HW 4 鏈結串列多項式

1. 解題說明

1. 定義節點結構體（Node）

   coef：係數

   exp：指數

   link：指向下一個節點的指標

2. 建立多項式類別（Polynomial）

   插入節點：根據指數的大小順序插入新的節點，以保證多項式按指數降序排列。

   輸入/輸出：通過重載輸入和輸出運算子來實現多項式的讀取和顯示。

3. 基本操作函數

   加法 減法 乘法

4. 程式執行流程

   輸入：使用者輸入兩個多項式，程式將其轉換為鏈結串列表示。

   運算：使用者選擇加法、減法、乘法或求值操作，程式進行相應的運算並輸出結果。

   輸出：將結果以標準數學形式顯示在控制台上。

2. 建立 node

```
struct Node {
    int coef; // 系数
    int exp;  // 指数
    Node* link; // 指向下一个节点的指针
};

class Polynomial {
private:
    Node* head;  // 指向头节点的指针

    // 插入节点的私有方法
    void InsertNode(int coef, int exp);

    // 用于管理空闲空间链表的私有方法
    static Node* avail;
    static Node* GetNode();
    static void RetNode(Node* node);

public:
    Polynomial();  // 构造函数
    Polynomial(const Polynomial& a);  // 拷贝构造函数
    ~Polynomial(); // 析构函数

    Polynomial& operator=(const Polynomial& a);  // 赋值运算符
    Polynomial operator+(const Polynomial& b) const;  // 加法运算符
    Polynomial operator-(const Polynomial& b) const;  // 減法运算符
    Polynomial operator*(const Polynomial& b) const;  // 乘法运算符
    float Evaluate(float x) const;  // 评估多项式值

    friend std::istream& operator>>(std::istream& is, Polynomial& x);  // 輸入运算符重载
    friend std::ostream& operator<<(std::ostream& os, const Polynomial& x);  // 輸出运算符重载
};
```

插入節點

```
// 插入节点的方法
void Polynomial::InsertNode(int coef, int exp) {
    Node* newNode = GetNode();
    newNode->coef = coef;
    newNode->exp = exp;

    Node* prev = head;
    Node* current = head->link;

    while (current != head && current->exp > exp) {
        prev = current;
        current = current->link;
    }

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

加減乘

```
// 加法运算符重载
Polynomial Polynomial::operator+(const Polynomial& b) const {
    Polynomial result;
    Node* aTerm = head->link;
    Node* bTerm = b.head->link;

    while (aTerm != head || bTerm != b.head) {
        if (aTerm != head && (bTerm == b.head || aTerm->exp > bTerm->exp)) {
            result.InsertNode(aTerm->coef, aTerm->exp);
            aTerm = aTerm->link;
        }
        else if (bTerm != b.head && (aTerm == head || bTerm->exp > aTerm->exp)) {
            result.InsertNode(bTerm->coef, bTerm->exp);
            bTerm = bTerm->link;
        }
        else {
            int newCoef = aTerm->coef + bTerm->coef;
            if (newCoef != 0)
                result.InsertNode(newCoef, aTerm->exp);
            aTerm = aTerm->link;
            bTerm = bTerm->link;
        }
    }
    return result;
}
```

```cpp
// 減法運算符重載
Polynomial Polynomial::operator-(const Polynomial& b) const {
    Polynomial result;
    Node* aTerm = head->link;
    Node* bTerm = b.head->link;

    while (aTerm != head || bTerm != b.head) {
        if (aTerm != head && (bTerm == b.head || aTerm->exp > bTerm->exp)) {
            result.InsertNode(aTerm->coef, aTerm->exp);
            aTerm = aTerm->link;
        }
        else if (bTerm != b.head && (aTerm == head || bTerm->exp > aTerm->exp)) {
            result.InsertNode(-bTerm->coef, bTerm->exp);
            bTerm = bTerm->link;
        }
        else {
            int newCoef = aTerm->coef - bTerm->coef;
            if (newCoef != 0)
                result.InsertNode(newCoef, aTerm->exp);
            aTerm = aTerm->link;
            bTerm = bTerm->link;
        }
    }
    return result;
}
```

```cpp
// 乘法運算符重載
Polynomial Polynomial::operator*(const Polynomial& b) const {
    Polynomial result;

    for (Node* aTerm = head->link; aTerm != head; aTerm = aTerm->link) {
        Polynomial temp;
        for (Node* bTerm = b.head->link; bTerm != b.head; bTerm = bTerm->link) {
            temp.InsertNode(aTerm->coef * bTerm->coef, aTerm->exp + bTerm->exp);
        }
        result = result + temp;
    }
    return result;
}
```

多項式評估

```cpp
// 评估多项式值
float Polynomial::Evaluate(float x) const {
    float result = 0.0;
    Node* curr = head->link;
    while (curr != head) {
        result += curr->coef * pow(x, curr->exp);
        curr = curr->link;
    }
    return result;
}
```

主程式

```cpp
int main() {
    Polynomial p1, p2, p3;
    std::cout << "輸入第一個多項式: ";
    std::cin >> p1;
    std::cout << "輸入第二個多項式: ";
    std::cin >> p2;

    p3 = p1 + p2;
    std::cout << "兩個多項式的和: " << p3 << std::endl;

    p3 = p1 - p2;
    std::cout << "兩個多項式的差: " << p3 << std::endl;

    p3 = p1 * p2;
    std::cout << "兩個多項式的積: " << p3 << std::endl;

    float x;
    std::cout << "輸入值來評估第一個多項式 ";
    std::cin >> x;
    std::cout << "評估結果: " << p1.Evaluate(x) << std::endl;

    return 0;
}
```

3. 效能分析

空間複雜度:

節點的空間：每個節點包含一個整數的係數(coef)、一個整數的指數(exp)，以及一個指標(link)。

整數的空間複雜度為 O(1)。

指標的空間複雜度為 O(1)。

因此，每個節點的空間複雜度為 O(1)。

鏈結串列的空間：

對於 n 個項的多項式，共需 n 個節點。

所以鏈結串列的總空間複雜度為 O(n)。

時間複雜度:

插入節點：O(n)

加法/減法：O(m + n)

乘法：O(m * n)

評估：O(n)

輸入/輸出：O(n)

4. 測試與過程

```
输入第一个多项式: 3 4 3 2 2 1 1
输入第二个多项式: 3 1 3 4 2 5 0

两个多项式的和: 4x^3 + 6x^2 + 5 + x^4

两个多项式的差: 3x^4 + 1x^3 - 2x^2 - 5

两个多项式的积: 4x^6 + 10x^5 + 14x^4 + 11x^3 + 5x^2

输入一个值来评估第一个多项式: 2
评估结果: 44
```