

## 解題說明：

利用這段程式可以算出多項式的加法跟乘法以及值。

## 效能分析：

$m$  和  $n$  分別是兩個多項式的非零項數量。

空間複雜度：單個多項式的儲存空間： $O(n)$

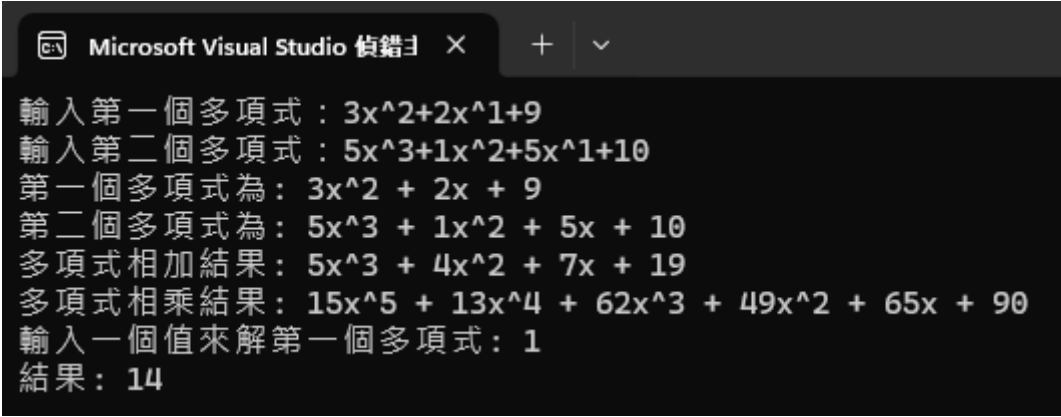
運算時的額外空間：

- 相加的暫存空間為  $O(m+n)O(m+n)$ 。
- 相乘的暫存空間為  $O(m \times n)O(m \times n)$ 。

時間複雜度：

- 乘法： $O(m+n)O(m+n)$
- 加法： $O(m \times n)$
- Eval： $O(m)$

## 結果展示：



```
Microsoft Visual Studio 偵錯 1  X + v
輸入第一個多項式：3x^2+2x^1+9
輸入第二個多項式：5x^3+1x^2+5x^1+10
第一個多項式為：3x^2 + 2x + 9
第二個多項式為：5x^3 + 1x^2 + 5x + 10
多項式相加結果：5x^3 + 4x^2 + 7x + 19
多項式相乘結果：15x^5 + 13x^4 + 62x^3 + 49x^2 + 65x + 90
輸入一個值來解第一個多項式：1
結果：14
```

## 心得：

這次的作業需要用到很多的類別，而我在這部分還不夠熟悉，導致我花費了需多時間在這次的作業上。

## 申論及開發報告：

### 需求分析

- 開發一個多項式處理系統，支持以下功能：
  1. 使用直觀的輸入格式（如  $3x^2 + 2x + 1$ ）。
  2. 支持基本運算（加法、乘法）。
  3. 計算多項式在某點的值。
  4. 提供動態記憶體支持，適應不同大小的多項式。

### 設計與實現

- 設計多項式結構：
  - 使用 **Term** 類別表示單項式，包括係數和指數。
  - 使用 **Polynomial** 類別管理多項式，內部以動態陣列儲存多項式的所有項。
- 操作功能：
  - 重載  $>>$  和  $<<$  運算子，實現自然語義的多項式輸入與輸出。
  - 使用雙指針法實現高效的多項式加法。
  - 提供多項式相乘功能並合併同類項。

## 程式片段：

```
1  #include <iostream>
2  #include <sstream>
3  #include <cmath> // 用於 pow 函數
4  using namespace std;
5
6  // 定義 Term 類別，表示多項式中的單一項
7  class Term {
8  public:
9      float coef; // 係數
10     int exp;    // 指數
11
12     Term(float c = 0, int e = 0) : coef(c), exp(e) {}
13 };
14
15 // 定義 Polynomial 類別，表示整個多項式
16 class Polynomial {
17 private:
18     Term* termArray; // 動態陣列存放多項式的每一項
19     int capacity;    // 陣列的容量
20     int terms;       // 非零項的數量
21
22     // 擴展動態陣列的容量
23     void resize(int newCapacity) {
24         Term* newArray = new Term[newCapacity];
25         for (int i = 0; i < terms; ++i) {
26             newArray[i] = termArray[i];
27         }
28         delete[] termArray;
29         termArray = newArray;
30         capacity = newCapacity;
31     }
32
33 public:
```

```
32
33 public:
34     // 建構子，初始化為空多項式
35     Polynomial() : termArray(new Term[1]), capacity(1), terms(0) {}
36
37     // 解構子，釋放動態記憶體
38     ~Polynomial() {
39         delete[] termArray;
40     }
41
42     // 輸入多項式
43     friend istream& operator>>(istream& is, Polynomial& poly) {
44         string input;
45         getline(is, input); // 讀取整行輸入
46         stringstream ss(input);
47         float coef;
48         int exp;
49         char x, caret;
50
51         poly.terms = 0; // 重設非零項數量
52         while (ss >> coef) {
53             if (poly.terms >= poly.capacity) {
54                 poly.resize(poly.capacity * 2); // 擴展容量
55             }
56
57             // 嘗試解析 'x^' 的部分
58             if (ss.peek() == 'x') {
59                 ss >> x; // 跳過 'x'
60                 if (ss.peek() == '^') {
61                     ss >> caret >> exp; // 讀取 '^' 和指數
62                 }
63             }
64         }
65     }
66 }
```

```
63     else {
64         exp = 1; // 若無 '^', 指數為 1
65     }
66 }
67 else {
68     exp = 0; // 若無 'x', 則為常數項
69 }
70 poly.termArray[poly.terms++] = Term(coef, exp);
71
72 // 跳過可能存在的 '+' 符號
73 if (ss.peek() == '+') {
74     ss.ignore();
75 }
76 }
77 return is;
78 }
79
80 // 輸出多項式
81 friend ostream& operator<<(ostream& os, const Polynomial& poly) {
82     for (int i = 0; i < poly.terms; ++i) {
83         if (i > 0 && poly.termArray[i].coef > 0) os << " + ";
84         os << poly.termArray[i].coef;
85         if (poly.termArray[i].exp > 0) {
86             os << "x";
87             if (poly.termArray[i].exp > 1) {
88                 os << "^" << poly.termArray[i].exp;
89             }
90         }
91     }
92     return os;
93 }
```

```
95 // 多項式相加
96 Polynomial PolynomialADD(const Polynomial& poly) const {
97     Polynomial result;
98     int i = 0, j = 0;
99
100     // 合併兩個多項式的非零項
101     while (i < terms && j < poly.terms) {
102         if (termArray[i].exp == poly.termArray[j].exp) {
103             float sumCoef = termArray[i].coef + poly.termArray[j].coef;
104             if (sumCoef != 0) {
105                 result.addTerm(sumCoef, termArray[i].exp);
106             }
107             ++i; ++j;
108         }
109         else if (termArray[i].exp > poly.termArray[j].exp) {
110             result.addTerm(termArray[i].coef, termArray[i].exp);
111             ++i;
112         }
113         else {
114             result.addTerm(poly.termArray[j].coef, poly.termArray[j].exp);
115             ++j;
116         }
117     }
118
119     // 加入剩下的項
120     while (i < terms) {
121         result.addTerm(termArray[i].coef, termArray[i].exp);
122         ++i;
123     }
124     while (j < poly.terms) {
125         result.addTerm(poly.termArray[j].coef, poly.termArray[j].exp);
126         ++j;
127     }
128 }
```

```
124     while (j < poly.terms) {
125         result.addTerm(poly.termArray[j].coef, poly.termArray[j].exp);
126         ++j;
127     }
128     return result;
129 }
130
131 // 多項式相乘
132 Polynomial PolynomialMult(const Polynomial& poly) const {
133     Polynomial result;
134     for (int i = 0; i < terms; ++i) {
135         for (int j = 0; j < poly.terms; ++j) {
136             float newCoef = termArray[i].coef * poly.termArray[j].coef;
137             int newExp = termArray[i].exp + poly.termArray[j].exp;
138             result.addTerm(newCoef, newExp);
139         }
140     }
141     result.combineLikeTerms(); // 合併同類項
142     return result;
143 }
144
145 // 計算多項式在某個值的結果
146 float Eval(float x) const {
147     float result = 0;
148     for (int i = 0; i < terms; ++i) {
149         result += termArray[i].coef * pow(x, termArray[i].exp);
150     }
151     return result;
```

```
154 // 新增一項到多項式中
155 void addTerm(float coef, int exp) {
156     for (int i = 0; i < terms; ++i) {
157         if (termArray[i].exp == exp) {
158             termArray[i].coef += coef;
159             if (termArray[i].coef == 0) { // 移除係數為 0 的項
160                 for (int j = i; j < terms - 1; ++j) {
161                     termArray[j] = termArray[j + 1];
162                 }
163                 --terms;
164             }
165             return;
166         }
167     }
168     if (terms >= capacity) {
169         resize(capacity * 2);
170     }
171     termArray[terms++] = Term(coef, exp);
172 }
173
174 // 合併同類項
175 void combineLikeTerms() {
176     for (int i = 0; i < terms; ++i) {
177         for (int j = i + 1; j < terms; ++j) {
178             if (termArray[i].exp == termArray[j].exp) {
179                 termArray[i].coef += termArray[j].coef;
180                 for (int k = j; k < terms - 1; ++k) {
181                     termArray[k] = termArray[k + 1];
182                 }
183                 --terms;
184                 --j;
185             }
186         }
187     }
```



```
192  int main() {  
193      Polynomial p1, p2;  
194      cout << "輸入第一個多項式：";  
195      cin >> p1;  
196      cout << "輸入第二個多項式：";  
197      cin >> p2;  
198  
199      Polynomial sum = p1.PolynomialADD(p2);  
200      Polynomial product = p1.PolynomialMult(p2);  
201  
202      cout << "第一個多項式為：" << p1 << endl;  
203      cout << "第二個多項式為：" << p2 << endl;  
204      cout << "多項式相加結果：" << sum << endl;  
205      cout << "多項式相乘結果：" << product << endl;  
206  
207      float evalValue;  
208      cout << "輸入一個值來解第一個多項式：";  
209      cin >> evalValue;  
210      cout << "結果：" << p1.Eval(evalValue) << endl;  
211  
212      return 0;  
213  }  
214
```