資料結構作業2
Polynomial類別實作

41243111 王家偉

Nov. 26, 2024

1.解題說明

```
class Polynomial {
// p(x) = a_0 x^{e_0} + ··· + a_n x^{e_n}; a set of ordered pairs of <e_i, a_i>,
// where a_i is a nonzero float coefficient and e_i is a non-negative integer exponent.
public:
    Polynomial();
    // Construct the polynomial p(x) = 0.

    Polynomial Add(Polynomial poly);
    // Return the sum of the polynomials *this and poly.

    Polynomial Mult(Polynomial poly);
    // Return the product of the polynomials *this and poly.

    float Eval(float f);
    // Evaluate the polynomial *this at f and return the result.
};
```

Figure 1. Abstract data type of *Polynomial* class

```
class Polynomial ; // forward declaration

class Term {
friend Polynomial;
private:
    float coef;   // coefficient
    int exp;      // exponent
};
```

The private data members of *Polynomial* are defined as follows:

```
private:
    Term *termArray;   // array of nonzero terms
    int capacity;      // size of termArray
    int terms;         // number of nonzero terms
```

Figure 2. The private data members of *Polynomial* class

1.實現多項式類（Polynomial class）及其抽象資料類型（ADT），並根據圖1和圖2顯示的私有資料成員來實現。

2.編寫C++函數來輸入和輸出以圖2表示的多項式。您的函數應該重載 << 和 >> 運算符。

詳細實作參見檔案
Polynomial.cpp

2.程式實作

1. 多項式類別實作

圖2中要求的資料成員實作:

```
// ========== 多項式基本單位
class Term
{
friend Polynomial;
friend ostream& operator<<(ostream& output, const Polynomial& poly);
private:
    float coef; // 係數
    int exp;    // 次數
};
```

```
// ========== 多項式類別
class Polynomial
{
friend ostream& operator<<(ostream& output, const Polynomial& poly);
private:
    Term *termArray; // 存放多項式的陣列
    int capacity;    // 陣列配置大小
    int terms;       // 非零項數
```

圖1中要求的函式成員實作:

```
Polynomial()
{
    termArray = new Term[1];
    capacity = 1;
    terms = 0;

    termArray->coef=0;
    termArray->exp=0;
}
```

```
// ---------- 多項式相加
Polynomial Add(Polynomial poly)
{
    Soft();
    poly.Soft();

    Polynomial result;
    int i = 0, j = 0;

    while (i < this->terms || j < poly.terms)
    {
        if (j >= poly.terms || this->termArray[i].exp > poly.termArray[j].exp)
        {
            result.newTerm(this->termArray[i].coef, this->termArray[i].exp);
            i++;
        }
        else if (i >= poly.terms || this->termArray[i].exp < poly.termArray[j].exp)
        {
            result.newTerm(poly.termArray[j].coef, poly.termArray[j].exp);
            j++;
        }
        else
        {
            float sumCoef = this->termArray[i].coef + poly.termArray[j].coef;
            if (sumCoef != 0) result.newTerm(sumCoef, this->termArray[i].exp);
            i++; j++;
        }
    }

    return result;
}
```

```
// ---------- 多項式相乘
Polynomial Mult(Polynomial poly)
{
    Soft();
    poly.Soft();

    Polynomial result;

    for (int i = 0; i < this->terms; ++i)
    {
        for (int j = 0; j < poly.terms; ++j)
        {
            float newCoef = this->termArray[i].coef * poly.termArray[j].coef;
            int newExp = this->termArray[i].exp + poly.termArray[j].exp;

            bool contain = false;
            for (int k = 0; k < result.terms; ++k)
            {
                if (result.termArray[k].exp == newExp)
                {
                    result.termArray[k].coef += newCoef;
                    contain = true;
                    break;
                }
            }
            if (!contain) result.newTerm(newCoef, newExp);
        }
    }

return result;
}
```

```
// ---------- 計算 x 代入某數的結果
float Eval(float x)
{
    float result = 0;
    for (int i = 0; i < this->terms; ++i)
        result += termArray[i].coef * pow(x, termArray[i].exp);
    return result;
}
```

2.<< 和 >>的重載

```
// ========== 重載 >>Polynomial
istream& operator>>(istream& input, Polynomial& poly)
{
    int theCoef, theExp;
    cout << "兩數 m 和 n 關係為 mx^n, 兩者都為0時停止讀取: " << endl << "請輸入 m 和 n : ";
    while(input>>theCoef>>theExp)
    {
        if(theCoef==0 && theExp==0) break;
        poly.newTerm(theCoef,theExp);
        cout << "請輸入 m 和 n : ";
    }
    return input;
}

// ========== 重載 <<Polynomial
ostream& operator<<(ostream& output, const Polynomial& poly)
{
    for (int i = 0; i < poly.terms; ++i)
    {
        if (i > 0 && poly.termArray[i].coef > 0) output << " + ";
        output << poly.termArray[i].coef << "x^" << poly.termArray[i].exp;
    }
    return output;
}
```

```
// ========== 主程式
int main()
{
    Polynomial a,b;

    cout << "請輸入最簡多項式a:" << endl;
    cin >> a;
    cout << "請輸入最簡多項式b:" << endl;
    cin >> b;

    a.Soft();
    b.Soft();
    cout << "多項式a: " << a << endl << "多項式b: " << b << endl;

    cout << "兩式相加: " << a.Add(b) << endl;
    cout << "兩式相乘: " << a.Mult(b) << endl;

    float x;
    cout << "請輸入一數將代入兩式相乘結果計算: ";
    cin >> x;
    cout << "代入 x = " << x << " 計算結果為: " << a.Mult(b).Eval(x) << endl;

    return 0;
}
```

3.效能分析

1.Polynomial::newTerm函式：

n 為 terms
T(n)：O(n)
S(n)：O(n)

2.Polynomial::Soft函式：

n 為 terms
T(n)：$O(n^2)$
S(n)：O(1)

3.Polynomial::Add函式：

n_1 和 n_2 分別為兩個多項式的項數
T(n)：$O(n_1^2 + n_2^2)$
S(n)：$O(n_1 + n_2)$

4.Polynomial::Mult函式：

n_1 和 n_2 分別為兩個多項式的項數
T(n)：$O(n_1 * n_2 + r^2)$，r 為結果多項式的項數
S(n)：$O(n_1 * n_2)$

5.Polynomial::Eval函式：

n_1 為 terms，n_2 為 exp
T(n)：$O(n_1 * \log(n_2))$
S(n)：O(1)

6.operator>> 運算子overloading：

n 為用戶輸入的項數
T(n)：O(n²)
S(n)：O(n)

7.operator<< 運算子overloading：

n 為多項式的項數
T(n)：O(n)
S(n)：O(1)

8.整個程式：

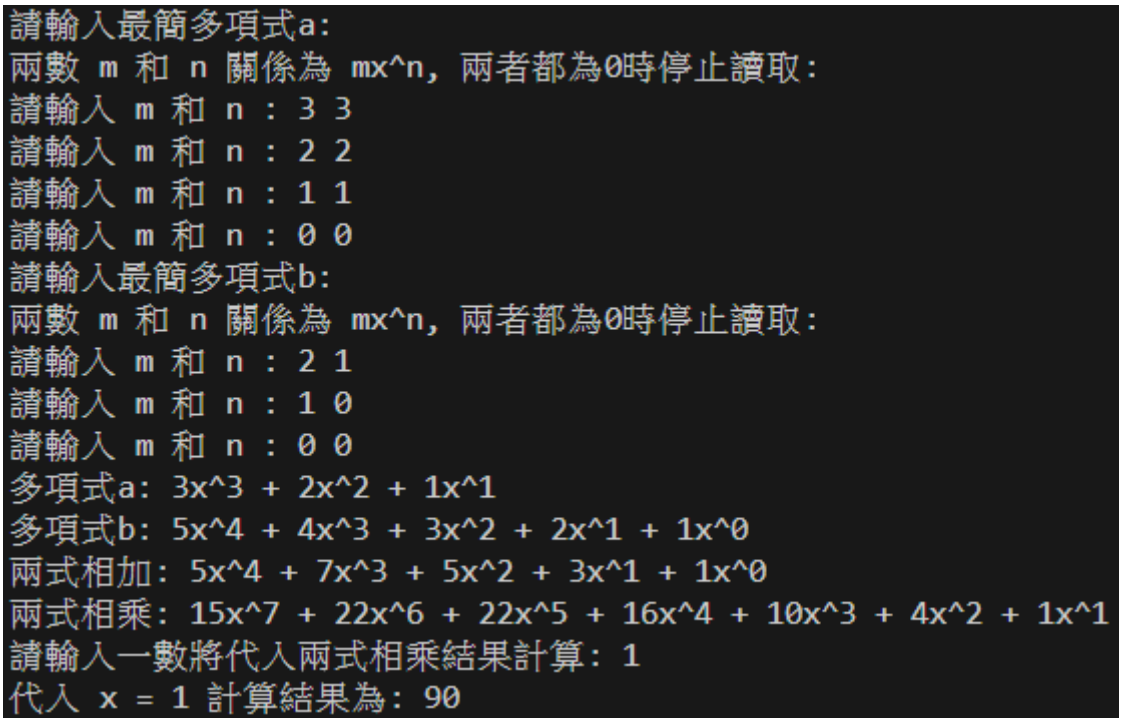
適合項數較少的多項式操作，如果在大量項的情況下效率可能需要提升。

4.測試與驗證

輸入兩多項式
a: 3x^3 + 2x^2 + 1x^1
b: 5x^4 + 4x^3 + 3x^2 + 2x^1 + 1x^0

相加與相乘結果沒問題

代入 x = 1 等於係數相加
15+22+22+16+10+4+1 = 90



5.申論及開發報告

課本的教學以及之前TA課做的題目練習在這次的作業上幫了我很多，指明了大方向讓我能去延伸並完成這次實作，報告製作方面效能分析部分有使用生成式AI輔助完成。

以下為一些函式的邏輯說明:

1.Polynomial::Add函式:
演算法是以排序完成為前提寫的，最開始先使用sort()排序多項式，使用while(i 或 j < terms)判斷兩式是否有未計算的項數，if判斷有兩個邏輯，第一部分左側邏輯為檢察其中一個多項式是否讀取完成，如果是則只讀取剩下的另一個多項式，第二部分右側邏輯是以次數大的一方優先讀取，如果次數相同則兩項相加，最後回傳result多項式。

2.Polynomial::Mult函式:
同上以排序完成為前提寫的先使用sort()，兩式的各項都要相乘使用雙重for迴圈，將相乘出來的項存在newCoef和newExp中，第三個for迴圈用於判斷result多項式中是否包含同樣次數的項，如果包含就將newCoef係數加上去並將布林值contain設定為真，如果result多項式中不包含則contain為假，最後判斷如果contain為假將相乘出來的項加入result多項式，三重迴圈完成後回傳result多項式。