

資料結構 HW2

一、 解題說明

加法功能(Add)

透過兩個指標移動，如果兩個指標指到的項次方數相同，則將係數相加，加到結果多項式中；如果 poly1 的該項指標次方數 $<$ poly2 的該項指標次方數，則將 poly2 該項加入到結果多項式中；如果 poly1 的該項指標次方數 $>$ poly2 的該項指標次方數，則將 poly1 該項加入到結果多項式中。

乘法功能(Mult)

透過一個布林陣列，且搭配哈希函數對應出一個固定的 index，查找布林陣列的內容，來判斷相乘後是否有重複次方數的項，如果有，則將相乘後的項與原本多項式相加；如果沒有，則直接相乘該項。

求值(Eval)

使用輸入的值，將其代入多項式的每項中，計算每項的數值，並將其全部加起來。

輸入

使用多載輸入運算子" $>>$ "，每次先忽略開頭的" $($ "，接著讀入每項係數，忽略後面的" X "和" $^$ "，再讀入次方數，直到讀入為" $)$ "時，停止輸入。

輸出

依序將多項式，輸出多項式的係數，並加上" X "和" $^$ "，如果指數次方為 0，則只輸出係數，並判斷是否為最後項，如果是，則不輸出" $+$ "，反之，則在每項後輸出" $+$ "。

二、 Algorithm Design & Programming

```
class Polynomial;
class Term { //定義Term物件
    friend Polynomial;
    friend ostream& operator<<(ostream& os, const Polynomial& p); //多載輸出運算子
    friend istream& operator>>(istream& is, const Polynomial& p); //多載輸入運算子
private:
    float coef; //係數
    int exp; //指數次方
};
```

圖 1 定義 Term 物件

```
class Polynomial { //定義Polynomial類別
    friend ostream& operator<<(ostream& os, const Polynomial& p); //多載輸出運算子
    friend istream& operator>>(istream& is, const Polynomial& p); //多載輸入運算子
private:
    Term* termArray; //項的陣列
    int capacity; //記憶體大小
    int terms; //多項式項數
public:
    Polynomial();
    float Eval(float f); //多項式求值
    void NewTerm(const float newCoef, const int newExp); //加入新項
    Polynomial Mult(Polynomial poly); //多項式相乘
    Polynomial Add(Polynomial b); //多項式相加
};
```

圖 2 定義 Polynomial 類別

```
Polynomial::Polynomial() : capacity(2), terms(0) { //建構子
    termArray = new Term[capacity];
}
```

圖 3 Polynomial 建構子

```
void Polynomial::NewTerm(const float newCoef, const int newExp=0) { //加入新項
    if (terms == capacity) //空間不足
    {
        capacity *= 2; //空間翻倍
        Term* tmp = new Term[capacity];
        copy(termArray, termArray + terms, tmp);
        delete[] termArray; //刪除原陣列，釋放記憶體
        termArray = tmp;
    }
    termArray[terms].coef = newCoef; //放入係數
    termArray[terms++].exp = newExp; //放入次方數
}
```

圖 4 加入多項式新的項

```

float Polynomial::Eval(float f) { //多項式求值
    float t = 0;
    for (int i = 0; i < terms; i++) //將f的值依序代入相加
    {
        t += termArray[i].coef * pow(f, termArray[i].exp);
    }
    return t;
}

```

圖 5 多項式求值

```

Polynomial Polynomial::Add(Polynomial b) { //多項式相加
    Polynomial c; //結果多項式
    int apos = 0, bpos = 0;
    while ((apos < terms) && (bpos < b.terms)) //次方相同係數相加
    {
        if (termArray[apos].exp == b.termArray[bpos].exp)
        {
            float t = termArray[apos].coef + b.termArray[bpos].coef;
            if (t)
            {
                c.NewTerm(t, termArray[apos].exp); //加入結果多項式中
            }
            apos++;
            bpos++;
        }
        else if (termArray[apos].exp < b.termArray[bpos].exp) //poly1次方 < poly2次方，poly2加入結果多項式中
        {
            c.NewTerm(b.termArray[bpos].coef, b.termArray[bpos].exp);
            bpos++;
        }
        else //poly1次方 > poly2次方，poly1加入結果多項式中
        {
            c.NewTerm(termArray[apos].coef, termArray[apos].exp);
            apos++;
        }
    }
    for (; apos < terms; apos++) //將poly1剩餘項加入結果多項式中
    {
        c.NewTerm(termArray[apos].coef, termArray[apos].exp);
    }
    for (; bpos < b.terms; bpos++) //將poly2剩餘項加入結果多項式中
    {
        c.NewTerm(b.termArray[bpos].coef, b.termArray[bpos].exp);
    }
    return c;
}

```

圖 6 多項式相加

```

Polynomial Polynomial::Mult(Polynomial poly) { //多項式相乘
    Polynomial res;
    int use_cap = terms*poly.terms; //已用過的次方旗標陣列大小
    bool* use = new bool[use_cap](); //用於檢查是否有重複次方數的項
    int* hash = new int[use_cap](); //紀錄哈希表中對應的值
    for (int i = 0; i < terms; i++) //每項——相乘
    {
        for (int j = 0; j < poly.terms; j++)
        {
            int mult_exp = termArray[i].exp + poly.termArray[j].exp; //相乘後的次方
            if (use[hash_function(mult_exp, use_cap, hash)]) //已用過，將相乘完的項使用加法功能加入多項式
            {
                Polynomial temp;
                temp.NewTerm(termArray[i].coef * poly.termArray[j].coef, mult_exp);
                res = res.Add(temp);
            }
            else //未用過，將相乘完的項加入多項式
            {
                res.NewTerm(termArray[i].coef * poly.termArray[j].coef, mult_exp);
                use[mult_exp] = true;
            }
        }
    }
    return res;
}

```

圖 7 多項式相乘

```

int hash_function(int x, int t, int*u) //計算線性哈希函數
{
    if (u[x % t] == 0 || u[x % t] - 1 == x)
    {
        u[x % t] = x + 1;
        return x % t;
    }
    else
    {
        for (int i = 1; i < t; i++)
        {
            if (u[(x + i) % t] == 0)
            {
                u[(x + i) % t] = x + 1;
                return (x + i) % t;
            }
        }
    }
}

```

圖 8 定義哈希函數

```

ostream& operator<<(ostream& os, const Polynomial& p) { //多載輸出運算子
    for (int i = 0; i < p.terms; i++) {
        if (p.termArray[i].exp == 0) {
            os << p.termArray[i].coef;
            continue;
        }
        if (i == p.terms - 1) { //最後項不加"+"
            os << p.termArray[i].coef << "X^" << p.termArray[i].exp;
        }
        else {
            os << p.termArray[i].coef << "X^" << p.termArray[i].exp << "+";
        }
    }
    return os;
}

```

圖 9 多載輸出運算子

```

istream& operator>>(istream& os, Polynomial& p) { //多載輸入運算子
    float coeftmp;
    int exptmp;
    os.ignore();
    while(1)
    {
        os >> coeftmp; //讀入係數
        os.ignore(2); //忽略 X^
        os >> exptmp; //讀入次方
        p.NewTerm(coeftmp, exptmp); //加入新term
        char c;
        os >> c;
        if (c == ')') break; //讀到括號尾跳出
    }
    os.get(); //吃掉換行
    return os;
}

```

圖 10 多載輸入運算子

```

int main() {
    //宣告三個多項式物件
    Polynomial poly1;
    Polynomial poly2;
    Polynomial poly3;
    int x;
    cout << "輸入格式 (aX^n1+bX^n2+cX^n3)" << endl;
    cout << "poly1:";
    cin >> poly1;
    cout << "poly2:";
    cin >> poly2;
    cout << "(" << poly1 << " + (" << poly2 << " = ";
    poly3 = poly1.Add(poly2);
    cout << poly3 << endl;
    cout << "(" << poly1 << " X (" << poly2 << " = ";
    poly3 = poly1.Mult(poly2);
    cout << poly3 << endl;

    cout << "輸入要代入的數:" ;
    cin >> x;
    cout << "X = " << x << ":" << endl;
    cout << "(" << poly1 << " + (" << poly2 << " = ";
    poly3 = poly1.Add(poly2);
    cout << poly3.Eval(x) << endl;
    cout << "(" << poly1 << " X (" << poly2 << " = ";
    poly3 = poly1.Mult(poly2);
    cout << poly3.Eval(x) << endl;
    return 0;
}

```

圖 11 主程式

三、效能分析

	時間複雜度	空間複雜度
加入新項	$O(\text{terms})$	$S(2 * \text{capacity})$
加法功能	$O(\text{terms} + b.\text{terms})$	$S(3)$
乘法功能	$O(\text{terms} * b.\text{terms})$	$S(2 * \text{capacity})$
哈希函數	$O(\text{terms} * b.\text{terms})$	$S(3)$
求值	$O(\text{terms})$	$S(1)$

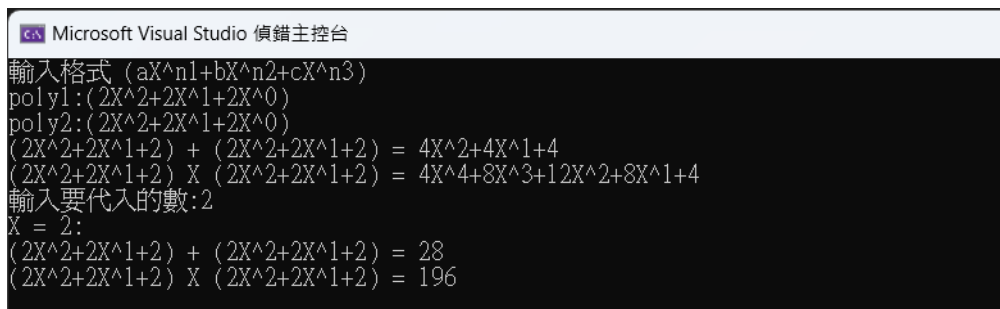
四、測試與驗證

輸入:

poly1: $(2X^2 + 2X^1 + 2X^0)$

Poly2: $(2X^2 + 2X^1 + 2X^0)$

代入數字:2



```
Microsoft Visual Studio 偵錯主控台
輸入格式 (aX^n1+bX^n2+cX^n3)
poly1:(2X^2+2X^1+2X^0)
poly2:(2X^2+2X^1+2X^0)
(2X^2+2X^1+2) + (2X^2+2X^1+2) = 4X^2+4X^1+4
(2X^2+2X^1+2) X (2X^2+2X^1+2) = 4X^4+8X^3+12X^2+8X^1+4
輸入要代入的數:2
X = 2:
(2X^2+2X^1+2) + (2X^2+2X^1+2) = 28
(2X^2+2X^1+2) X (2X^2+2X^1+2) = 196
```

圖 12 輸出結果

五、效能量測

	執行次數	執行時間(ms)	平均時間(ms)
加法功能	10000	6	0.0006
乘法功能	10000	61	0.0061
求值	10000	2	0.0002

六、心得討論

經過這次作業，讓重新複習了使用類別與運算子的多載，我這次原本想使用 STL 中的 map 函數，但想到近期演算法剛好學到哈希函數，剛好透過這次實作機會嘗試實現，此外，也花了不少時間解決輸入多載無法正常讀入的問題，最終以透過讀取前後“(“、”、””，才將此問題解決。我認為這次作業是非常好的機會，讓我們實現課本的範例，讓我們更了解每行程式的意義。