資料結構報告

41243117 吳承璿

11.27 2024

解題說明

- termArray：動態陣列，儲存多項式的所有項。
- terms：當前多項式的項數。
- capacity：termArray 的容量，用於動態調整陣列大小。
- ensureCapacity(int newCapacity)：確保陣列有足夠的容量以容納新的項，若不足則擴容。
- Polynomial(int cap = 10)： 預設容量為 10，初始化多項式的項數和動態陣列。
- ~Polynomial()： 釋放動態陣列的記憶體。

1. **void AddTerm(float coef, int exp)**

- 將新項加入多項式中。
- 若該次數的項已存在，則合併（將係數相加）。
- 若係數加和為零，則移除該項。
- 動態擴容，確保空間足夠。

2. **float Eval(float f) const**

計算多項式在某一點的值（代入變數 f）。

3. **Polynomial Add(const Polynomial& other) const**

- 實現多項式的加法運算。
- 將兩個多項式的所有項逐項相加。

4. **Polynomial Mult(const Polynomial& other) const**

- 實現多項式的乘法運算。
- 將兩個多項式的每一項進行分配律相乘，並將結果加到新多項式中。

5. void Print() const

- 輸出多項式的表示形式，例如 3x^2 + 2x^1 + 1。

運算子重載

- **輸入 (>>)**： 允許使用者輸入多項式的項數及每項的係數與指數。
- **輸出 (<<)**： 以數學形式顯示多項式（如 3x^2 + 2x^1 + 1）。

演算法與設計

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

class Polynomial {
private:
    struct Term {
        float coef;
        int exp;
    };
    Term* termArray;
    int terms;
    int capacity;

    void ensureCapacity(int newCapacity) {
        if (newCapacity > capacity) {
            capacity = newCapacity;
            Term* newArray = new Term[capacity];
            for (int i = 0; i < terms; ++i) {
                newArray[i] = termArray[i];
            }
            delete[] termArray;
            termArray = newArray;
        }
    }

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

    float Eval(float f) const {
        float result = 0;

        // 計算多項式的值
        for (int i = 0; i < terms; i++) {
            result += termArray[i].coef * pow(f, termArray[i].exp);
        }

        return result;
    }
```

```
// 添加一個新項到多項式
void AddTerm(float coef, int exp) {
    if (coef == 0) return; // 如果係數為 0 則忽略

    // 檢查是否已經存在相同指數的項
    for (int i = 0; i < terms; ++i) {
        if (termArray[i].exp == exp) {
            termArray[i].coef += coef;
            if (termArray[i].coef == 0) {
                // 如果係數變為 0，則移除該項
                for (int j = i; j < terms - 1; ++j) {
                    termArray[j] = termArray[j + 1];
                }
                terms--;
            }
            return;
        }
    }

    // 如果數組已滿，擴展數組
    if (terms == capacity) {
        ensureCapacity(capacity * 2);
    }

    // 添加新項
    termArray[terms].coef = coef;
    termArray[terms].exp = exp;
    terms++;
}

// 輸出多項式
void Print() const {
    for (int i = 0; i < terms; i++) {
        if (i > 0 && termArray[i].coef > 0) cout << " + ";
        if (termArray[i].coef < 0) cout << " - ";
        cout << abs(termArray[i].coef) << "x^" << termArray[i].exp;
    }
    cout << endl;
}
```

```cpp
// 多項式加法
Polynomial Add(const Polynomial& other) const {
    Polynomial result;
    result.ensureCapacity(terms + other.terms);
    for (int i = 0; i < terms; ++i) {
        result.AddTerm(termArray[i].coef, termArray[i].exp);
    }
    for (int i = 0; i < other.terms; ++i) {
        result.AddTerm(other.termArray[i].coef, other.termArray[i].exp);
    }
    return result;
}

// 多項式乘法
Polynomial Mult(const Polynomial& other) const {
    Polynomial result;
    result.ensureCapacity(terms * other.terms);
    for (int i = 0; i < terms; ++i) {
        for (int j = 0; j < other.terms; ++j) {
            result.AddTerm(termArray[i].coef * other.termArray[j].coef, termArray[i].exp + other.termArray[j].exp);
        }
    }
    return result;
}

// 輸入運算符重載
friend istream& operator>>(istream& is, Polynomial& poly) {
    cout << "Enter number of terms: ";
    is >> poly.terms;

    if (poly.terms > poly.capacity) {
        poly.ensureCapacity(poly.terms);
    }

    for (int i = 0; i < poly.terms; i++) {
        cout << "Enter coefficient and exponent for term " << i + 1 << ": ";
        is >> poly.termArray[i].coef >> poly.termArray[i].exp;
    }
    return is;
}
```

```
// 輸入運算符重載
friend istream& operator>>(istream& is, Polynomial& poly) {
    cout << "Enter number of terms: ";
    is >> poly.terms;

    if (poly.terms > poly.capacity) {
        poly.ensureCapacity(poly.terms);
    }

    for (int i = 0; i < poly.terms; i++) {
        cout << "Enter coefficient and exponent for term " << i + 1 << ": ";
        is >> poly.termArray[i].coef >> poly.termArray[i].exp;
    }
    return is;
}

// 輸出運算符重載
friend ostream& operator<<(ostream& os, const Polynomial& poly) {
    for (int i = 0; i < poly.terms; i++) {
        if (i > 0 && poly.termArray[i].coef > 0) {
            os << " + ";
        }
        os << poly.termArray[i].coef << "x^" << poly.termArray[i].exp;
    }
    return os;
}

// 析構函數：釋放動態數組
~Polynomial() {
    delete[] termArray;
}
```

```cpp
// 主函數測試
int main() {
    Polynomial p1, p2;

    // 固定輸入第一個多項式 p1(x) = 3x^2 + 2x + 1
    p1.AddTerm(3, 2);
    p1.AddTerm(2, 1);
    p1.AddTerm(1, 0);

    // 固定輸入第二個多項式 p2(x) = x^2 + 2x + 3
    p2.AddTerm(1, 2);
    p2.AddTerm(2, 1);
    p2.AddTerm(3, 0);

    cout << "First Polynomial: " << p1 << endl;
    cout << "Second Polynomial: " << p2 << endl;

    // 測試多項式加法
    Polynomial sum = p1.Add(p2);
    cout << "Sum: " << sum << endl;

    // 測試多項式乘法
    Polynomial product = p1.Mult(p2);
    cout << "Product: " << product << endl;

    float x = 2.0; // 固定輸入 x 的值
    cout << "p1(" << x << ") = " << p1.Eval(x) << endl;

    return 0;
}
```

效能分析

	時間複雜度	空間複雜度

AddTerm	O(n)	O(n)
Eval	O(n)	O(1)
Add	O(n+m*n)	O(n+m)
Mult	O(n*m)	O(n*m)
>>	O(n)	O(n)
<<	O(n)	O(1)

心得與申論

關於這次的作業，我大部分是參考課本上的舉例來完成的，雖然能夠完成基本的代數運算與數值計算功能，也具可擴展性，但是在效能與空間利用方面仍有進一步改進的空間。儘管目前我還是對於這一方面不太熟悉，未來我會更努力加強自身的程式能力。