資料結構 HW3

資工二甲 謝於潔

Jan 7,2025

目錄

解題說明

依題目要求設計一個 Polynomial 類別來表示和操作單變量多項式，

並且每個多項式存儲為帶有表頭節點的循環鏈結串列

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.

驗算法設計與實作

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

//節點類別，表示多項式的單項
class Node
{
public:
    float coef;  //係數
    int exp;  //指數
    Node* link;  //指向下一個節點的連結

    //初始化係數、指數與連結
    Node(float c = 0, int e = 0, Node* l = nullptr)
        : coef(c), exp(e), link(l) {}
};

//多項式類別
class Polynomial
{
private:
    Node* head;  //頭節點，作為循環鏈結串列的起點

    //將新的節點加到鏈結最後面
    void Attach(float coef, int exp, Node*& tail)
    {
        if (abs(coef) < 1e-6) return;  //係數接近0，則忽略該項
        Node* newNode = new Node(coef, exp);
        tail->link = newNode;  //將最後節點的鏈結指向新節點
        tail = newNode;  //更新最後節點為新節點
    }

    //清除輸入緩衝區
    void clearInputBuffer()
    {
        cin.clear();  //清除輸入的錯誤狀態
        cin.ignore(numeric_limits<streamsize>::max(), '\n');  //忽略緩衝區中的內容直到換行符
    }

public:
    //建構子
    Polynomial() {
        head = new Node();
        head->link = head;
    }

    // 解構函子
    ~Polynomial() {
        Node* curr = head->link;
        while (curr != head)
        {
            Node* temp = curr;   //儲存當前節點
            curr = curr->link;
            delete temp;  //刪除節點內容
        }
        delete head;
    }
```

```
// 複製建構函數
Polynomial(const Polynomial& other)
{
    head = new Node();
    head->link = head;
    Node* tail = head;
    Node* curr = other.head->link;
    while (curr != other.head)
    {
        Attach(curr->coef, curr->exp, tail);  //將當前節點複製並加到新多項式
        curr = curr->link;
    }
    tail->link = head;  //關閉循環鏈結串列
}

//運算子
Polynomial& operator=(const Polynomial& other) {
    if (this != &other)  //防止自我賦值
    {
        Node* curr = head->link;
        while (curr != head)
        {
            Node* temp = curr;
            curr = curr->link;
            delete temp;
        }

        // 複製另一個多項式的數據
        Node* tail = head;  //最後節點為最前節點
        curr = other.head->link;  //從另一個多項式的第一個節點開始
        while (curr != other.head)
        {
            Attach(curr->coef, curr->exp, tail);  //將當前節點複製加入到新多項式
            curr = curr->link;
        }
        tail->link = head;  //關閉循環環鏈結串列
    }
    return *this;  //返回當前引用
}
```

```
// 輸入運算子
friend istream& operator>>(istream& is, Polynomial& x)
{
    int n;
    do
    {
        cout << "輸入多項式的項數（正整數）：";
        if (!(is >> n) || n <= 0)  //檢查輸入是不是有效的正整數
        {
            cout << "請輸入有效的正整數！" << endl;
            x.clearInputBuffer();  //清除輸入緩衝區
            continue;
        }
        break;  //輸入有效就退出循環
    } while (true);

    Node* tail = x.head;  //最後節點為最前節點
    int lastExp = INT_MAX;  //用於檢查指數是不是較小，初始設為最大值

    for (int i = 0; i < n; ++i)
    {
        float coef;  //儲存係數
        int exp;  //儲存指數
        bool validInput = false;  //檢查輸入是否有效

        do
        {
            cout << "輸入第" << (i + 1) << "項的係數與指數（例如 3 2 表示 3x^2）：";
            if (!(is >> coef >> exp))  //檢查輸入是不是有效的數值
            {
                cout << "輸入無效，請重試！" << endl;
                x.clearInputBuffer();
                continue;
            }

            if (exp >= lastExp)  //檢查指數是否比前面的還小
            {
                cout << "指數必須小於前一項！前一項指數為：" << lastExp << endl;
                continue;
            }

            validInput = true; //輸入有效就退出循環
            lastExp = exp;  //更新最後的指數
        } while (!validInput);

        if (abs(coef) >= 1e-6)  //忽略係數為0的項
        {
            x.Attach(coef, exp, tail);  //將有效項加到多項式
        }
    }
    tail->link = x.head;  //關閉合循環鏈結串列
    return is;
}
```

```
// 輸出運算子，將多項式輸出到輸出流
fricnd ostrcam& opcrator<<(ostrcam& os, const Polynomial& x)
{
    Nodc* curr = x.hcad->link;  //從第一個有效節點開始
    if (curr == x.hcad)  //如果多項式為空
    {
        os << "0";
        rcturn os;
    }

    bool first = truc;  //用於標記是否為第一項
    whilc (curr != x.hcad)
    {
        if (curr->cocf == 0)  //忽略係數為0的項
        {
            curr = curr->link;
            continuc;  //跳過本次迭代
        }

        if (!first && curr->cocf > 0) os << "+";  //如果非第一項且係數為正，則輸出+
        first = falsc;  //設置標記為非第一項

        if (curr->cocf != 1 || curr->cxp == 0)  //係數不為1或指數為0
        {
            if (curr->cocf == -1 && curr->cxp != 0)  //係數為-1且指數不為0
            {
                os << "-";
            }
            clsc {
                os << curr->cocf;
            }
        }

        if (curr->cxp > 0)  //如果指數大於0
        {
            os << "x";
            if (curr->cxp > 1) os << "^" << curr->cxp;  //如果指數大於1則輸出指數
        }

        curr = curr->link;
    }
    rcturn os;
}
```

```
    // 加法運算子
    Polynomial operator+(const Polynomial& b) const
    {
        Polynomial result;  //用於存儲結果的多項式
        Node* aPtr = head->link;
        Node* bPtr = b.head->link;
        Node* tail = result.head;

        while (aPtr != head && bPtr != b.head)  //同時跑兩個
        {
            if (aPtr->exp == bPtr->exp)  //如果指數相等
            {
                float sum = aPtr->coef + bPtr->coef;  //計算係數和
                if (abs(sum) >= 1e-6)  //若和不為0
                {
                    result.Attach(sum, aPtr->exp, tail);  //將和附加到結果多項式
                }
                aPtr = aPtr->link;
                bPtr = bPtr->link;
            }
            else if (aPtr->exp > bPtr->exp)  //如果a多項式指數較大
            {
                result.Attach(aPtr->coef, aPtr->exp, tail); // 加入a多項式的當前項
                aPtr = aPtr->link;
            }
            else  //如果b多項式指數較大
            {
                result.Attach(bPtr->coef, bPtr->exp, tail); // 加入b多項式的當前項
                bPtr = bPtr->link;
            }
        }

        while (aPtr != head)  //若a還有剩餘項
        {
            result.Attach(aPtr->coef, aPtr->exp, tail);  //加到結果多項式
            aPtr = aPtr->link;
        }

        while (bPtr != b.head)  //若b還有剩餘項
        {
            result.Attach(bPtr->coef, bPtr->exp, tail);  //加到結果多項式
            bPtr = bPtr->link;
        }

        tail->link = result.head;  //關閉結果多項式鏈結串列
        return result;
    }

    // 計算多項式的值，給定x
    float Evaluate(float x) const {
        float result = 0;
        Node* curr = head->link;
        while (curr != head)
        {
            result += curr->coef * pow(x, curr->exp);  //累加每個數值
            curr = curr->link;
        }
        return result;
    }
};
```

```cpp
// 主函數，測試多項式類別的功能
int main() {
    try {
        Polynomial p1, p2;
        cout << "輸入A多項式" << endl;
        cin >> p1;
        cout << "輸入B多項式" << endl;
        cin >> p2;

        cout << "\nA多項式為: " << p1 << endl;
        cout << "B多項式為: " << p2 << endl;

        Polynomial sum = p1 + p2;
        cout << "加法結果: " << sum << endl;

        float x; // 用於存儲用戶輸入的x值
        do {
            cout << "\n輸入X值計算A多項式結果: ";
            if (!(cin >> x))
            {
                cout << "請輸入有效的數值！" << endl;
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');  //忽略緩衝區中的內容直到換行符號
                continue;
            }
            break;  //若輸入有效就退出循環
        } while (true);

        cout << "A(" << x << ") = " << p1.Evaluate(x) << endl;
    }
    catch (const exception& e)
    {
        cerr << "發生錯誤：" << e.what() << endl;
        return 1;
    }

    return 0;
}
```

效能分析

	時間複雜度	空間複雜度
多項式輸入及建立	O(n)	O(n)
多項式加法	O(n+m)	O(n+m)
多項式計算	O(n)	O(1)
多項式輸出	O(n)	O(1)

測試與過程

```
輸入A多項式
輸入多項式的項數（正整數）：2
輸入第1項的係數與指數 (例如 3 2 表示 3x^2)：2 3
輸入第2項的係數與指數 (例如 3 2 表示 3x^2)：1 2
輸入B多項式
輸入多項式的項數（正整數）：2
輸入第1項的係數與指數 (例如 3 2 表示 3x^2)：2 2
輸入第2項的係數與指數 (例如 3 2 表示 3x^2)：1 1

A多項式為： 2x^3+x^2
B多項式為： 2x^2+x
加法結果： 2x^3+3x^2+x

輸入X值計算A多項式結果： 1
A(1) = 3
```

驗證

第一個方程式為$2x^3+x^2$ ，第二個方程式為$2x^2+x$相加為$(2+0)x^3+(1+2)x^2+(0+1)x=2x^3+3x^2+x$，輸入一個數字計算第一個多項式的值

$2\times(1^3)+1\times(1^2)=2+1=3$

問題申論及開發

I. 功能實現

1. 基本功能：

支援使用者輸入多項式，係數與指數按降序排序。

支援多項式的加法運算，並自動刪除係數為 0 的項。

可計算多項式在指定 X 值下的數值結果

2. 資料結構：

採用單向循環鏈結串列儲存多項式，實現動態管理項數。

II. 現有問題

1. 加入多項式減法與乘法運算：

減法：與加法類似，只需將第二個多項式的係數取負後相加。

乘法：需要考慮展開每一項的運算，並對結果按指數合併整理。

2. 除法與餘數運算：

除法：需要實現多項式的長除法演算法。

餘數：保留除法過程中未被整除的部分。

III. 測試與可靠性

1. 測試覆蓋率提升

編寫自動化測試程式，覆蓋常見邊界條件（如空多項式、單項多項式、極大值指數）。

2. 性能壓力測試

測試多項式在大項數（如數百或數千項）時的效能。

心得

這個練習在類別的部分我其實不太熟悉，改了好幾次跟查了之前的上課的講義作業還有網路上很多資料來輔助我來做這份作業，做完之後感覺又複習了一次關於類別的相關知識，並且藉由這次的實作可以對程式碼更加的熟練，並且對語法更加的熟悉，對之後我的程式能力又更加進步，也希望這些能力可以應用在未來的工作上，讓我在為來的工作如魚得水更加容易。