

## 1. 解題說明及申論

這程式使用鏈結串列來實現多項式的存儲和操作，每一個節點 Node 代表多項式中的一項，包含係數和指數，還有一個指針用來連接下一個節點。而多項式類別 Polynomial 用一個環狀鏈結串列來儲存整個多項式，並透過頭節點作為起點。這樣可以動態管理記憶體，不用固定大小來存多項式，用在不同大小的多項式。程式裡面有合併、刪除、加法、減法、乘法和多項式帶值。

```
// 定義節點類別，用於鏈結串列的每一項
```

```
class Node {
```

```
public:
```

```
    double coef; // 係數
```

```
    int exp;      // 指數
```

```
    Node* next;   // 指向下一個節點的指針
```

```

        // 節點構造函數 係數為 0 指數為 0 指向下一個節點為
        nullptr

        Node(double c = 0, int e = 0, Node* n = nullptr) :
        coef(c), exp(e), next(n) {}

    };

    // 定義多項式類別

    class Polynomial {

    private:

        Node* head; // 指向頭節點的指針 多項式的起始位置

```

這段程式碼就是把整個多項式清掉。先看看有沒有東西，如果連頭節點都沒有，就直接結束。如果有的話，從頭節點的下一個節點開始，一個一個把多項式的每一項刪掉。程式先暫時記住當前節點的位置，然後跳到下一個節點，再把剛才的節點刪掉。這樣一路刪到所有的項目都沒了，最後連頭節點也刪掉，然後把頭設成空的，表示多項式已經清理完了。

// 清除多項式中的所有項目

```
void clear() {
```

```
    if (!head) return; // 如果 head 為空返回
```

```
    Node* current = head->next; // 從第一個有效節
```

點開始

```
    while (current != head) { // current = head 就
```

結束 代表走完所有節點

```
        Node* temp = current; // temp 保存當前節點
```

```
        current = current->next; // current 指向下一
```

個節點

```
        delete temp; // 刪除當前節點
```

```
    }
```

```
    delete head; // 刪除頭節點
```

```
    head = nullptr; // 重置 head}
```

這段程式碼是多項式類別的建構函數和解構函數。在建構函數中，初始化了一個頭節點 head，並將 next 指標指向自己，形成一個環狀鏈結結構，代表一個空的多項式。在解構函數中，呼叫 clear() 函式，清除多項式中的所有節點，釋放記憶體。

public:

```
// 構造函數 初始化頭節點 並指向自己
```

```
Polynomial() {
```

```
    head = new Node();
```

```
    head->next = head; // 環形結構
```

```
}
```

```
//清除所有節點
```

```
~Polynomial() {
```

```
    clear();}
```

這段程式碼是多項式中的項目插入和合併。當插入一個新項目時，程式會檢查有沒有同樣次方的項目，如果有相同次方的項目，就把它們的係數相加。如果加完後的係數變成 0，就刪除那一項目。如果沒有相同次方的項目，就建立一個新節點並插入正確的位置。整個過程要確保多項式有按次方的大小排序，相同次方的項目也不會重複的出現。

```
// 多項式合併

void insertTerm(double coef, int exp) {

    if (coef == 0) return; // 若係數為 0 返回

    Node* prev = head; // 指向前一個節點

    Node* current = head->next; // 指向頭節點的下
    一個節點

    // current 的次方大於插入的新項目的次方

    while (current != head && current->exp > exp) {

        prev = current; // prev 指向當前的節點
```

```
current = current->next;//current 指向下一
```

個節點

```
}
```

```
if (current != head && current->exp == exp) {
```

```
// 如果已經有相同次方的項就合併
```

```
current->coef += coef;
```

```
if (current->coef == 0) { // 如果合併後係
```

數為 0

```
prev->next = current->next;//指向下一項
```

```
delete current;//刪除此項
```

```
}
```

```
}
```

```
else {
```

```
// 新節點
```

```
Node* newNode = new Node(coef, exp,
```

```
current);  
  
        prev->next = newNode;  
  
    }  
  
}
```

這段程式就是在複製一個多項式。它會從頭開始把原來多項式的每一項抓過來，一個一個放進新的多項式裡，過程中新多項式會建立自己的頭節點跟環狀結構，然後在逐一複製係數和次方。然後新的多項式長得跟原來的多項式一模一樣，但它們是分開的不會被干擾，像是複製了一份備份一樣。

```
// 複製多項式
```

```
Polynomial(const Polynomial& other) {
```

```
    head = new Node();//新的頭節點
```

```
    head->next = head;//新頭節點的 next 指向自己
```

```

        Node* current = other.head->next; // current 指
向原多項式 other 的第一個有效節點

        while (current != other.head) { // 走過原多項式
的每個項

            insertTerm(current->coef, current->exp);

            // 將當前節點的 coef exp 插入新多項式

            current = current->next; // current 移動到原
下一個節點繼續處理

        }

    }

```

這段程式就是幫多項式進行複製的功能。一開始它會檢查兩個多項式是不是同一個東西。如果是，直接返回就好了。如果不是，就把現在的多項式清空，然後重新建立一個新的結構，再把目標多項式的每一項一項地複製過來。做完後這個多項式就跟目標多項式一樣了，但它們還是兩個獨立的。



//檢查多項式是否一致

```
Polynomial& operator=(const Polynomial& other) {
```

```
    if (this == &other) return *this; // 如果
```

\*this 和 other 是同一個對象 返回當前對象

```
    clear(); // 清除當前多項式
```

```
    head = new Node(); // 新的頭節點
```

```
    head->next = head; // 重建頭節點
```

```
    Node* current = other.head->next; // current 指
```

向原多項式 other 的第一個有效節點

```
    while (current != other.head) { // 走過原多項式
```

的每個項

```
        insertTerm(current->coef, current->exp);
```

```
    // 插入每一項
```

```
    current = current->next; // current 移動到原
```

下一個節點繼續處理

```
}
```

```
return *this; // 返回當前對象 }
```

這段程式碼是輸入多項式。讀取一個多項式並儲存到 Polynomial 物件中。會讀取整行的輸入並轉換成字串流。然後在逐一處理每一項，先判斷符號是正還是負，然後讀取係數。如果符號是 "-" 係數就要加上負號。如果有字元 "x"，再判斷有沒有次方的字元 "^"，如果有，輸入次方。如果沒有次方，預設次方是 1。最後會把這些讀取到的係數和次方傳到 insertTerm 函數來插入到多項式中。做到讀取完所有項目，最後會回傳多項式。

```
// 輸入多項式

friend istream& operator>>(istream& is, Polynomial&
poly) {

    string input;

    getline(is, input); // 讀取整行輸入

    istringstream ss(input); // 將輸入轉為字串流

    char sign = '+'; // 記錄項目前的符號

    while (ss) {

        double coef = 1.0, exp = 0; // 係數為 1 次
```

方為 0

```
if (ss.peek() == '+' || ss.peek() == '-') {  
  
    sign = ss.get(); // 讀取符號  
  
}
```

```
if (isdigit(ss.peek())) {  
  
    ss >> coef; // 讀取係數  
  
}
```

```
if (sign == '-') coef = -coef; // 如果符號  
式 '-' 係數加 '-'
```

```
if (ss.peek() == 'x') { //如果有變數 x  
  
    ss.get(); // 跳過 'x'  
  
    if (ss.peek() == '^') {  
  
        ss.get(); // 跳過 '^'  
  
        ss >> exp; // 讀取次方  
  
    }
```

```

else {

    exp = 1; // 如果有'X'但沒次方 次方
    設為 1

}

}

poly.insertTerm(coef, exp); // 傳入
insertTerm 合併

}

return is;//回傳多項式}

```

這段程式碼的作用是輸出多項式。它會依序檢查多項式的每一項，根據係數和次方大小來決定輸出的先後順序。程式會檢查每一項的符號，如果係數 $>0$ 也不是第一項的話加上 "+"，如果係數 $<0$ 的話，加上負號 "-"。然後係數都要用絕對值來顯示輸出數字，因為前面已經處理正負號的問題了，如果次方大於 0 輸出 "x"，如果次方大於 1 輸出 "^" 再輸入次方，然後在移到下一個項目繼續上述的動作。如果多項式是空的，程式會輸出 0。最後回傳多項式。

```

// 輸出多項式

friend ostream& operator<<(ostream& os, const
Polynomial& poly) {

    Node* current = poly.head->next; // 從第一個有
效節點開始

    bool first = true; // 標誌為第一項

    while (current != poly.head) { // 走過所有項

        if (current->coef > 0 && !first) os << " +
"; // 如果係數大於 0 且不是第一項加上 '+'

        if (current->coef < 0) os << (first ? "-" :
" - "); // 若係數小於 0 就看有沒有 '-'

        os << abs(current->coef); // 輸出係數加絕對
值

        if (current->exp > 0) os << "x"; // 如果
次方大於 0 輸出 x

        if (current->exp > 1) os << "^" << current-
>exp; // 如果次方大於 1 輸出 '^' 再輸入次方

```

```

        current = current->next; // 移動到下一項

        first = false; // 設置 first 為 false
    }

    if (first) os << "0"; /// 若多項式為空輸出 0

    return os;//回傳多項式
}

```

多項式的加法運算。要將兩個多項式相加。程式會建立一個空的多項式 `result` 來存儲加法的結果，然後分別用 `a` 和 `b` 指向兩個多項式的開頭。然後在比較兩個多項式，如果 `a` 中的項次方大於 `b` 中的項次方，就直接將 `a` 中的係數和次方加到 `result` 中。如果 `b` 中的項次方大於 `a` 中的項次方，也直接將 `b` 中的係數和次方加到 `result` 中。那如果兩個項次方相同時，會把係數相加然後結果加到 `result` 中，然後在跳到下一項繼續執行。最後程式會返回加法後的多項式。

// 多項式加法

```
Polynomial operator+(const Polynomial& other) const  
{
```

```
    Polynomial result; // 儲存結果
```

```
    Node* a = head->next;
```

```
    Node* b = other.head->next;
```

```
    while (a != head || b != other.head) { // 當沒  
        走完時
```

```
        if (a != head && (b == other.head || a->exp  
            > b->exp)) { // 如果當前 a 中的項次方大於 b 中的項次方
```

```
            result.insertTerm(a->coef, a->exp); //  
            直接將 a 中的係數和次方加到 result 中
```

```
            a = a->next; // a 移到下一項
```

```
        }
```

```
        else if (b != other.head && (a == head ||  
            a->exp < b->exp)) { // 如果 b 中的項次方大於 a 中的項次方
```

```
            result.insertTerm(b->coef, b->exp); //
```

直接將 b 中的係數和次方加到 result 中

```
        b = b->next;// b 移到下一項

    }

    else {

        result.insertTerm(a->coef + b->coef, a-
>exp); // 合併相同次方的項

        a = a->next;// a 移到下一項

        b = b->next;// b 移到下一項

    }

}

return result;//回傳結果

}
```



多項式的減法運算。要將兩個多項式相減。首先程式會建立一個空的多項式 `result` 來存儲減法的結果，然後分別用 `a` 和 `b` 指向兩個多項式的開頭。然後在比較兩個多項式，如果 `a` 中的項次方大於 `b` 中的項次方，就直接將 `a` 中的係數和次方加到 `result` 中。如果 `b` 中的項次方大於 `a` 中的項次方，那 `b` 的係數要多負號，因為是在減去 `b` 中的項。那如果兩個項次方相同時，會把係數相減然後結果加到 `result` 中，然後在跳到下一項繼續執行。最後程式會返回減法後的多項式。

```
// 多項式減法
```

```
Polynomial operator-(const Polynomial& other) const
{
    Polynomial result;// 儲存結果

    Node* a = head->next;

    Node* b = other.head->next;

    while (a != head || b != other.head) {

        if (a != head && (b == other.head || a->exp
```

> b->exp)) { //如果當前 a 中的項次方大於 b 中的項次方

result.insertTerm(a->coef, a->exp); //直

接將 a 中的係數和次方加到 result 中

a = a->next; // a 移到下一項

}

else if (b != other.head && (a == head ||

a->exp < b->exp)) { //如果 b 中的項次方大於 a 中的項次方

result.insertTerm(-b->coef, b->exp); //

b 中的係數取負和次方加到 result 中 係數需要取負因為是在

減去 b 中的項

b = b->next; // b 移到下一項

}

else { //如果 a 和 b 中的次方相同 則 a 和 b

中相同次方的項進行相減 結果加到 result 中

result.insertTerm(a->coef - b->coef, a->

exp);

a = a->next; // a 移到下一項

```

        b = b->next;// b 移到下一項

    }

}

return result;//回傳結果

}

```

多項式的乘法運算。是把兩個多項式的係數相乘，再把它們的指數加起來，然後結果加到新的多項式中。先建立一個空的多項式 `result` 來儲存結果，然後會走過多項式 `a` 中的每一項，對每一項再和多項式 `b` 中的每一項做運算。運算是將兩個多項式的係數相乘，次方相加，再把結果存入 `result` 中。最後返回運算結果。

```

// 多項式乘法

Polynomial operator*(const Polynomial& other) const
{

    Polynomial result;// 儲存結果

```

```

Node* a = head->next;

while (a != head) { // 遍歷 a 中的每一項

    Node* b = other.head->next;

    while (b != other.head) { // 遍歷 b 中的每一項

        result.insertTerm(a->coef * b->coef, a->exp + b->exp); // 係數相乘 指數相加 結果加到 result 中

        b = b->next; // b 移到下一項

    }

    a = a->next; // a 移到下一項

}

return result; // 回傳結果

}

```

這段程式碼用來計算多項式在指定  $x$  值運算後的結果。一開始宣告 `result` 用來累加每一項的計算結果，然後程式會走過多項式中的每一項，每一項都會用指定的  $x$  值來做運算，然後再把這些結果累加到 `result` 中，做完一項會跳到下一個項目繼續運算，最後返回結果。

```
// 計算多項式的值 帶入輸入的 x 值

double evaluate(int x) const {

    double result = 0.0f; // 儲存結果

    Node* current = head->next;

    while (current != head) { // 走過每一項

        result += current->coef * pow(x, current-
>exp); // 根據 x 的值計算每項結果並累加

        current = current->next; // 移動到下一項繼續
計算

    }

    return result; // 回傳結果 }};
```

這段程式碼的主要目的是進行多項式的基本運算加法、減法、乘法和帶入  $x$ ，並計算每個運算所需的時間，才可以知道效能量測是多少。程式首先要輸入兩個多項式  $a$  和  $b$ ，然後分別運算加法、減法和乘法運算和帶入  $x$ 。每個運算程式會記錄開始和結束的時間，算出每次運算的執行時間，這樣就可以知道每個運算所花的時間。然後程式要在輸入一個整數  $x$ ，再把  $x$  的值帶入多項式  $a$  和  $b$  中，再算出值。就可以看到多項式對  $x$  值的計算結果，也能看到計算這些值所需的時間。

```
int main() {  
  
    clock_t start, finish; // 用於計時的變數  
  
    Polynomial a, b; // 宣告兩個多項式  
  
    cout << "a=";  
  
    cin >> a; // 輸入多項式 a  
  
    cout << "b=";
```

```
cin >> b; // 輸入多項式 b

cout << "-----
-----" <<
endl;

// 加法操作並計算時間

start = clock();

Polynomial sum = a + b;

finish = clock();

cout << "a + b = " << sum << endl;

cout << "sum() 需時: " << (double)(finish - start)
/ CLOCKS_PER_SEC << " s" << endl;

// 減法操作並計算時間

start = clock();
```

```

Polynomial diff = a - b;

finish = clock();

cout << "a - b = " << diff << endl;

cout << "diff() 需時: " << (double)(finish - start)
/ CLOCKS_PER_SEC << " s" << endl;


// 乘法操作並計算時間

start = clock();

Polynomial prod = a * b;

finish = clock();

cout << "a * b = " << prod << endl;

cout << "prod() 需時: " << (double)(finish - start)
/ CLOCKS_PER_SEC << " s" << endl;


cout << "-----"
-----" <<

```



```
endl;
```

```
int x;
```

```
cout << "輸入 x 的值來計算多項式 a(x)、b(x)：";
```

```
cin >> x;
```

```
// 計算 a(x)並計算時間
```

```
start = clock();
```

```
cout << "a(" << x << ") = " << a.evaluate(x) <<
```

```
endl;
```

```
finish = clock();
```

```
cout << "a(x) 需時: " << (double)(finish - start) /
```

```
CLOCKS_PER_SEC << " s" << endl;
```

```
// 計算 b(x)並計算時間
```

```
start = clock();
```

```

        cout << "b(" << x << ") = " << b.evaluate(x) <<
endl;

        finish = clock();

        cout << "b(x) 需時: " << (double)(finish - start) /
CLOCKS_PER_SEC << " s" << endl;

        return 0;
}

```

## 2. Algorithm Design & Programming

```

1 #include <iostream>
2 #include <cmath>
3 #include <sstream>
4 #include <string>
5 #include <ctime>
6
7 using namespace std;
8
9 // 定義節點類別 用於鏈結串列的每一項
10 class Node {
11 public:
12     double coef; // 系數
13     int exp; // 指數
14     Node* next; // 指向下一個節點的指針
15
16     // 節點構造函數 係數為0 指數為0 指向下一個節點為nullptr
17     Node(double c = 0, int e = 0, Node* n = nullptr) : coef(c), exp(e), next(n) {}
18 };
19
20 // 定義多項式類別
21 class Polynomial {
22 private:
23     Node* head; // 指向頭節點的指針 多項式的起始位置
24
25     // 清除多項式中的所有項目
26     void clear() {
27         if (!head) return; // 如果head為空返回
28     }
29 };

```

```

// 清除多項式中的所有項目
void clear() {
    if (!head) return; // 如果head為空返回
    Node* current = head->next; // 從第一個有效節點開始
    while (current != head) { // current = head就結束 代表走完所有節點
        Node* temp = current; // temp 保存當前節點
        current = current->next; // current 指向下一個節點
        delete temp; // 刪除當前節點
    }
    delete head; // 刪除頭節點
    head = nullptr; // 重置head
}

public:
// 構造函數 初始化頭節點 並指向自己
Polynomial() {
    head = new Node();
    head->next = head; // 環形結構
}

// 清除所有節點
~Polynomial() {
    clear();
}

// 多項式合併

```

```

// 多項式合併
void insertTerm(double coef, int exp) {
    if (coef == 0) return; // 若係數為0返回
    Node* prev = head; // 指向前一個節點
    Node* current = head->next; // 指向頭節點的下一個節點

    // current 的次方大於插入的新項目的次方
    while (current != head && current->exp > exp) {
        prev = current; // prev 指向當前的節點
        current = current->next; // current 指向下一個節點
    }

    if (current != head && current->exp == exp) {
        // 如果次方相同就合併
        current->coef += coef; // 係數相加
        if (current->coef == 0) { // 如果合併後係數為0
            prev->next = current->next; // 指向下一項
            delete current; // 刪除此項
        }
    }
    else {
        // 新節點
        Node* newNode = new Node(coef, exp, current);
        prev->next = newNode;
    }
}

```

```

        prev->next = newNode;
    }
}

// 複製多項式
Polynomial(const Polynomial& other) {
    head = new Node();//新的頭節點
    head->next = head;//新頭節點的 next 指向自己
    Node* current = other.head->next; // current指向原多項式 other 的第一個有效節點
    while (current != other.head) { // 走過原多項式的每個項
        insertTerm(current->coef, current->exp); // 將當前節點的 coef exp插入新多項式
        current = current->next;//current 移動到原下一個節點繼續處理
    }
}

//檢查多項式是否一致
Polynomial& operator=(const Polynomial& other) {
    if (this == &other) return *this; // 如果 *this 和 other 是同一個對象 返回當前對象
    clear(); // 清除當前多項式
    head = new Node();//新的頭節點
    head->next = head; // 重建頭節點
    Node* current = other.head->next; // current指向原多項式 other 的第一個有效節點
    while (current != other.head) { // 走過原多項式的每個項
        insertTerm(current->coef, current->exp); // 插入每一項
        current = current->next;//current 移動到原下一個節點繼續處理
    }
    return *this;//返回當前對象
}

```

```

97         current = current->next;//current 移動到原下一個節點繼續處理
98     }
99     return *this;//返回當前對象
100 }
101
102
103
104 // 輸入多項式
105 friend istream& operator>>(istream& is, Polynomial& poly) {
106     string input;
107     getline(is, input); // 讀取整行輸入
108     istringstream ss(input); // 將輸入轉為字串流
109     char sign = '+'; // 記錄項目前的符號
110     while (ss) {
111         double coef = 1.0, exp = 0; // 係數為1 次方為0
112         if (ss.peek() == '+' || ss.peek() == '-') {
113             sign = ss.get(); // 讀取符號
114         }
115         if (isdigit(ss.peek())) {
116             ss >> coef; // 讀取係數
117         }
118         if (sign == '-') coef = -coef; // 如果符號是 '-' 係數加 '-'
119
120         if (ss.peek() == 'x') { //如果有字元 'x'
121             ss.get(); // 跳過 'x'
122             if (ss.peek() == '^') { //如果有字元 '^'
123                 ss.get(); // 跳過 '^'

```

```

121         ss.get(); // 跳過 'x'
122         if (ss.peek() == '^') { // 如果有字元 '^'
123             ss.get(); // 跳過 '^'
124             ss >> exp; // 讀取次方
125         }
126         else {
127             exp = 1; // 如果有 'x' 但沒次方 次方設為 1
128         }
129     }
130     poly.insertTerm(coef, exp); // 傳入 insertTerm 合併
131 }
132 return is; // 回傳多項式
133 }
134
135 // 輸出多項式
136 friend ostream& operator<<(ostream& os, const Polynomial& poly) {
137     Node* current = poly.head->next; // 從第一個有效節點開始
138     bool first = true; // 標誌為第一項
139     while (current != poly.head) { // 走過所有項
140         if (current->coef > 0 && !first) os << " + "; // 如果係數大於 0 且不是第一項加上 '+'
141         if (current->coef < 0) os << (first ? "-" : "- "); // 如果係數小於 0 就看有沒有 '-' 如果 true 輸出 '-' 不是的話輸出 ' - '
142         os << abs(current->coef); // 輸出係數加絕對值
143         if (current->exp > 0) os << "x"; // 如果次方大於 0 輸出 'x'
144         if (current->exp > 1) os << "^" << current->exp; // 如果次方大於 1 輸出 '^' 再輸入次方
145         current = current->next; // 移動到下一項
146         first = false; // 設置 first 為 false
147     }

```

```

145         current = current->next; // 移動到下一項
146         first = false; // 設置 first 為 false
147     }
148     if (first) os << "0"; // 如果多項式為空輸出 0
149     return os; // 回傳多項式
150 }
151
152 // 多項式加法
153 Polynomial operator+(const Polynomial& other) const {
154     Polynomial result; // 儲存結果
155     Node* a = head->next;
156     Node* b = other.head->next;
157     while (a != head || b != other.head) {
158         if (a != head && (b == other.head || a->exp > b->exp)) { // 如果當前 a 中的項次方大於 b 中的項次方
159             result.insertTerm(a->coef, a->exp); // 直接將 a 中的係數和次方加到 result 中
160             a = a->next; // a 移到下一項
161         }
162         else if (b != other.head && (a == head || a->exp < b->exp)) { // 如果 b 中的項次方大於 a 中的項次方
163             result.insertTerm(b->coef, b->exp); // 直接將 b 中的係數和次方加到 result 中
164             b = b->next; // b 移到下一項
165         }
166         else {
167             result.insertTerm(a->coef + b->coef, a->exp); // 合併相同次方的項
168             a = a->next; // a 移到下一項
169             b = b->next; // b 移到下一項
170         }
171     }

```

```

169         b = b->next; // b 移到下一項
170     }
171 }
172 return result; // 回傳結果
173 }
174
175 // 多項式減法
176 Polynomial operator-(const Polynomial& other) const {
177     Polynomial result; // 儲存結果
178     Node* a = head->next;
179     Node* b = other.head->next;
180     while (a != head || b != other.head) {
181         if (a != head && (b == other.head || a->exp > b->exp)) { // 如果當前 a 中的項次方大於 b 中的項次方
182             result.insertTerm(a->coef, a->exp); // 直接將 a 中的係數和次方加到 result 中
183             a = a->next; // a 移到下一項
184         }
185         else if (b != other.head && (a == head || a->exp < b->exp)) { // 如果 b 中的項次方大於 a 中的項次方
186             result.insertTerm(-b->coef, b->exp); // b 中的係數取負和次方加到 result 中 係數需要取負因為是在減去 b 中的項
187             b = b->next; // b 移到下一項
188         }
189         else { // 如果 a 和 b 中的次方相同 a 和 b 中相同次方的項要係數相減 結果加到 result 中
190             result.insertTerm(a->coef - b->coef, a->exp);
191             a = a->next; // a 移到下一項
192             b = b->next; // b 移到下一項
193         }
194     }

```

```

193     }
194 }
195 return result;//回傳結果
196 }
197
198 // 多項式乘法
199 Polynomial operator*(const Polynomial& other) const {
200     Polynomial result;// 儲存結果
201     Node* a = head->next;
202     while (a != head) { // 走過a的每一項
203         Node* b = other.head->next;
204         while (b != other.head) { // 走過b的每一項
205             result.insertTerm(a->coef * b->coef, a->exp + b->exp); // 係數相乘 指數相加 結果加到 result 中
206             b = b->next;// b 移到下一項
207         }
208         a = a->next;// a 移到下一項
209     }
210     return result;//回傳結果
211 }
212
213 // 計算多項式的值 帶入輸入的x值
214 double evaluate(int x) const {
215     double result = 0.0f;// 儲存結果
216     Node* current = head->next;
217     while (current != head) { // 走過每一項
218         result += current->coef * pow(x, current->exp); // 根據x的值計算每項結果再累加
219         current = current->next;//移動到下一項繼續計算

```

```

217     while (current != head) { // 走過每一項
218         result += current->coef * pow(x, current->exp); // 根據x的值計算每項結果再累加
219         current = current->next;//移動到下一項繼續計算
220     }
221     return result;//回傳結果
222 }
223
224
225 int main() {
226     clock_t start, finish; // 用於計時的變數
227     Polynomial a, b; // 宣告兩個多項式
228
229     cout << "a=";
230     cin >> a; // 輸入多項式a
231
232     cout << "b=";
233     cin >> b; // 輸入多項式b
234
235     cout << "-----" << endl;
236
237     // 加法操作並計算時間
238     start = clock();
239     Polynomial sum = a + b;
240     finish = clock();
241     cout << "a + b = " << sum << endl;
242     cout << "sum() 需時: " << (double)(finish - start) / CLOCKS_PER_SEC << " s" << endl;
243

```

```

235     cout << "-----" << endl;
236
237     // 加法操作並計算時間
238     start = clock();
239     Polynomial sum = a + b;
240     finish = clock();
241     cout << "a + b = " << sum << endl;
242     cout << "sum() 需時: " << (double)(finish - start) / CLOCKS_PER_SEC << " s" << endl;
243
244     // 減法操作並計算時間
245     start = clock();
246     Polynomial diff = a - b;
247     finish = clock();
248     cout << "a - b = " << diff << endl;
249     cout << "diff() 需時: " << (double)(finish - start) / CLOCKS_PER_SEC << " s" << endl;
250
251     // 乘法操作並計算時間
252     start = clock();
253     Polynomial prod = a * b;
254     finish = clock();
255     cout << "a * b = " << prod << endl;
256     cout << "prod() 需時: " << (double)(finish - start) / CLOCKS_PER_SEC << " s" << endl;
257
258     cout << "-----" << endl;
259
260     int x;
261     cout << "輸入 x 的值並計算多項式 (a+b)(x) 的值: ";

```

```

256     cout << "prod() 需時: " << (double)(finish - start) / CLOCKS_PER_SEC << " s" << endl;
257
258     cout << "-----" << endl;
259
260     int x;
261     cout << "輸入 x 的值來計算多項式 a(x) · b(x) : ";
262     cin >> x;
263
264     // 計算a(x)並計算時間
265     start = clock();
266     cout << "a(" << x << ") = " << a.evaluate(x) << endl;
267     finish = clock();
268     cout << "a(x) 需時: " << (double)(finish - start) / CLOCKS_PER_SEC << " s" << endl;
269
270     // 計算b(x)並計算時間
271     start = clock();
272     cout << "b(" << x << ") = " << b.evaluate(x) << endl;
273     finish = clock();
274     cout << "b(x) 需時: " << (double)(finish - start) / CLOCKS_PER_SEC << " s" << endl;
275
276     return 0;
277 }
278
279 5x^9-3x^8+2x^7+x^6-4x^5+6x^4-8x^3+7x^2-9x+10
280 -2x^10+4x^9-6x^8+8x^7-10x^6+12x^5-14x^4+16x^3-18x^2+20x-22
281 2
282 */

```

### 3. 效能分析 Time complexity & Space complexity

時間複雜度：

- 插入項 (insertTerm):  $O(n)$  — 需要遍歷鏈表找到插入位置。
- 加法 (operator +):  $O(n + m)$  — 同時遍歷兩個多項式的項，進行合併。
- 減法 (operator -):  $O(n + m)$  — 同加法，遍歷兩個多項式進行項的減法。
- 乘法 (operator \*):  $O(n * m)$  — 每一項與另一多項式的每一項相乘。
- 計算多項式值 (evaluate):  $O(n)$  — 遍歷所有項計算結果。

空間複雜度：

- 節點存儲： $O(n)$  — 每個多項式的項都需一個節點。
- 操作後的結果多項式： $O(n + m)$ （加法、減法）， $O(n * m)$ （乘法） — 依據結果的項數。

#### 4. 測試與驗證

Input:

➤  $5x^9 - 3x^8 + 2x^7 + x^6 - 4x^5 + 6x^4 - 8x^3 + 7x^2 - 9x + 10$

➤  $-2x^{10} + 4x^9 - 6x^8 + 8x^7 - 10x^6 + 12x^5 - 14x^4 + 16x^3 - 18x^2 + 20x - 22$

➤ 2



Output:

```
Microsoft Visual Studio 偵錯主控台
a=5x^9-3x^8+2x^7+x^6-4x^5+6x^4-8x^3+7x^2-9x+10
b=-2x^10+4x^9-6x^8+8x^7-10x^6+12x^5-14x^4+16x^3-18x^2+20x-22
-----
a + b = -2x^10 + 9x^9 - 9x^8 + 10x^7 - 9x^6 + 8x^5 - 8x^4 + 8x^3 - 11x^2 + 11x - 12
sum() 需時: 0 s
a - b = 2x^10 + 1x^9 + 3x^8 - 6x^7 + 11x^6 - 16x^5 + 20x^4 - 24x^3 + 25x^2 - 29x + 32
diff() 需時: 0 s
a * b = -10x^19 + 26x^18 - 46x^17 + 64x^16 - 74x^15 + 72x^14 - 54x^13 + 22x^12 + 28x^11 - 98x^10 +
168x^9 - 358x^8 + 510x^7 - 644x^6 + 710x^5 - 702x^4 + 638x^3 - 514x^2 + 398x - 220
prod() 需時: 0 s
-----
輸入 x 的值來計算多項式 a(x)、b(x): 2
a(2) = 2036
a(x) 需時: 0.002 s
b(2) = -918
b(x) 需時: 0.001 s
C:\Users\Lenovo\OneDrive\Desktop\Project1\Debug\Project1.exe (處理序 2528) 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...
```

## 5. 效能量測

相加的過程用了 0 秒

相減的過程用了 0 秒

相乘的過程用了 0 秒

把數值帶入 a 多項式的過程用了 0.002 秒

把數值帶入 b 多項式的過程用了 0.001 秒

## 6. 心得討論

這次作業讓我學到了怎麼用鏈結串列來寫程式。首先是用 Node 類別代表多項式的每一項，然後再用 Polynomial 類別把整個多項式包起來。那在這程式我是用頭節點當作開始也可以當作結束。透過 C++ 的運算子多載，可以讓多項式的加減乘運算變得像數學公式一樣簡單。還有程式裡也處理了記憶體的問題，比如用解構子和 `clear()` 來清理釋放記憶體。最讓我困惑的部分是拷貝建構子和拷貝指派運算子，因為我之前沒寫過這種程式，結果去 chatgpt 查詢後發現它的難點是要處理深層複製和自我指派的問題，不能讓兩個物件指向同一塊記憶體，比如說：如果直接複製指標，兩個物件就會指向同一塊記憶體，這樣的話如果有一個物件被刪除，另一個物件就會釋放或著也被刪除掉，這樣就會出現問題。而且如果兩個物件都去刪除那塊記憶體，就會造成錯誤。所以為了解決這個情況，需要做深層複製，也就是複製每個節點，這樣每個物件就有自己的記憶體，就不會互相干擾到。程式就是 `if (this == &other)` 來檢查是否自我指派，同時要確保每次拷貝都新建節點，才成功解決了這個問題。這次作業讓我學習到鏈結串列，還有上述說的困難點，也加深了對物件導向的理解，對我來說是個蠻有挑戰性，也有收穫的經驗。