解題說明

問題：設計一個多項式運算的程式，包含加法、乘法與求值。

想法：

1. 使用類別封裝多項式及其項目（係數與指數）。
2. 提供多項式的建立、新增項、加法、乘法與求值功能。
3. 使用動態記憶體管理來處理多項式項目。

Algorithm Design & Programming

CONTENT

NewTerm()

新增多項式的項目

Add()

將兩個多項式相加並
返回新多項式

Mult()

將兩個多項式相乘並
返回新多項式

Eval()

對多項式進行求值，
返回結果

效能分析&Source code + Comment

```cpp
void Polynomial::NewTerm(const float newCoef, const int newExp) {
    if (this->terms == this->capacity) { // 空間不足時重新配置空間
        this->capacity *= 2;
        Term * temp = new Term[this->capacity];
        copy(this->termArray, this->termArray + terms, temp);
        delete[] this->termArray;
        this->termArray = temp;
    }
    this->termArray[this->terms].coef = newCoef;
    this->termArray[this->terms++].exp = newExp;
}
```

```
Polynomial的newTerm()
請輸入新增項的係數: 1
請輸入新增項的指數: 0
新增前的多項式= 3x^5 + 6x^2 + 8
新增後的多項式= 3x^5 + 6x^2 + 8 + 1
-------------------------
```

Polynomial 的 NewTerm()是讓多項式類別可以增加項次，當空間不足時，我們將空間大小設為原始空間的兩倍大，將原始的資料搬移到新的資料串，並 釋放存放原始資料串的記憶體，這個時間複雜度最好的狀 況O(1)，但最差也還是O(1)；空間複雜度最好的情況是S(n+4)，最壞是S(n+4+2^(⌈ lgn ⌉ +1))。

效能分析

```cpp
float Polynomial::Eval(float f) { // 多項式求值
    float res = 0.0f;
     for (int i = 0; i < this->terms; i++) { // 走訪terms
         float temp = this->termArray[i].coef;
            for (int j = 0; j < this->termArray[i].exp; j++) // f^n
                temp *= f;
                 res += temp;
     }
       return res; // 回傳最終計算值
   }
```

```
Polynomial的Eval()
請輸入f值: 1
多項式: 3x^5 + 6x^2 + 8 + 1
結果= 18
```

Polynomial 的 Eval()是若給 f 求出多項式的值，我是先透過 res 去儲存結果值， 就是去走訪多項式，先用temp變數儲存係數，在根據項次的指數去乘上輸入的f值， 以模擬計算，最後在加到res，跑完就會知道值是多少，最後回傳結果

```cpp
Polynomial Polynomial::Add(Polynomial poly) {
    Polynomial res;
    int* loc;
    loc = new int[poly.terms + this->terms];
    float* data;
    data = new float[poly.terms + this->terms];
    int use_len = 0;
    for (int i = 0; i < this->terms; i++) {
        int t = -1;
        for (int j = 0; j < use_len; j++) {
            if (this->termArray[i].exp == loc[j]) {
                t = j;
                continue;
            }
        }
        if (t == -1) {
            loc[use_len] = this->termArray[i].exp;
            data[use_len++] = this->termArray[i].coef;
        }
        else
            data[t] += this->termArray[i].coef;
    }
     for (int i = 0; i < poly.terms; i++) {  // 把參數poly放入陣列
        int t = -1; // -1表示未找到
        for (int j = 0; j < use_len; j++) {  // 走訪重複判斷
            if (poly.termArray[i].exp == loc[j]) {
                t = j;
                continue;
            }
        }
        if (t == -1) {
            loc[use_len] = poly.termArray[i].exp;
            data[use_len++] = poly.termArray[i].coef;
        }
        else
            data[t] += poly.termArray[i].coef;
    }
     for (int i = 0; i < use_len; i++)  // 存入新的多項式類別
            res.NewTerm(data[i], loc[i]);
    return res;
}
```

Polynomial 的 Add()是自己加參數的多項式。A區域是建立一些初始設定，res是 經由多項式相加的結果，loc&data陣列是暫存放多項式加的內容，它們的大小空間最 差的情況是兩個多項式terms的大小相加。B區域是針對自己的多項式，走訪自己的所有項次，將每個內容放到暫存陣列 中，t 指的是target 要存放指數的索引位置，若 t=-1，代表沒有重複的項次，所以直接放 入暫存，並將use_len遞增，否則，有重複的項次，所以找到指數在loc的索引值，透 過索引值對應data的值，加上這次的係數值，若不清楚可以看圖15，C區跟B區差 不多，只是B區是this自己，C區是參數的poly，D區就是將暫存的data和loc存入 res 多項式。

這個Add()函式時間複雜度為O(n+m)；空間複雜度為O(4(n+m)+12)。

```
Polynomial的Add()
p1 = 3x^5 + 6x^2 + 8 + 1
p2 = 9x^4 + 8x^2 + 3x
(3x^5 + 6x^2 + 8 + 1) + (9x^4 + 8x^2 + 3x) = 3x^5 + 14x^2 + 9 + 9x^4 + 3x
Add()需時: 0.001s
```

```cpp
Polynomial Polynomial::Mult(Polynomial poly) {
        Polynomial res;
        int* loc = new int[poly.terms * this->terms];  // 指數的陣列
        float* data = new float[poly.terms * this->terms];  // 係數的陣列
        int use_len = 0;  // 陣列使用長度
       for (int i = 0; i < this->terms; i++) {  // 走訪自己的Term陣列
           for (int j = 0; j < poly.terms; j++) {  // 走訪參數poly的Term陣列
              float t_coef = this->termArray[i].coef * poly.termArray[j].coef;  // 計算係數
                  int t_exp = this->termArray[i].exp + poly.termArray[j].exp;  // 計算指數
                     int t = -1;  // -1表示未找到
        for (int k = 0; k < use_len; k++) {  // 走訪重複判斷
            if (t_exp == loc[k]) {
                  t = k;
                  continue;
               }
            }
            if (t == -1) {
             loc[use_len] = t_exp;
             data[use_len++] = t_coef;
            }
            else
               data[t] += t_coef;
        }
    }
       for (int i = 0; i < use_len; i++) {
           res.NewTerm(data[i], loc[i]);
       }
            return res;
}
```

Polynomial 的 Mult()是將自己的多項式乘以參數的多項式，for 迴圈前的都是初 始化的設置，跟Add()前面的一樣，只是這邊的空間大小是相乘，主要過程是透過雙 層for 迴圈模擬乘法運算，第一層是走訪自己的多項式，第二層是走訪參數的多項式， 內部先計算係數和指數，再根據指數去找有沒有重複，若重複找重複的地方，接來就 是根據重複與否，是重複就根據t加到對應位置，若沒重複就直接放入，過程是差不多的。

Mult()函式時間複雜度是O(n*m)；空間複雜度是O(2(n+m+nm)+13)。

```
Polynomial的Mult():
p1 = 3x^5 + 6x^2 + 8 + 1
p2 = 9x^4 + 8x^2 + 3x
(3x^5 + 6x^2 + 8 + 1) * (9x^4 + 8x^2 + 3x) = 27x^9 + 24x^7 + 63x^6 + 129x^4 + 18x^3 + 72x^2 + 27x
Mult()需時: 0.002s
```

心得討論

1 反思與改進

使用 STL（如 std::vector）替代動態陣列，簡化記憶體管理

2

增加對輸入格式的檢查，避免錯誤輸入

3

提供更多操作，例如多項式的除法與微分

4 整體感想

實現了多項式運算的核心功能，增強了對動態記憶體與演算法的理解

程式碼

```cpp
#include <iostream>
#include <string>
#include <exception>
#include <ctime>
using namespace std;

class Polynomial; // 前向宣告

class Term {
    friend Polynomial;
    friend ostream& operator<<(ostream& os, const Polynomial& p);
private:
    float coef; // 係數
    int exp;    // 指數
};

class Polynomial {
    friend ostream& operator<<(ostream& os, const Polynomial& p);
    friend istream& operator>>(istream& input, Polynomial& p);
private:
    Term* termArray; // 多項式中非零項的陣列
    int capacity;    // termArray 的大小
    int terms; // 非零項的數量
public:
    Polynomial(); // Construct the polynomial p(x) = 0.
    Polynomial Add(Polynomial poly); // Return the sum of the polynomial *this and poly.
    Polynomial Mult(Polynomial poly); // Return the product of the polynomials this and poly.
    float Eval(float f); // Evaluate the polynomial this at f and return the result.
    void NewTerm(const float newCoef, const int newExp); // 新增項目
};

Polynomial::Polynomial() : capacity(2), terms(0) {
    this->termArray = new Term[capacity];
}

Polynomial Polynomial::Add(Polynomial poly) {
    Polynomial res;
    int* loc;
    loc = new int[poly.terms + this->terms];
    float* data;
    data = new float[poly.terms + this->terms];
    int use_len = 0;
    for (int i = 0; i < this->terms; i++) {
        int t = -1;
        for (int j = 0; j < use_len; j++) {
            if (this->termArray[i].exp == loc[j]) {
                t = j;
                continue;
            }
        }
        if (t == -1) {
            loc[use_len] = this->termArray[i].exp;
            data[use_len++] = this->termArray[i].coef;
        }
        else
            data[t] += this->termArray[i].coef;
    }
    for (int i = 0; i < poly.terms; i++) {  // 把參數poly放入陣列
        int t = -1; // -1表示未找到
        for (int j = 0; j < use_len; j++) {  // 走訪重複判斷
            if (poly.termArray[i].exp == loc[j]) {
                t = j;
                continue;
            }
        }
        if (t == -1) {
            loc[use_len] = poly.termArray[i].exp;
            data[use_len++] = poly.termArray[i].coef;
        }
        else
            data[t] += poly.termArray[i].coef;
    }
    for (int i = 0; i < use_len; i++)  // 存入新的多項式陣列
        res.NewTerm(data[i], loc[i]);
    return res;
}

Polynomial Polynomial::Mult(Polynomial poly) {
    Polynomial res;
    int* loc = new int[poly.terms * this->terms];  // 指數的陣列
    float* data = new float[poly.terms * this->terms];  // 係數的陣列
    int use_len = 0;  // 陣列使用長度
    for (int i = 0; i < this->terms; i++) {  // 走訪自己的Term陣列
        for (int j = 0; j < poly.terms; j++) {  // 走訪參數poly的Term陣列
            float t_coef = this->termArray[i].coef * poly.termArray[j].coef;  // 計算係數
            int t_exp = this->termArray[i].exp + poly.termArray[j].exp;  // 計算指數
            int t = -1;  // -1表示未找到
            for (int k = 0; k < use_len; k++) {  // 走訪重複判斷
                if (t_exp == loc[k]) {
                    t = k;
                    continue;
                }
            }
            if (t == -1) {
                loc[use_len] = t_exp;
                data[use_len++] = t_coef;
            }
            else
                data[t] += t_coef;
        }
    }
    for (int i = 0; i < use_len; i++) {
        res.NewTerm(data[i], loc[i]);
    }
    return res;
}
float Polynomial::Eval(float f) { // 多項式求值
    float res = 0.0f;
    for (int i = 0; i < this->terms; i++) { // 走訪terms
        float temp = this->termArray[i].coef;
        for (int j = 0; j < this->termArray[i].exp; j++) // f^n
            temp *= f;
        res += temp;
    }
    return res; // 回傳最終計算值
}
void Polynomial::NewTerm(const float newCoef, const int newExp) {
    if (this->terms == this->capacity) { // 空間不足時重新配置空間
        this->capacity *= 2;
        Term * temp = new Term[this->capacity];
        copy(this->termArray, this->termArray + terms, temp);
        delete[] this->termArray;
        this->termArray = temp;
    };
    this->termArray[this->terms].coef = newCoef;
    this->termArray[this->terms++].exp = newExp;
}
ostream& operator<<(ostream& output, const Polynomial& p) {
    for (int i = 0; i < p.terms; i++) {
        if (p.termArray[i].coef == 0) continue;// 判斷係數是否為 0，跳過該項
        if (i == 0) { // 處理第一項
            output << p.termArray[i].coef;
        }
        else {
            if (p.termArray[i].coef > 0) {// 處理後續項：根據係數的正負，決定是否加 "+"
                output << " + " << p.termArray[i].coef;
            }
            else {
                output << " - " << -p.termArray[i].coef;
            }
        }
        if (p.termArray[i].exp != 0) {// 處理指數
            output << "x";
            if (p.termArray[i].exp != 1) {
                output << "^" << p.termArray[i].exp;
            }
        }
    }
    return output;
}
istream & operator>>(istream & input, Polynomial & p) {
    float t_coef;
    int t_exp;
    char tmp;
    bool plus = true;
    while (1) {
        input >> t_coef;
        if (!plus) {
            t_coef *= -1;
            plus = true;
        }
        input.get(tmp);
        if (tmp == '\n') { // 遇到換行符，代表結束一項
            p.NewTerm(t_coef, 0); // 所有項都存儲完畢
            break; // 跳出迴圈
}
        input.ignore(1); // 忽略指數前的符號
        input >> t_exp; // 讀入指數
        p.NewTerm(t_coef, t_exp);
        input.get(tmp);
        if (tmp == '\n') break ; // 若下一個項目為負，代表下一項係數為負
        if (tmp == '-')  plus = false; // 判斷是否多項式結尾
    }
return input;
}
int main() {
    clock_t start, finish;
    cout << "輸入格式 ax^n2+bx^n1+cx^n0+d (若為常數，可省略x^0)\n";
    Polynomial p1, p2;
    cout << "p1: ";
    cin >> p1;
    cout << "p2: ";
    cin >> p2;
    cout << "p1 = " << p1 << endl;
    cout << "p2 = " << p2 << endl;
    cout << "------------------------\n";
    cout << "Polynomial的newTerm()\n";
    float t_coef = 0.0f;
    int t_exp = 0;
    cout << "請輸入新增項的係數: ";
    cin >> t_coef;
    cout << "請輸入新增項的指數: ";
    cin >> t_exp;
    cout << "新增前的多項式= " << p1 << endl;
    p1.NewTerm(t_coef, t_exp);
    cout << "新增後的多項式= " << p1 << endl;
    cout << "------------------------\n";
    cout << "Polynomial的Eval()\n";
    float f = 0.0f;
    cout << "請輸入x值: ";
    cin >> f;
    cout << "多項式: " << p1 << endl;
    cout << "結果= " << p1.Eval(f) << endl;
    cout << "------------------------\n";
    cout << "Polynomial的Add()\n";
     cout << "p1 = " << p1 << endl;
     cout << "p2 = " << p2 << endl;
    start = clock();
     cout << "(" << p1 << ") + (" << p2 << ") = " << p1.Add(p2) << endl;
    finish = clock();
    cout << "Add()耗時: " << (double)(finish - start) /
        CLOCKS_PER_SEC << "s" << endl;
     cout << "------------------------\n";
     cout << "Polynomial的Mult():\n";
     cout << "p1 = " << p1 << endl;
    cout << "p2 = " << p2 << endl;
     start = clock();
    cout << "(" << p1 << ") * (" << p2 << ") = " << p1.Mult(p2) << endl;
     finish = clock();
    cout << "Mult()耗時: " << (double)(finish - start) / CLOCKS_PER_SEC <<
        "s" << endl;
    cout << "------------------------\n";
    return 0;
}
```

執行結果

Microsoft Visual Studio 偵錯主控台

```
輸入格式 ax^n2+bx^n1+cx^n0+d (若為常數，可省略x^0)
p1: 3x^5+6x^2+8
p2: 9x^4+8x^2+3x^1
p1 = 3x^5 + 6x^2 + 8
p2 = 9x^4 + 8x^2 + 3x
---------------------
Polynomial的newTerm()
請輸入新增項的係數: 1
請輸入新增項的指數: 0
新增前的多項式= 3x^5 + 6x^2 + 8
新增後的多項式= 3x^5 + 6x^2 + 8 + 1
---------------------
Polynomial的Eval()
請輸入f值: 1
多項式: 3x^5 + 6x^2 + 8 + 1
結果= 18
---------------------
Polynomial的Add()
p1 = 3x^5 + 6x^2 + 8 + 1
p2 = 9x^4 + 8x^2 + 3x
(3x^5 + 6x^2 + 8 + 1) + (9x^4 + 8x^2 + 3x) = 3x^5 + 14x^2 + 9 + 9x^4 + 3x
Add()需時: 0.001s
---------------------
Polynomial的Mult():
p1 = 3x^5 + 6x^2 + 8 + 1
p2 = 9x^4 + 8x^2 + 3x
(3x^5 + 6x^2 + 8 + 1) * (9x^4 + 8x^2 + 3x) = 27x^9 + 24x^7 + 63x^6 + 129x^4 + 18x^3 + 72x^2 + 27x
Mult()需時: 0.002s
---------------------
```