

# 資料結構報告

資工二甲 41243101 伍翊瑄

日期 : 2025/01/07

## 目錄

1. 解題說明	3
2. 演算法設計與實作	5
3. 效能分析	15
4. 測試與驗證	17
5. 效能量測	18
6. 心得討論	19

# 一. 解題說明

## 題目背景：

設計一個多項式運算系統，使用含有標頭節點的單一鏈結環狀串列(circular linked lists with header nodes)來表示多項式。多項式的每一項將個別存於一個節點中。系統需具備基本操作功能，包括加法、減法、乘法運算以及針對指定數值進行計算。

## 主要功能：

1. **多項式表示：** 透過帶有標頭節點的單一環狀鏈結串列實現多項式的存儲與管理。(優點: 可以靈活地操作多項式的插入、刪除與合併。標頭節點使得操作更為方便，特別是在多項式為空的情況下。)
2. **多項式運算：** 執行兩個多項式的加法與減法和乘法運算。
3. **數值計算：** 輸入變數  $x$  的值，計算多項式在該點的數值結果。
4. 支援多項式的輸入與輸出格式化。

## 解決方案概述：

透過物件導向程式設計的方式來實現，將多項式分解為兩部分：

- 單項式 ( Term )：表示多項式中的每一項，包含係數與指數，並存儲於單一鏈結環狀串列的節點中。
- 多項式 ( Polynomial )：整體的多項式表示，內部使用帶有標頭節點的環狀鏈結串列來管理單項式，實現動態存儲與操作。

多項式表示：採用環狀鏈結串列結構，每一個節點代表一個單項式，

節點包含三個部分：

- 係數 coef：對應於單項式的係數。
- 指數 exp：對應於單項式的指數。
- 節點指標 link：指向(鏈結串列的)下一個單項式節點，最後一個節點指向標頭節點形成環狀結構。

## 二. 演算法設計與實作

### 1. Term 結構：

- 多項式的基本單位，包含項的係數與指數。
- 指向下一個節點的指標。

```
5 // 單一項(節點)結構 //struct 強調 Term 裡的資料成員是公用的
6 struct Term
7 {
8     int coef;    // 係數
9     int exp;     // 指數
10    Term* link;  // 下一個節點指標
11 };
```

### 2. Polynomial 類別：表示整個多項式，

包含以下功能：

- `istream& operator>>(istream& is, Polynomial& x)` :

從輸入流 ( 如鍵盤或檔案 ) 讀取多項式的各項係數和指數 ,

並將其存儲到物件中。

```
273 // 輸入運算子
274 istream& operator>>(istream& is, Polynomial& x)
275 {
276     int n;
277     cout << "輸入非零項的數量: ";
278     is >> n;
279     for (int i = 0; i < n; ++i)
280     {
281         int coef, exp;
282         cout << "輸入係數與指數: ";
283         is >> coef >> exp;
284         x.insertTerm(coef, exp);
285     }
286     return is;
287 }
```

- ostream& operator<<(ostream& os, const

Polynomial& x): 將多項式物件轉換為可讀的字串格式，並輸出到輸出流（如螢幕或檔案）。

```
290 ostream& operator<<(ostream& os, const Polynomial& x)
291 {
292     Term* current = x.head->link;
293     bool firstTerm = true; // 用於檢查是否是第一個項
294     while (current != x.head)
295     {
296         if (!firstTerm && current->coef > 0) os << "+"; // 非第一項且係數為正，添加 '+'
297         if (current->coef != 1 || current->exp == 0) // 係數不為 1 或 指數為 0 時，顯示係數
298         {
299             os << current->coef;
300         }
301         if (current->exp > 0) // 如果指數大於 0，顯示 x
302         {
303             os << "x";
304             if (current->exp > 1) // 指數大於 1 時顯示次方
305             {
306                 os << "^" << current->exp;
307             }
308         }
309         current = current->link;
310         firstTerm = false; // 更新為非第一項
311     }
312     return os;
313 }
314 }
```

- `Polynomial(const Polynomial& a)`：用於創建多項式物件的拷貝，確保兩個物件獨立且不共享內部資源(記憶體)。

```
47 // 複製建構子
48 Polynomial::Polynomial(const Polynomial& a) : Polynomial()
49 {
50     Term* current = a.head->link;
51     while (current != a.head) // 處理多項式每一項 // 遍歷 a 的所有有效節點，直到重新到達頭節點 a.head
52     {
53         insertTerm(current->coef, current->exp);
54         current = current->link; // 移動到下一個節點
55     }
56 }
```

- `const Polynomial& operator=(const Polynomial& a)`：

將另一個多項式的內容賦值給當前物件。用於物件間的賦值

操作，例如 `p1 = p2`。

```
71 // 賦值運算子重載
72 const Polynomial& Polynomial::operator=(const Polynomial& a)
73 {
74     if (this != &a) // 避免對自身進行賦值 (如 p1 = p1)
75     {
76         this->~Polynomial(); // 清除當前多項式物件 *this(自己) 的舊內容，包括所有的節點記憶體
77
78         // 為當前物件重新建立頭節點 // 重新初始化為空環狀鏈結串列
79         head = getNode();
80         head->link = head;
81
82         // 複製另一個多項式的內容
83         Term* current = a.head->link;
84         while (current != a.head)
85         {
86             insertTerm(current->coef, current->exp);
87             current = current->link;
88         }
89     }
90     return *this; // 返回當前物件
91 }
```



- ~Polynomial(): 釋放多項式物件所使用的所有資源，

避免記憶體洩漏。

```
58      // 解構子：釋放所有節點
59      Polynomial::~Polynomial()
60      {
61          Term* current = head->link;
62          while (current != head)
63          {
64              Term* temp = current;
65              current = current->link;
66              returnNode(temp);
67          }
68          returnNode(head);
69      }
```

- `void insertTerm(int coef, int exp)`: 向多項式中插入一個新項目（由係數和指數組成），並根據指數大小保持多項式的降冪順序。如果指數相同，則合併同類項；如果合併後的係數為零，則刪除該項。

```
113 // 插入單項式(添加新項)，按指數降序排列
114 void Polynomial::insertTerm(int coef, int exp)
115 {
116     Term* current = head;
117     while (current->link != head && current->link->exp > exp) // && 下一項指數 > 新項指數
118     {
119         current = current->link; // 移動到下一項
120     }
121
122     // 若下一項指數 != 新項指數，表示找到插入位置
123     // 檢查新項在此插入位置，是否需要被合併（下一項指數 = 新項指數）
124     if (current->link != head && current->link->exp == exp)
125     {
126         current->link->coef += coef; // 合併同類項（下一項係數 + 新項係數）
127         if (current->link->coef == 0) // 合併後，若係數為 0，刪除該項
128         {
129             Term* temp = current->link; // 記錄要刪除的項(節點)
130             current->link = temp->link; // 將鏈結指向刪除項(節點)的下一項
131             returnNode(temp);
132         }
133     }
134     else
135     {
136         Term* temp = getNode(); // 從可用空間或記憶體中分配一個新節點(項)
137         temp->coef = coef; // 設置新項的係數
138         temp->exp = exp;
139         temp->link = current->link; // 先將新項的下一項設置為 current 的下一項
140         current->link = temp; // 再更新 current->link，使其指向新項
141     }
142 }
```

- operator+ (const Polynomial& b) const : 計算兩個多項式的和。

```
144 // 加法運算
145 Polynomial Polynomial::operator+(const Polynomial& b) const
146 {
147     Polynomial result;
148     Term* p1 = head->link;
149     Term* p2 = b.head->link;
150
151     while (p1 != head || p2 != b.head) // 只要有一多項式不為空就可以算
152     {
153         if (p1 != head && (p2 == b.head || p1->exp > p2->exp))
154         {
155             result.insertTerm(p1->coef, p1->exp);
156             p1 = p1->link;
157         }
158         else if (p2 != b.head && (p1 == head || p2->exp > p1->exp))
159         {
160             result.insertTerm(p2->coef, p2->exp);
161             p2 = p2->link;
162         }
163         else // p1指數 = p2指數
164         {
165             result.insertTerm(p1->coef + p2->coef, p1->exp);
166             p1 = p1->link;
167             p2 = p2->link;
168         }
169     }
170     return result;
171 }
```

- operator- (const Polynomial& b) const : 計算兩個多項式的差。

```
211 // 減法運算
212 Polynomial Polynomial::operator-(const Polynomial& b) const
213 {
214     Polynomial result;
215     Term* p1 = head->link;
216     Term* p2 = b.head->link;
217
218     while (p1 != head || p2 != b.head)
219     {
220         if (p1 != head && (p2 == b.head || p1->exp > p2->exp))
221         {
222             result.insertTerm(p1->coef, p1->exp);
223             p1 = p1->link;
224         }
225         else if (p2 != b.head && (p1 == head || p2->exp > p1->exp))
226         {
227             result.insertTerm(-p2->coef, p2->exp);
228             p2 = p2->link;
229         }
230         else
231         {
232             result.insertTerm(p1->coef - p2->coef, p1->exp);
233             p1 = p1->link;
234             p2 = p2->link;
235         }
236     }
237     return result;
238 }
```

- `operator* (const Polynomial& b) const`：計算兩個多項式的積。

```
240 // 乘法運算
241 Polynomial Polynomial::operator*(const Polynomial& b) const
242 {
243     Polynomial result;
244     Term* p1 = head->link;
245
246     while (p1 != head)
247     {
248         Term* p2 = b.head->link;
249         while (p2 != b.head)
250         {
251             result.insertTerm(p1->coef * p2->coef, p1->exp + p2->exp);
252             p2 = p2->link;
253         }
254         p1 = p1->link;
255     }
256     return result;
257 }
```

- `float Eval(float x) const`：計算多項式在某一給定變數值下的結果。

```
259 // 計算多項式值
260 float Polynomial::Eval(float x) const
261 {
262     float result = 0;
263     Term* current = head->link;
264
265     while (current != head)
266     {
267         result += current->coef * pow(x, current->exp);
268         current = current->link;
269     }
270     return result;
271 }
```

## HW.cpp 的 main.cpp

```
317  int main()
318  {
319      Polynomial p1, p2;
320
321      cout << "輸入第一個多項式:" << endl;
322      cin >> p1;
323      cout << endl << "輸入第二個多項式:" << endl;
324      cin >> p2;
325
326      Polynomial sum = p1 + p2;
327      Polynomial diff = p1 - p2;
328      Polynomial prod = p1 * p2;
329
330      cout << "第一個多項式: " << p1 << endl;
331      cout << "第二個多項式: " << p2 << endl;
332      cout << "和: " << sum << endl;
333      cout << "差: " << diff << endl;
334      cout << "積: " << prod << endl;
335
336      float x;
337      cout << "計算多項式值 (輸入 x 的值): ";
338      cin >> x;
339      cout << "p1(" << x << ") = " << p1.Eval(x) << endl;
340
341      return 0;
342  }
```

### 三. 效能分析

#### 時間複雜度

##### 1. 加法與減法、乘法：

- 加法：  $T(P) = O(n+m)$ ，
- 減法：  $T(P) = O(n+m)$ ，
- 乘法：  $T(P) = O(n \times m)$ ，

其中  $n$ 、 $m$  分別為兩多項式的項目數。

##### 2. 計算多項式值 (Eval)：

- $T(P) = O(n)$ ， $n$  為當前多項式中的非零項數量。

遍歷多項式的每一項，進行指數運算和乘法運算。

##### 3. 插入單項式 (insertTerm)：

- $T(P) = O(n)$ ， $n$  為當前多項式中的非零項數量。

插入時遍歷多項式以找到適合位置，並根據需要進行合併。最壞情況是插入到尾部。

##### 4. 輸入輸出運算子 ( $>>$ 和 $<<$ )：

- 輸入：  $T(P) = O(n^2)$ ， $n$  為需要插入的項數。
- 輸出：  $T(P) = O(n)$ ， $n$  為當前多項式中的非零項數量。

## 空間複雜度

### 1. 單個多項式 (Term) :

- 每個多項式的節點需要儲存 `coef`, `exp` 和 `link` , 單個節點 :

$$S(P) = O(1) \text{ 。}$$

- 如果多項式有  $n$  項 , 整個多項式 :

$$S(P) = O(n) \text{ 。}$$

### 2. 加法與減法、乘法 : ( 最多/最壞情況 )

- 加法 :  $S(P) = O(n+m)$  ,
- 減法 :  $S(P) = O(n+m)$  ,
- 乘法 :  $S(P) = O(n \times m)$  ,

其中  $n$ 、 $m$  分別為兩多項式的項目數。

### 3. 複製建構子與賦值運算

$$S(P) = O(n) \text{ 。}$$

需要為每個節點分配空間 ,  $S(P)$ 與原多項式 1.相同。

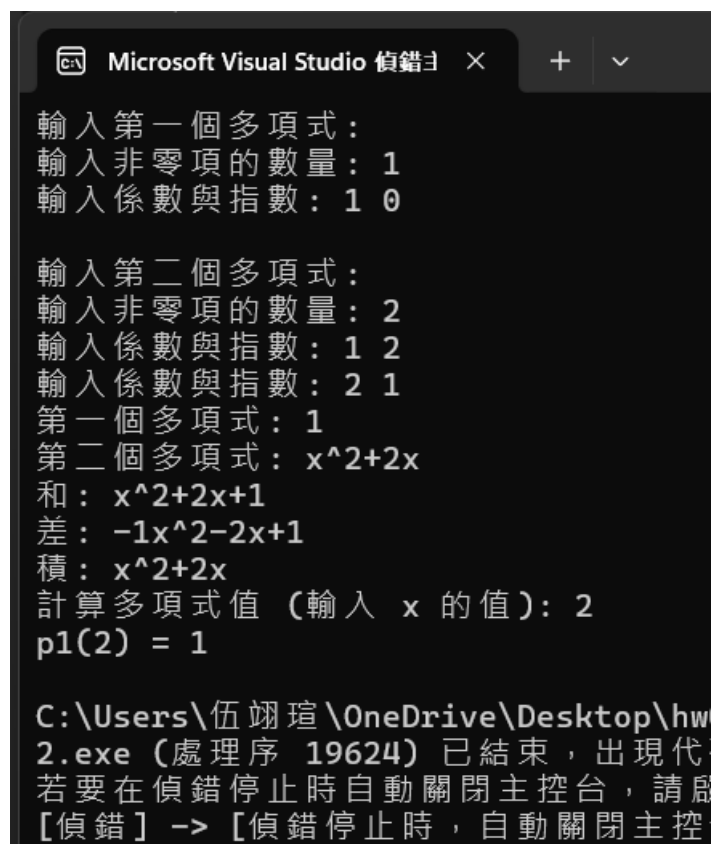
### 4. 可用空間鏈結串列

$$S(P) = O(k) \text{ , } k \text{ 為釋放的節點數。}$$

可用空間儲存已釋放的節點 ,  $S(P)$ 與  $k$  成正比。



## 四. 測試與驗證



```
Microsoft Visual Studio 偵錯  ×  +  ▾  
輸入第一個多項式：  
輸入非零項的數量：1  
輸入係數與指數：1 0  
  
輸入第二個多項式：  
輸入非零項的數量：2  
輸入係數與指數：1 2  
輸入係數與指數：2 1  
第一個多項式：1  
第二個多項式：x^2+2x  
和：x^2+2x+1  
差：-1x^2-2x+1  
積：x^2+2x  
計算多項式值（輸入 x 的值）：2  
p1(2) = 1  
  
C:\Users\伍翊瑄\OneDrive\Desktop\hw  
2.exe (處理序 19624) 已結束，出現代  
若要在偵錯停止時自動關閉主控台，請啟  
[偵錯] -> [偵錯停止時，自動關閉主控
```

## 驗證

## 五. 效能量測

### 1. 空間效能測試

- 測試內容：
  - 測試多項式運算過程中的記憶體使用量。
  - 特別關注乘法運算，因為生成的結果可能大幅增加項數。
- 測試方法：記錄高峰記憶體使用量，確認是否有記憶體洩漏。

### 2. 邊界條件測試

- 測試內容：
  - 空多項式運算（例如所有係數為 0）。
  - 僅包含常數項的多項式（例如  $3x^0$ ）。
  - 高次多項式運算（例如指數  $10^6$ ）。
  - 大量項數的多項式（例如  $n, m > 10^5$ ）。
- 測試結果範例：
  - 空多項式：正確輸出。
  - 高次多項式：執行時間隨  $n$  增加呈線性或二次增長。

※資料輔助：ChatGPT

## 六. 心得討論

### 為什麼這樣設計這個程式

- 環狀鏈結串列的選擇：
  - 環狀結構能方便地處理多項式的循環操作（例如遍歷、插入）。
  - 使用頭節點簡化了邊界條件的處理，插入和刪除操作不需要特別處理空多項式的情況。
- 節點復用（可用空間機制）：
  - 使用 avail 鏈結串列，降低頻繁記憶體分配和釋放的成本，提高效能。
  - 減少記憶體碎片，特別是當多項式運算頻繁且項數變化較大時。
- 按指數降序排列的插入策略：
  - 保證了多項式在任何時候都是有序的，便於後續運算（加法、減法、輸出）。
  - 插入過程中自動合併同類項，減少後續運算的額外步驟。
  -

- 靈活的運算支援：
  - 支援加、減、乘等運算，並實現了運算符重載，使得程式使用時語法直觀且易於理解。

## 2. 優點

- 模組化設計：
  - 各功能如插入、加法、減法、乘法和求值獨立實現，便於維護和擴展。
- 效率高：
  - 利用鏈結串列和節點復用，減少了記憶體分配釋放的開銷。
- 易用性：
  - 使用運算符重載和直觀的輸入/輸出設計，讓使用者操作簡單。

※資料輔助：ChatGPT