資料結構 第三份 作業

姓名:林國善

8/22/2024

1 題目

Homework 3

[***Programming Project***] Develop a C++ class *Polynomial* to represent and manipulate univariate polynomials with integer coefficients (use circular linked lists with header nodes). Each term of the polynomial will be represented as a node. Thus, a node in this system will have three data members as below:

coef	exp	link

Each polynomial is to be represented as a circular list with header node. To delete polynomials efficiently, we need to use an available-space list and associated functions as described in Section 4.5. The external (i.e., for input or output) representation of a univariate polynomial will be assumed to be a sequence of integers of the form: $n, c_1, e_1, c_2, e_2, c_3, e_3, \ldots, c_n, e_n$, where e_i represents an exponent and c_i a coefficient; n gives the number of terms in the polynomial. The exponents are in decreasing order—$e_1 > e_2 > \cdots > e_n$.

Write and test the following functions:

(a) *istream*& **operator>>**(*istream*& *is*, *Polynomial*& *x*): Read in an input polynomial and convert it to its circular list representation using a header node.

(b) *ostream*& **operator<<**(*ostream*& *os*, *Polynomial*& *x*): Convert *x* from its linked list representation to its external representation and output it.

(c) *Polynomial*::*Polynomial*(**const** *Polynomial*& *a*) [Copy Constructor]: Initialize the polynomial ***this** to the polynomial *a*.

(d) **const** *Polynomial*& *Polynomial*::**operator=**(**const** *Polynomial*& *a*) **const** [Assignment Operator]: Assign polynomial *a* to ***this**.

(e) *Polynomial*::~*Polynomial*() [Destructor]: Return all nodes of the polynomial ***this** to the available-space list.

(f) *Polynomial* **operator+** (**const** *Polynomial*& *b*) **const** [Addition]: Create and return the polynomial ***this** + *b*.

(g) *Polynomial* **operator**– (**const** *Polynomial*& *b*) **const** [Subtraction] : Create and return the polynomial ***this** – *b*.

(h) *Polynomial* **operator***(**const** *Polynomial*& *b*) **const** [Multiplication]: Create and return the polynomial ***this** * *b*.

(i) **float** *Polynomial*::*Evaluate*(**float** *x*) **const**: Evaluate the polynomial ***this** at *x* and return the result.

2 演算法設計與實作

想法

這段程式碼實現了一個多項式類別 Polynomial，並支援多項式的加法、減法、乘法，以及在某一點的值的計算。每個多項式的項次（如 4x^3 中的 4x^3）由一個結構體 Term 表示，並使用環狀鏈結串列來儲存多項式的所有項。程式碼使用了運算子重載，使多項式的操作更加直觀和易於理解。

環狀鏈結串列的基本概念

環狀鏈結串列是一種特殊的鏈結串列結構，其中最後一個節點的 next 指標指向鏈結串列的頭節點，這樣整個串列形成一個閉合的環。相較於普通的單向鏈結串列，環狀鏈結串列具有一些特點，例如可以從串列的任意位置開始遍歷，並能夠快速返回頭節點。所以透過這裡定義了 Term 新增一個項到環狀鏈結串列中

newTerm 函數負責將新的項加入到多項式中，。

主要步驟與邏輯

1. **檢查系數**: 如果係數為零，不會加入新項，直接返回。

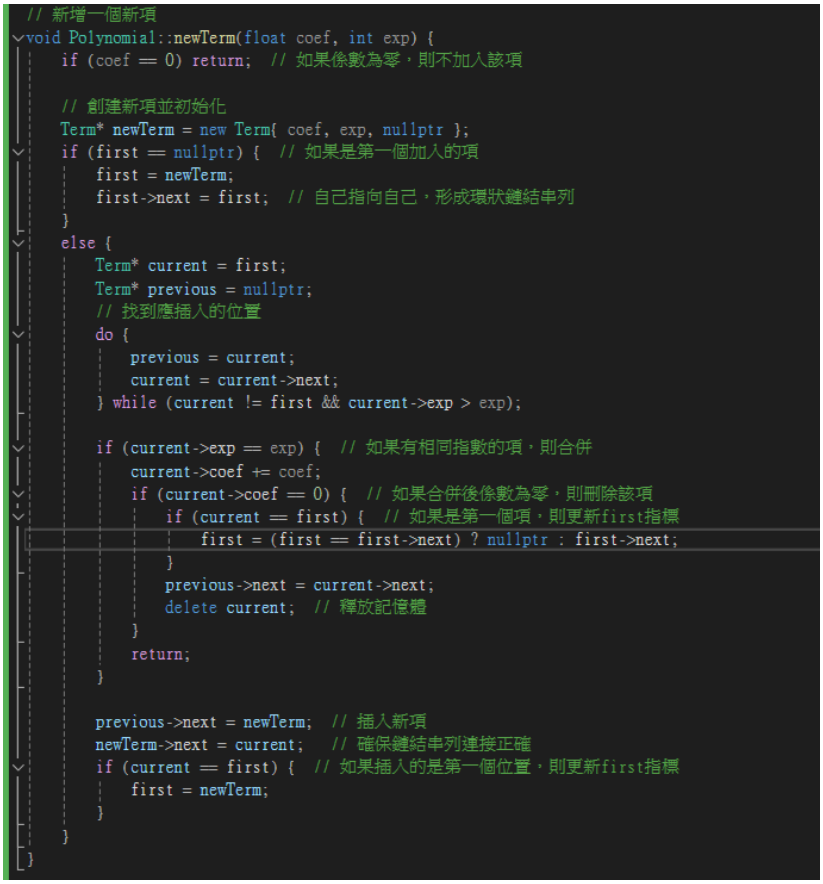
2. **創建新項**: 使用輸入的 coef 和 exp 創建新的 Term 結構

體。

3. **檢查是否為第一項**：如果這是多項式中的第一個項，則 first 指向該項，並且該項的 next 指向自己，形成環狀。

插入新項：

- 從 first 開始遍歷，根據指數從大到小的順序找到插入點。
- 如果找到相同指數的項，則合併這兩個項（即相加系數）。
- 如果合併後的系數為零，則刪除該項，並調整指標。
- 否則，將新項插入到適當的位置，並確保環狀結構的完整性。



```
// 新增一個新項
void Polynomial::newTerm(float coef, int exp) {
    if (coef == 0) return;  // 如果係數為零，則不加入該項

    // 創建新項並初始化
    Term* newTerm = new Term{ coef, exp, nullptr };
    if (first == nullptr) {  // 如果是第一個加入的項
        first = newTerm;
        first->next = first;  // 自己指向自己，形成環狀鏈結串列
    }
    else {
        Term* current = first;
        Term* previous = nullptr;
        // 找到應插入的位置
        do {
            previous = current;
            current = current->next;
        } while (current != first && current->exp > exp);

        if (current->exp == exp) {  // 如果有相同指數的項，則合併
            current->coef += coef;
            if (current->coef == 0) {  // 如果合併後係數為零，則刪除該項
                if (current == first) {  // 如果是第一個項，則更新first指標
                    first = (first == first->next) ? nullptr : first->next;
                }
                previous->next = current->next;
                delete current;  // 釋放記憶體
            }
            return;
        }

        previous->next = newTerm;  // 插入新項
        newTerm->next = current;   // 確保鏈結串列連接正確
        if (current == first) {  // 如果插入的是第一個位置，則更新first指標
            first = newTerm;
        }
    }
}
```

3 效能分析

加法與減法的時間複雜度：O(n + m)，其中 n 和 m 分別是兩個多項式的項數。這是因為加法與減法需要遍歷兩個多項式的每一項。

乘法的時間複雜度：O(n * m)，其中 n 和 m 分別是兩個多項式的項數。這是因為每一個項都需要與另一個多項式的所有項相乘。

空間複雜度：O(n)，用來儲存多項式的環狀鏈結串列所需的空間。

4 測試與過程

```
P1(x) = 1 + 2x + 3x^2 + 4x^3
P2(x) = 5 - 1x - 2x^2
P1(x) + P2(x) = 2x + 4x^3 + 1x^2 - 1x + 6
P1(x) - P2(x) = 2x + 4x^3 + 5x^2 + 1x - 4
P1(x) * P2(x) = - 1x - 8x^5 - 10x^4 + 13x^3 + 11x^2 + 10x + 5
P1 在 x = 2 處的值: 49
P2 在 x = 2 處的值: -5
和在 x = 2 處的值: 44
差在 x = 2 處的值: 54
積在 x = 2 處的值: -245
```

5 心得討論

運算子重載使得程式碼更具可讀性，同時也展示了環狀鏈結串列在數學結構中的應用。也透過 TA 的程式改變從中也學習到了許多的技巧和要如何操作。