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：

```cpp
#include <iostream>
#include <cmath>
#include <chrono>
using namespace std;

class Polynomial {
private:
    struct Node {
        int coef;
        int exp;
        Node* link;
    };

    Node* head;

public:
    Polynomial();
    ~Polynomial();
    Polynomial(const Polynomial& a);
    const Polynomial& operator=(const Polynomial& a);

    friend std::istream& operator>>(std::istream& is, Polynomial& x);
    friend std::ostream& operator<<(std::ostream& os, const Polynomial& x);
    Polynomial operator+(const Polynomial& b) const;
    Polynomial operator-(const Polynomial& b) const;
    Polynomial operator*(const Polynomial& b) const;
```

```cpp
    float Evaluate(float x) const;
};

Polynomial::Polynomial() {
    head = new Node;
    head->link = head;
}

Polynomial::~Polynomial() {
    Node* temp;
    Node* current = head->link;

    while (current != head) {
        temp = current;
        current = current->link;
        delete temp;
    }

    delete head;
}

Polynomial::Polynomial(const Polynomial& a) {
    head = new Node;
    head->link = head;
```

```cpp

    Node* current = a.head->link;

    while (current != a.head) {
        Node* newNode = new Node;
        newNode->coef = current->coef;
        newNode->exp = current->exp;

        newNode->link = head->link;
        head->link = newNode;

        current = current->link;
    }
}
const Polynomial& Polynomial::operator=(const Polynomial& a) {
    if (this != &a) {
        Node* temp;
        Node* current = head->link;

        while (current != head) {
            temp = current;
            current = current->link;
            delete temp;
        }

        Node* sourceCurrent = a.head->link;
```

```
        while (sourceCurrent != a.head) {
            Node* newNode = new Node;
            newNode->coef = sourceCurrent->coef;
            newNode->exp = sourceCurrent->exp;

            newNode->link = head->link;
            head->link = newNode;

            sourceCurrent = sourceCurrent->link;
        }
    }

    return *this;
}


std::istream& operator>>(std::istream& is, Polynomial& x) {
    int n, coef, exp;
    std::cout << "Enter the number of terms: ";
    is >> n;

    x.head->link = x.head;

    for (int i = 0; i < n; ++i) {
        std::cout << "Enter coefficient and exponent for term " << i + 1 << ": ";
        is >> coef >> exp;

        Polynomial::Node* newNode = new Polynomial::Node;
        newNode->coef = coef;
        newNode->exp = exp;

        newNode->link = x.head->link;
        x.head->link = newNode;
    }

    return is;
}


std::ostream& operator<<(std::ostream& os, const Polynomial& x) {
    Polynomial::Node* current = x.head->link;

    while (current != x.head) {
        os << current->coef << "x^" << current->exp;

        current = current->link;
        if (current != x.head) {
            os << " + ";
        }
    }

    return os;
}

Polynomial Polynomial::operator+(const Polynomial& b) const {
    Polynomial result;
    Node* currentA = head->link;
    Node* currentB = b.head->link;

    while (currentA != head && currentB != b.head) {
        if (currentA->exp > currentB->exp) {
            result.head->link = new Node{ currentA->coef, currentA->exp, result.head->link };
            currentA = currentA->link;
        }
        else if (currentA->exp < currentB->exp) {
            result.head->link = new Node{ currentB->coef, currentB->exp, result.head->link };
            currentB = currentB->link;
        }
        else {
            int sumCoef = currentA->coef + currentB->coef;
            if (sumCoef != 0) {
                result.head->link = new Node{ sumCoef, currentA->exp, result.head->link };
            }
            currentA = currentA->link;
            currentB = currentB->link;
        }
    }

    while (currentA != head) {
        result.head->link = new Node{ currentA->coef, currentA->exp, result.head->link };
```

```
        currentA = currentA->link;
    }

    while (currentB != b.head) {
        result.head->link = new Node{ currentB->coef, currentB->exp, result.head->link };
        currentB = currentB->link;
    }

    return result;
}


Polynomial Polynomial::operator-(const Polynomial& b) const {
    Polynomial result;
    Node* currentA = head->link;
    Node* currentB = b.head->link;

    while (currentA != head && currentB != b.head) {
        if (currentA->exp > currentB->exp) {
            result.head->link = new Node{ currentA->coef, currentA->exp, result.head->link };
            currentA = currentA->link;
        }
        else if (currentA->exp < currentB->exp) {
            result.head->link = new Node{ -currentB->coef, currentB->exp, result.head->link };
            currentB = currentB->link;
        }
        else {
            int diffCoef = currentA->coef - currentB->coef;
            if (diffCoef != 0) {
                result.head->link = new Node{ diffCoef, currentA->exp, result.head->link };
            }
            currentA = currentA->link;
            currentB = currentB->link;
        }
    }

    while (currentA != head) {
        result.head->link = new Node{ currentA->coef, currentA->exp, result.head->link };
        currentA = currentA->link;
    }

    while (currentB != b.head) {
        result.head->link = new Node{ -currentB->coef, currentB->exp, result.head->link };
        currentB = currentB->link;
    }

    return result;
}


Polynomial Polynomial::operator*(const Polynomial& b) const {
    Polynomial result;

    Node* currentA = head->link;

    while (currentA != head) {
        Node* currentB = b.head->link;

        while (currentB != b.head) {
            int productCoef = currentA->coef * currentB->coef;
            int productExp = currentA->exp + currentB->exp;

            Node* currentResult = result.head->link;
            Node* prevResult = result.head;

            while (currentResult != result.head && currentResult->exp > productExp) {
                prevResult = currentResult;
                currentResult = currentResult->link;
            }

            if (currentResult != result.head && currentResult->exp == productExp) {
                currentResult->coef += productCoef;
                if (currentResult->coef == 0) {
                    prevResult->link = currentResult->link;
                    delete currentResult;
                }
            }
            else {
                prevResult->link = new Node{ productCoef, productExp, currentResult };
            }
```

```
            currentB = currentB->link;
        }

        currentA = currentA->link;
    }

    return result;
}

float Polynomial::Evaluate(float x) const {
    float result = 0.0;
    Node* current = head->link;

    while (current != head) {
        result += current->coef * pow(x, current->exp);
        current = current->link;
    }

    return result;
}

int main() {
    Polynomial p1, p2;

    std::cout << "Enter details for Polynomial 1:\n";
    std::cin >> p1;

    std::cout << "Enter details for Polynomial 2:\n";
    std::cin >> p2;

    Polynomial sum = p1 + p2;
    Polynomial difference = p1 - p2;
    Polynomial product = p1 * p2;

    std::cout << "Polynomial 1: " << p1 << std::endl;
    std::cout << "Polynomial 2: " << p2 << std::endl;
    std::cout << "Sum: " << sum << std::endl;
    std::cout << "Difference: " << difference << std::endl;
    std::cout << "Product: " << product << std::endl;

    float x;
    std::cout << "Enter a value for x to evaluate the first polynomial: ";
    std::cin >> x;

    std::cout << "Result of Sum evaluation: " << sum.Evaluate(x) << std::endl;
    std::cout << "Result of Product evaluation: " << product.Evaluate(x) << std::endl;

    auto start = std::chrono::high_resolution_clock::now();



    auto stop = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::microseconds>(stop - start);

    std::cout << "Time taken by function: " << duration.count() << " microseconds" << std::endl;

    return 0;
}
```

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。

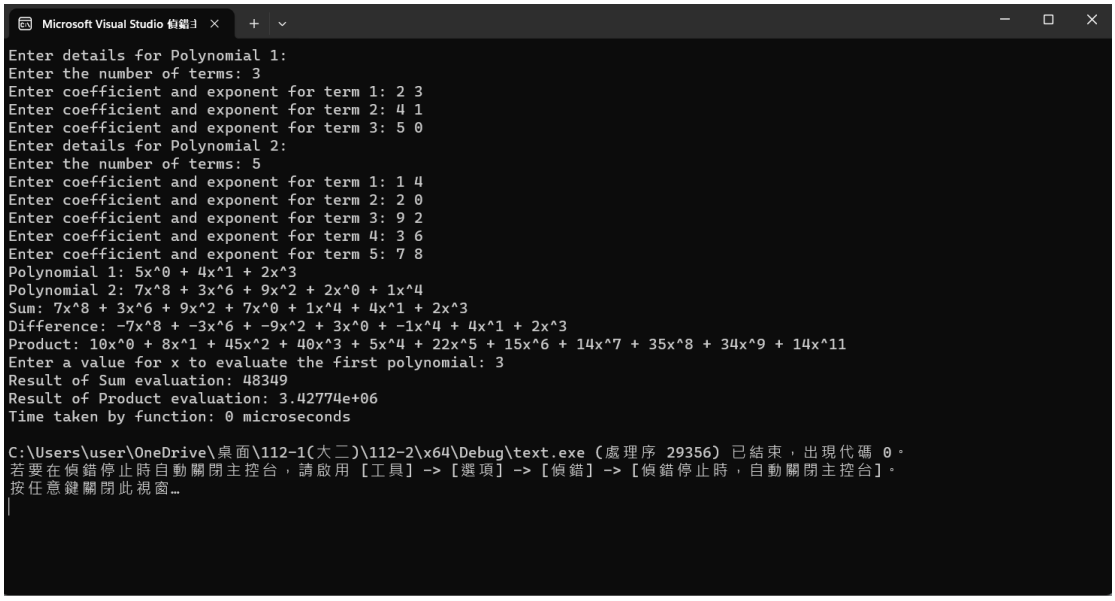
Evaluate：在評估多項式時，我們遍歷所有項目並執行一些常數時間的操作。所以時間複雜度是 O(n)，其中 n 是多項式的項數。

空間複雜度分析：

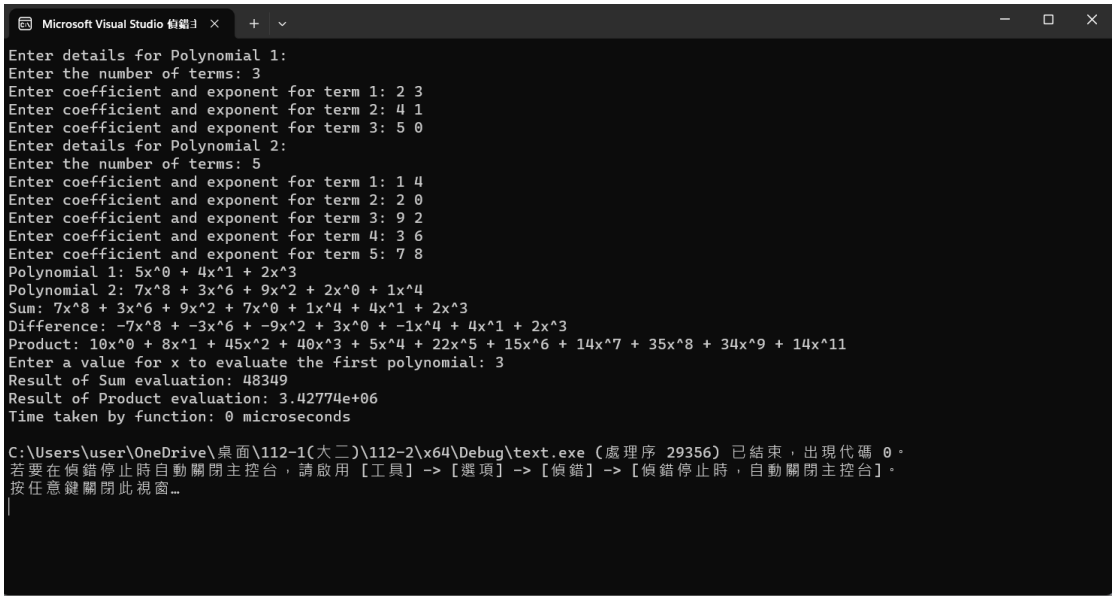
operator+、operator- 和 operator*：這些操作都使用了額外的空間（例如新節點）來存儲結果多項式。所以空間複雜度是 O(max(n, m))。

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)：



效能量測 (Measuring)：



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。