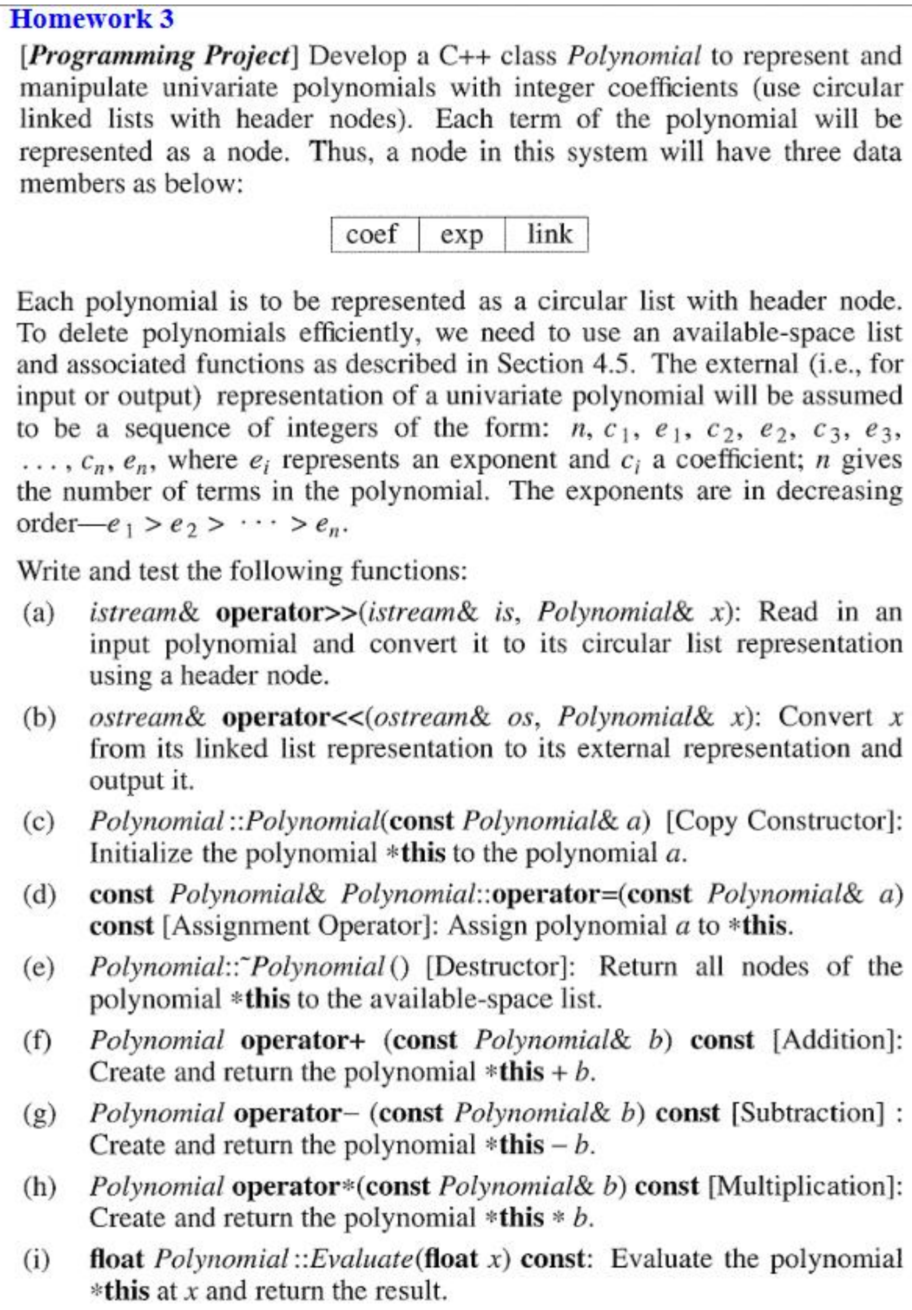
資料結構 HW3

41243128 徐聖硯

一、題目



Homework 3

[***Programming Project***] Develop a C++ class *Polynomial* to represent and manipulate univariate polynomials with integer coefficients (use circular linked lists with header nodes). Each term of the polynomial will be represented as a node. Thus, a node in this system will have three data members as below:

coef	exp	link

Each polynomial is to be represented as a circular list with header node. To delete polynomials efficiently, we need to use an available-space list and associated functions as described in Section 4.5. The external (i.e., for input or output) representation of a univariate polynomial will be assumed to be a sequence of integers of the form: $n, c_1, e_1, c_2, e_2, c_3, e_3, \ldots, c_n, e_n$, where e_i represents an exponent and c_i a coefficient; n gives the number of terms in the polynomial. The exponents are in decreasing order—$e_1 > e_2 > \cdots > e_n$.

Write and test the following functions:

(a) *istream*& **operator>>**(*istream*& *is*, *Polynomial*& *x*): Read in an input polynomial and convert it to its circular list representation using a header node.

(b) *ostream*& **operator<<**(*ostream*& *os*, *Polynomial*& *x*): Convert *x* from its linked list representation to its external representation and output it.

(c) *Polynomial*::*Polynomial*(**const** *Polynomial*& *a*) [Copy Constructor]: Initialize the polynomial ***this** to the polynomial *a*.

(d) **const** *Polynomial*& *Polynomial*::**operator=**(**const** *Polynomial*& *a*) **const** [Assignment Operator]: Assign polynomial *a* to ***this**.

(e) *Polynomial*::~*Polynomial*() [Destructor]: Return all nodes of the polynomial ***this** to the available-space list.

(f) *Polynomial* **operator+** (**const** *Polynomial*& *b*) **const** [Addition]: Create and return the polynomial ***this** + *b*.

(g) *Polynomial* **operator**– (**const** *Polynomial*& *b*) **const** [Subtraction] : Create and return the polynomial ***this** – *b*.

(h) *Polynomial* **operator***(**const** *Polynomial*& *b*) **const** [Multiplication]: Create and return the polynomial ***this** * *b*.

(i) **float** *Polynomial*::*Evaluate*(**float** *x*) **const**: Evaluate the polynomial ***this** at *x* and return the result.

需實作的功能：

(a) istream& operator>>(istream& is, Polynomial& x)

讀入多項式並將其轉換為循環鏈表表示。

(b) ostream& operator<<(ostream& os, Polynomial& x)

將多項式從鏈表形式轉換為輸出格式並輸出。

(c) 拷貝構造函數 Polynomial::Polynomial(const Polynomial& a)

將一個多項式 a 的內容拷貝到當前對象。

(d) 賦值運算符重載 Polynomial::operator=(const Polynomial& a)

將多項式 a 的內容賦值給當前對象。

(e) 解構函數 Polynomial::~Polynomial()

將多項式的所有節點歸還到可用空間鏈表。

(f) 加法運算符重載 Polynomial operator+(const Polynomial& b)

創建並返回兩個多項式的加法結果。

(g) 減法運算符重載 Polynomial operator-(const Polynomial& b)

創建並返回兩個多項式的減法結果。

(h) 乘法運算符重載 Polynomial operator*(const Polynomial& b)

創建並返回兩個多項式的乘法結果。

(i) 多項式求值 Polynomial::Evaluate(float x)

輸入一個浮點數 x，計算並返回多項式在該點的值。

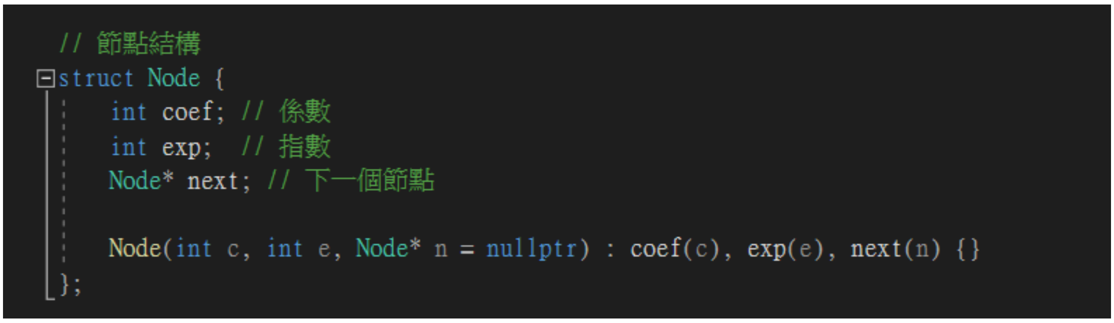
要使用 Circular Linked List with Header Nodes 來實作

二、解題說明

這個題目要實作 Polynomial 類別，用來表示多項式，他必須使用 Circular Linked List with Header Nodes 來儲存，他會包含三個屬性 coef、exp、link。

三、Algorithm Design & Programming

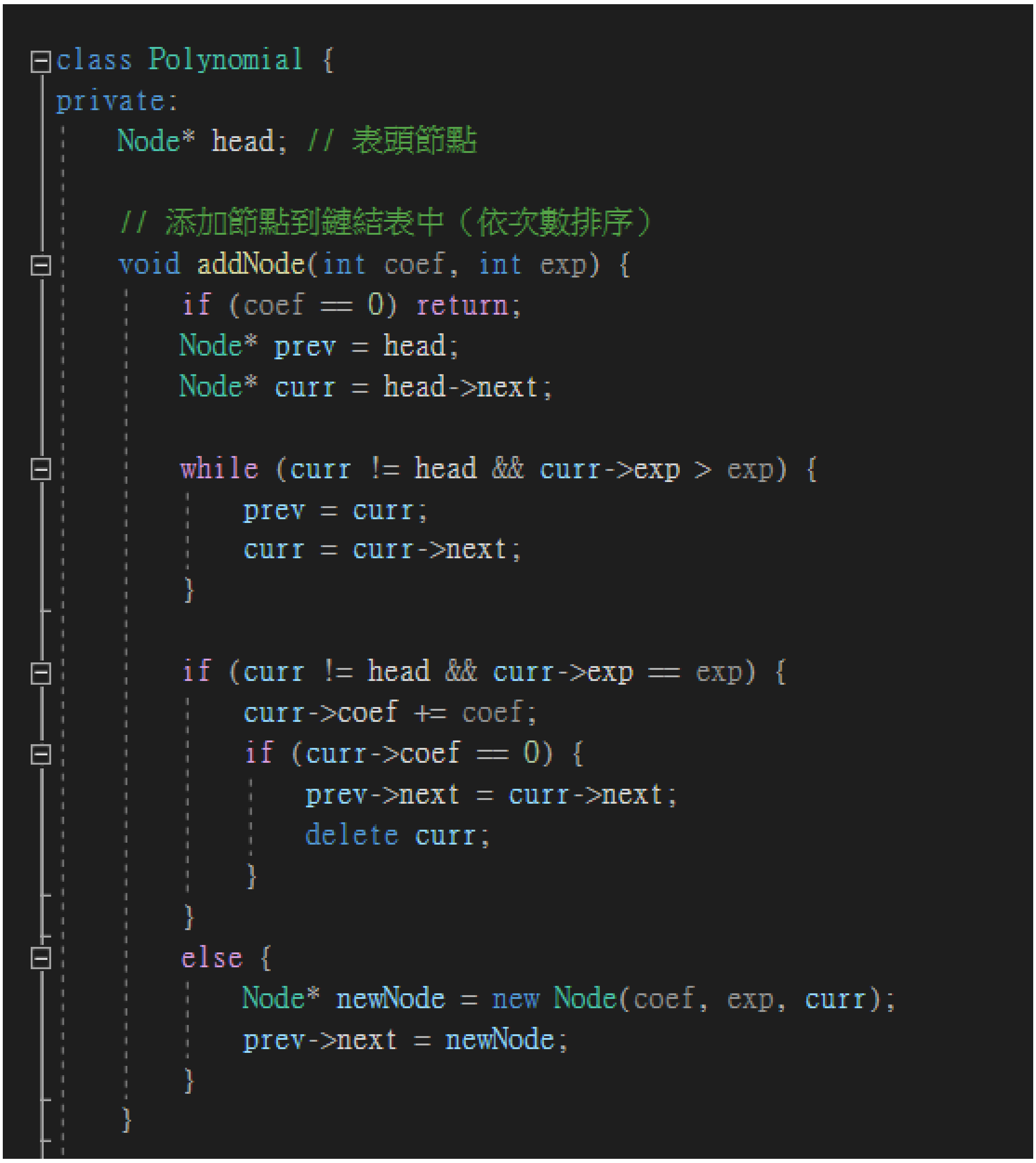
這個是節點結構



```
// 節點結構
struct Node {
    int coef; // 係數
    int exp;  // 指數
    Node* next; // 下一個節點

    Node(int c, int e, Node* n = nullptr) : coef(c), exp(e), next(n) {}
};
```

他是用來儲存多項式的一項 像是 5x^2 coef = 2,exp = 3

然後 next 指向下一個



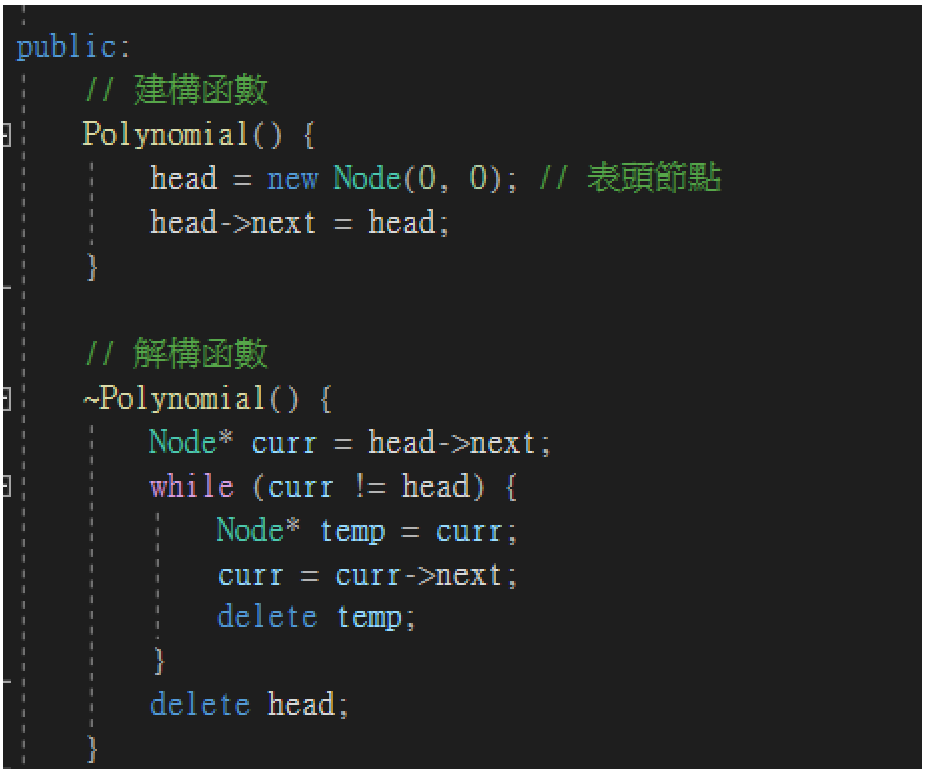
```
class Polynomial {
private:
    Node* head; // 表頭節點

    // 添加節點到鏈結表中（依次數排序）
    void addNode(int coef, int exp) {
        if (coef == 0) return;
        Node* prev = head;
        Node* curr = head->next;

        while (curr != head && curr->exp > exp) {
            prev = curr;
            curr = curr->next;
        }

        if (curr != head && curr->exp == exp) {
            curr->coef += coef;
            if (curr->coef == 0) {
                prev->next = curr->next;
                delete curr;
            }
        }
        else {
            Node* newNode = new Node(coef, exp, curr);
            prev->next = newNode;
        }
    }
```

這個是我的 Polynomial 的添加節點到多項式的部分，

它的功能是將一項新增到多項式內，並按照指數大小排列

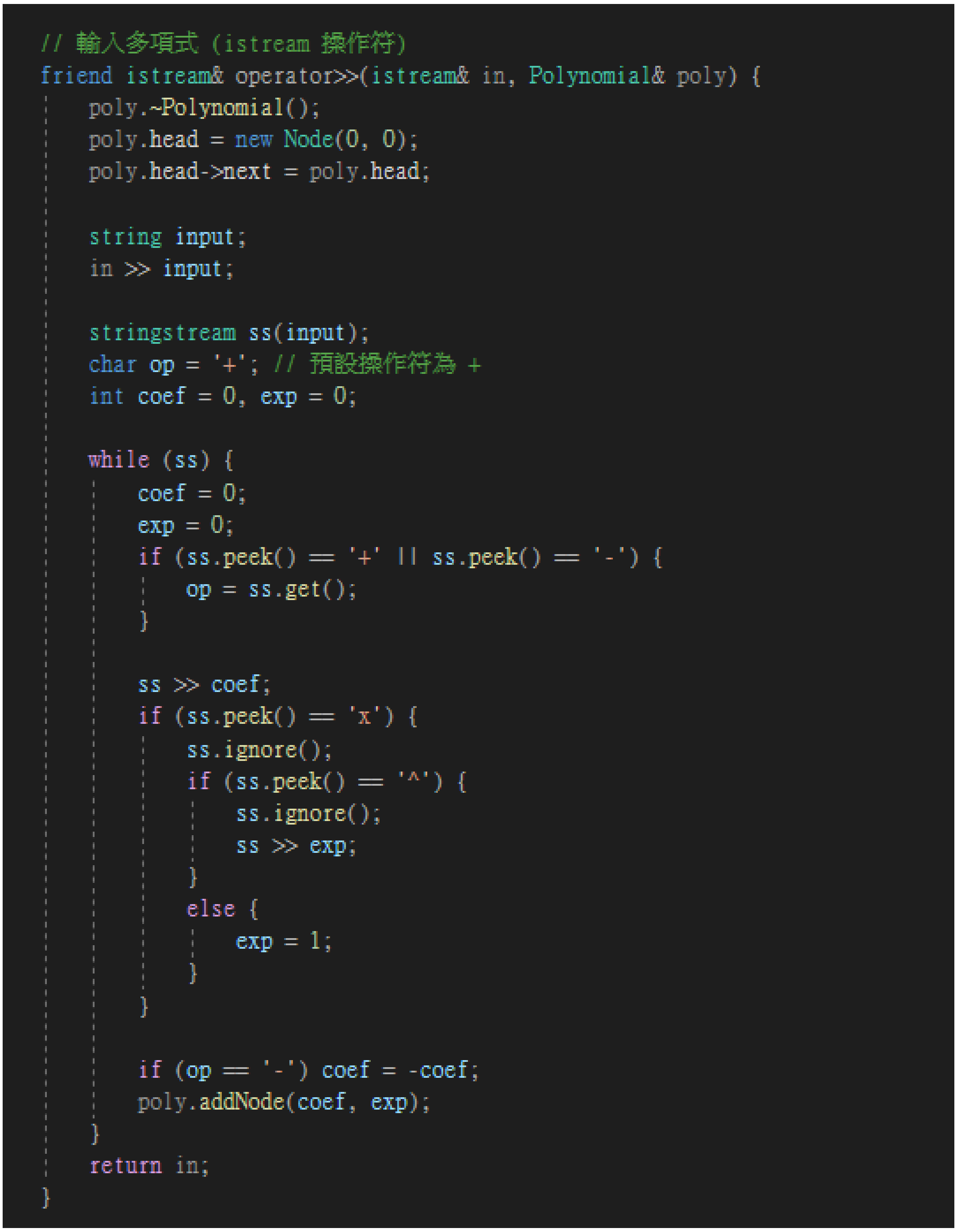
，若指數的項目已經存在，就會合併，如果係數是 0，就會

刪除此項



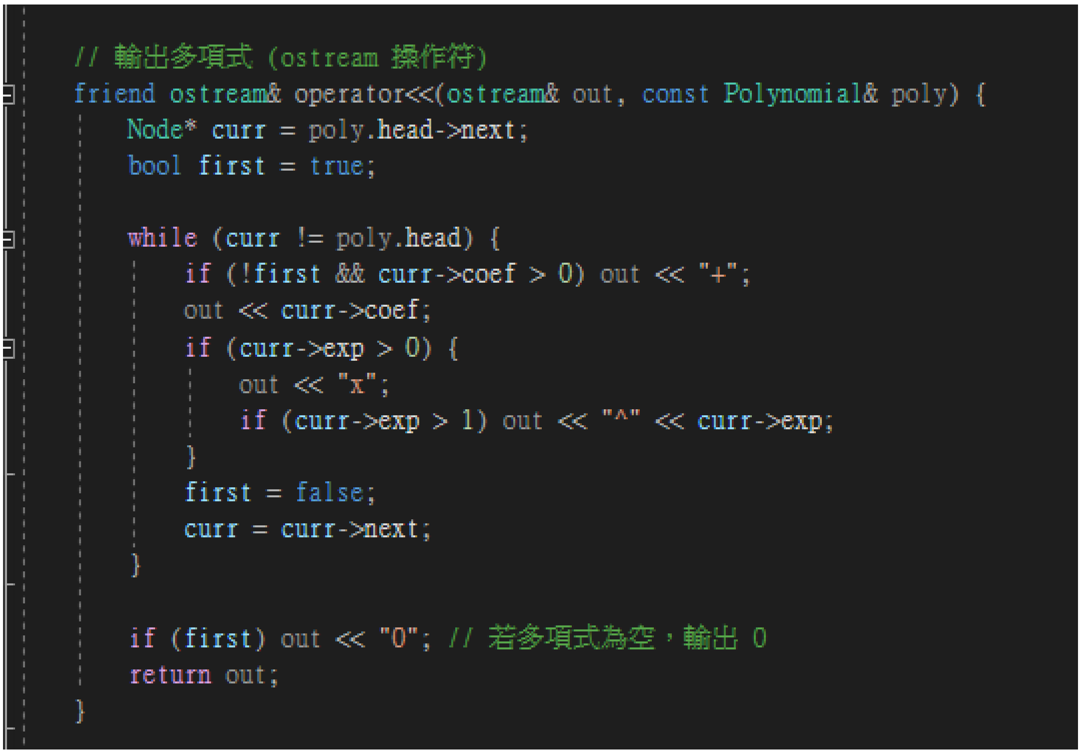
```
public:
    // 建構函數
    Polynomial() {
        head = new Node(0, 0); // 表頭節點
        head->next = head;
    }

    // 解構函數
    ~Polynomial() {
        Node* curr = head->next;
        while (curr != head) {
            Node* temp = curr;
            curr = curr->next;
            delete temp;
        }
        delete head;
    }
```

這是建構解構函式。



```
// 輸入多項式 (istream 操作符)
friend istream& operator>>(istream& in, Polynomial& poly) {
    poly.~Polynomial();
    poly.head = new Node(0, 0);
    poly.head->next = poly.head;

    string input;
    in >> input;

    stringstream ss(input);
    char op = '+'; // 預設操作符為 +
    int coef = 0, exp = 0;

    while (ss) {
        coef = 0;
        exp = 0;
        if (ss.peek() == '+' || ss.peek() == '-') {
            op = ss.get();
        }

        ss >> coef;
        if (ss.peek() == 'x') {
            ss.ignore();
            if (ss.peek() == '^') {
                ss.ignore();
                ss >> exp;
            }
            else {
                exp = 1;
            }
        }

        if (op == '-') coef = -coef;
        poly.addNode(coef, exp);
    }
    return in;
}
```

這是 istream& operator>>(istream& in,Polynomial& poly)的實作，他就是去判斷多項式的格式，並且解析他，並把他轉成連結串列來表示。

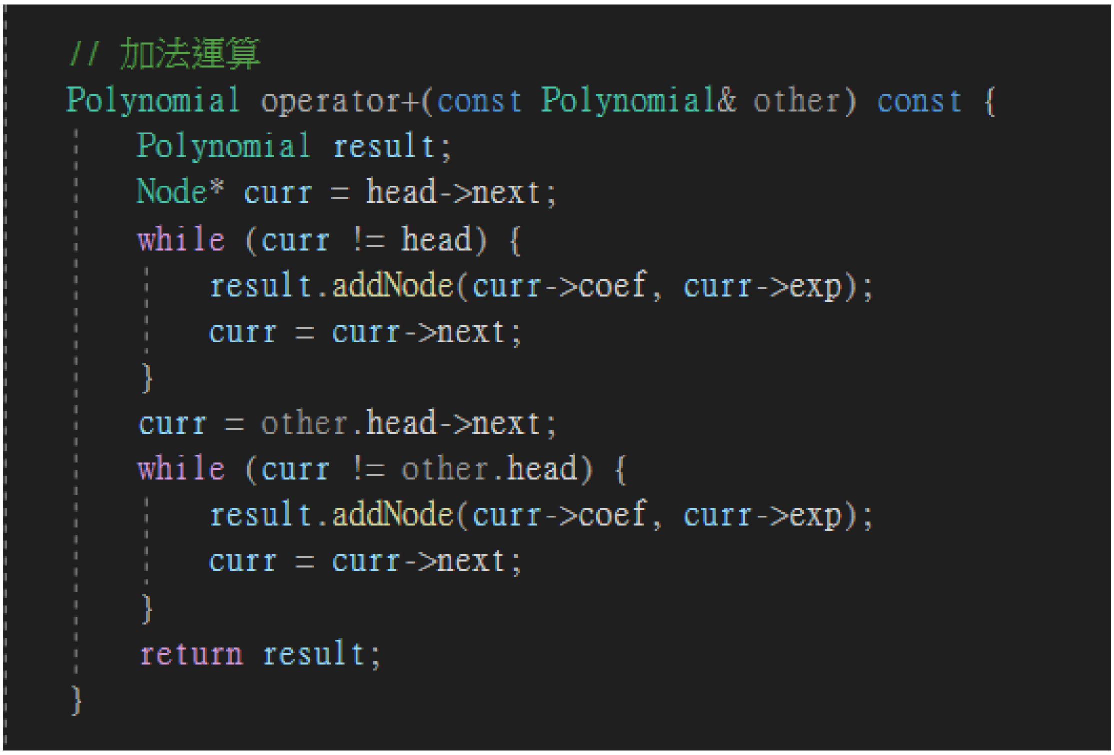


```
// 輸出多項式 (ostream 操作符)
friend ostream& operator<<(ostream& out, const Polynomial& poly) {
    Node* curr = poly.head->next;
    bool first = true;

    while (curr != poly.head) {
        if (!first && curr->coef > 0) out << "+";
        out << curr->coef;
        if (curr->exp > 0) {
            out << "x";
            if (curr->exp > 1) out << "^" << curr->exp;
        }
        first = false;
        curr = curr->next;
    }

    if (first) out << "0"; // 若多項式為空，輸出 0
    return out;
}
```

ostream& operator<<(ostream& out,const Polynomial& poly)

的實作，他主要就是按照要求的格式來輸出多項式。

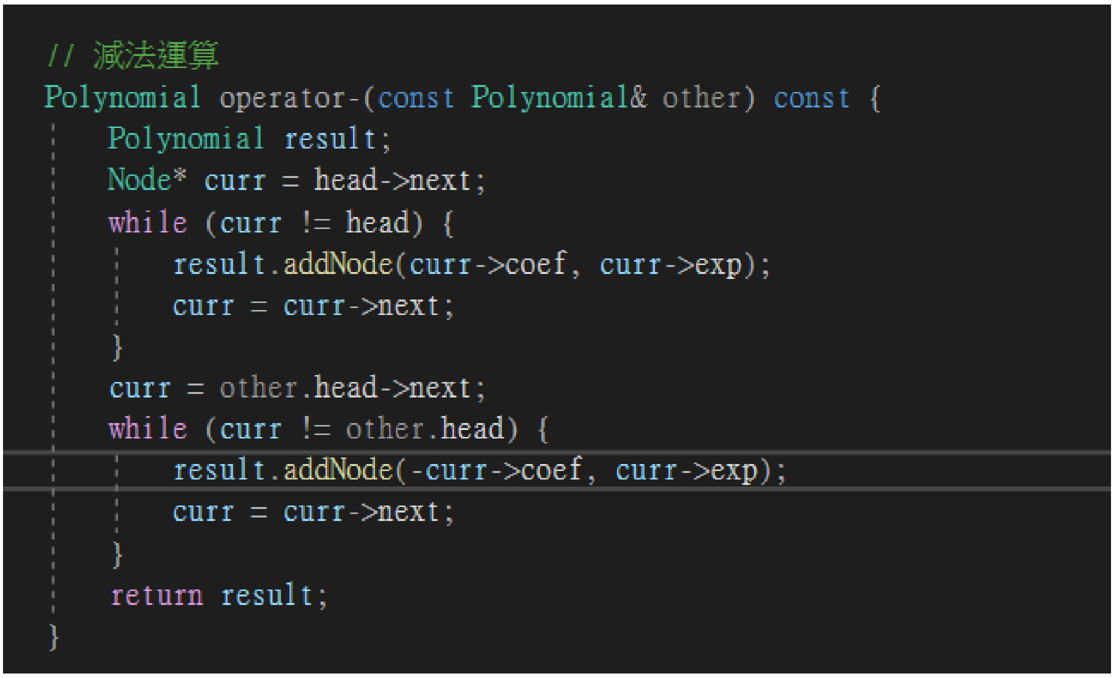
範例: 2x^4-7x+2



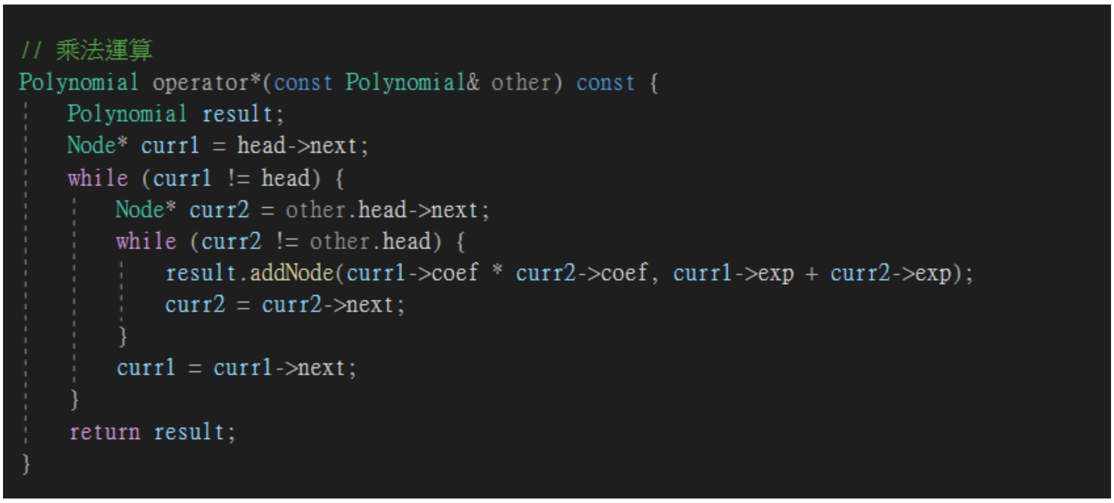
```
// 加法運算
Polynomial operator+(const Polynomial& other) const {
    Polynomial result;
    Node* curr = head->next;
    while (curr != head) {
        result.addNode(curr->coef, curr->exp);
        curr = curr->next;
    }
    curr = other.head->next;
    while (curr != other.head) {
        result.addNode(curr->coef, curr->exp);
        curr = curr->next;
    }
    return result;
}
```

這是加法部分的實作，他會 run 過兩個多項式的節點，

然後在把結果新增到多項式中。減法就是把係數取負。

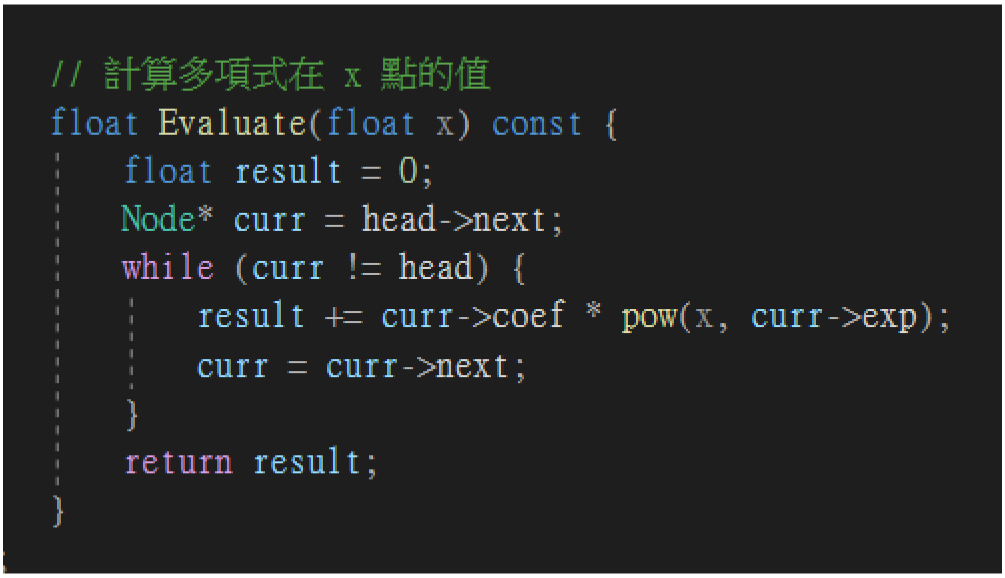


```
// 減法運算
Polynomial operator-(const Polynomial& other) const {
    Polynomial result;
    Node* curr = head->next;
    while (curr != head) {
        result.addNode(curr->coef, curr->exp);
        curr = curr->next;
    }
    curr = other.head->next;
    while (curr != other.head) {
        result.addNode(-curr->coef, curr->exp);
        curr = curr->next;
    }
    return result;
}
```



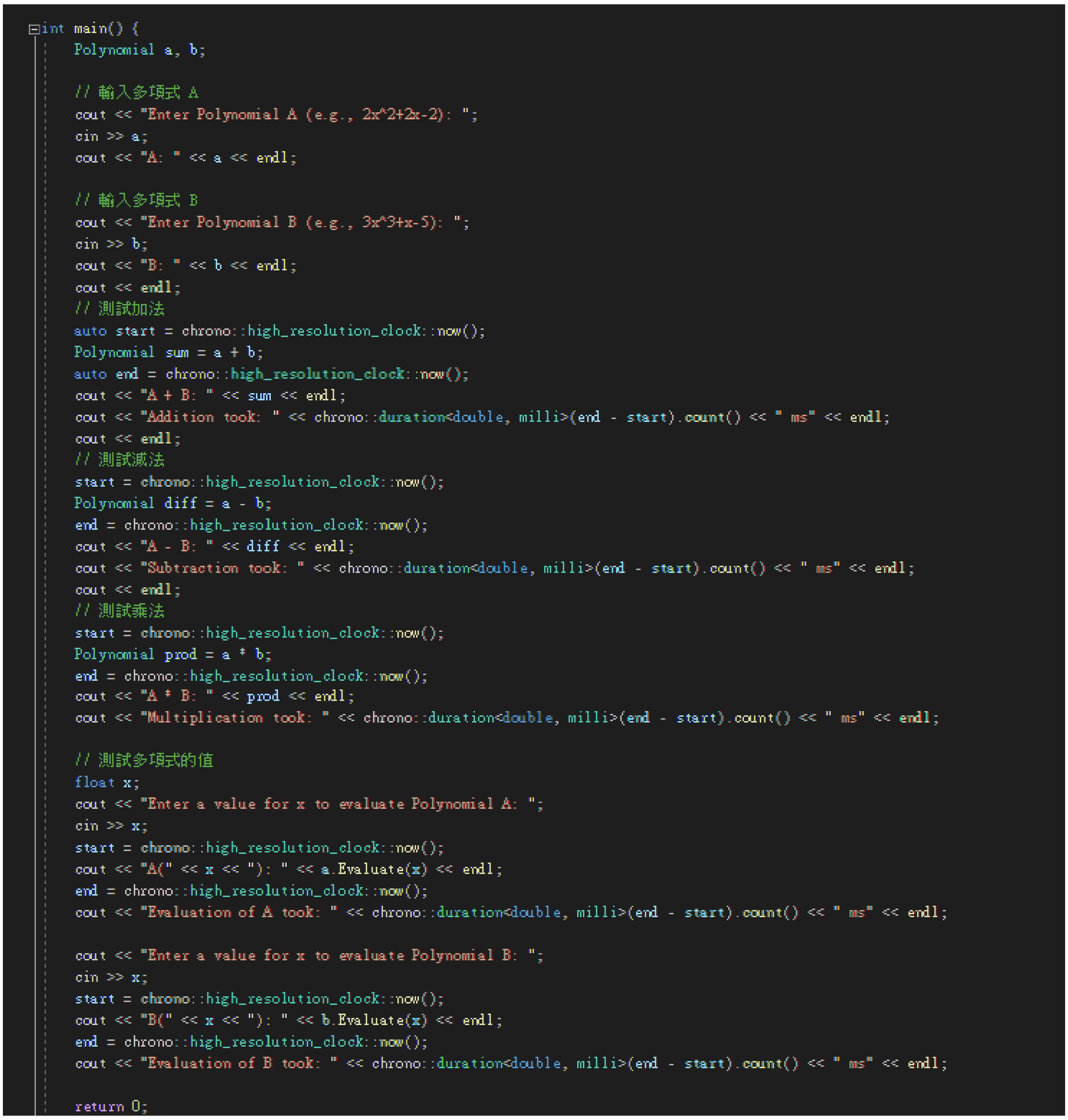
```
// 乘法運算
Polynomial operator*(const Polynomial& other) const {
    Polynomial result;
    Node* curr1 = head->next;
    while (curr1 != head) {
        Node* curr2 = other.head->next;
        while (curr2 != other.head) {
            result.addNode(curr1->coef * curr2->coef, curr1->exp + curr2->exp);
            curr2 = curr2->next;
        }
        curr1 = curr1->next;
    }
    return result;
}
```

這是乘法的部分，他將每個節點進行項目成項目的計算，然後再塞到結果。



```cpp
// 計算多項式在 x 點的值
float Evaluate(float x) const {
    float result = 0;
    Node* curr = head->next;
    while (curr != head) {
        result += curr->coef * pow(x, curr->exp);
        curr = curr->next;
    }
    return result;
}
```

他使用了 pow 來計算每一項的值，並累加來計算結果。



```cpp
int main() {
    Polynomial a, b;

    // 輸入多項式 A
    cout << "Enter Polynomial A (e.g., 2x^2+2x-2): ";
    cin >> a;
    cout << "A: " << a << endl;

    // 輸入多項式 B
    cout << "Enter Polynomial B (e.g., 3x^3+x-5): ";
    cin >> b;
    cout << "B: " << b << endl;
    cout << endl;
    // 測試加法
    auto start = chrono::high_resolution_clock::now();
    Polynomial sum = a + b;
    auto end = chrono::high_resolution_clock::now();
    cout << "A + B: " << sum << endl;
    cout << "Addition took: " << chrono::duration<double, milli>(end - start).count() << " ms" << endl;
    cout << endl;
    // 測試減法
    start = chrono::high_resolution_clock::now();
    Polynomial diff = a - b;
    end = chrono::high_resolution_clock::now();
    cout << "A - B: " << diff << endl;
    cout << "Subtraction took: " << chrono::duration<double, milli>(end - start).count() << " ms" << endl;
    cout << endl;
    // 測試乘法
    start = chrono::high_resolution_clock::now();
    Polynomial prod = a * b;
    end = chrono::high_resolution_clock::now();
    cout << "A * B: " << prod << endl;
    cout << "Multiplication took: " << chrono::duration<double, milli>(end - start).count() << " ms" << endl;

    // 測試多項式的值
    float x;
    cout << "Enter a value for x to evaluate Polynomial A: ";
    cin >> x;
    start = chrono::high_resolution_clock::now();
    cout << "A(" << x << "): " << a.Evaluate(x) << endl;
    end = chrono::high_resolution_clock::now();
    cout << "Evaluation of A took: " << chrono::duration<double, milli>(end - start).count() << " ms" << endl;

    cout << "Enter a value for x to evaluate Polynomial B: ";
    cin >> x;
    start = chrono::high_resolution_clock::now();
    cout << "B(" << x << "): " << b.Evaluate(x) << endl;
    end = chrono::high_resolution_clock::now();
    cout << "Evaluation of B took: " << chrono::duration<double, milli>(end - start).count() << " ms" << endl;

    return 0;
```

這是我的 main，讓使用者輸入多項式，並測試及輸出多項式 a 跟 b 的相乘、相減、相加功能是否正常，並且計算個個執行的效能及耗時。

四、效能分析

時間複雜度:

輸入:

如果多項式有 N 項，最多的情況為 O(n^2)

輸出:

需要 run 過整個，所以為 O(n)

加減法:

若兩多項式分別為 n 跟 m 時間複雜度為 O(n+m)

乘法:

因為要對 A 跟 B 的每一項做逐項相乘，

所以 A 有 n 項 B 有 m 項，最多的情況 n x m 項，

最多 O(n・m)

總時間複雜度 O(n^2・m^2)

計算多項式的值:

要 run 過多項式的每一項 O(exp)

若有 n 項，最多的情況 k 很大，時間複雜度 O(n・k)

空間複雜度:

多項式的表示，每個節點都要儲存係數、指數、指標

O(n)，n 是多項是項數

加減法:

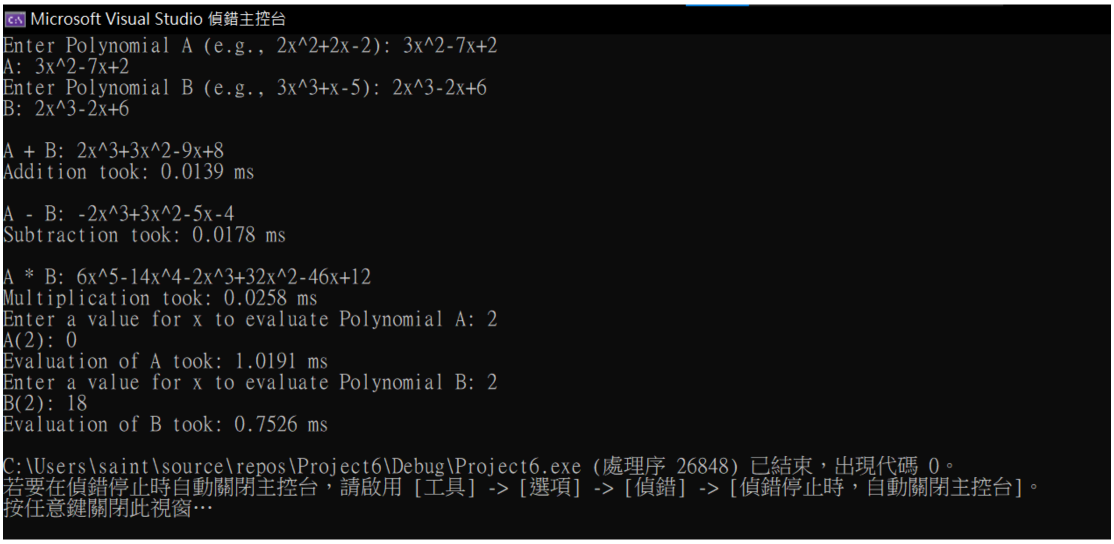
最多的情況為 O(n+m)

(就是兩個多項式的總和

乘法:

最多的情況為 O(n・m)

五、執行與驗證



```
Enter Polynomial A (e.g., 2x^2+2x-2): 3x^2-7x+2
A: 3x^2-7x+2
Enter Polynomial B (e.g., 3x^3+x-5): 2x^3-2x+6
B: 2x^3-2x+6

A + B: 2x^3+3x^2-9x+8
Addition took: 0.0139 ms

A - B: -2x^3+3x^2-5x-4
Subtraction took: 0.0178 ms

A * B: 6x^5-14x^4-2x^3+32x^2-46x+12
Multiplication took: 0.0258 ms
Enter a value for x to evaluate Polynomial A: 2
A(2): 0
Evaluation of A took: 1.0191 ms
Enter a value for x to evaluate Polynomial B: 2
B(2): 18
Evaluation of B took: 0.7526 ms

C:\Users\saint\source\repos\Project6\Debug\Project6.exe (處理序 26848) 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…
```

A: $3x^2-7x+2$, $x=2$, $3\times2^2-7\times2+2 = 12-14+2 = \underline{0}$

B: $2x^3-2x+6$, $x=2$, $2\times2^3-2\times2+6 = 16-4+6 = \underline{18}$

$$A+B=(3x^2-7x+2)+(2x^3-2x+6)=\underline{2x^3+3x^2-9x+8}$$

$$A-B=(3x^2-7x+2)-(2x^3-2x+6)=\underline{-2x^3+3x^2-9x-4}$$

$$A\times B=(3x^2-7x+2)\times(2x^3-2x+6)$$

$$\Rightarrow 6x^5-6x^3+18x^2-14x^4+14x^2-42x+4x^3-4x+12$$

$$=\underline{6x^5-14x^4-2x^3+32x^2-46x+12}$$