**1.解題說明**

使用Term類別儲存每一項的coef和exp

只儲存非零項，並按照指數大小降序排列

使用動態陣列實現，可根據需要自動擴展容量

實作加法、乘法和求值等基本運算

利用運算子重載實現輸入輸出

**2.Algorithm Design & Programming**

#include <iostream>

using namespace std;

// 前向聲明

class Polynomial;

// 定義項次類別：用來表示多項式中的每一項

class Term {

friend ostream& operator<<(ostream& out, const Polynomial& poly);

friend istream& operator>>(istream& in, Polynomial& poly);

friend Polynomial;

private:

float coef; // 係數

int exp; // 指數

// Term 類別的建構子

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

};

class Polynomial {

public:

// 預設建構子：建立一個空的多項式 p(x) = 0

Polynomial() : capacity(10), terms(0) {

termArray = new Term[capacity];

}

// 解構子：釋放動態配置的記憶體

~Polynomial() {

delete[] termArray;

}

// 複製建構子：深層複製一個多項式

Polynomial(const Polynomial& other) : capacity(other.capacity), terms(other.terms) {

termArray = new Term[capacity];

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

termArray[i] = other.termArray[i];

}

}

// 指定運算子

Polynomial& operator=(const Polynomial& other) {

if (this != &other) {

delete[] termArray;

capacity = other.capacity;

terms = other.terms;

termArray = new Term[capacity];

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

termArray[i] = other.termArray[i];

}

}

return \*this;

}

// 多項式相加

Polynomial Add(Polynomial poly) {

Polynomial result;

int aPos = 0, bPos = 0;

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

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

float coef = termArray[aPos].coef + poly.termArray[bPos].coef;

if (coef != 0) {

result.NewTerm(coef, termArray[aPos].exp);

}

aPos++; bPos++;

}

else if (termArray[aPos].exp < poly.termArray[bPos].exp) {

result.NewTerm(poly.termArray[bPos].coef, poly.termArray[bPos].exp);

bPos++;

}

else {

result.NewTerm(termArray[aPos].coef, termArray[aPos].exp);

aPos++;

}

}

for (; aPos < terms; aPos++) {

result.NewTerm(termArray[aPos].coef, termArray[aPos].exp);

}

for (; bPos < poly.terms; bPos++) {

result.NewTerm(poly.termArray[bPos].coef, poly.termArray[bPos].exp);

}

return result;

}

// 多項式相乘

Polynomial Mult(Polynomial poly) {

Polynomial result;

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

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

float coef = termArray[i].coef \* poly.termArray[j].coef;

int exp = termArray[i].exp + poly.termArray[j].exp;

Polynomial temp;

temp.NewTerm(coef, exp);

result = result.Add(temp);

}

}

return result;

}

// 計算多項式在某一點的值

float Eval(float f) {

float result = 0;

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

float power = 1;

for (int j = 0; j < termArray[i].exp; j++) {

power \*= f;

}

result += termArray[i].coef \* power;

}

return result;

}

// 輸入輸出運算子重載宣告

friend istream& operator>>(istream& in, Polynomial& poly);

friend ostream& operator<<(ostream& out, const Polynomial& poly);

private:

Term\* termArray; // 非零項次陣列

int capacity; // 陣列容量

int terms; // 非零項次個數

// 新增一個項次到多項式中

void NewTerm(float coef, int exp) {

// 先檢查是否已存在相同次方的項

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

if (termArray[i].exp == exp) {

// 如果存在，則係數相加

float newCoef = termArray[i].coef + coef;

if (newCoef != 0) {

// 如果新係數不為零，更新係數

termArray[i].coef = newCoef;

}

else {

// 如果新係數為零，移除此項

for (int j = i; j < terms - 1; j++) {

termArray[j] = termArray[j + 1];

}

terms--;

}

return;

}

else if (termArray[i].exp < exp) {

// 找到正確的插入位置

break;

}

}

// 如果不存在相同次方的項且係數不為零，則新增

if (coef != 0) {

if (terms == capacity) {

capacity \*= 2;

Term\* newArray = new Term[capacity];

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

newArray[i] = termArray[i];

}

delete[] termArray;

termArray = newArray;

}

int i;

for (i = terms - 1; i >= 0 && termArray[i].exp < exp; i--) {

termArray[i + 1] = termArray[i];

}

termArray[i + 1] = Term(coef, exp);

terms++;

}

}

};

// 實作輸入運算子

istream& operator>>(istream& in, Polynomial& poly) {

poly.terms = 0;

int numTerms;

cout << "請輸入項次的個數: ";

in >> numTerms;

cout << "請依序輸入每一項的係數和次方（從高次方到低次方）:\n";

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

float coef;

int exp;

cout << "第 " << i + 1 << " 項（係數 次方）: ";

in >> coef >> exp;

if (coef != 0) {

poly.NewTerm(coef, exp);

}

}

return in;

}

// 實作輸出運算子

ostream& operator<<(ostream& out, const Polynomial& poly) {

if (poly.terms == 0) {

out << "0";

return out;

}

bool first = true;

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

if (first) {

if (poly.termArray[i].coef < 0) {

out << "-";

}

}

else {

if (poly.termArray[i].coef < 0) {

out << " - ";

}

else {

out << " + ";

}

}

float absCoef = abs(poly.termArray[i].coef);

if (absCoef != 1 || poly.termArray[i].exp == 0) {

out << absCoef;

}

if (poly.termArray[i].exp > 0) {

out << "x";

if (poly.termArray[i].exp > 1) {

out << "^" << poly.termArray[i].exp;

}

}

first = false;

}

return out;

}

int main() {

Polynomial p1, p2;

cout << "\n====== 多項式計算機 ======\n\n";

cout << "請輸入第一個多項式：\n";

cin >> p1;

cout << "\n請輸入第二個多項式：\n";

cin >> p2;

cout << "\n====== 計算結果 ======\n";

cout << "p1(x) = " << p1 << endl;

cout << "p2(x) = " << p2 << endl;

Polynomial sum = p1.Add(p2);

cout << "p1 + p2 = " << sum << endl;

Polynomial product = p1.Mult(p2);

cout << "p1 \* p2 = " << product << endl;

float x;

cout << "\n請輸入要帶入的 x 值: ";

cin >> x;

cout << "p1(" << x << ") = " << p1.Eval(x) << endl;

cout << "p2(" << x << ") = " << p2.Eval(x) << endl;

cout << "(p1 + p2)(" << x << ") = " << sum.Eval(x) << endl;

cout << "(p1 \* p2)(" << x << ") = " << product.Eval(x) << endl;

return 0;

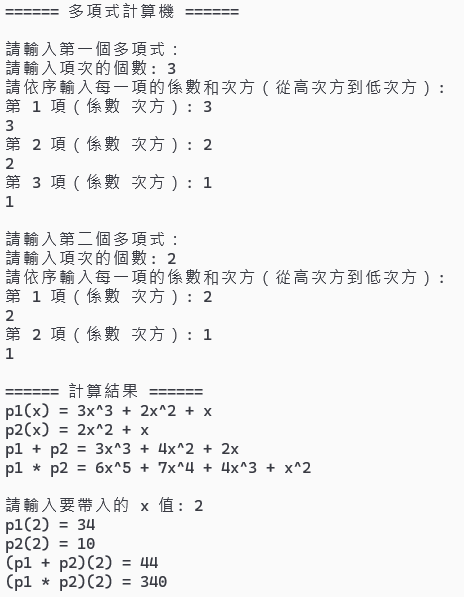
}

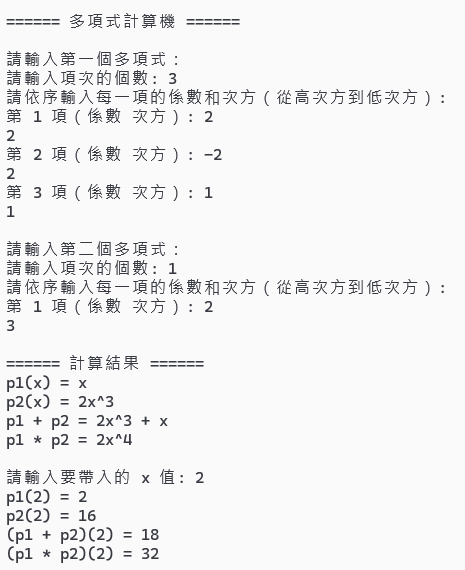
**3.效能分析**

相加: 空間複雜度O(n + m) 時間複雜度O(n + m)

相乘: 空間複雜度O(nm) 時間複雜度O(nm)

**4.測試與驗證**





**5.心得**

這個程式實現了多項式的加法、乘法和帶入運算，透過這個實作，我學到如何設計和操作多項式的基本運算，這次經驗深化了我對資料結構的理解，但我不清楚要怎麼做效能量測 有點可惜。