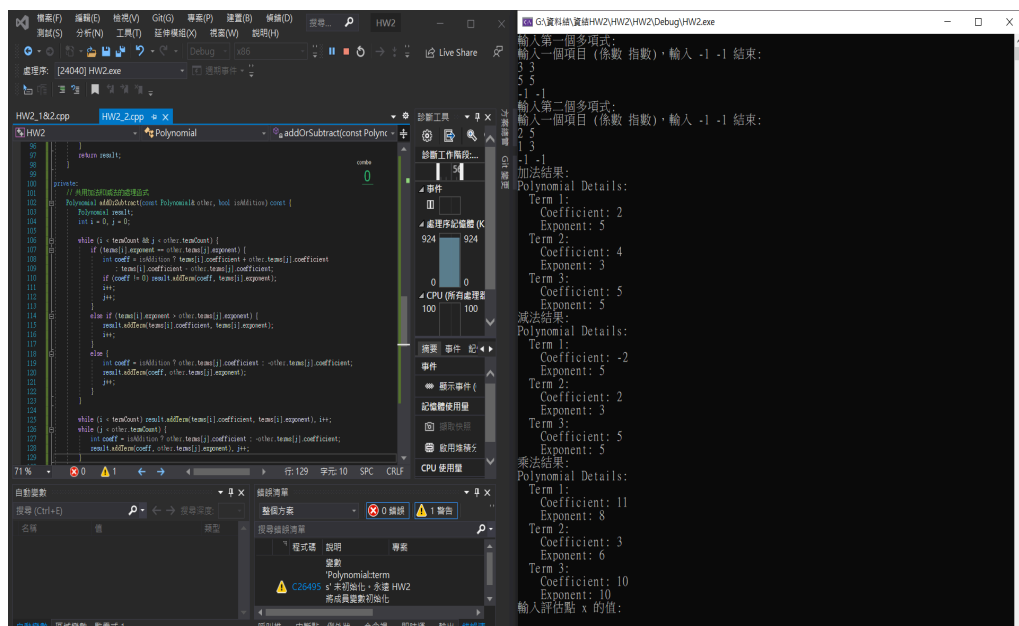
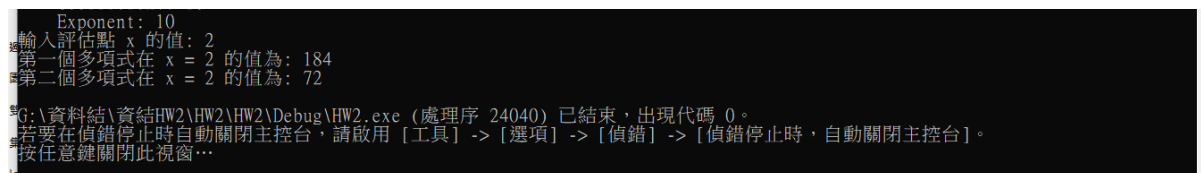


Figure4.3HW2.cpp



$$5x^5 + (5x^3 + 2x^3) + 3x^2$$

我輸入的加法是

$$(5x^3 \cdot 5x^5) + (3x^2 \cdot 2x^3) + (5x^3 \cdot 2x^3)$$

乘法為

$$5 \cdot 2^5 + 7 \cdot 2^3 + 3 \cdot 2^2 = 148$$

x代2=

22,2024

CHAPTER 5 申論

申論：多項式 (Polynomial) 作為數學中的重要概念，常出現在各類計算和分析中。而在程式設計的世界中，將多項式抽象化並加以實作，不僅能幫助理解資料結構與演算法，還能體現程式設計的靈活性與可擴展性。本次作業圍繞多項式的 ADT (抽象資料型態) 與運算符多載進行實作，讓我們在學習的過程中收穫良多。

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，它強調定義操作的行為而非內部細節。在本次作業中，多項式被抽象為一組由「係數」與「指數」構成的項集合。我們利用陣列結構儲存多項式的每一項，並實現了包括新增項目、顯示內容等基本操作，使多項式的邏輯結構清晰、易於管理。

運算符多載：讓操作更直觀

運算符多載是 C++ 的一大特色，它允許重新定義運算符的行為，使程式碼更具可讀性與直觀性。在這次作業中，我們實現了輸入 (>>)、輸出 (<<)、加法 (+)、減法 (-) 與乘法 (*) 的運算符多載。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讀取，而輸出時，透過 `cout << p1` 以數學形式呈現多項式。這種設計使操作更貼近數學的表達方式，同時也提升了程式的使用體驗。

實作中的挑戰與收穫

在實作過程中，最大的挑戰來自於多項式運算的設計與優化。例如，加法和乘法需要處理不同項目的合併，尤其是指數相同的項；而在計算多項式值 (Eval 函數) 時，如何確保結果的正確性和程式的效能，也是重要的考量點。這些問題的解決不僅幫助我們深入理解資料結構與演算法，也鍛煉了邏輯思維與問題解決能力。

程式設計的價值

透過本次作業，我更深刻地體會到程式設計的價值在於將抽象的概念具體化，並將複雜的運算轉化為簡潔直觀的操作。ADT 和運算符多載的結合，不僅展示了資料結構的靈活性，也讓我們感受到程式設計的創造力。

CHAPTER 6心得

心得: 這次作業讓我深入理解了多項式 ADT 和運算符多載的應用。

透過實作, 我學會如何利用 ADT 進行資料的抽象與操作, 將多項式拆解成多個項目並進行加法、乘法等運算。

在中學的時候都用計算機算多項式, 沒想到在大學的今天會寫出計算多項式的程式碼, 令我覺得有點好笑。

儘管在實作過程中遇到了一些困難, 例如不理解題目要求或程式邏輯出錯, 但這也讓我明白了與助教、同學交流的重要性。

透過討論和反思, 不僅解決了問題, 也鞏固了對程式設計的基本概念。

22,2024

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和接近數學表達。

確保程式結構清晰、邏輯完整，並具備可擴展性。

開發過程

-程式架構設計

首先參考題目提供的 ADT 定義，我們設計了一個 Polynomial 類別，使用陣列結構儲存多項式的每一項。

在多項式的每一項中，包含係數與指數，並實現了新增項目 (addTerm) 與顯示多項式內容 (operator<<) 的功能。

-運算符多載

為了讓多項式操作更加直觀，我們透過運算符多載實現了加法 (operator+)、減法 (operator-) 與乘法 (operator*)。

同時，輸入 (operator>>) 與輸出 (operator<<) 運算符也被重新定義，使多項式的輸入輸出更符合數學習慣。

-功能擴展

在基本顯示功能完成後，我們進一步實現了多項式的運算功能，並加入了 Eval(float x) 函數，用於計算多項式在指定點的值。在 Eval 的實作過程中，我們引入了迴圈計算法則，確保程式效能與正確性。

-問題與挑戰

在做到後期的時候，我其實對題目的需求理解出現了一些偏差。第一題我以為是完成基礎的多項式顯示功能；第二題才是加入運算多項式的功能。後來在與助教的溝通中，我才明白兩題其實可以合併處理，讓程式更具整體性與邏輯性，這也讓我重新規劃了程式的結構設計。

梁詠琳