

解題說明：

利用這段程式可以算出多項式的加法跟乘法。

效能分析：

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：

- 加法： $O(n_1+n_2)$
- 乘法： $O(n_1 \times n_2)$

時間複雜度：

- 乘法和加法： $O(n_1 \times n_2)$
- 新增項次： $O(k)$ k 為當前項次

結果展示：

```
輸入格式 ax^n2+bx^n1+cx^n0+d ( 若為常數，可省略 x^0 )
p1 : 3x^3+2x^2+3x+4
p2 : 5x+1
p1 = 3x^3+2x^2+3x^1+4
p2 = 5x^1+1

-----
新增項目
請輸入項次的係數：2
請輸入項次的指數：4
新增前的多項式 = 3x^3+2x^2+3x^1+4
新增後的多項式 = 3x^3+2x^2+3x^1+4+2x^4

-----
求值
請輸入要帶入的值：1
多項式：3x^3+2x^2+3x^1+4+2x^4
結果 = 14

-----
相加測試
(3x^3+2x^2+3x^1+4+2x^4) + (5x^1+1) = 3x^3+2x^2+8x^1+5+2x^4

-----
相乘測試
(3x^3+2x^2+3x^1+4+2x^4) * (5x^1+1) = 17x^4+13x^3+17x^2+23x^1+4+10x^5
```

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。

程式片段：

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  class Polynomial;
6
7  class Term {
8  |   friend Polynomial;
9  |   friend ostream& operator<<(ostream& os, const Polynomial& p);
10 | private:
11 |   float coef; // 係數
12 |   int exp;    // 指數
13 | };
14
15 class Polynomial { // 多項式類別
16 |   friend ostream& operator<<(ostream& os, const Polynomial& p);
17 |   friend istream& operator>>(istream& input, Polynomial& p);
18 | private:
19 |   Term* termArray; // 儲存非零項的陣列
20 |   int capacity;    // 陣列容量
21 |   int terms;       // 非零項數量
22 |
23 | public:
24 |   Polynomial(); // 預設建構函式：p(x) = 0
25 |   Polynomial Add(Polynomial poly); // 多項式相加
26 |   Polynomial Mult(Polynomial poly); // 多項式相乘
27 |   float Eval(float f);              // 多項式求值
28 |   void NewTerm(const float newCoef, const int newExp); // 新增項目
29 | };
```

```
32  ~ Polynomial::Polynomial() : capacity(2), terms(0) {
33      |      this->termArray = new Term[capacity];
34      |  }
35
36      // 新增項目函式
37  ~ void Polynomial::NewTerm(const float newCoef, const int newExp) {
38      ~ |      if (this->terms == this->capacity) { // 空間不足時擴展容量
39      |      |      this->capacity *= 2;
40      |      |      Term* temp = new Term[this->capacity];
41      |      |      copy(this->termArray, this->termArray + terms, temp);
42      |      |      delete[] this->termArray;
43      |      |      this->termArray = temp;
44      |      |  }
45      |      this->termArray[this->terms].coef = newCoef; // 設定係數
46      |      this->termArray[this->terms++].exp = newExp; // 設定指數
47      |  }
48
49      // 多項式相加
50  ~ Polynomial Polynomial::Add(Polynomial poly) {
51      |      Polynomial res;
52      |      int* loc = new int[poly.terms + this->terms]; // 儲存指數
53      |      float* data = new float[poly.terms + this->terms]; // 儲存係數
54      |      int use_len = 0; // 使用長度
55
56      |      // 將當前多項式的項加入陣列
57      ~ |      for (int i = 0; i < this->terms; i++) {
58      |      |      int t = -1; // -1 表示未找到
59      ~ |      |      for (int j = 0; j < use_len; j++) {
60      ~ |      |      |      if (this->termArray[i].exp == loc[j]) {
61      |      |      |      |      t = j;
62      |      |      |      |      break;
63      |      |      |      }
```

```
64     }
65     if (t == -1) { // 新的指數
66         loc[use_len] = this->termArray[i].exp;
67         data[use_len++] = this->termArray[i].coef;
68     }
69     else {
70         data[t] += this->termArray[i].coef; // 相同指數合併係數
71     }
72 }
73
74 // 處理第二個多項式
75 for (int i = 0; i < poly.termArray[i].exp; i++) {
76     int t = -1;
77     for (int j = 0; j < use_len; j++) {
78         if (poly.termArray[i].exp == loc[j]) {
79             t = j;
80             break;
81         }
82     }
83     if (t == -1) {
84         loc[use_len] = poly.termArray[i].exp;
85         data[use_len++] = poly.termArray[i].coef;
86     }
87     else {
88         data[t] += poly.termArray[i].coef;
89     }
90 }
91
92 // 將結果存入 res
93 for (int i = 0; i < use_len; i++)
94     res.NewTerm(data[i], loc[i]);
95
96 delete[] loc;
```

```
97     delete[] data;
98
99     return res;
100 }
101
102 // 多項式相乘
103 Polynomial Polynomial::Mult(Polynomial poly) {
104     Polynomial res;
105     int* loc = new int[poly.terms * this->terms];
106     float* data = new float[poly.terms * this->terms];
107     int use_len = 0;
108
109     for (int i = 0; i < this->terms; i++) {
110         for (int j = 0; j < poly.terms; j++) {
111             float t_coef = this->termArray[i].coef * poly.termArray[j].coef; // 係數相乘
112             int t_exp = this->termArray[i].exp + poly.termArray[j].exp; // 指數相加
113
114             int t = -1;
115             for (int k = 0; k < use_len; k++) {
116                 if (t_exp == loc[k]) {
117                     t = k;
118                     break;
119                 }
120             }
121             if (t == -1) { // 新的項目
122                 loc[use_len] = t_exp;
123                 data[use_len++] = t_coef;
124             }
125         }
126     }
127 }
```

```
125     } else {
126         data[t] += t_coef;
127     }
128 }
129 }
130
131 for (int i = 0; i < use_len; i++)
132     res.NewTerm(data[i], loc[i]);
133
134 delete[] loc;
135 delete[] data;
136
137 return res;
138 }
139
140 // 多項式求值
141 float Polynomial::Eval(float f) {
142     float res = 0.0f;
143     for (int i = 0; i < this->terms; i++) {
144         float temp = this->termArray[i].coef;
145         for (int j = 0; j < this->termArray[i].exp; j++)
146             temp *= f; // 計算次方
147         res += temp;
148     }
149     return res;
150 }
151
152 // 輸入運算子
153 istream& operator>>(istream& input, Polynomial& p) {
154     float coef = 0.0f;
155     int exp = 0;
156     char ch;
```

```
157     bool isNegative = false;
158
159     while (true) {
160         input >> coef;
161         if (isNegative) coef = -coef;
162         isNegative = false;
163
164         input.get(ch);
165         if (ch == 'x') {
166             input.get(ch);
167             if (ch == '^') {
168                 input >> exp;
169             }
170             else {
171                 exp = 1;
172                 input.unget();
173             }
174         }
175         else {
176             exp = 0;
177             input.unget();
178         }
179
180         p.NewTerm(coef, exp);
181
182         input.get(ch);
183         if (ch == '\n' || input.eof()) break;
184         if (ch == '-') {
185             isNegative = true;
186         }
187         else if (ch != '+') {
188             cout << "輸入格式錯誤！請使用正確的多項式格式。\\n";
189             break;
```

```

190     }
191     }
192     return input;
193 }
194
195 // 輸出運算子
196 ostream& operator<<(ostream& output, const Polynomial& p) {
197     for (int i = 0; i < p.terms; i++) {
198         if (p.termArray[i].exp == 0) {
199             output << (p.termArray[i].coef < 0 ? "" : "+") << p.termArray[i].coef;
200         }
201         else {
202             if (i == 0) {
203                 output << p.termArray[i].coef << "x^" << p.termArray[i].exp;
204             }
205             else {
206                 output << (p.termArray[i].coef < 0 ? "" : "+") << p.termArray[i].coef << "x^" << p.termArray[i].exp;
207             }
208         }
209     }
210     return output;
211 }
212
213 int main() {
214     cout << "輸入格式 ax^n2+bx^n1+cx^n0+d (若為常數, 可省略 x^0) \n";
215     Polynomial p1, p2;
216     cout << "p1: ";
217     cin >> p1;
218     cout << "p2: ";
219     cin >> p2;

```

```

220
221     cout << "p1 = " << p1 << endl;
222     cout << "p2 = " << p2 << endl;
223
224     cout << "-----\n";
225     cout << "新增項目\n";
226     float t_coef = 0.0f;
227     int t_exp = 0;
228     cout << "請輸入項次的係數: ";
229     cin >> t_coef;
230     cout << "請輸入項次的指數: ";
231     cin >> t_exp;
232     cout << "新增前的多項式 = " << p1 << endl;
233     p1.NewTerm(t_coef, t_exp);
234     cout << "新增後的多項式 = " << p1 << endl;
235
236     cout << "-----\n";
237     cout << "求值\n";
238     float f = 0.0f;
239     cout << "請輸入要帶入的值: ";
240     cin >> f;
241     cout << "多項式: " << p1 << endl;
242     cout << "結果 = " << p1.Eval(f) << endl;
243
244     cout << "-----\n";
245     cout << "相加測試\n";
246     cout << "(" << p1 << ") + (" << p2 << ") = " << p1.Add(p2) << endl;
247
248     cout << "-----\n";
249     cout << "相乘測試\n";
250     cout << "(" << p1 << ") * (" << p2 << ") = " << p1.Mult(p2) << endl;
251
252     return 0;

```