資料結構 HW2

資工二甲 謝於潔

Nov 26,2024

目錄

解題說明

依照題目給的兩個類別去擴展主程式寫出指定條件

Problems

1. **Implement the Polynomial class its ADT and private data members are shown in Figure 1 and 2, respectively.**
2. **Write C++ functions to input and output polynomials represented as Figure 2. Your functions should overload the << and >> operators.**

```
//用來表示單項式(單個項目)的類別
class Term
{
    friend class Polynomial;   //讓 Polynomial類別直接存取Term的私有成員
    friend ostream& operator<<(ostream&, const Polynomial&);
    friend istream& operator>>(istream&, Polynomial&);

private:
    float coef;  //係數
    int exp;     //次方
};

// 表示多項式的 Polynomial 類別
class Polynomial
{
public:
    Polynomial();  //建構子
    Polynomial Add(const Polynomial& poly) const;   //多項式相加
    Polynomial Mult(const Polynomial& poly) const;   //多項式相乘
    float Eval(float f) const;  //計算多項式在某點的值
    friend ostream& operator<<(ostream& out, const Polynomial& poly);  //輸出多項式
    friend istream& operator>>(istream& in, Polynomial& poly);  //輸入多項式

private:
    Term* termArray;  //儲存多項式的非零項
    int capacity;    //termArray的大小
    int terms;       //當前非零項的數量

    void resize();   //調整 termArray 的大小
};
```

驗算法設計與實作

```
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

class Polynomial;  //先呼叫Polynomial類別

//用來表示單項式(單個項目)的類別
class Term
{
    friend class Polynomial;   //讓 Polynomial類別直接存取Term的私有成員
    friend ostream& operator<<(ostream&, const Polynomial&);
    friend istream& operator>>(istream&, Polynomial&);

private:
    float coef;  //係數
    int exp;     //次方
};

// 表示多項式的 Polynomial 類別
class Polynomial
{
public:
    Polynomial();  //建構子
    Polynomial Add(const Polynomial& poly) const;   //多項式相加
    Polynomial Mult(const Polynomial& poly) const;   //多項式相乘
    float Eval(float f) const;  //計算多項式在某點的值
    friend ostream& operator<<(ostream& out, const Polynomial& poly);  //輸出多項式
    friend istream& operator>>(istream& in, Polynomial& poly);  //輸入多項式

private:
    Term* termArray;  //儲存多項式的非零項
    int capacity;     //termArray的大小
    int terms;        //當前非零項的數量

    void resize();   //調整 termArray 的大小
};

//預設建構子：建立空多項式
Polynomial::Polynomial()
{
    capacity = 10;                    //預設最多能儲存10項
    terms = 0;                        //初始時多項事是空的
    termArray = new Term[capacity];  //配置記憶體
}

//動態調整termArray大小的函數
void Polynomial::resize()
{
    capacity *= 2;  //將容量*2擴增
    Term* newArray = new Term[capacity];
    for (int i = 0; i < terms; i++)
    {
        newArray[i] = termArray[i];
    }
    delete[] termArray;     //刪除原本內容
    termArray = newArray;  //拷貝舊資料到新陣列
}
```

```
//多項式相加
Polynomial Polynomial::Add(const Polynomial& poly) const
{
    Polynomial result;
    int i = 0, j = 0;
    while (i < terms && j < poly.terms)
    {
        if (termArray[i].exp == poly.termArray[j].exp)  //一樣次方相加
        {
            float sumCoef = termArray[i].coef + poly.termArray[j].coef;
            if (sumCoef != 0)
            {
                result.termArray[result.terms].coef = sumCoef;
                result.termArray[result.terms].exp = termArray[i].exp;
                result.terms++;
            }
            i++;
            j++;
        }
        else if (termArray[i].exp > poly.termArray[j].exp)  //this多項式的次方較高
        {
            result.termArray[result.terms++] = termArray[i++];
        }
        else  //傳入多項式的次方較高
        {
            result.termArray[result.terms++] = poly.termArray[j++];
        }
        if (result.terms == result.capacity)
        {
            result.resize();
        }
    }
    while (i < terms)  //把剩下的加入
    {
        result.termArray[result.terms++] = termArray[i++];
        if (result.terms == result.capacity) result.resize();
    }
    while (j < poly.terms)
    {
        result.termArray[result.terms++] = poly.termArray[j++];
        if (result.terms == result.capacity) result.resize();
    }
    return result;
}

//多項式相乘
Polynomial Polynomial::Mult(const Polynomial& poly) const
{
    Polynomial result;
    for (int i = 0; i < terms; i++)
    {
        for (int j = 0; j < poly.terms; j++)
        {
            float prodCoef = termArray[i].coef * poly.termArray[j].coef;  //係數相乘
            int prodExp = termArray[i].exp + poly.termArray[j].exp;       //次方相加
            bool found = false;
            for (int k = 0; k < result.terms; k++)
            {
                if (result.termArray[k].exp == prodExp)  //如果找到相同次方的合併在一起
                {
                    result.termArray[k].coef += prodCoef;
                    found = true;
                    break;
                }
            }
            if (!found)  //如果沒有相同的就新增一項
```

```
            {
                result.termArray[result.terms].coef = prodCoef;
                result.termArray[result.terms].exp = prodExp;
                result.terms++;
                if (result.terms == result.capacity) result.resize();
            }
        }
    }
    return result;
}

//計算多項式在某點的值
float Polynomial::Eval(float f) const
{
    float result = 0;
    for (int i = 0; i < terms; i++)
    {
        result += termArray[i].coef * pow(f, termArray[i].exp);
    }
    return result;
}

//輸入多項式
istream& operator>>(istream& in, Polynomial& poly)
{
    cout << "輸入多項式的項數: ";
    in >> poly.terms;
    if (poly.terms > poly.capacity)
    {
        delete[] poly.termArray;
        poly.capacity = poly.terms;
        poly.termArray = new Term[poly.capacity];
    }

    cout << "依次輸入係數與次方（例如：3 2 表示 3x^2）\n";
    for (int i = 0; i < poly.terms; i++)
    {
        in >> poly.termArray[i].coef >> poly.termArray[i].exp;
    }
    return in;
}

//輸出多項式
ostream& operator<<(ostream& out, const Polynomial& poly)
{
    for (int i = 0; i < poly.terms; i++)
    {
        out << poly.termArray[i].coef << "x^" << poly.termArray[i].exp;
        if (i < poly.terms - 1) out << " + ";
    }
    return out;
}

//主程式
int main()
{
    Polynomial p1, p2, p3;
    cout << "輸入第一個多項式:\n";
    cin >> p1;
    cout << "輸入第二個多項式:\n";
    cin >> p2;

    cout << "第一個多項式: " << p1 << endl;
    cout << "第二個多項式: " << p2 << endl;

    p3 = p1.Add(p2);
```

```
    p3 = p1.Mult(p2);
    cout << "乘法結果: " << p3 << endl;

    float evalPoint;
    cout << "輸入一個數字計算第一個多項式的值: ";
    cin >> evalPoint;
    cout << "計算結果: " << p1.Eval(evalPoint) << endl;

    return 0;
}
```

效能分析

時間複雜度

輸入：$O(m)$

加法：$O(m_1 + m_2)$

乘法：$O(m_1 \times m_2)$

計算多項式的值：$O(m)$

空間複雜度

S(P) = $O(m)$用來儲存多項式的項

測試與過程

```
輸入第一個多項式:
輸入多項式的項數: 2
依次輸入係數與次方(例如:3 2 表示 3x^2)
3 2
2 0
輸入第二個多項式:
輸入多項式的項數: 2
依次輸入係數與次方(例如:3 2 表示 3x^2)
1 1
1 0
第一個多項式: 3x^2 + 2x^0
第二個多項式: 1x^1 + 1x^0
加法結果: 3x^2 + 1x^1 + 3x^0
乘法結果: 3x^3 + 3x^2 + 2x^1 + 2x^0
輸入一個數字計算第一個多項式的值: 2
計算結果: 14
```

驗證

第一個方程式為$3x^2 + 2x^0$ ，第二個方程式為$1x^1 + 1x^0$相加為$(3+0)x^2 + (1+0)x^1 + (2+1)x^0 = 3x^2 + x^1 + 3x^0$，相乘為$(3\times1)x^{2+1} + (3\times1)x^{2+0} + (3\times1)x^{0+1} + (2\times1)x^{0+0} = 3x^3 + 3x^2 + 2x^1 + 2x^0$，輸入一個數字計算第一個多項式的值$3\times(2^2) + 2\times(2^0) = 3\times4 + 2\times1 = 14$

問題申論及開發

I. 類別結構

1.使用 Term 類別表示單個多項式項，包含兩個私有成員：
coef（浮點型）：表示多項式項的係數。
exp（整數型）：表示多項式項的指數。
2.使用 Polynomial 類別表示整個多項式，包含：
termArray：動態分配的指針陣列，用於存儲多項式的所有項。
capacity：多項式陣列的最大容量。
3.terms：目前多項式中有效項的數量。

II. 項式的運算

1.加法運算：兩個多項式的項，將具有相同指數的項合併。
2.乘法運算：實現兩個多項式的相乘，根據乘法規則更新係數與指數，並合併相同指數的項。
3.數值計算：針對多項式的每項，代入指定的值 x，計算$a \cdot x^e$並累加。

III.輸入與輸出

1.使用運算符重載來簡化使用者輸入與輸出多項式的方式。
2.>> 運算符：輸入多項式的項數及每項的係數與指數。
3.<< 運算符：輸出格式化後的多項式。

IV. 實現

1.多項式輸入（>>）：使用迴圈輸入每一項，檢查係數和指數的有效性，並將其存入 termArray。
2.多項式輸出（<<）：格式化輸出，例如 3x2+2x03x^2 + 2x^03x2+2x0。
3.加法與乘法運算：使用迴圈遍歷兩個多項式的所有項，將結果存入新的多項式中，並動態分配記憶體以擴容。
4.數值計算（Eval 方法）：使用指數運算 pow(x,e)計算值，並累加至結果變數。

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。