

解題說明:

(a) `istream& operator>>(istream& is, Polynomial& x)`：用戶輸入多項式的係數和指數，初始化多項式對象。

(b) `ostream& operator<<(ostream& os, const Polynomial& x)`：將多項式對象轉換為外部表示形式，輸出到螢幕。

(c) `Polynomial::Polynomial(const Polynomial& a)`：複製構造函數，將一個多項式初始化為另一個多項式。

(d) `const Polynomial& Polynomial::operator=(const Polynomial& a) const`：賦值運算子，將一個多項式賦值給另一個多項式。

(e) `Polynomial::~~Polynomial()`：析構函數，釋放多項式的所有節點。

(f) `Polynomial operator+(const Polynomial& b) const`：多項式相加，返回和。

(g) `Polynomial operator-(const Polynomial& b) const`：多項式相減，返回差。

(h) `Polynomial operator*(const Polynomial& b) const`：多項式相乘，返回積。

(i) `float Polynomial::Evaluate(float x) const`：在給定的 x 值處評估多項式，返回結果。

Algorithm Design & Programming :

```
1  #include <iostream>
2  #include <cmath>
3  #include <chrono>
4  using namespace std;
5
6  class Polynomial {
7  private:
8      struct Node {
9          int coef;
10         int exp;
11         Node* link;
12     };
13
14     Node* head;
15
16 public:
17     Polynomial();
18     ~Polynomial();
19     Polynomial(const Polynomial& a);
20     const Polynomial& operator=(const Polynomial& a);
21
22     friend std::istream& operator>>(std::istream& is, Polynomial& x);
23     friend std::ostream& operator<<(std::ostream& os, const Polynomial& x);
24     Polynomial operator+(const Polynomial& b) const;
25     Polynomial operator-(const Polynomial& b) const;
26     Polynomial operator*(const Polynomial& b) const;
27
28     float Evaluate(float x) const;
29 };
30
31 Polynomial::Polynomial() {
32     head = new Node;
33     head->link = head;
34 }
35
36 Polynomial::~Polynomial() {
37     Node* temp;
38     Node* current = head->link;
39
40     while (current != head) {
41         temp = current;
42         current = current->link;
43         delete temp;
44     }
45
46     delete head;
47 }
48
49 Polynomial::Polynomial(const Polynomial& a) {
50     head = new Node;
51     head->link = head;
52
53     Node* current = a.head->link;
54
55     while (current != a.head) {
56         Node* newNode = new Node;
57         newNode->coef = current->coef;
58         newNode->exp = current->exp;
59
60         newNode->link = head->link;
61         head->link = newNode;
62
63         current = current->link;
64     }
65 }
66
67 const Polynomial& Polynomial::operator=(const Polynomial& a) {
68     if (this != &a) {
69         Node* temp;
70         Node* current = head->link;
71
72         while (current != head) {
73             temp = current;
74             current = current->link;
75             delete temp;
76         }
77
78         Node* sourceCurrent = a.head->link;
```

```

81
82     while (sourceCurrent != a.head) {
83         Node* newNode = new Node;
84         newNode->coef = sourceCurrent->coef;
85         newNode->exp = sourceCurrent->exp;
86
87         newNode->link = head->link;
88         head->link = newNode;
89
90         sourceCurrent = sourceCurrent->link;
91     }
92 }
93
94 return *this;
95 }
96
97
98 std::istream& operator>>(std::istream& is, Polynomial& x) {
99     int n, coef, exp;
100     std::cout << "Enter the number of terms: ";
101     is >> n;
102
103     x.head->link = x.head;
104
105     for (int i = 0; i < n; ++i) {
106         std::cout << "Enter coefficient and exponent for term " << i + 1 << ": ";
107         is >> coef >> exp;
108
109         Polynomial::Node* newNode = new Polynomial::Node;
110         newNode->coef = coef;
111         newNode->exp = exp;
112
113         newNode->link = x.head->link;
114         x.head->link = newNode;
115     }
116
117     return is;
118 }
119
120
121 std::ostream& operator<<(std::ostream& os, const Polynomial& x) {
122     Polynomial::Node* current = x.head->link;
123
124     while (current != x.head) {
125         os << current->coef << "x^" << current->exp;
126
127         current = current->link;
128         if (current != x.head) {
129             os << " + ";
130         }
131     }
132
133     return os;
134 }
135
136 Polynomial Polynomial::operator+(const Polynomial& b) const {
137     Polynomial result;
138     Node* currentA = head->link;
139     Node* currentB = b.head->link;
140
141     while (currentA != head && currentB != b.head) {
142         if (currentA->exp > currentB->exp) {
143             result.head->link = new Node{ currentA->coef, currentA->exp, result.head->link };
144             currentA = currentA->link;
145         }
146         else if (currentA->exp < currentB->exp) {
147             result.head->link = new Node{ currentB->coef, currentB->exp, result.head->link };
148             currentB = currentB->link;
149         }
150         else {
151             int sumCoef = currentA->coef + currentB->coef;
152             if (sumCoef != 0) {
153                 result.head->link = new Node{ sumCoef, currentA->exp, result.head->link };
154             }
155             currentA = currentA->link;
156             currentB = currentB->link;
157         }
158     }
159
160     while (currentA != head) {
161         result.head->link = new Node{ currentA->coef, currentA->exp, result.head->link };

```

```

162         currentA = currentA->link;
163     }
164
165     while (currentB != b.head) {
166         result.head->link = new Node{ currentB->coef, currentB->exp, result.head->link };
167         currentB = currentB->link;
168     }
169
170     return result;
171 }
172
173
174 Polynomial Polynomial::operator-(const Polynomial& b) const {
175     Polynomial result;
176     Node* currentA = head->link;
177     Node* currentB = b.head->link;
178
179     while (currentA != head && currentB != b.head) {
180         if (currentA->exp > currentB->exp) {
181             result.head->link = new Node{ currentA->coef, currentA->exp, result.head->link };
182             currentA = currentA->link;
183         }
184         else if (currentA->exp < currentB->exp) {
185             result.head->link = new Node{ -currentB->coef, currentB->exp, result.head->link };
186             currentB = currentB->link;
187         }
188         else {
189             int diffCoef = currentA->coef - currentB->coef;
190             if (diffCoef != 0) {
191                 result.head->link = new Node{ diffCoef, currentA->exp, result.head->link };
192             }
193             currentA = currentA->link;
194             currentB = currentB->link;
195         }
196     }
197
198     while (currentA != head) {
199         result.head->link = new Node{ currentA->coef, currentA->exp, result.head->link };
200         currentA = currentA->link;
201     }
202
203     while (currentB != b.head) {
204         result.head->link = new Node{ -currentB->coef, currentB->exp, result.head->link };
205         currentB = currentB->link;
206     }
207
208     return result;
209 }
210
211
212 Polynomial Polynomial::operator*(const Polynomial& b) const {
213     Polynomial result;
214
215     Node* currentA = head->link;
216
217     while (currentA != head) {
218         Node* currentB = b.head->link;
219
220         while (currentB != b.head) {
221             int productCoef = currentA->coef * currentB->coef;
222             int productExp = currentA->exp + currentB->exp;
223
224             Node* currentResult = result.head->link;
225             Node* prevResult = result.head;
226
227             while (currentResult != result.head && currentResult->exp > productExp) {
228                 prevResult = currentResult;
229                 currentResult = currentResult->link;
230             }
231
232             if (currentResult != result.head && currentResult->exp == productExp) {
233                 currentResult->coef += productCoef;
234                 if (currentResult->coef == 0) {
235                     prevResult->link = currentResult->link;
236                     delete currentResult;
237                 }
238             }
239             else {
240                 prevResult->link = new Node{ productCoef, productExp, currentResult };
241             }
242         }

```

```

243         currentB = currentB->link;
244     }
245     currentA = currentA->link;
246 }
247 }
248
249 return result;
250 }
251
252 float Polynomial::Evaluate(float x) const {
253     float result = 0.0;
254     Node* current = head->link;
255
256     while (current != head) {
257         result += current->coef * pow(x, current->exp);
258         current = current->link;
259     }
260
261     return result;
262 }
263
264 int main() {
265     Polynomial p1, p2;
266
267     std::cout << "Enter details for Polynomial 1:\n";
268     std::cin >> p1;
269
270     std::cout << "Enter details for Polynomial 2:\n";
271     std::cin >> p2;
272
273     Polynomial sum = p1 + p2;
274     Polynomial difference = p1 - p2;
275     Polynomial product = p1 * p2;
276
277     std::cout << "Polynomial 1: " << p1 << std::endl;
278     std::cout << "Polynomial 2: " << p2 << std::endl;
279     std::cout << "Sum: " << sum << std::endl;
280     std::cout << "Difference: " << difference << std::endl;
281     std::cout << "Product: " << product << std::endl;
282
283     float x;
284     std::cout << "Enter a value for x to evaluate the first polynomial: ";
285     std::cin >> x;
286
287     std::cout << "Result of Sum evaluation: " << sum.Evaluate(x) << std::endl;
288     std::cout << "Result of Product evaluation: " << product.Evaluate(x) << std::endl;
289
290     auto start = std::chrono::high_resolution_clock::now();
291
292
293     auto stop = std::chrono::high_resolution_clock::now();
294     auto duration = std::chrono::duration_cast<std::chrono::microseconds>(stop - start);
295
296
297     std::cout << "Time taken by function: " << duration.count() << " microseconds" << std::endl;
298
299     return 0;
300 }

```

效能分析(Analysis)：

時間複雜度分析：

operator+ 和 operator-：在這兩個操作中，我們需要遍歷兩個多項式的所有項目。對於每一項，我們執行一些常數時間的操作（例如分配新節點、計算和差），所以整體時間複雜度是 $O(\max(n, m))$ ，其中 n 和 m 分別是兩個多項式的項數。

operator*：對於乘法操作，我們需要遍歷兩個多項式的所有項目，對每一項執行嵌套的內循環以處理乘法。因此，整體時間複雜度是 $O(n * m)$ ，其中 n 和 m 分別是兩個多項式的項數。

Evaluate：在評估多項式時，我們遍歷所有項目並執行一些常數時間的操作。所以時間複雜度是 $O(n)$ ，其中 n 是多項式的項數。

空間複雜度分析：

operator+、operator- 和 operator*：這些操作都使用了額外的空間（例如新節點）來存儲結果多項式。所以空間複雜度是 $O(\max(n, m))$ 。

Evaluate：評估多項式時，我們沒有使用額外的空間，只是使用一些臨時變量。因此，空間複雜度是 $O(1)$ 。

測試與驗證(Testing and Proving)：

```
Microsoft Visual Studio 偵錯器
Enter details for Polynomial 1:
Enter the number of terms: 3
Enter coefficient and exponent for term 1: 2 3
Enter coefficient and exponent for term 2: 4 1
Enter coefficient and exponent for term 3: 5 0
Enter details for Polynomial 2:
Enter the number of terms: 5
Enter coefficient and exponent for term 1: 1 4
Enter coefficient and exponent for term 2: 2 0
Enter coefficient and exponent for term 3: 9 2
Enter coefficient and exponent for term 4: 3 6
Enter coefficient and exponent for term 5: 7 8
Polynomial 1: 5x^0 + 4x^1 + 2x^3
Polynomial 2: 7x^8 + 3x^6 + 9x^2 + 2x^0 + 1x^4
Sum: 7x^8 + 3x^6 + 9x^2 + 7x^0 + 1x^4 + 4x^1 + 2x^3
Difference: -7x^8 + -3x^6 + -9x^2 + 3x^0 + -1x^4 + 4x^1 + 2x^3
Product: 10x^0 + 8x^1 + 45x^2 + 40x^3 + 5x^4 + 22x^5 + 15x^6 + 14x^7 + 35x^8 + 34x^9 + 14x^11
Enter a value for x to evaluate the first polynomial: 3
Result of Sum evaluation: 48349
Result of Product evaluation: 3.42774e+06
Time taken by function: 0 microseconds

C:\Users\user\OneDrive\桌面\112-1(大二)\112-2\x64\Debug\text.exe (處理序 29356) 已結束，出現代碼 0。
若要在偵錯停止時自動關閉主控台，請啟用 [工具] -> [選項] -> [偵錯] -> [偵錯停止時，自動關閉主控台]。
按任意鍵關閉此視窗...
```

效能量測 (Measuring)：

```
Microsoft Visual Studio 偵錯器
Enter details for Polynomial 1:
Enter the number of terms: 3
Enter coefficient and exponent for term 1: 2 3
Enter coefficient and exponent for term 2: 4 1
Enter coefficient and exponent for term 3: 5 0
Enter details for Polynomial 2:
Enter the number of terms: 5
Enter coefficient and exponent for term 1: 1 4
Enter coefficient and exponent for term 2: 2 0
Enter coefficient and exponent for term 3: 9 2
Enter coefficient and exponent for term 4: 3 6
Enter coefficient and exponent for term 5: 7 8
Polynomial 1: 5x^0 + 4x^1 + 2x^3
Polynomial 2: 7x^8 + 3x^6 + 9x^2 + 2x^0 + 1x^4
Sum: 7x^8 + 3x^6 + 9x^2 + 7x^0 + 1x^4 + 4x^1 + 2x^3
Difference: -7x^8 + -3x^6 + -9x^2 + 3x^0 + -1x^4 + 4x^1 + 2x^3
Product: 10x^0 + 8x^1 + 45x^2 + 40x^3 + 5x^4 + 22x^5 + 15x^6 + 14x^7 + 35x^8 + 34x^9 + 14x^11
Enter a value for x to evaluate the first polynomial: 3
Result of Sum evaluation: 48349
Result of Product evaluation: 3.42774e+06
Time taken by function: 0 microseconds

C:\Users\user\OneDrive\桌面\112-1(大二)\112-2\x64\Debug\text.exe (處理序 29356) 已結束，出現代碼 0。
若要在偵錯停止時自動關閉主控台，請啟用 [工具] -> [選項] -> [偵錯] -> [偵錯停止時，自動關閉主控台]。
按任意鍵關閉此視窗...
```

心得討論：

在實作多項式類別的過程中，透過 C++ 的物件導向設計，我們成功實現了多項式的基本操作，包括加法、減法、乘法、輸入輸出，並加入了複製構造函數和賦值運算子。為了提高效能，我們使用了合適的資料結構，適當的算法，以及效能測量。這次的實作加深了我對類別設計和程式效能優化的理解，同時提供了一個實用的例子，展現了如何在實際問題中應用 C++ 的強大功能。透過效能分析，我們也更深入了解程式的運行效率，並學到了如何進一步優化代碼。