

# 資料結構報告

姓名:童呈偉

日期:2024/08/11

# 目錄

1. 解題說明
2. 設計與實作
3. 效能分析
4. 測試與驗證
5. 效能量測
6. 心得

# 1.解題說明

題目翻譯如下:開發一個 C++ 類別 Polynomial 來表示和操作具有整數係數的單變量多項式(使用帶有頭節點的環形鏈表)。多項式的每一項將表示為一個節點。因此,在這個系統中,一個節點將有以下三個數據成員:

| 係數 | 指數 | 鏈接 |

每個多項式都將表示為一個帶有頭節點的環形鏈表。為了有效地刪除多項式,我們需要使用一個可用空間列表和第 4.5 節中描述的相關函數。單變量多項式的外部(即輸入或輸出)表示將假定為以下形式的整數序列: $n, c_1, e_1, c_2, e_2, c_3, e_3, \dots, c_n, e_n$ , 其中  $e_i$  代表指數,  $c_i$  代表係數; $n$  給出多項式中的項數。指數按遞減順序排列—— $e_1 > e_2 > \dots > e_n$ 。

撰寫並測試以下函數：

(a) `istream& operator>>(istream& is, Polynomial& x)`: 讀取輸入的多項式，並使用頭節點將其轉換為環形鏈表表示。

(b) `ostream& operator<<(ostream& os, Polynomial& x)`: 將 `x` 從鏈表表示轉換為外部表示並輸出。

(c) `Polynomial::Polynomial(const Polynomial& a)`: [複製構造函數] 初始化多項式 `*this` 為多項式 `a`。

(d) `const Polynomial& Polynomial::operator=(const Polynomial& a)`  
`const`: [賦值運算符] 將多項式 `a` 賦值給 `*this`。

(e) `Polynomial::~~Polynomial()`: [解構函數] 將多項式 `*this` 的所有節點返回到可用空間列表。

(f) `Polynomial operator+(const Polynomial& b) const`: [加法] 創建並返回多項式 `*this + b`。

(g) `Polynomial operator-(const Polynomial& b) const`: [減法] 創建並返回多項式 `*this - b`。

(h) `Polynomial operator*(const Polynomial& b) const`: [乘法] 創建並返回多項式 `*this * b`。

(i) `float Polynomial::Evaluate(float x) const`: 在 `x` 評估多項式 `*this` 並返回結果。

## 依照題目的類別設定如下

```
class Term //定義類別 Term
{
    friend class Polynomial; //Polynomial可存取Term的私有成員
    friend ostream& operator<<(ostream&, const Polynomial&); // << 可存取 Term的私有成員
    friend istream& operator>>(istream&, Polynomial&); // >> 可存取Term的私有成員
private://的私有成員
    float coef; //多項式係數
    int exp; //多項式指數
    Term* link; //指向下一個 Term 對象的指針，這是用於鏈表結構的一部分
};
```

fig1.1 類別創建:Polynomiallink.cpp

```
17  class Polynomial //定義類別 Polynomial
18  {
19  public://公有成員
20      Polynomial(); //建構函數，用於初始化資料
21      Polynomial(const Polynomial& poly); //複製現有Polynomial物件創建新物件
22      ~Polynomial(); //解構函數
23
24      const Polynomial& operator=(const Polynomial& poly);
25      //首先會清空當前對象中的所有項目，然後複製給定對象 (poly) 的所有項目到當前對象中，並返回當前對象 (*this) 的參考。
26      Polynomial operator+(const Polynomial& b) const; //加法
27      Polynomial operator-(const Polynomial& b) const; //減法
28      Polynomial operator*(const Polynomial& b) const; //乘法
29      float Evaluate(float x) const; //計算多項式在給定數值 x 上的值的函數
30
31      friend ostream& operator<<(ostream& os, const Polynomial& poly);
32      friend istream& operator>>(istream& is, Polynomial& poly);
33
34  private:
35      Term* head;
36      //用來作為多項式項的鏈表的頭部。head 節點本身並不存儲實際的多項式項目，但它用來標記鏈表的開始並提供一個循環結構。
37      void insertTerm(float coef, int exp);
38      //根據係數和指數在鏈表中找到正確的位置來插入新的 Term 對象。這個方法會處理合併相同指數的項，並在需要時刪除係數為零的項。
39      void clear();
40      //清空多項式中所有項目的私有方法。它會遍歷鏈表，刪除所有 Term 對象，然後重設 head 節點的鏈接，使其指向自身。
41  };
42
```

fig1.2 link方式的多項式類別創建:Polynomiallink.cpp

## 2.設計與實作

依照(a)~(i)的要求程式碼如下:

```
52 //複製一個現有的 Polynomial 對象 poly 的數據到新創建的 Polynomial 對象中。
53 Polynomial::Polynomial(const Polynomial& poly)
54 {
55     head = new Term;
56     head->link = head;
57     Term* current = poly.head->link;
58     //設置 current 指針為傳入 poly 對象鏈表中的第一個實際節點。
59     while (current != poly.head) //遍歷 poly 對象中的所有 Term 節點 (直到返回到 head)。
60     {
61         insertTerm(current->coef, current->exp);
62         //將每個 Term 的係數和指數插入到當前 Polynomial 對象中。
63         current = current->link;
64         //移動到下一個節點
65     }
66 }
67
68 Polynomial::~Polynomial()
69 {
70     clear();
71     //刪除多項式中的所有 Term 對象。這個方法會遍歷鏈表並刪除每個節點，確保沒有內存洩漏。
72     delete head;
73     //刪除 head 指針所指向的節點。這裡的 head 節點在構造函數中是用 new 分配的，因此需要顯式地刪除來釋放內存。
74 }
75
```

fig2.1解構函數和複製構造:Polynomiallink.cpp

```
76 void Polynomial::clear()
77 //刪除多項式中的每一個 Term 對象，以便釋放佔用的內存並確保鏈表的狀態被重置為空
78 {
79     Term* current = head->link;
80     //第一個實際 Term 節點
81     while (current != head) //遍歷鏈表中的所有 Term 節點。循環的條件是 current 不等於 head，這樣可以確保循環鏈表直到回到 head 節點為止
82     {
83         Term* temp = current;
84         //在刪除當前節點後仍然能夠保留對該節點的指針。這樣可以確保在刪除節點之前能夠正確地移動到下一個節點。
85         current = current->link;
86         //更新 current 指針，將其設置為當前節點的 link 成員
87         delete temp;
88         //刪除當前節點以釋放內存
89     }
90     head->link = head;
91     //重新建立一個空的循環鏈表結構。這樣，鏈表的狀態被重置為空，並且 head 節點指向自身。
92 }
```

fig2.2 link方式的多項式重製節點:Polynomiallink.cpp

```

93 //Polynomial::表示insertTerm 函數是 Polynomial 類的一部分，而不是全局函數或其他類的函數。
94 ▼ void Polynomial::insertTerm(float coef, int exp)
95 //用於在多項式中插入一個新的項，或者更新已有項的係數。如果插入或更新後的係數為零，則該項會被刪除。
96 {
97     if (coef == 0) return; //係數 coef 為零，則不需要插入或更新任何項，因此直接返回。
98     Term* prev = head; //prev 指向鏈表的頭部節點 (head)，用來追蹤當前節點的前一個節點
99     Term* current = head->link; //current 指向鏈表中的第一個實際節點
100     while (current != head && current->exp > exp) //檢查每個節點的指數 (exp)
101     { //如果 current 節點的指數大於新項的指數，則繼續前進，直到找到正確的位置。
102         prev = current;
103         current = current->link;
104     }
105     if (current != head && current->exp == exp)
106         //找到的節點 (current) 的指數等於新項的指數，則更新該節點的係數。
107     {
108         current->coef += coef;
109         if (current->coef == 0)
110             //更新後的係數為零，則刪除該節點並調整鏈表中的鏈接，以移除該項。
111         {
112             prev->link = current->link;
113             delete current;
114         }
115     }
116     else //如果 current 節點的指數不等於新項的指數，則創建一個新的 Term 節點。
117     { //設置新節點的係數和指數，並將新節點插入到 prev 節點和 current 節點之間。
118         Term* newTerm = new Term;
119         newTerm->coef = coef;
120         newTerm->exp = exp;
121         newTerm->link = current;
122         prev->link = newTerm;
123     }
124 }

```

fig2.3 link方式的多項式插入節點(新項):Polynomiallink.cpp

```

126     const Polynomial& Polynomial::operator=(const Polynomial& poly)
127     //將一個 Polynomial 對象的內容賦值給另一個 Polynomial 對象。
128     {
129         if (this != &poly) //確保 this 指針不等於 poly 的地址。這樣可以防止在賦值時出現自我賦值的情況
130         {
131             clear(); //刪除當前對象中的所有項目，釋放內存。
132             Term* current = poly.head->link; //設置 current 指針為 poly 對象的第一個實際節點。
133             while (current != poly.head) //遍歷 poly 對象中的所有項，並將每個項插入到當前對象中
134             {
135                 insertTerm(current->coef, current->exp);
136                 current = current->link;
137             }
138         }
139         return *this; //返回當前對象的參考 (*this)，這樣可以支持連鎖賦值操作 (例如 a = b = c;)。
140     }

```

fig2.4 賦值運算符:Polynomiallink.cpp

```

245     ostream& operator<<(ostream& os, const Polynomial& poly) //將 Polynomial 對象輸出到 ostream 的功能
246     {
247         Term* current = poly.head->link; //指針 current 指向多項式的第一個實際節點
248         while (current != poly.head) //遍歷多項式的所有項，直到達到頭節點 (poly.head)。
249         {
250             //如果不是第一個項且係數大於 0，則先輸出 " + " 來分隔項。
251             if (current != poly.head->link && current->coef > 0) os << " + ";
252             os << current->coef << "x^" << current->exp;
253             current = current->link;
254             //輸出當前項的係數和指數。格式是 係數x^指數。
255             //更新 current 指針，指向下一個項。
256         }
257         return os;
258     }

```

fig2.5 輸出運算符<<:Polynomiallink.cpp



```

260 //讀取數據來初始化 Polynomial 對象的功能
261 istream& operator>>(istream& is, Polynomial& poly)
262 { //從輸入流中讀取整數 n，表示多項式的項數。
263     int n;
264     cout << "輸入項數: ";
265     is >> n;
266     for (int i = 0; i < n; i++) //根據項數 n 讀取每一項的係數和指數。
267     {
268         float coef;
269         int exp;
270         cout << "輸入係數跟指數: ";
271         is >> coef >> exp;
272         poly.insertTerm(coef, exp); //讀取到的係數和指數插入到 Polynomial 對象中。
273     }
274     return is;
275 }
276

```

fig2.5 輸入運算符>>:Polynomiallink.cpp

```

142 //const 是函數修飾符，表示這個函數不會修改其成員變量。
143 Polynomial Polynomial::operator+(const Polynomial& b) const //計算當前多項式 (*this) 和另一個多項式 (b) 的和
144 {
145     Polynomial result; //對象 result，用來存儲加法結果。
146     Term* aTerm = head->link; //aTerm 指向當前多項式的第一個實際節點。
147     Term* bTerm = b.head->link; //bTerm 指向另一個多項式 b 的第一個實際節點。
148     while (aTerm != head && bTerm != b.head) //循環遍歷兩個多項式，直到其中一個多項式遍歷完成。
149     {
150         if (aTerm->exp == bTerm->exp) //aTerm 和 bTerm 的指數相同，係數相加，並插入到 result 中。然後更新 aTerm 和 bTerm 指針。
151         {
152             result.insertTerm(aTerm->coef + bTerm->coef, aTerm->exp);
153             aTerm = aTerm->link;
154             bTerm = bTerm->link;
155         }
156         else if (aTerm->exp > bTerm->exp) //aTerm 的指數大於 bTerm 的指數，則將 aTerm 的項插入到 result 中，並更新 aTerm 指針。
157         {
158             result.insertTerm(aTerm->coef, aTerm->exp);
159             aTerm = aTerm->link;
160         }
161         else //bTerm 的指數大於 aTerm 的指數，則將 bTerm 的項插入到 result 中，並更新 bTerm 指針。
162         {
163             result.insertTerm(bTerm->coef, bTerm->exp);
164             bTerm = bTerm->link;
165         }
166     }
167     //在前面的 while 循環中可能有一個多項式還有剩餘的項未被處理。這些剩餘的項會被單獨處理並插入到 result 中。
168     while (aTerm != head) //aTerm 還剩下的項。
169     {
170         result.insertTerm(aTerm->coef, aTerm->exp);
171         aTerm = aTerm->link;
172     }
173     while (bTerm != b.head) //bTerm 還剩下的項。
174     {
175         result.insertTerm(bTerm->coef, bTerm->exp);
176         bTerm = bTerm->link;
177     }
178     return result;
179 }

```

fig2.5 加法函式+:Polynomiallink.cpp

```

181 //計算當前多項式 (*this) 和另一個多項式 (b) 的差
182 Polynomial Polynomial::operator-(const Polynomial& b) const
183 {
184     Polynomial result;
185     Term* aTerm = head->link;
186     Term* bTerm = b.head->link;
187     while (aTerm != head && bTerm != b.head)
188     {
189         if (aTerm->exp == bTerm->exp) //指數相同，係數進行相減，結果插入到 result 中。然後更新 aTerm 和 bTerm 指針。
190         {
191             result.insertTerm(aTerm->coef - bTerm->coef, aTerm->exp);
192             aTerm = aTerm->link;
193             bTerm = bTerm->link;
194         }
195         else if (aTerm->exp > bTerm->exp) //aTerm 的指數大於 bTerm 的指數，則將 aTerm 的項插入到 result 中，並更新 aTerm 指針。
196         {
197             result.insertTerm(aTerm->coef, aTerm->exp);
198             aTerm = aTerm->link;
199         }
200         else //bTerm 的指數大於 aTerm 的指數，則將 -bTerm 的項插入到 result 中 (這裡的 -bTerm->coef 表示 bTerm 的項係數取反)，並更新 bTerm 指針。
201         {
202             result.insertTerm(-bTerm->coef, bTerm->exp);
203             bTerm = bTerm->link;
204         }
205     }
206     while (aTerm != head) //第一個 while 循環處理 aTerm 還剩下的項，直接插入到 result。
207     {
208         result.insertTerm(aTerm->coef, aTerm->exp);
209         aTerm = aTerm->link;
210     }
211     while (bTerm != b.head) //處理 bTerm 還剩下的項，將這些項的係數取反後插入 result。
212     {
213         result.insertTerm(-bTerm->coef, bTerm->exp);
214         bTerm = bTerm->link;
215     }
216     return result;
217 }

```

fig2.5 減法函式:-Polynomiallink.cpp

```

Polynomial Polynomial::operator*(const Polynomial& b) const //當前多項式 (*this) 和參數 b 的乘積。
{
    Polynomial result;
    for (Term* aTerm = head->link; aTerm != head; aTerm = aTerm->link) //外層 for 循環遍歷當前多項式 (*this) 的所有項 (由 aTerm 指針遍歷)
    {
        for (Term* bTerm = b.head->link; bTerm != b.head; bTerm = bTerm->link) //內層 for 循環遍歷參數多項式 (b) 的所有項 (由 bTerm 指針遍歷)。
        {
            //對於每對項 aTerm 和 bTerm，計算它們的乘積 aTerm->coef * bTerm->coef，並將它們的指數相加 aTerm->exp + bTerm->exp。
            result.insertTerm(aTerm->coef * bTerm->coef, aTerm->exp + bTerm->exp);
            //result.insertTerm 函數將這些計算出的項插入到結果多項式 result 中。
        }
    }
    return result;
}

```

fig2.5 乘法函式\*:Polynomiallink.cpp

```

234 //計算多項式在給定點 x 的值。它返回多項式在 x 位置的結果。
235 ▼ float Polynomial::Evaluate(float x) const
236 {
237     float result = 0; //變量 result 用於累加多項式在 x 點的計算結果，初始值為 0。
238     for (Term* current = head->link; current != head; current = current->link) //遍歷多項式的所有項（由 current 指針遍歷）
239     { //對於每個項 current，計算 current->coef * pow(x, current->exp)
240         result += current->coef * pow(x, current->exp);
241     }
242     return result;
243 }
244

```

fig2.5給定x值:Polynomiallink.cpp

```

43 //初始化 Polynomial 對象的成員變量。在這裡主要是用來設置多項式的鏈表頭部節點。
44 ▼ Polynomial::Polynomial()
45 {
46     head = new Term;
47     //這個 Term 對象作為鏈表的頭部節點，用於標記鏈表的起始位置。
48     head->link = head;
49     //形成一個循環鏈表。這樣設置可以使得當鏈表中沒有其他節點時，head->link 仍然指向自己，簡化了邊界條件處理。
50 }
51

```

fig2.6初始化:Polynomiallink.cpp

### 3.效能分析

時間複雜度和空間複雜度如下：

$m$ 為多項式 $this$ 的項數， $n$ 為多項式 $b$ 的項數。

時間複雜度：

插入項： $O(m)$ ，最壞情況：需要遍歷整個多項式鏈表。

加法： $O(m + n)$ ，需要同時遍歷兩個多項式( $this$  和  $b$ )。

減法： $O(m + n)$ ，需要同時遍歷兩個多項式( $this$  和  $b$ )。

乘法： $O(m * n)$ ，每個項需要與另一個多項式的每個項相乘。

求值： $O(m)$ ，遍歷多項式的所有項。

輸出： $O(m)$ ，將每個項的係數和指數輸出到流中。

空間複雜度：

插入項： $O(1)$ ，每次插入只需要常數空間。

加法： $O(m + n)$ ，結果多項式最多包含  $m + n$  項。

減法： $O(m + n)$ ，結果多項式最多包含  $m + n$  項。

乘法： $O(m * n)$ ，項數最多可能是  $m * n$ 。

求值： $O(1)$ ，只需要存儲當前計算的結果。

輸出： $O(m)$ ，輸出操作本身不需要額外的空間。

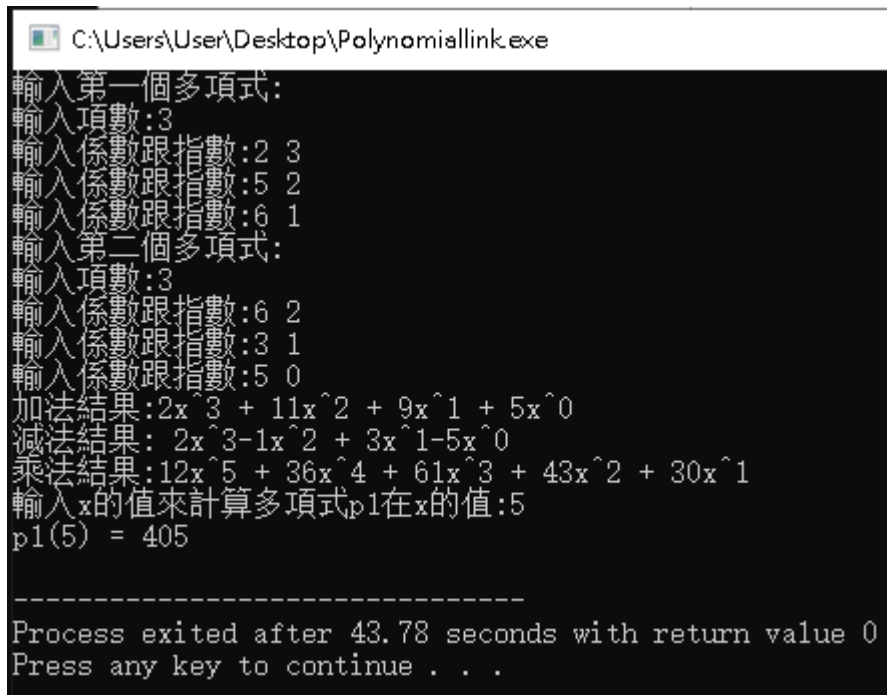
### 3.測試與驗證

主函式如下:

```
277  int main() {
278      Polynomial p1, p2, p3;
279      cout << "輸入第一個多項式:\n";
280      cin >> p1;
281
282      cout << "輸入第二個多項式:\n";
283      cin >> p2;
284
285      p3 = p1 + p2;
286      cout << "加法結果: " << p3 << endl;
287
288      p3 = p1 - p2;
289      cout << "減法結果: " << p3 << endl;
290
291      p3 = p1 * p2;
292      cout << "乘法結果: " << p3 << endl;
293
294      float x;
295      cout << "輸入x的值來計算多項式p1在x的值: ";
296      cin >> x;
297      float result = p1.Evaluate(x); //計算多項式 p1 在 x 點的值, 結果存儲在 result 中。這是通過 Evaluate 成員函數來實現的。
298      cout << "p1(" << x << ") = " << result << endl;
299
300      return 0;
301  }
```

fig3.1 主函式:Polynomiallink.cpp

結果如下:



```
C:\Users\User\Desktop\Polynomiallink.exe
輸入第一個多項式:
輸入項數:3
輸入係數跟指數:2 3
輸入係數跟指數:5 2
輸入係數跟指數:6 1
輸入第二個多項式:
輸入項數:3
輸入係數跟指數:6 2
輸入係數跟指數:3 1
輸入係數跟指數:5 0
加法結果:2x^3 + 11x^2 + 9x^1 + 5x^0
減法結果: 2x^3-1x^2 + 3x^1-5x^0
乘法結果:12x^5 + 36x^4 + 61x^3 + 43x^2 + 30x^1
輸入x的值來計算多項式p1在x的值:5
p1(5) = 405

-----
Process exited after 43.78 seconds with return value 0
Press any key to continue . . .
```

fig3.2 結果:Polynomiallink.exe

驗證:

$$\text{多項式1: } 2x^3 + 5x^2 + 6x$$

$$\text{多項式2: } 6x^2 + 3x + 5$$

$$\begin{aligned}\text{加法: } & 2x^3 + (5x^2 + 6x^2) + (6x + 3x) + 5 \\ & = 2x^3 + 11x^2 + 9x + 5\end{aligned}$$

$$\begin{aligned}\text{減法: } & (2x^3 - 0x^3) + (5x^2 - 6x^2) + (6x - 3x) + (5 - 0) \\ & = 2x^3 - x^2 + 5\end{aligned}$$

$$\begin{aligned}\text{乘法: } & (2x^3 * 6x^2) + (2x^3 * 3x) + (2x^3 * 5) \\ & + (5x^2 * 6x^2) + (5x^2 * 3x) + (5x^2 * 5) \\ & + (6x * 6x^2) + (6x * 3x) + (6x * 5) \\ & = 12x^5 + 6x^4 + 10x^3 + 30x^4 + 15x^3 + 25x^2 + 36x^3 + 18x^2 + 30x \\ & = 12x^5 + 36x^4 + 61x^3 + 43x^2 + 30x\end{aligned}$$

給定x值=5

$$\begin{aligned}& 2x^3 + 5x^2 + 6x \\ & = 2 * 125 + 5 * 25 + 30 \\ & = 250 + 125 + 30 \\ & = 405\end{aligned}$$

## 5.效能量測

```
Compilation results...
-----
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\User\Desktop\Polynomiallink.exe
- Output Size: 1.85982227325439 MiB
- Compilation Time: 9.66s
```

fig5.4 編譯時間:Polynomiallink.cpp

## 6.心得

這一次的製作一樣是由chat GPT協助，但是因為上一個作業有完成跟之前的註釋解析，這一次的程式碼理解，沒有那麼困難，同時藉由手算了解鏈結方式的步驟，讓思路清晰許多。