四資工二乙 張大軒 **41143229 HW3**

[程式功能]

1. 資料結構與基本操作:
   - 每個多項式由一個鏈結串列 (Linked List) 表示，串列中的每個節點包含一個係數 (coef)、一個指數 (exp)，以及指向下一個節點的指標 (link)。
   - head 是指向鏈結串列頭節點的指標，這個節點不存放有效數據，只是用來表示鏈結串列的起點。
2. 建構子與解構子:
   - Polynomial()：預設建構子，初始化一個空的多項式。
   - ~Polynomial()：解構子，釋放鏈結串列中所有節點的記憶體。
   - Polynomial(const Polynomial& a)：複製建構子，將另一個多項式的內容複製到新的多項式中。
   - const Polynomial& operator=(const Polynomial& a)：賦值運算子，將另一個多項式的內容賦值給當前多項式。
3. 輸入與輸出運算子:
   - istream& operator>>(istream& is, Polynomial& x)：重載輸入運算子，允許從輸入流中讀取多項式。
   - ostream& operator<<(ostream& os, const Polynomial& x)：重載輸出運算子，將多項式輸出到輸出流中。
4. 多項式的算術運算:
   - Polynomial operator+(const Polynomial& b) const：實現兩個多項式的加法。
   - Polynomial operator-(const Polynomial& b) const：實現兩個多項式的減法。
   - Polynomial operator*(const Polynomial& b) const：實現兩個多項式的乘法。
5. 多項式的評估:
   - float Evaluate(float x) const：計算多項式在給定 x 值時的結果。
6. 主程式:
   - 主程式用於輸入兩個多項式，並計算它們的和、差、積，然後在給定的 x 值處評估結果。

[程式實作]

```cpp
#include <iostream>
#include <cmath>
using namespace std;
class Polynomial {
private:
    struct Node {
        int coef;   // 係數
        int exp;    // 指數
        Node* link;  // 指向下一個節點的指標
    };

    Node* head;  // 多項式的頭節點指標
public:
    Polynomial();                    // 預設建構子
    ~Polynomial();                   // 解構子
    Polynomial(const Polynomial& a); // 複製建構子
    const Polynomial& operator=(const Polynomial& a); // 賦值運算子
    friend istream& operator>>(istream& is, Polynomial& x); // 輸入運算子
    friend ostream& operator<<(ostream& os, const Polynomial& x); // 輸出運算子
    Polynomial operator+(const Polynomial& b) const; // 加法運算子
    Polynomial operator-(const Polynomial& b) const; // 減法運算子
    Polynomial operator*(const Polynomial& b) const; // 乘法運算子
    float Evaluate(float x) const; // 計算多項式在指定點的值
};

// 多項式類別的預設建構子
Polynomial::Polynomial() {
    head = new Node;
    head->link = head;
}

// 多項式類別的解構子
Polynomial::~Polynomial() {
    Node* temp;
    Node* current = head->link;

    // 釋放節點內存
    while (current != head) {
        temp = current;
        current = current->link;
        delete temp;
    }
    delete head; // 釋放頭節點內存
}
```

```cpp
// 多項式輸入運算子
istream& operator>>(istream& is, Polynomial& x) {
    int n, coef, exp;
    cout << "輸入有幾項: ";
    is >> n;

    for (int i = 0; i < n; ++i) {
        cout << "第" << i + 1 << "項是多少" << ": ";
        is >> coef >> exp;

        // 將新節點插入多項式中
        Polynomial::Node* newNode = new Polynomial::Node{ coef, exp, nullptr };
        newNode->link = x.head->link;
        x.head->link = newNode;
    }

    return is;
}
```

```cpp
// 多項式輸出運算子
ostream& operator<<(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;
}
```

```cpp
// 多項式複製建構子
Polynomial::Polynomial(const Polynomial& a) {
    head = new Node;
    head->link = head;

    Node* current = a.head->link;

    while (current != a.head) {
        // 將新節點插入多項式中
        Node* newNode = new Node{ current->coef, current->exp, nullptr };
        newNode->link = head->link;
        head->link = newNode;

        current = current->link;
    }
}
```

```cpp
// 多項式賦值運算子
const Polynomial& Polynomial::operator=(const Polynomial& a) {
    if (this != &a) {
        Node* temp;
        Node* current = head->link;

        // 釋放原有節點內存
        while (current != head) {
            temp = current;
            current = current->link;
            delete temp;
        }

        Node* sourceCurrent = a.head->link;

        // 複製新的節點
        while (sourceCurrent != a.head) {
            // 將新節點插入多項式中
            Node* newNode = new Node{ sourceCurrent->coef, sourceCurrent->exp, nullptr };
            newNode->link = head->link;
            head->link = newNode;

            sourceCurrent = sourceCurrent->link;
        }
    }

    return *this;
}
```

```cpp
// 多項式加法運算子
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) {
            // 將新節點插入多項式中
            Node* newNode = new Node{ currentA->coef, currentA->exp, nullptr };
            newNode->link = result.head->link;
            result.head->link = newNode;
            currentA = currentA->link;}
        else if (currentA->exp < currentB->exp) {
            // 將新節點插入多項式中
            Node* newNode = new Node{ currentB->coef, currentB->exp, nullptr };
            newNode->link = result.head->link;
            result.head->link = newNode;
            currentB = currentB->link;}
        else {
            int sumCoef = currentA->coef + currentB->coef;
            if (sumCoef != 0) {
                // 將新節點插入多項式中
                Node* newNode = new Node{ sumCoef, currentA->exp, nullptr };
                newNode->link = result.head->link;
                result.head->link = newNode;}
            currentA = currentA->link;
            currentB = currentB->link;}
    }
```

```
    while (currentA != head) {
        // 將新節點插入多項式中
        Node* newNode = new Node{ currentA->coef, currentA->exp, nullptr };
        newNode->link = result.head->link;
        result.head->link = newNode;
        currentA = currentA->link;
    }

    while (currentB != b.head) {
        // 將新節點插入多項式中
        Node* newNode = new Node{ currentB->coef, currentB->exp, nullptr };
        newNode->link = result.head->link;
        result.head->link = newNode;

        currentB = currentB->link;
    }

    return result;
}
// 多項式減法運算子
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) {
            // 將新節點插入多項式中
            Node* newNode = new Node{ currentA->coef, currentA->exp, nullptr };
            newNode->link = result.head->link;
            result.head->link = newNode;
            currentA = currentA->link;}
        else if (currentA->exp < currentB->exp) {
            // 將新節點插入多項式中
            Node* newNode = new Node{ -currentB->coef, currentB->exp, nullptr };
            newNode->link = result.head->link;
            result.head->link = newNode;
            currentB = currentB->link;}
        else {
            int diffCoef = currentA->coef - currentB->coef;
            if (diffCoef != 0) {
                // 將新節點插入多項式中
                Node* newNode = new Node{ diffCoef, currentA->exp, nullptr };
                newNode->link = result.head->link;
                result.head->link = newNode;}
            currentA = currentA->link;
            currentB = currentB->link;}
    }
```

```
    while (currentA != head) {
        // 將新節點插入多項式中
        Node* newNode = new Node{ currentA->coef, currentA->exp, nullptr };
        newNode->link = result.head->link;
        result.head->link = newNode;
        currentA = currentA->link;}
    while (currentB != b.head) {
        // 將新節點插入多項式中
        Node* newNode = new Node{ -currentB->coef, currentB->exp, nullptr };
        newNode->link = result.head->link;
        result.head->link = newNode;
        currentB = currentB->link;}
    return result;
}
```

```
// 多項式乘法運算子
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;
            // 尋找插入點
            while (currentResult != result.head && currentResult->exp > productExp) {
                prevResult = currentResult;
                currentResult = currentResult->link;
            }
            if (currentResult != result.head && currentResult->exp == productExp) {
                // 指數相同，合併同類項
                currentResult->coef += productCoef;
                if (currentResult->coef == 0) {
                    // 當係數為零時，刪除節點
                    prevResult->link = currentResult->link;
                    delete currentResult;                }}
            else {
                // 插入新節點
                prevResult->link = new Node{ productCoef, productExp, currentResult };
            }
            currentB = currentB->link;}
        currentA = currentA->link;}
    return result;
}
```

```cpp
// 計算多項式在指定點的值
float Polynomial::Evaluate(float x) const {
    float result = 0.0;
    Node* current = head->link;

    while (current != head) {
        // 以指定點計算多項式值
        result += current->coef * pow(x, current->exp);
        current = current->link;
    }
    return result;
}
// 主程式
int main() {
    Polynomial p1, p2;
    cout << "輸入第一項有幾項:\n";
    cin >> p1;
    cout << "輸入第二項有幾項:\n";
    cin >> p2;
    Polynomial sum = p1 + p2;
    Polynomial difference = p1 - p2;
    Polynomial product = p1 * p2;
    cout << "多項式一: " << p1 << endl;
    cout << "多項式二: " << p2 << endl;
    cout << "和: " << sum << endl;
    cout << "差: " << difference <<endl;
    cout << "乘法: " << product <<endl;
    float x;
    cout << "輸入X要帶多少: ";
    cin >> x;
    cout << "加完答案: " << sum.Evaluate(x) <<endl;
    cout << "減完答案: " << difference.Evaluate(x) <<endl;
    cout << "乘完答案: " << product.Evaluate(x) <<endl;
    return 0;
}
```

[程式執行畫面]

```
輸入第一項有幾項:
輸入有幾項: 3
第1項是多少: 1
1
第2項是多少: 2
2
第3項是多少: 3
3
輸入第二項有幾項:
輸入有幾項: 2
第1項是多少: 1
1
第2項是多少: 2
2
多項式一: 3x^3 + 2x^2 + 1x^1
多項式二: 2x^2 + 1x^1
和: 3x^3 + 4x^2 + 2x^1
差: 3x^3
乘法: 1x^2 + 4x^3 + 7x^4 + 6x^5
輸入X要帶多少: 10
加完答案: 3420
減完答案: 3000
乘完答案: 674100
```

[時間複雜度]

- 加、減法 : O(m+n)
- 乘法 : O(m*n)
- 評估 : O(m)

[空間複雜度]

- 加、減法 : O(m+n)
- 乘法 : O(m*n)
- 評估 : O(1)

[心得]

上完暑假這 6 週的課程，對於資料結構與演算法的概念，比起大二第一次修的時候來的好很多，學到了很多不同的資料排序方式，還有程式實作，雖然程式 coding 的部分還是相較不熟悉，但經過上機考以及這三次的程式作業，比起之前的自己，有覺得進步很多。