Homework3

四資工二甲 **41243149** 廖張竹

解題說明

多項式是一個代數表達式，包含係數和指數。為了處理這些多項式，我們使用了鏈結串列來儲存每一項。每一項由 coef (係數) 和 exp (指數) 組成。對於每個多項式，我們創建了一個 Node 結構，並將這些 Node 節點串接起來，形成鏈結串列結構。

基本運算：

- **加法**：將兩個多項式按照指數對齊，對應的項進行係數相加。
- **減法**：透過將第二個多項式的係數取反後進行加法實現。
- **乘法**：對於每一項，將第一個多項式中的每一項與第二個多項式中的每一項相乘，並將結果儲存到新多項式中。
- **評估**：計算多項式在給定 x 值處的值，對每一項計算 coef * x^exp，並累加結果。

程式實作

1. **Polynomial 類別**：這個類別定義了多項式的基本操作。它包含以下成員：
    - **Node 結構**：每一項的表示。
    - **AddTerm 方法**：將新項加入到多項式中。
    - **Clear 方法**：釋放多項式中所有的節點。
    - 各種運算符重載（+, -, *）來實現多項式的加法、減法、乘法。
    - **Evaluate 方法**：計算多項式在某個 x 值的結果。
2. **多項式的輸入與輸出**：
    - 使用 istream 和 ostream 重載運算子來處理多項式的輸入與輸出。
3. **記憶體管理**：
    - 使用動態記憶體分配來創建多項式的每一項，並在需要時釋放記憶體，避免內存洩漏。

測試與驗證

為了驗證程式的正確性，可以進行以下測試：

```
PS D:\資料結構\homework3> ./hw3.exe
Enter the first polynomial:
Enter number of terms: 3
Enter coefficient and exponent: 2 4
Enter coefficient and exponent: 3 1
Enter coefficient and exponent: -1 0
Enter the second polynomial:
Enter number of terms: 2
Enter coefficient and exponent: 3 2
Enter coefficient and exponent: 1 6
First polynomial: 2x^4 + 3x - 1
Second polynomial: x^6 + 3x^2
Sum: x^6 + 2x^4 + 3x^2 + 3x - 1
Difference: x^6 + 2x^4 - 3x^2 + 3x - 1
Product: 2x^10 + 3x^7 + 5x^6 + 9x^3 - 3x^2
Enter a value to evaluate the first polynomial: 3
Value of the first polynomial at 3: 170
PS D:\資料結構\homework3>
```

效能分析

- **時間複雜度**：
  - 加法：遍歷兩個多項式的每一項，時間複雜度為 $O(n + m)$，其中 n 和 m 分別是兩個多項式的項數。
  - 減法：類似加法，時間複雜度為 $O(n + m)$。
  - 乘法：對每一項進行乘法運算，總時間複雜度為 $O(n * m)$。
  - 評估：需要遍歷所有項進行計算，時間複雜度為 $O(n)$。
- **空間複雜度**：
  - 每個多項式的空間複雜度為 $O(n)$，其中 n 是多項式中的項數。

    申論與開發報告
  - 本程式透過鏈結串列實現多項式的表示與基本運算，展現了面向對象程式設計的應用。使用者可以方便地輸入多項式並執行各種運算，程式自動處理記憶體管理，確保運行時不會出現記憶體洩漏問題。透過多項式運算符的重載，使用者不必了解底層實現，僅需關注多項式的表示與結果。
  - 然而，程式的效能在處理大規模多項式時可能受到影響，尤其是在進行乘法運算時，時間複雜度為 $O(n * m)$。未來可以考慮引入

其他數據結構或算法來優化性能，例如使用哈希表來加速查找和插入操作。

- 此外，程式對多項式的項數有一定限制，當多項式項數過多時，記憶體使用和計算時間可能會增加。這需要根據實際情況調整數據結構或進行優化。
- **結論**
- 這段程式碼有效地解決了多項式運算的問題，能夠處理加法、減法、乘法及評估運算。透過正確的測試和效能分析，程序的正確性和效率得到了驗證。未來可進行優化以提高處理大型多項式時的效率。

程式碼:

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

struct Node{
    int coef;  // 係數
    int exp;   // 指數
    Node* link; // 下一節點
};

class Polynomial{
    private:
        Node* header; // link 的頭
        void AddTerm(int coef, int exp); // 新增多項式
        void Clear(); // 清空多項式
    public:
        Node* getHeader() const { return header; }
        Polynomial(); // 建構子
        Polynomial(const Polynomial& a); // copy 建構子
        ~Polynomial(); // 解構子
        Polynomial& operator=(const Polynomial& a); // 值的運算
        Polynomial operator+(const Polynomial& b) const; // 加法
        Polynomial operator-(const Polynomial& b) const; // 減法
        Polynomial operator*(const Polynomial& b) const; // 乘法
        float Evaluate(float x) const; // 多項式的值

        friend istream& operator>>(istream& is, Polynomial& x); // 輸入
        friend ostream& operator<<(ostream& os, const Polynomial& x); // 輸出
};

// 預設建構子，初始化多項式，建立一個表頭節點，並使其指向自己 (形成循環)
Polynomial::Polynomial() : header(new Node{0, 0, nullptr}) {
    header->link = header;
}

// 複製建構子，利用賦值運算子完成複製
Polynomial::Polynomial(const Polynomial& a) : Polynomial() {
    *this = a;
}
```

```cpp
// 解構子，釋放記憶體，清空多項式並刪除表頭節點
Polynomial::~Polynomial() {
    Clear();
    delete header;
}

// 賦值運算子，將多項式 a 賦值給當前多項式
Polynomial& Polynomial::operator=(const Polynomial& a) {
    if (this != &a) {  // 避免自我賦值
        Clear(); // 先清除當前多項式
        for (Node* p = a.header->link; p != a.header; p = p->link) {
            AddTerm(p->coef, p->exp); // 複製 a 的每一項到當前多項式
        }
    }
    return *this;
}
```

```cpp
// 多項式加法運算，返回當前多項式與 b 的和
Polynomial Polynomial::operator+(const Polynomial& b) const {
    Polynomial result; // 存放結果的多項式
    Node* p1 = header->link;
    Node* p2 = b.header->link;

    // 遍歷兩個多項式，按照指數大小逐項相加
    while (p1 != header || p2 != b.header) {
        if (p1 == header || (p2 != b.header && p2->exp > p1->exp)) {
            result.AddTerm(p2->coef, p2->exp); // 加入 p2 的項
            p2 = p2->link;
        } else if (p2 == b.header || p1->exp > p2->exp) {
            result.AddTerm(p1->coef, p1->exp); // 加入 p1 的項
            p1 = p1->link;
        } else {
            int newCoef = p1->coef + p2->coef; // 指數相同，係數相加
            if (newCoef) result.AddTerm(newCoef, p1->exp);
            p1 = p1->link;
            p2 = p2->link;
        }
    }
    return result;
}
```

```cpp
// 多項式減法運算，返回當前多項式與 b 的差
Polynomial Polynomial::operator-(const Polynomial& b) const {
    Polynomial neg_b(b); // 複製 b
    for (Node* p = neg_b.header->link; p != neg_b.header; p = p->link) {
        p->coef = -p->coef; // 將 b 的每一項變號
    }
    return *this + neg_b; // 呼叫加法運算子
}

// 多項式乘法運算，返回當前多項式與 b 的乘積
Polynomial Polynomial::operator*(const Polynomial& b) const {
    Polynomial result;
    for (Node* p1 = header->link; p1 != header; p1 = p1->link) {
        for (Node* p2 = b.header->link; p2 != b.header; p2 = p2->link) {
            result.AddTerm(p1->coef * p2->coef, p1->exp + p2->exp);
        }
    }
    return result;
}

// 計算多項式在 x 處的值
float Polynomial::Evaluate(float x) const {
    float result = 0;
    for (Node* p = header->link; p != header; p = p->link) {
        result += p->coef * pow(x, p->exp); // 逐項累加
    }
    return result;
}
```

```cpp
// 新增項目到多項式中
void Polynomial::AddTerm(int coef, int exp) {
    if (!coef) return;
    Node* p = header;
    while (p->link != header && p->link->exp > exp) p = p->link;
    if (p->link != header && p->link->exp == exp) {
        p->link->coef += coef;
        if (!p->link->coef) { // 如果係數變為 0，刪除此項
            Node* temp = p->link;
            p->link = temp->link;
            delete temp;
        }
    } else {
        p->link = new Node{coef, exp, p->link}; // 插入新項
    }
}

// 清空多項式，釋放所有項目
void Polynomial::Clear() {
    while (header->link != header) {
        Node* temp = header->link;
        header->link = temp->link;
        delete temp;
    }
}

// 多項式輸入
istream& operator>>(istream& is, Polynomial& x) {
    x.Clear();
    int n, coef, exp;
    cout << "Enter number of terms: ";
    is >> n;
    while (n--) {
        cout << "Enter coefficient and exponent: ";
        is >> coef >> exp;
        x.AddTerm(coef, exp);
    }
    return is;
}
```

```cpp
// 多項式輸出
ostream& operator<<(ostream& os, const Polynomial& x) { // 修改此處，const 引用
    bool first = true;
    for (Node* p = x.getHeader()->link; p != x.getHeader(); p = p->link) {
        if (!first) os << (p->coef > 0 ? " + " : " - ");
        if (abs(p->coef) != 1 || p->exp == 0) os << abs(p->coef);
        if (p->exp) os << "x" << (p->exp != 1 ? "^" + std::to_string(p->exp) : "");
        first = false;
    }
    if (first) os << "0";
    return os;
}

int main() {
    Polynomial p1, p2; // 宣告兩個 Polynomial 物件 p1 和 p2，表示兩個多項式
    cout << "Enter the first polynomial:" << endl;
    cin >> p1; // 輸入第一個多項式

    cout << "Enter the second polynomial:" << endl;
    cin >> p2; // 輸入第二個多項式

    // 輸出兩個多項式
    cout << "First polynomial: " << p1 << endl;
    cout << "Second polynomial: " << p2 << endl;

    // 計算並輸出兩個多項式的和、差和積
    cout << "Sum: " << (p1 + p2) << endl;
    cout << "Difference: " << (p1 - p2) << endl;
    cout << "Product: " << (p1 * p2) << endl;

    // 評估第一個多項式在特定 x 值下的結果
    float x;
    cout << "Enter a value to evaluate the first polynomial: ";
    cin >> x;
    cout << "Value of the first polynomial at " << x << ": " << p1.Evaluate(x) << endl;

    return 0; // 程式結束
}
```