解題思路與程式碼:

```cpp
#include<iostream>
#include <cmath>
using namespace std;
class Node {
public:
    float coef;
    int exp;
    Node* link;//指向下一個位子
};
class Polynomial {
private:
    Node* hand;
public:
    Node* getHand() const { return hand; } // 返回 hand 指針
    Polynomial();//建構值
    ~Polynomial();//解構值
    Polynomial(const Polynomial& a);//複製建構值
    void Added(float coef, int exp);//新增函數
    Polynomial& operator=(const Polynomial& a);//等號
    Polynomial operator+(const Polynomial& b);//加法
    Polynomial operator-(const Polynomial& b);//減法
    Polynomial operator*(const Polynomial& b);//乘法
    float Eval(float x)const;//給值代入
    friend ostream& operator<<(ostream& os, const Polynomial& b);
    friend istream& operator>>(istream& is, Polynomial& b);
};

//建構子初始化指標
Polynomial::Polynomial() : hand(nullptr) {}

//解構子 釋放記憶體
Polynomial::~Polynomial()
{
    Node* temp;
    while (hand != nullptr)
    {
        temp = hand;
        hand = hand->link;
        delete temp;
    }
}

// 複製建構子  複製另一個多項式
Polynomial::Polynomial(const Polynomial& a) {
    hand = nullptr;
    Node* temp = a.hand;
    while (temp != nullptr) {
        Added(temp->coef, temp->exp);
```

```
49          temp = temp->link;
50      }
51  }
52
53  // 新增節點函數
54  void Polynomial::Added(float coef, int exp) {
55      Node* newNode = new Node;
56      newNode->coef = coef;
57      newNode->exp = exp;
58      newNode->link = nullptr;
59
60      if (hand == nullptr || hand->exp < exp) {
61          newNode->link = hand;
62          hand = newNode;
63      }
64      else {
65          Node* temp = hand;
66          while (temp->link != nullptr && temp->link->exp >= exp) {
67              temp = temp->link;
68          }
69          if (temp->exp == exp) {
70              temp->coef += coef;  // 如果指數相同，合併係數
71              delete newNode;   // 釋放多餘的節點
72          }
```

```
73          else {
74              newNode->link = temp->link;
75              temp->link = newNode;
76          }
77      }
78  }
79
80  //多項式的相加
81  Polynomial Polynomial::operator+(const Polynomial& b) {
82      Polynomial result;
83      Node* tempA = hand;
84      Node* tempB = b.getHand();
85
86      while (tempA != nullptr && tempB != nullptr) {
87          if (tempA->exp > tempB->exp) {
88              result.Added(tempA->coef, tempA->exp);
89              tempA = tempA->link;
90          }
91          else if (tempA->exp < tempB->exp) {
92              result.Added(tempB->coef, tempB->exp);
93              tempB = tempB->link;
94          }
95          else {
96              result.Added(tempA->coef + tempB->coef, tempA->exp);
```

```
97                  tempA = tempA->link;
98                  tempB = tempB->link;
99              }
100         }
101
102         while (tempA != nullptr) {
103             result.Added(tempA->coef, tempA->exp);
104             tempA = tempA->link;
105         }
106
107         while (tempB != nullptr) {
108             result.Added(tempB->coef, tempB->exp);
109             tempB = tempB->link;
110         }
111
112         return result;
113     }
114
115     //多項式的相減
116     Polynomial Polynomial::operator-(const Polynomial& b) {
117         Polynomial result;
118         Node* tempA = hand;
119         Node* tempB = b.getHand();
120
```

```
121         while (tempA != nullptr && tempB != nullptr) {
122             if (tempA->exp > tempB->exp) {
123                 result.Added(tempA->coef, tempA->exp);
124                 tempA = tempA->link;
125             }
126             else if (tempA->exp < tempB->exp) {
127                 result.Added(-tempB->coef, tempB->exp);
128                 tempB = tempB->link;
129             }
130             else {
131                 result.Added(tempA->coef - tempB->coef, tempA->exp);
132                 tempA = tempA->link;
133                 tempB = tempB->link;
134             }
135         }
136
137         while (tempA != nullptr) {
138             result.Added(tempA->coef, tempA->exp);
139             tempA = tempA->link;
140         }
141
142         while (tempB != nullptr) {
143             result.Added(-tempB->coef, tempB->exp);
144             tempB = tempB->link;
```

```
145         }
146
147         return result;
148     }
149
150     Polynomial Polynomial::operator*(const Polynomial& b) {
151         Polynomial result;
152         Node* tempA = hand;   // 第一個多項式的指標
153         Node* tempB = b.getHand();// 第二個多項式的指標
154
155         while (tempA != nullptr) {
156             tempB = b.getHand();  // 重置 tempB 為第二個多項式的頭節點
157             while (tempB != nullptr) {
158                 // 相乘：係數相乘，指數相加
159                 float newCoef = tempA->coef * tempB->coef;
160                 int newExp = tempA->exp + tempB->exp;
161
162                 // 將結果加入到結果多項式
163                 result.Added(newCoef, newExp);
164
165                 tempB = tempB->link; // 移動到第二個多項式的下一個項目
166             }
167
168             tempA = tempA->link; // 移動到第一個多項式的下一個項目
```

```
169         }
170
171         return result;
172     }
173
174     Polynomial& Polynomial::operator=(const Polynomial& a) {
175         if (this == &a) {
176             return *this; // 防止自我賦值
177         }
178
179         // 清除當前多項式的節點
180         Node* temp;
181         while (hand != nullptr) {
182             temp = hand;
183             hand = hand->link;
184             delete temp;
185         }
186
187         hand = nullptr;
188
189         // 複製 a 的節點
190         Node* tempA = a.hand;
191         while (tempA != nullptr) {
192             Added(tempA->coef, tempA->exp);
```

```
193             tempA = tempA->link;
194         }
195
196         return *this;
197     }
198
199
200     // Evaluate the polynomial at a given value of x
201     float Polynomial::Eval(float x) const {
202         float result = 0.0f;
203         Node* temp = hand;
204         while (temp != nullptr) {
205             result += temp->coef * pow(x, temp->exp);
206             temp = temp->link;
207         }
208         return result;
209     }
210
211     // Output stream overload to print the polynomial
212     ostream& operator<<(ostream& os, const Polynomial& b) {
213         Node* temp = b.hand;
214         bool first = true;
215         while (temp != nullptr) {
216             if (first) {
```

```
217                 first = false;
218             }
219             else {
220                 os << " + ";
221             }
222             os << temp->coef << "x^" << temp->exp;
223             temp = temp->link;
224         }
225         return os;
226     }
227
228     // Input stream overload to read a polynomial
229     istream& operator>>(istream& in, Polynomial& b) {
230         // 清空原來的多項式資料
231             int num;
232             cout << "請輸入有幾項指數: ";
233             in >> num;
234             for (int i = 0; i < num; i++)
235             {//一個一個存
236                 cout << "請輸入系數與指數(EX. 2 2 3 1 1 0) ";
237                 int exp;
238                 float coef;
239                 in >> coef >> exp;
240                 b.Added(coef, exp);//新增
```

HW3

```
241             }
242             return in;
243     }
244
```

執行解果:

```
請輸入第一個多項式
請輸入有幾項指數: 2
請輸入系數與指數(EX. 2 2 3 1 1 0) 1 2 1 0
請輸入系數與指數(EX. 2 2 3 1 1 0) 請輸入第二個多項式
請輸入有幾項指數: 2
請輸入系數與指數(EX. 2 2 3 1 1 0) 3 3 2 1
請輸入系數與指數(EX. 2 2 3 1 1 0) Polynomial 1: 1x^2 + 1x^0
Polynomial 2: 3x^3 + 2x^1
Sum of the polynomials: 3x^3 + 1x^2 + 2x^1 + 1x^0
Difference of the polynomials: -3x^3 + 1x^2 + -2x^1 + 1x^0
Product of the polynomials: 3x^5 + 5x^3 + 2x^1
Enter a value for x to evaluate the polynomials: 1
Polynomial 1 evaluated at x = 1: 2
Polynomial 2 evaluated at x = 1: 5
```