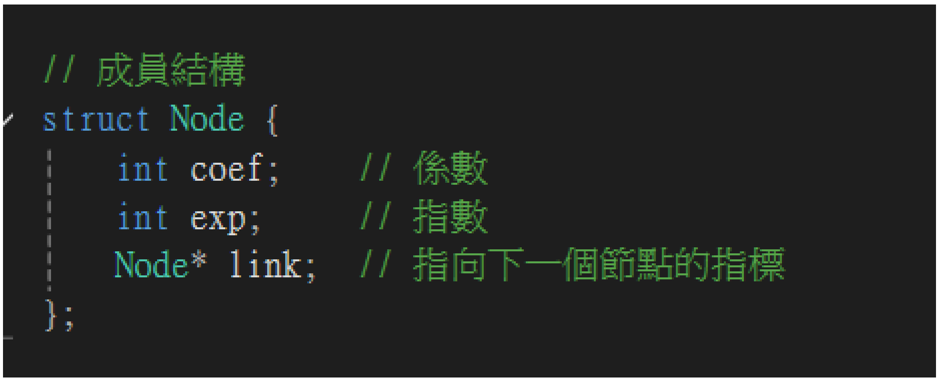
資料結構 homework3

這次功課是要用連結串列來時做多項式的加法減法和除法,我是使用了環狀連結串列來處存多項式。

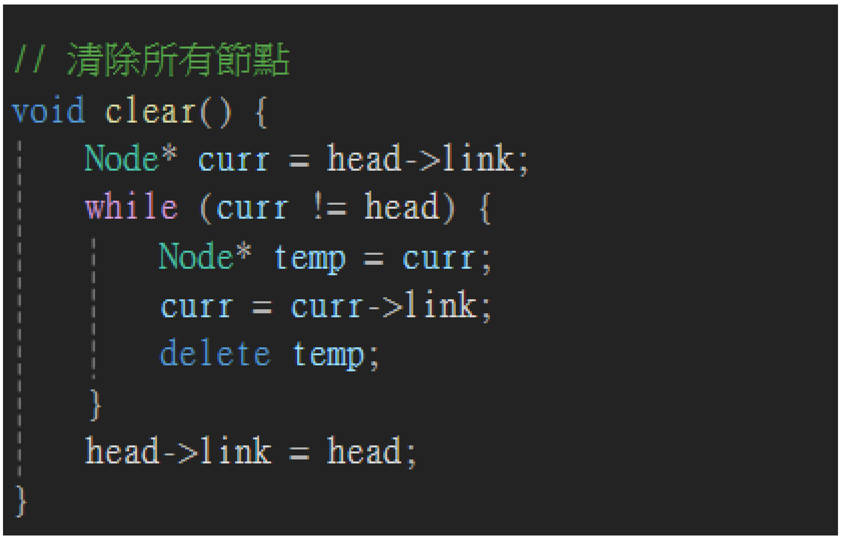
程式說明:



```
// 成員結構
struct Node {
    int coef;     // 係數
    int exp;      // 指數
    Node* link;   // 指向下一個節點的指標
};
```

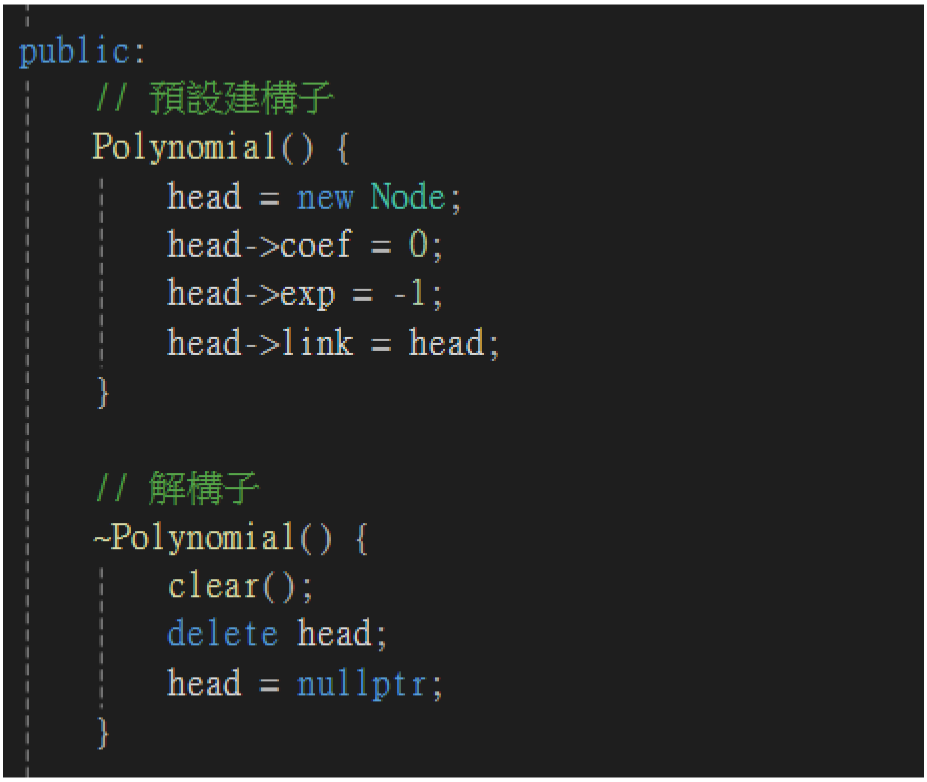
我使用了 struct 來自訂資料結構成員,coef 表示係數,exp 表示指數, link 是指向下一個節點。struct 可以讓使用者創造自己定義的資料型別，但無法定義函式

Polynomial 類別:



```cpp
// 清除所有節點
void clear() {
    Node* curr = head->link;
    while (curr != head) {
        Node* temp = curr;
        curr = curr->link;
        delete temp;
    }
    head->link = head;
}
```

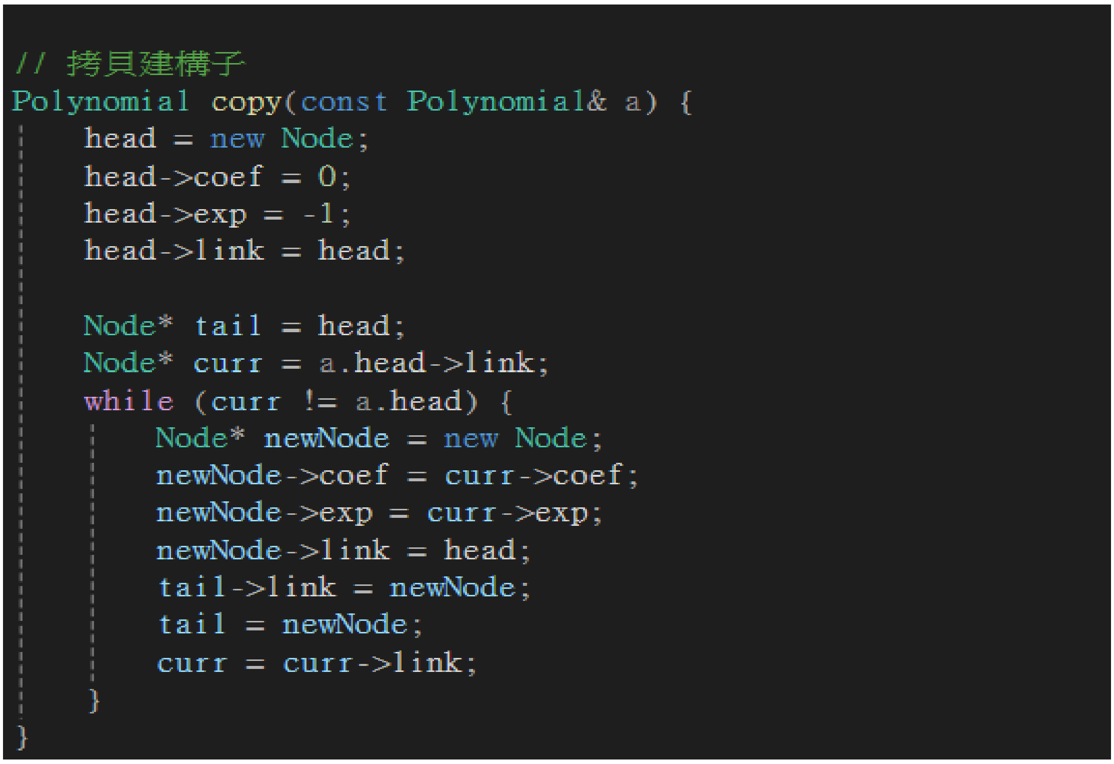
清空多項式的所有項目，釋放動態記憶體，避免記憶體洩漏。



```cpp
public:
    // 預設建構子
    Polynomial() {
        head = new Node;
        head->coef = 0;
        head->exp = -1;
        head->link = head;
    }

    // 解構子
    ~Polynomial() {
        clear();
        delete head;
        head = nullptr;
    }
```

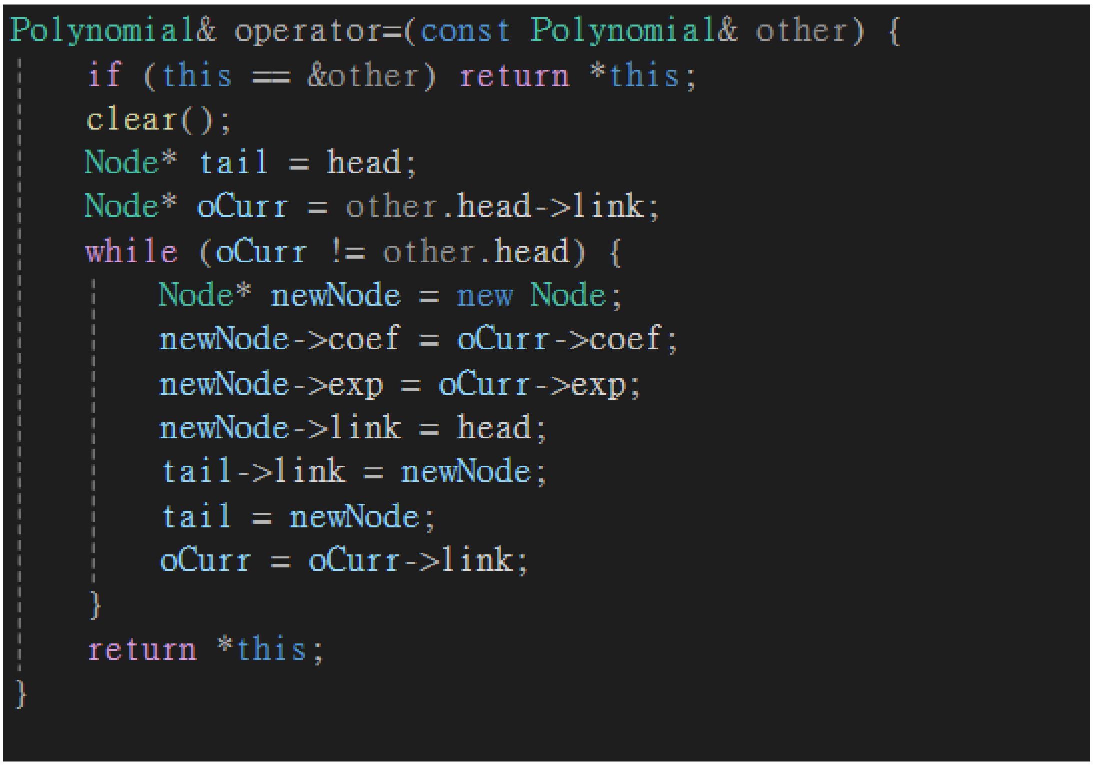
建構子：建立一個空的多項式，其頭節點的指數為 -1。

解構子：釋放所有節點並刪除頭節點。



```
// 拷貝建構子
Polynomial copy(const Polynomial& a) {
    head = new Node;
    head->coef = 0;
    head->exp = -1;
    head->link = head;

    Node* tail = head;
    Node* curr = a.head->link;
    while (curr != a.head) {
        Node* newNode = new Node;
        newNode->coef = curr->coef;
        newNode->exp = curr->exp;
        newNode->link = head;
        tail->link = newNode;
        tail = newNode;
        curr = curr->link;
    }
}
```

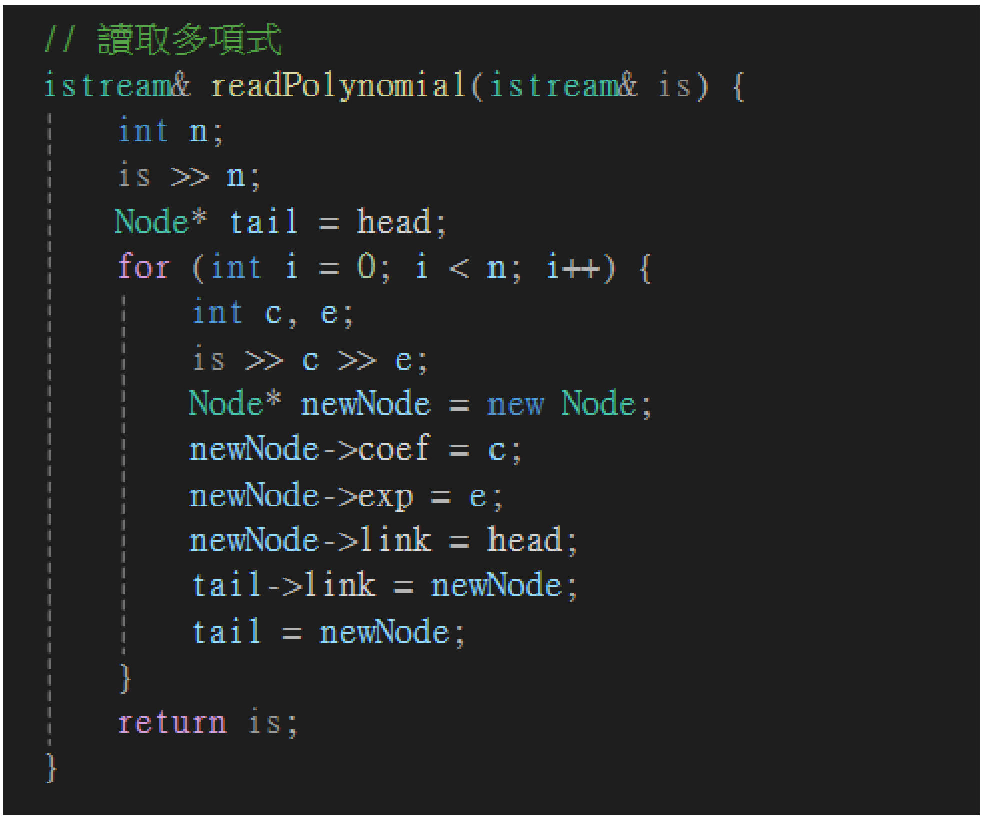
清空多項式的所有項目，釋放動態記憶體，避免記憶體洩漏。



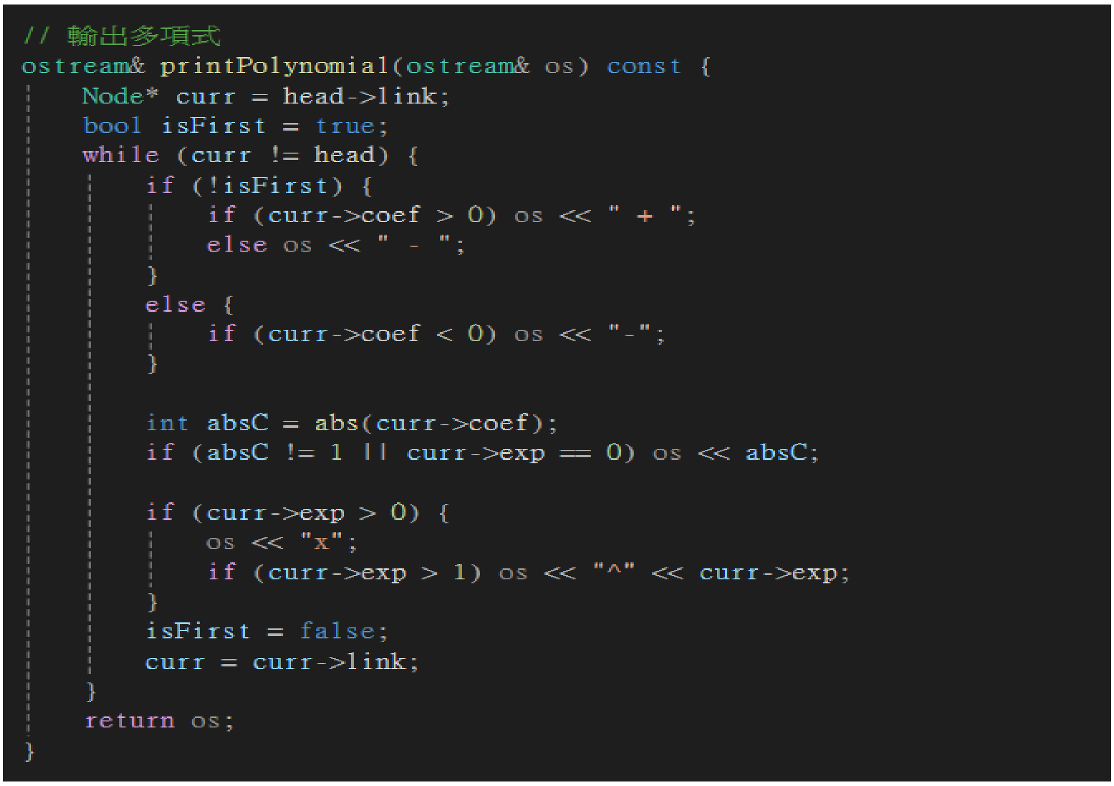
```
Polynomial& operator=(const Polynomial& other) {
    if (this == &other) return *this;
    clear();
    Node* tail = head;
    Node* oCurr = other.head->link;
    while (oCurr != other.head) {
        Node* newNode = new Node;
        newNode->coef = oCurr->coef;
        newNode->exp = oCurr->exp;
        newNode->link = head;
        tail->link = newNode;
        tail = newNode;
        oCurr = oCurr->link;
    }
    return *this;
}
```

賦值運算子,將多項式 a 賦值給多項式* this。

這是輸入多項式和輸出多項式的函式。

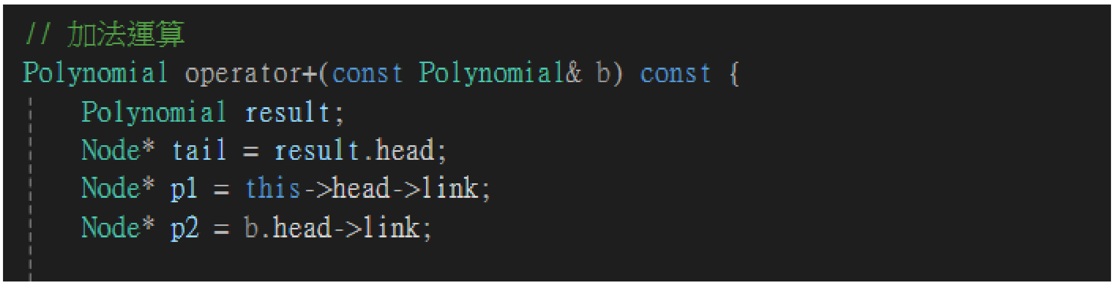


```
// 讀取多項式
istream& readPolynomial(istream& is) {
    int n;
    is >> n;
    Node* tail = head;
    for (int i = 0; i < n; i++) {
        int c, e;
        is >> c >> e;
        Node* newNode = new Node;
        newNode->coef = c;
        newNode->exp = e;
        newNode->link = head;
        tail->link = newNode;
        tail = newNode;
    }
    return is;
}
```



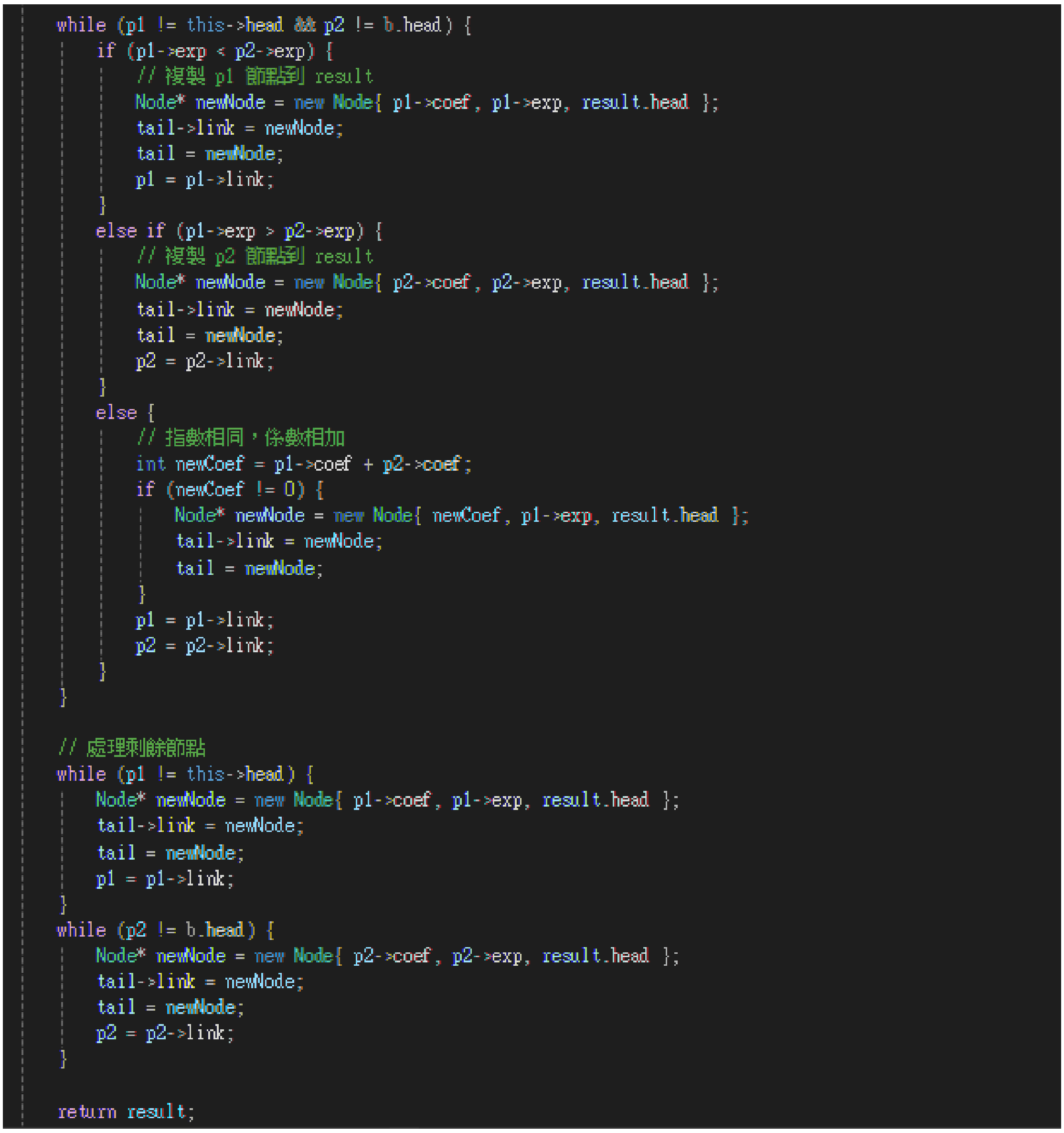
```
// 輸出多項式
ostream& printPolynomial(ostream& os) const {
    Node* curr = head->link;
    bool isFirst = true;
    while (curr != head) {
        if (!isFirst) {
            if (curr->coef > 0) os << " + ";
            else os << " - ";
        }
        else {
            if (curr->coef < 0) os << "-";
        }

        int absC = abs(curr->coef);
        if (absC != 1 || curr->exp == 0) os << absC;

        if (curr->exp > 0) {
            os << "x";
            if (curr->exp > 1) os << "^" << curr->exp;
        }
        isFirst = false;
        curr = curr->link;
    }
    return os;
}
```

這是輸入多項式和輸出多項式的函式。



```
// 加法運算
Polynomial operator+(const Polynomial& b) const {
    Polynomial result;
    Node* tail = result.head;
    Node* p1 = this->head->link;
    Node* p2 = b.head->link;
```

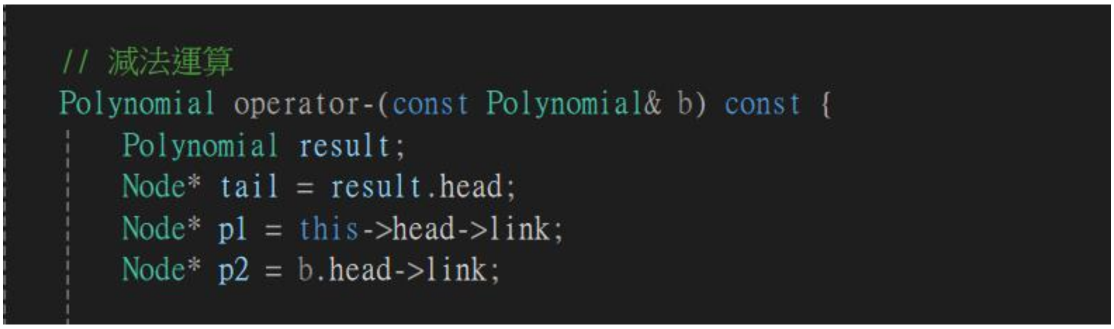
定義多項式加法運算子。



```
    while (p1 != this->head && p2 != b.head) {
        if (p1->exp < p2->exp) {
            // 複製 p1 節點到 result
            Node* newNode = new Node{ p1->coef, p1->exp, result.head };
            tail->link = newNode;
            tail = newNode;
            p1 = p1->link;
        }
        else if (p1->exp > p2->exp) {
            // 複製 p2 節點到 result
            Node* newNode = new Node{ p2->coef, p2->exp, result.head };
            tail->link = newNode;
            tail = newNode;
            p2 = p2->link;
        }
        else {
            // 指數相同，係數相加
            int newCoef = p1->coef + p2->coef;
            if (newCoef != 0) {
                Node* newNode = new Node{ newCoef, p1->exp, result.head };
                tail->link = newNode;
                tail = newNode;
            }
            p1 = p1->link;
            p2 = p2->link;
        }
    }

    // 處理剩餘節點
    while (p1 != this->head) {
        Node* newNode = new Node{ p1->coef, p1->exp, result.head };
        tail->link = newNode;
        tail = newNode;
        p1 = p1->link;
    }
    while (p2 != b.head) {
        Node* newNode = new Node{ p2->coef, p2->exp, result.head };
        tail->link = newNode;
        tail = newNode;
        p2 = p2->link;
    }

    return result;
```

比較指數大小，如果 p1->exp > p2->exp，複製 p1 節點，將其加入 result。如果 p1->exp < p2->exp，複製 p2 節點，將其入

result。如果指數相等，兩節點的係數相加，結果不為零時才加入 result。移動指標，每次處理一個節點後，將指標 p1 或 p2 移到下一個節點，直至某一個多項式結束。若其中一個多項式還有未處理完的節點，則將剩餘節點依序複製到 result。當所有節點處理完後，將返回相加後的多項式。

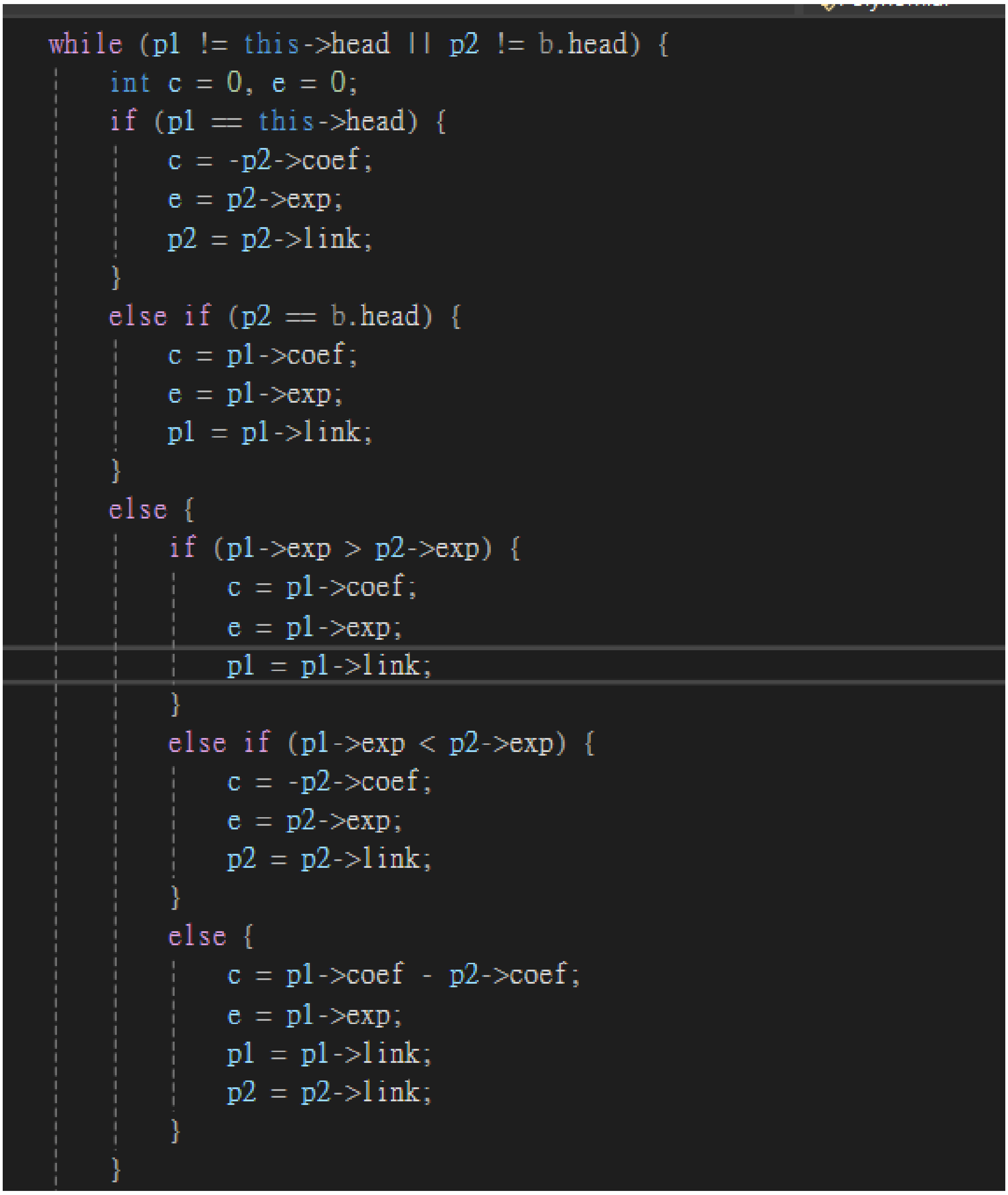


```
// 減法運算
Polynomial operator-(const Polynomial& b) const {
    Polynomial result;
    Node* tail = result.head;
    Node* p1 = this->head->link;
    Node* p2 = b.head->link;
```

result：用來存放結果的多項式。

tail：指向 result 的當前尾節點。

p1 和 p2：分別指向當前多項式*this 與被減多項式 b 的第一個節點。



```
while (p1 != this->head || p2 != b.head) {
    int c = 0, e = 0;
    if (p1 == this->head) {
        c = -p2->coef;
        e = p2->exp;
        p2 = p2->link;
    }
    else if (p2 == b.head) {
        c = p1->coef;
        e = p1->exp;
        p1 = p1->link;
    }
    else {
        if (p1->exp > p2->exp) {
            c = p1->coef;
            e = p1->exp;
            p1 = p1->link;
        }
        else if (p1->exp < p2->exp) {
            c = -p2->coef;
            e = p2->exp;
            p2 = p2->link;
        }
        else {
            c = p1->coef - p2->coef;
            e = p1->exp;
            p1 = p1->link;
            p2 = p2->link;
        }
    }
```

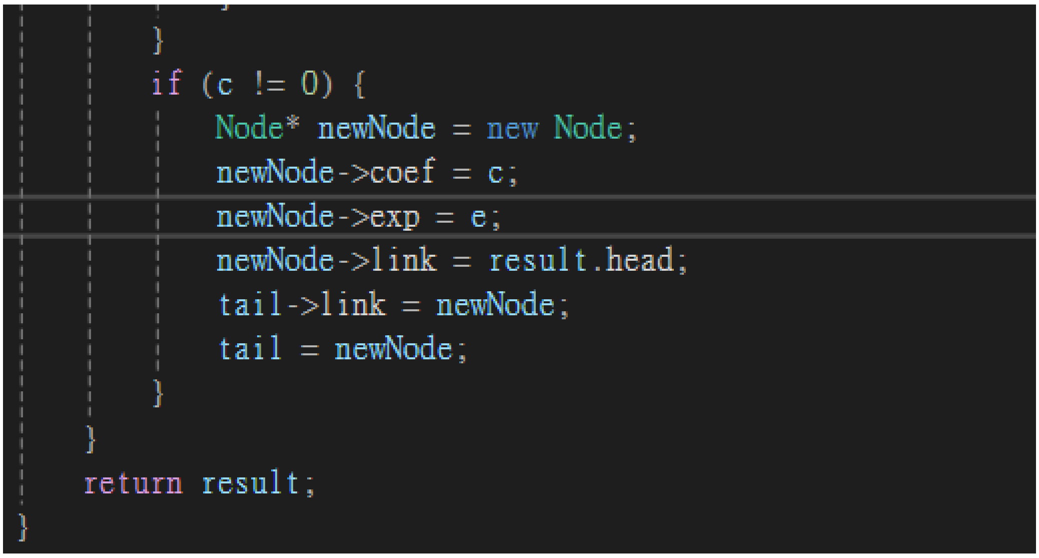
跟多項式相加的邏輯差不多一樣。

當 p1->exp>p2->exp 指數較大的項直接取自 p1，因為 p2 中沒有相同指數的項。

當 p1->exp<p2->exp 指數較大的項直接取自 p2，但系數需要

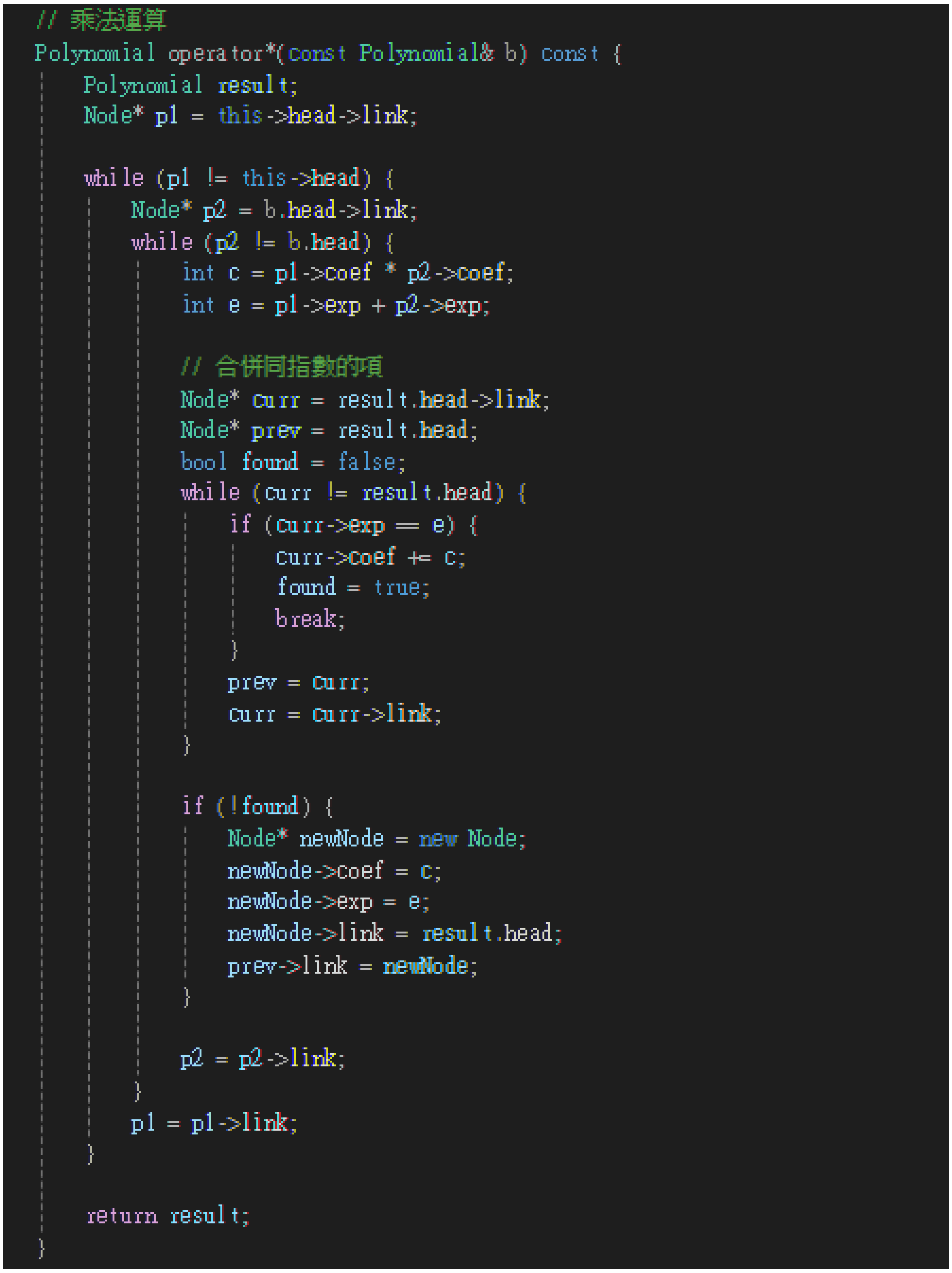
取負，因為是減法。

當 p1−>exp==p2−>exp 指數相同時，計算 p1->coef - p2->coef，並保留指數。將兩個指標同時移動到下一個節點。



```
        }
        if (c != 0) {
            Node* newNode = new Node;
            newNode->coef = c;
            newNode->exp = e;
            newNode->link = result.head;
            tail->link = newNode;
            tail = newNode;
        }
    }
    return result;
}
```

如果系數 c 不為零，則創建一個新的節點，將其加入到結果多項式的鏈表中。新節點：coef 存儲計算得到的系數。exp：存儲指數 link：指向結果多項式的節點。



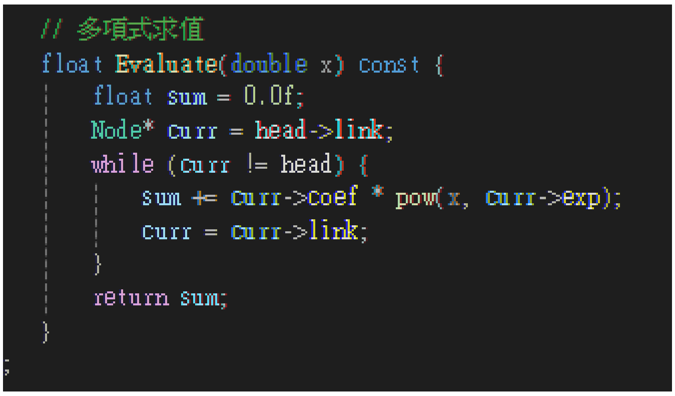
```
// 乘法運算
Polynomial operator*(const Polynomial& b) const {
    Polynomial result;
    Node* p1 = this->head->link;

    while (p1 != this->head) {
        Node* p2 = b.head->link;
        while (p2 != b.head) {
            int c = p1->coef * p2->coef;
            int e = p1->exp + p2->exp;

            // 合併同指數的項
            Node* curr = result.head->link;
            Node* prev = result.head;
            bool found = false;
            while (curr != result.head) {
                if (curr->exp == e) {
                    curr->coef += c;
                    found = true;
                    break;
                }
                prev = curr;
                curr = curr->link;
            }

            if (!found) {
                Node* newNode = new Node;
                newNode->coef = c;
                newNode->exp = e;
                newNode->link = result.head;
                prev->link = newNode;
            }

            p2 = p2->link;
        }
        p1 = p1->link;
    }

    return result;
}
```

每次迴圈，取 p1 和 p2 所代表的多項式項的係數（coef）和指數（exp），然後計算它們的乘積：

係數相乘：c = p1->coef * p2->coef

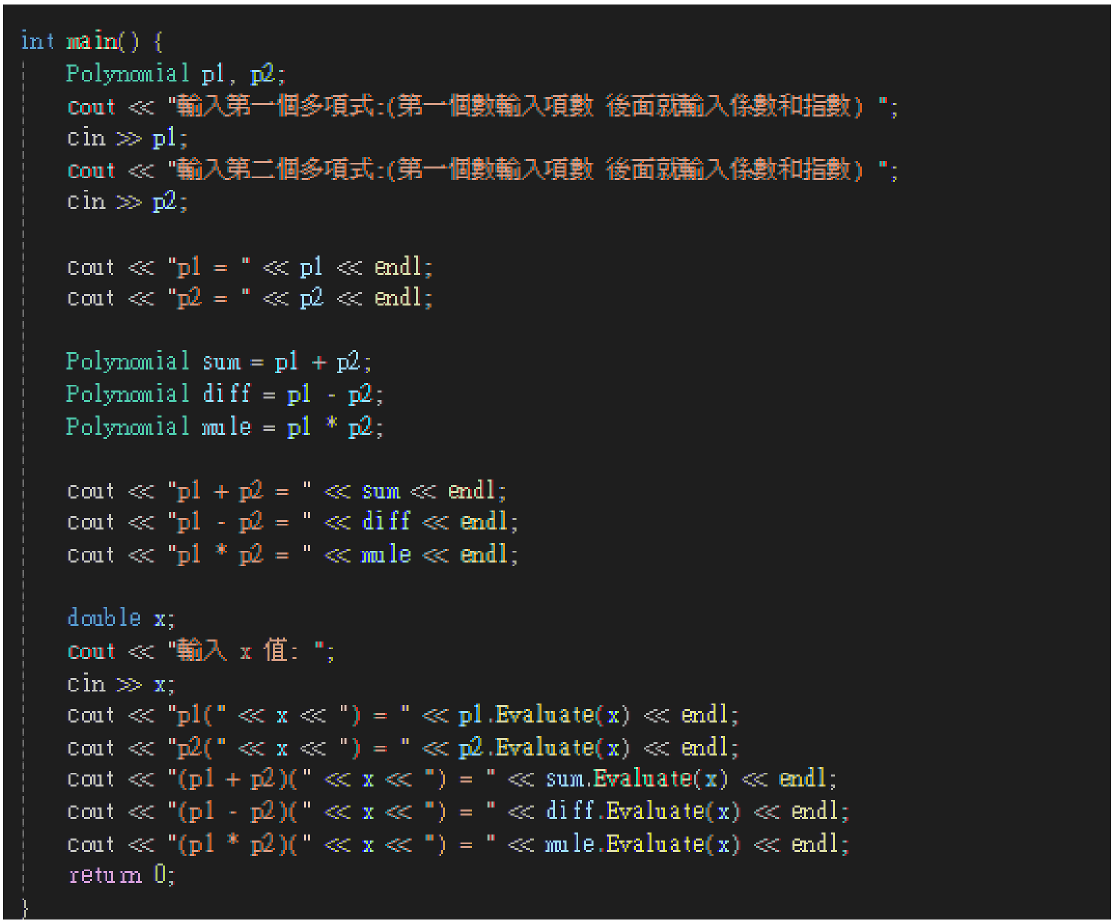
指數相加：e = p1->exp + p2->exp

如果在結果多項式中找不到相同指數的項，則創建一個新的節點 newNode，並將其係數設為 c，指數設為 e。這個新節點會被插入到適當的位置。



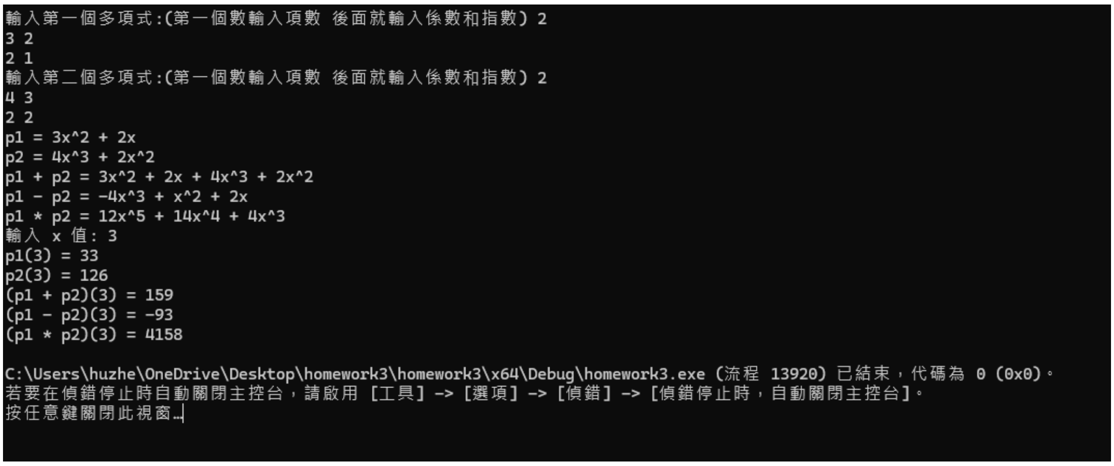
```
    // 多項式求值
    float Evaluate(double x) const {
        float sum = 0.0f;
        Node* curr = head->link;
        while (curr != head) {
            sum += curr->coef * pow(x, curr->exp);
            curr = curr->link;
        }
        return sum;
    }
;
```

帶入使用著輸入的值 並計算出來。



```
int main() {
    Polynomial p1, p2;
    cout << "輸入第一個多項式:(第一個數輸入項數 後面就輸入係數和指數) ";
    cin >> p1;
    cout << "輸入第二個多項式:(第一個數輸入項數 後面就輸入係數和指數) ";
    cin >> p2;

    cout << "p1 = " << p1 << endl;
    cout << "p2 = " << p2 << endl;

    Polynomial sum = p1 + p2;
    Polynomial diff = p1 - p2;
    Polynomial mule = p1 * p2;

    cout << "p1 + p2 = " << sum << endl;
    cout << "p1 - p2 = " << diff << endl;
    cout << "p1 * p2 = " << mule << endl;

    double x;
    cout << "輸入 x 值: ";
    cin >> x;
    cout << "p1(" << x << ") = " << p1.Evaluate(x) << endl;
    cout << "p2(" << x << ") = " << p2.Evaluate(x) << endl;
    cout << "(p1 + p2)(" << x << ") = " << sum.Evaluate(x) << endl;
    cout << "(p1 - p2)(" << x << ") = " << diff.Evaluate(x) << endl;
    cout << "(p1 * p2)(" << x << ") = " << mule.Evaluate(x) << endl;
    return 0;
}
```

主程式的部分較使用者依序輸入兩個多項式，並把相加相減鄉城結果顯示出來，最後使用者輸入 x 值，帶入到多項式裡面，求出答案。



```
輸入第一個多項式:(第一個數輸入項數 後面就輸入係數和指數) 2
3 2
2 1
輸入第二個多項式:(第一個數輸入項數 後面就輸入係數和指數) 2
4 3
2 2
p1 = 3x^2 + 2x
p2 = 4x^3 + 2x^2
p1 + p2 = 3x^2 + 2x + 4x^3 + 2x^2
p1 - p2 = -4x^3 + x^2 + 2x
p1 * p2 = 12x^5 + 14x^4 + 4x^3
輸入 x 值: 3
p1(3) = 33
p2(3) = 126
(p1 + p2)(3) = 159
(p1 - p2)(3) = -93
(p1 * p2)(3) = 4158

C:\Users\huzhe\OneDrive\Desktop\homework3\homework3\x64\Debug\homework3.exe (流程 13920) 已結束，代碼為 0 (0x0)。
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…
```

這是輸出結果。

心得:

這個作業讓我更深入理解了如何使用 C++的環狀鏈結串列來實現多項式類別，並且學習了如何進行多項式的加法減法和乘法的運算。過程中遇到許多困難，我都上網查早資料，或是詢問朋友，在不行我也會詢問 AI 幫我解決，這個過程不僅加強了我對資料結構的理解，也讓我熟悉了如何處理指標和動態記憶體管理。透過這些練習，讓我更加了解了連結串列式如何運行的。