資料結構報告

張哲維

AUG, 22, 2024

CONTENTS

CHAPTER 1

解題說明

用Polynomial來表示多項式，進行加法、乘法　處理多項式的輸入 輸出使用運算符號來方便這些操作。

Ex:

兩個多項式 p1(x) = 4x^2 + 3x + 1 和 p2(x) = x^2 + 2，我們希望實現以下功能：

1. **加法**: p1 + p2 = (4+1)x^2 + 3x + (1+2) = 5x^2 + 3x + 3
2. **減法: p1 - p2 = 3x**^2+3x-
3. **乘法**: p1 * p2 = (4x^2 + 3x + 1)(x^2 + 2) = 8x + x^2 + 4 = 3x^4 + 2x^3 + 13x^2 + 8x + 4

1

CHAPTER2

解題說明

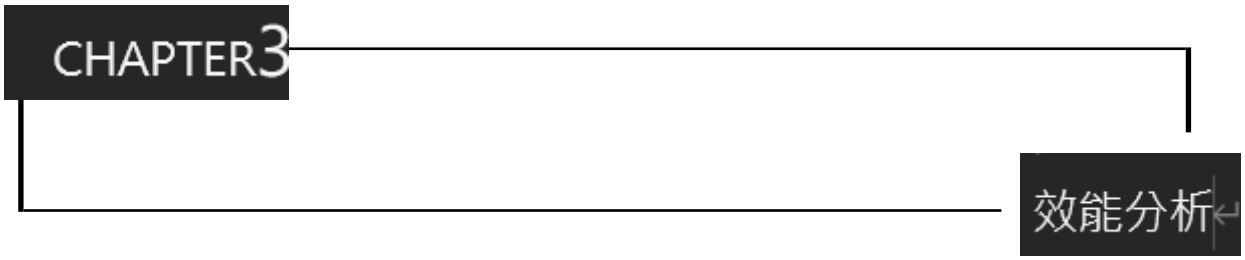
```cpp
#include <iostream>
#include <algorithm>
#include <cmath>
using namespace std;

class Polynomial; // 前向宣告

class Term {
    friend class Polynomial;
public:
    float coef;
    int exp;
};

class Polynomial {
private:
    Term* termArray; // 非零項的數組
    int capacity;    // termArray 的大小
    int terms;       // 非零項的數量

public:
    Polynomial(int cap = 10) : capacity(cap), terms(0) {
        termArray = new Term[capacity];
    }

    ~Polynomial() {
        delete[] termArray;
    }

    friend istream& operator>>(istream& input, Polynomial& p);
    friend ostream& operator<<(ostream& output, const Polynomial& p);

    Polynomial operator+(const Polynomial& other) const;
    Polynomial operator-(const Polynomial& other) const;
```

```
};

istream& operator>>(istream& input, Polynomial& p) {
    int numTerms;
    cout << "輸入項數: ";
    input >> numTerms;

    for (int i = 0; i < numTerms; ++i) {
        float coef;
        int exp;
        cout << "輸入係數和指數: ";
        input >> coef >> exp;
        p.NewTerm(coef, exp);
    }

    sort(p.termArray, p.termArray + p.terms, [](const Term& a, const Term& b) {
        return a.exp > b.exp;
    });

    return input;
}

ostream& operator<<(ostream& output, const Polynomial& p) {
    for (int i = 0; i < p.terms; ++i) {
        output << p.termArray[i].coef << "x^" << p.termArray[i].exp;
        if (i != p.terms - 1) output << " + ";
    }
    return output;
}

void Polynomial::NewTerm(float theCoeff, int theExp) {
    for (int i = 0; i < terms; ++i) {
        if (termArray[i].exp == theExp) {
            termArray[i].coef += theCoeff;
            return;
```

```
    }
    if (terms == capacity) {
        capacity *= 2;
        Term* newArray = new Term[capacity];
        copy(termArray, termArray + terms, newArray);
        delete[] termArray;
        termArray = newArray;
    }
    termArray[terms].coef = theCoeff;
    termArray[terms++].exp = theExp;
}

Polynomial Polynomial::operator+(const Polynomial& other) const {
    Polynomial c;
    int aPos = 0, bPos = 0;
    while (aPos < terms && bPos < other.terms) {
        if (termArray[aPos].exp == other.termArray[bPos].exp) {
            float t = termArray[aPos].coef + other.termArray[bPos].coef;
            if (t) c.NewTerm(t, termArray[aPos].exp);
            aPos++;
            bPos++;
        } else if (termArray[aPos].exp < other.termArray[bPos].exp) {
            c.NewTerm(other.termArray[bPos].coef, other.termArray[bPos].exp);
            bPos++;
        } else {
            c.NewTerm(termArray[aPos].coef, termArray[aPos].exp);
            aPos++;
        }
    }
    for (; aPos < terms; aPos++)
        c.NewTerm(termArray[aPos].coef, termArray[aPos].exp);
    for (; bPos < other.terms; bPos++)
        c.NewTerm(other.termArray[bPos].coef, other.termArray[bPos].exp);
```

```
    while (aPos < terms && bPos < other.terms) {
        if (termArray[aPos].exp == other.termArray[bPos].exp) {
            float t = termArray[aPos].coef - other.termArray[bPos].coef;
            if (t) c.NewTerm(t, termArray[aPos].exp);
            aPos++;
            bPos++;
        } else if (termArray[aPos].exp < other.termArray[bPos].exp) {
            c.NewTerm(-other.termArray[bPos].coef, other.termArray[bPos].exp);
            bPos++;
        } else {
            c.NewTerm(termArray[aPos].coef, termArray[aPos].exp);
            aPos++;
        }
    }
    for (; aPos < terms; aPos++)
        c.NewTerm(termArray[aPos].coef, termArray[aPos].exp);
    for (; bPos < other.terms; bPos++)
        c.NewTerm(-other.termArray[bPos].coef, other.termArray[bPos].exp);
    return c;
}

Polynomial Polynomial::operator*(const Polynomial& other) const {
    Polynomial c;
    for (int i = 0; i < terms; ++i) {
        for (int j = 0; j < other.terms; ++j) {
            float newCoef = termArray[i].coef * other.termArray[j].coef;
            int newExp = termArray[i].exp + other.termArray[j].exp;
            c.NewTerm(newCoef, newExp);
        }
    }
    return c;
}
```

```cpp

float Polynomial::evaluate(float x) const {
    float result = 0.0;
    for (int i = 0; i < terms; ++i) {
        result += termArray[i].coef * pow(x, termArray[i].exp);
    }
    return result;
}

int main() {
    Polynomial p1, p2;

    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 << "多項式1: " << p1 << endl;
    cout << "多項式2: " << p2 << endl;

    Polynomial p3 = p1 + p2;
    Polynomial p4 = p1 - p2;
    Polynomial p5 = p1 * p2;

    cout << "多項式加法結果: (" << p1 << ") + (" << p2 << ") = " << p3 << endl;
    cout << "多項式減法結果: (" << p1 << ") - (" << p2 << ") = " << p4 << endl;
    cout << "多項式乘法結果: (" << p1 << ") * (" << p2 << ") = " << p5 << endl;


    return 0;
}

```



時間複雜度

加法 減法:O(N+M)
乘法O(N*M)

空間複雜度
O(n)

心得

讓我們了解多項式的運算 讓我們更方便在生活上