# 解題想法

Term 類別設計:Term類別表示多項式中的一項，包含兩個私有成員變數：coef(系數)和 exp(指數)，構造函數初始化系數和指數。

Polynomial 類別設計:Polynomial 類別用於表示多項式，包含私有成員變數：termArray（存儲多項式的項的數組）、capacity（容量）、terms（當前項的數量）。構造函數動態分配存儲空間並初始化容量和項數。用AddTerm 方法向多項式中添加一項。用Add 方法進行多項式加法運算。用Mult 方法進行多項式乘法運算。 用Eval 方法計算多項式在特定值處的結果。 用Print 方法列出多項式。

實現 AddTerm 函數：檢查是否超過多項式的容量，如果超過，則拋出異常。 然後將新項添加到 termArray 中。

實現 Add 函數：項式的加法操作。同時遍歷兩個多項式的項，將指數相同的項進行合併，指數不同的項直接添加到結果多項式中。

實現 Mult 函數：多項式的乘法操作。遍歷兩個多項式的每一項，計算乘積項並添加到結果多項式中。

實現 Eval 函數：計算多項式在給定 x 值處的值。遍歷多項式的每一項，計算每一項在 x 處的值，然後求和。

實現 Print 函數：按格式輸出多項式。

# 演算法設計與實作

#include <iostream>

#include <vector>

#include <stdexcept>

#include <cmath>

using namespace std;

class Polynomial;

class Term {

friend class Polynomial; // 將 Polynomial 設為友元類別，使其能夠訪問 Term 的私有成員

private:

float coef;

int exp;

public:

Term(float c = 0, int e = 0) : coef(c), exp(e) {}

};

class Polynomial {

public:

Polynomial(int capacity = 10);

Polynomial Add(const Polynomial& other) const;

Polynomial Mult(const Polynomial& other) const;

float Eval(float x) const;

// 添加一項到多項式中

void AddTerm(float coef, int exp);

// 列出多項式

void Print() const;

private:

Term\* termArray; // 存儲多項式的項

int capacity; // 容量

int terms; // 當前項的數量

};

// Polynomial 類別的構造函數，初始化容量和項數

Polynomial::Polynomial(int cap) : capacity(cap), terms(0) {

termArray = new Term[capacity]; // 動態分配存儲空間

}

// 向多項式中添加一項

void Polynomial::AddTerm(float coef, int exp) {

if (terms >= capacity) {

throw overflow\_error("error"); // 當超過容量時拋出異常

}

termArray[terms++] = Term(coef, exp); // 添加新項到數組中

}

// 多項式加法

Polynomial Polynomial::Add(const Polynomial& other) const {

Polynomial c;

int aPos = 0, bPos = 0; // 兩個多項式的當前位置

// 合併兩個多項式

while (aPos < terms && bPos < other.terms) {

if (termArray[aPos].exp == other.termArray[bPos].exp) {

// 如果指數相同，系數相加

c.AddTerm(termArray[aPos].coef + other.termArray[bPos].coef, termArray[aPos].exp);

aPos++;

bPos++;

}

else if (termArray[aPos].exp > other.termArray[bPos].exp) {

// 如果當前項的指數較大，添加到結果多項式中

c.AddTerm(termArray[aPos].coef, termArray[aPos].exp);

aPos++;

}

else {

// 否則添加另一個多項式的當前項

c.AddTerm(other.termArray[bPos].coef, other.termArray[bPos].exp);

bPos++;

}

}

// 添加剩餘項

while (aPos < terms) {

c.AddTerm(termArray[aPos].coef, termArray[aPos].exp);

aPos++;

}

while (bPos < other.terms) {

c.AddTerm(other.termArray[bPos].coef, other.termArray[bPos].exp);

bPos++;

}

return c;

}

// 多項式乘法

Polynomial Polynomial::Mult(const Polynomial& other) const {

Polynomial result;

for (int i = 0; i < terms; i++) {

for (int j = 0; j < other.terms; j++) {

float coef = termArray[i].coef \* other.termArray[j].coef; // 系數相乘

int exp = termArray[i].exp + other.termArray[j].exp; // 指數相加

result.AddTerm(coef, exp); // 添加乘積項

}

}

return result;

}

// 多項式求值

float Polynomial::Eval(float x) const {

float result = 0.0;

// 計算多項式在 x 處的值

for (int i = 0; i < terms; i++) {

result += termArray[i].coef \* pow(x, termArray[i].exp); // 根據公式計算

}

return result;

}

// 列出多項式

void Polynomial::Print() const {

for (int i = 0; i < terms; i++) {

cout << termArray[i].coef << "x^" << termArray[i].exp;

if (i < terms - 1) {

cout << " + ";

}

}

cout << endl;

}

int main() {

Polynomial p1;

p1.AddTerm(3, 2);

p1.AddTerm(5, 1);

p1.AddTerm(6, 0);

Polynomial p2;

p2.AddTerm(2, 1);

p2.AddTerm(1, 0);

Polynomial sum = p1.Add(p2);

Polynomial product = p1.Mult(p2);

cout << "Polynomial 1: ";

p1.Print();

cout << "Polynomial 2: ";

p2.Print();

cout << "Sum: ";

sum.Print();

cout << "Product: ";

product.Print();

float value = p1.Eval(2.0);

cout << "Value of Polynomial 1 at x=2: " << value << endl;

return 0;

}

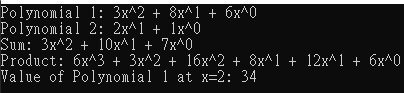
效能分析

加法：時間複雜度為𝑂(𝑛+𝑚)O(n+m)，空間複雜度為𝑂(𝑛+𝑚)O(n+m)。

乘法：時間複雜度為𝑂(𝑛⋅𝑚)O(n⋅m)，空間複雜度為𝑂(𝑛⋅𝑚)O(n⋅m)。

求值：時間複雜度為𝑂(𝑛)O(n)，空間複雜度為𝑂(1)O(1)。

# 測試與過程



(3x^2+8x^1+6)+(2x^1+1)=3x^2+10x^1+7

(3x^2+8x^1+6)\*(2x^1+1)=6x^3+19x^2+20x^1+6