資料結構報告

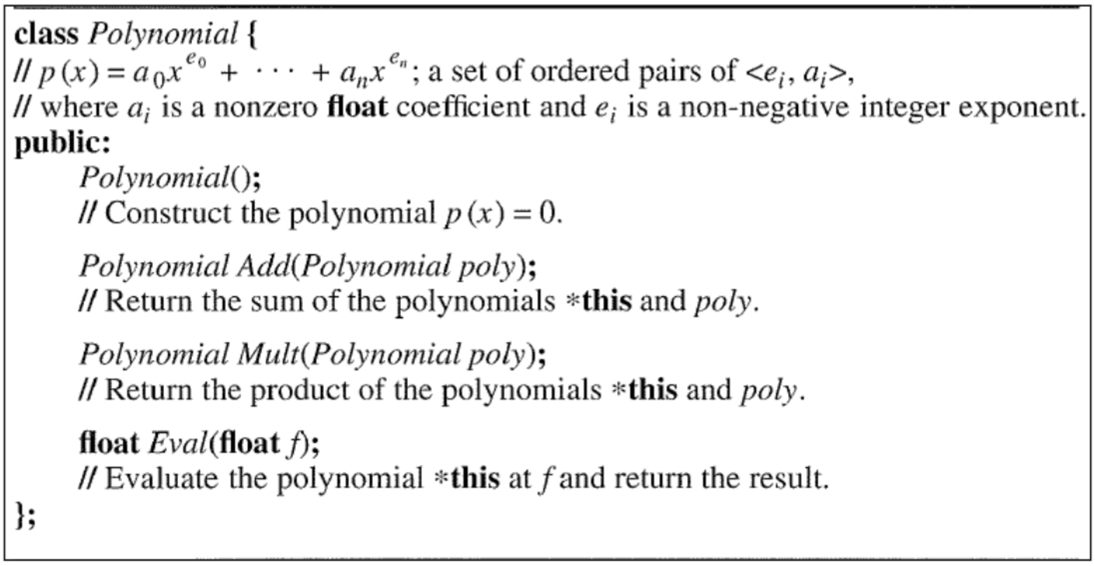
楊政愷

August 6, 2024

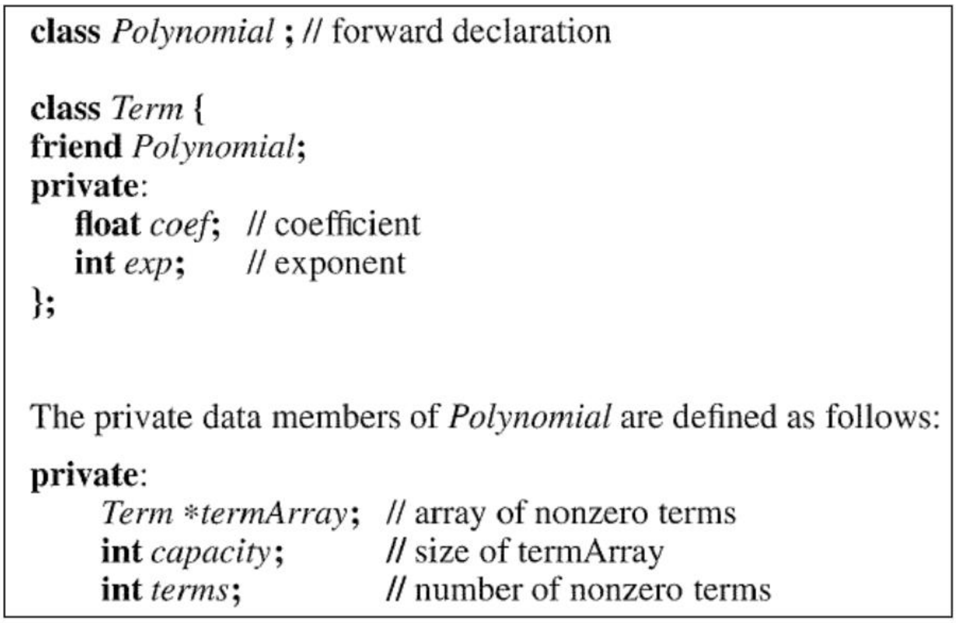
目錄

1. 解題說明

利用上課學到的方法來完成這份功課，並參考課本和 ppt 寫出大概，還上網查了很多解決方式才完成。



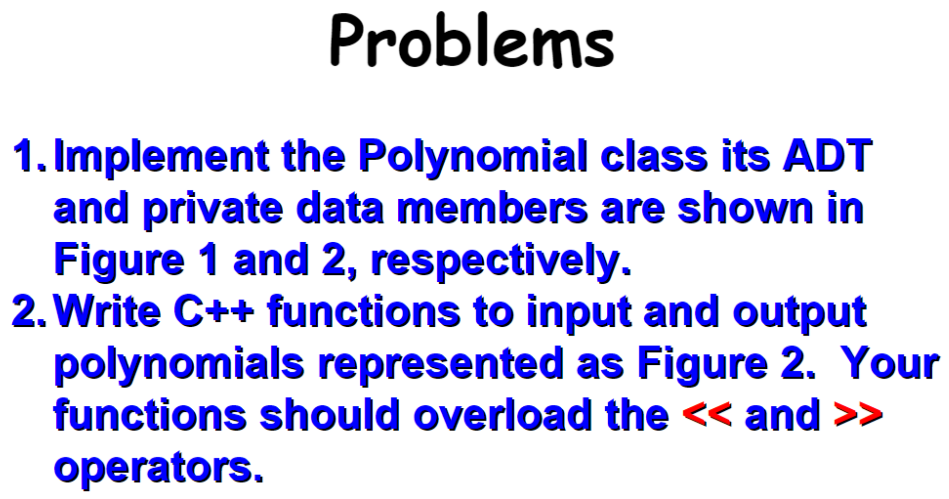
```
class Polynomial {
// $p(x) = a_0x^{e_0} + \cdots + a_nx^{e_n}$; a set of ordered pairs of <$e_i$, $a_i$>,
// where $a_i$ is a nonzero float coefficient and $e_i$ is a non-negative integer exponent.
public:
      Polynomial();
      // Construct the polynomial $p(x) = 0$.

      Polynomial Add(Polynomial poly);
      // Return the sum of the polynomials *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.
};
```



```
class Polynomial ; // forward declaration

class Term {
friend Polynomial;
private:
    float coef;   // coefficient
    int exp;      // exponent
};
```

The private data members of Polynomial are defined as follows:

```
private:
      Term *termArray;   // array of nonzero terms
      int capacity;      // size of termArray
      int terms;         // number of nonzero terms
```

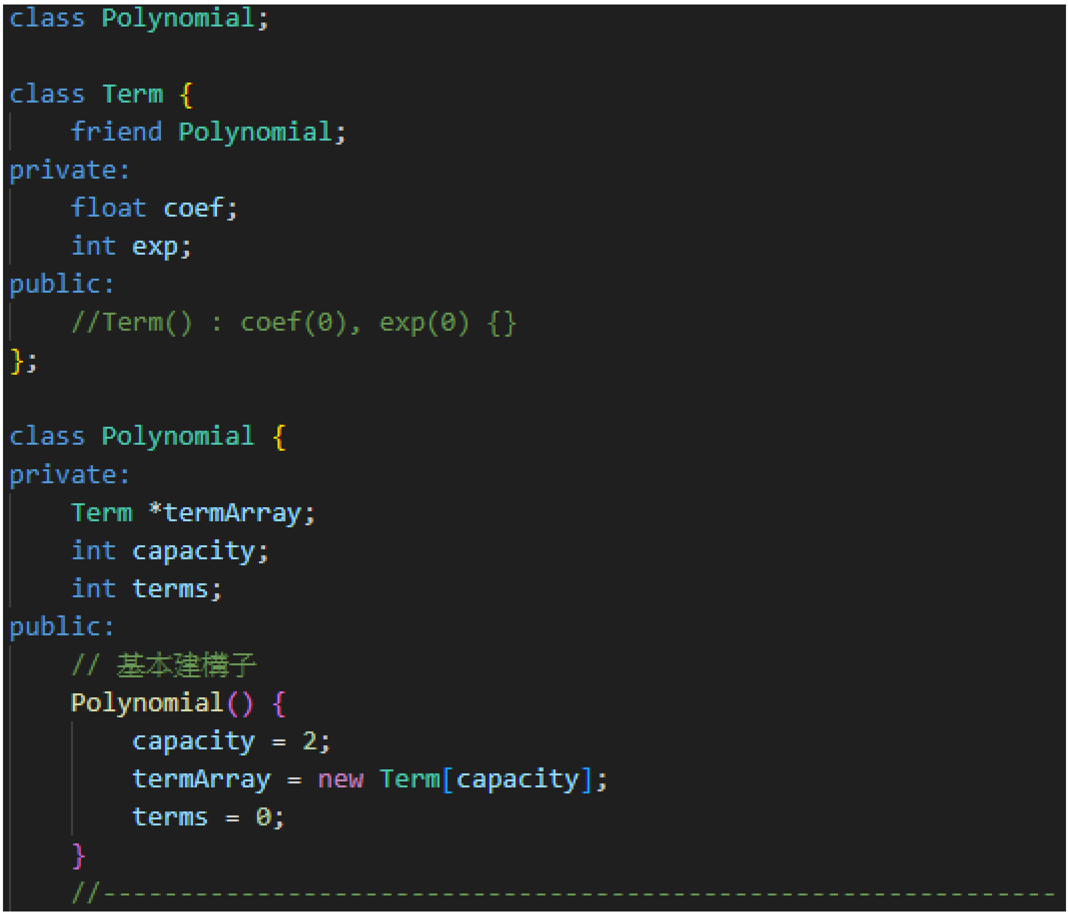


Problems

1. Implement the Polynomial class its ADT and private data members are shown in Figure 1 and 2, respectively.
2. Write C++ functions to input and output polynomials represented as Figure 2. Your functions should overload the << and >> operators.

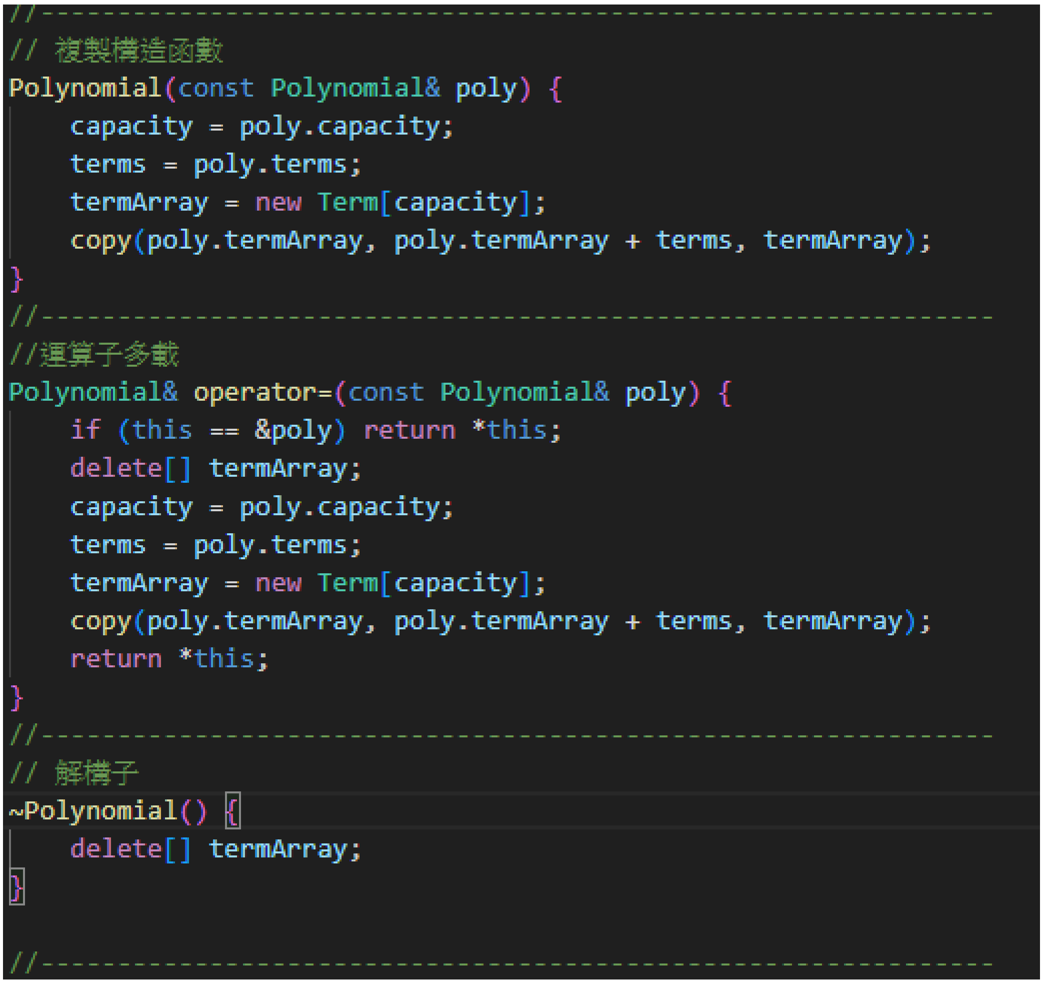
我今天才發現要實作<<與>>的多載，但我需要去上班，會在之後補上。

2. 演算法設計與實作



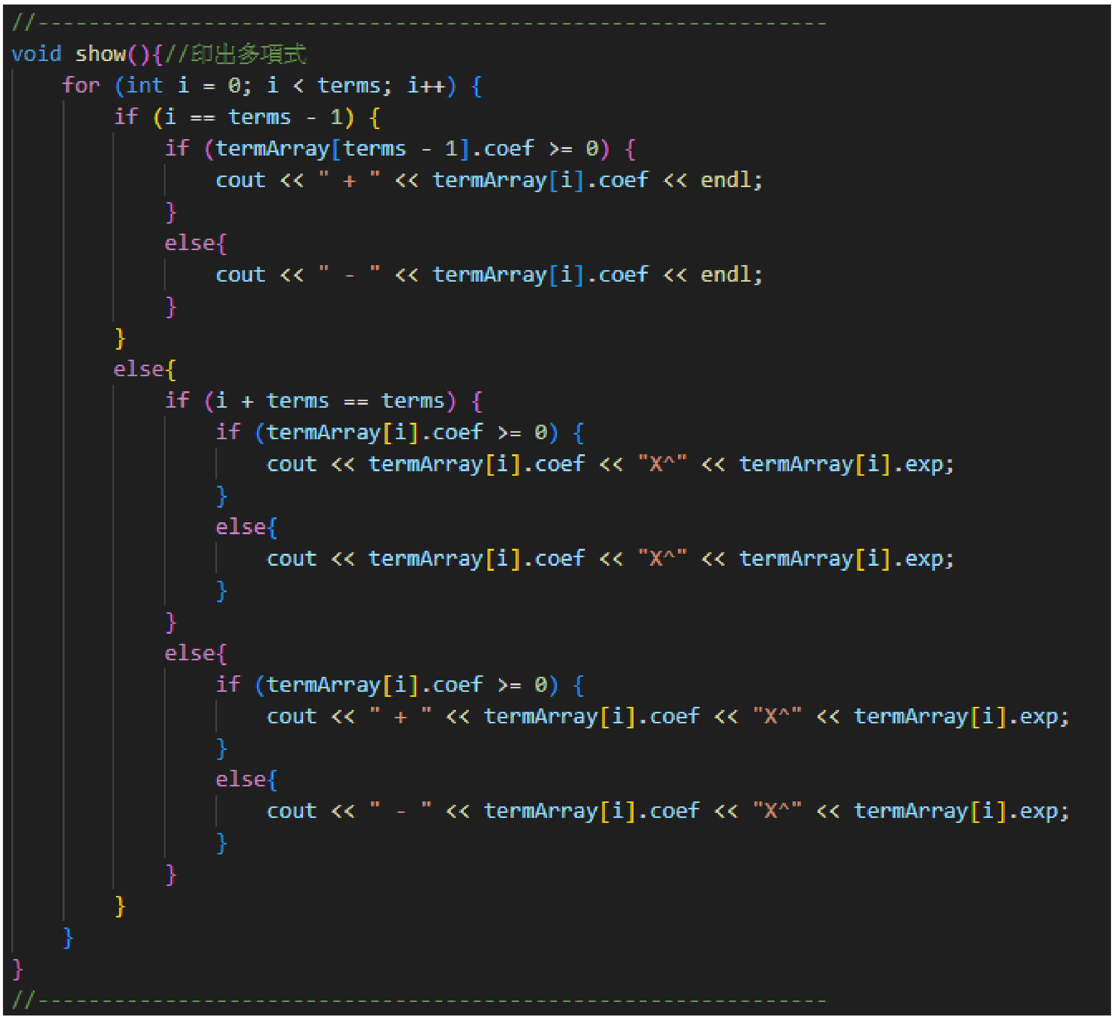
```
class Polynomial;

class Term {
    friend Polynomial;
private:
    float coef;
    int exp;
public:
    //Term() : coef(0), exp(0) {}
};

class Polynomial {
private:
    Term *termArray;
    int capacity;
    int terms;
public:
    // 基本建構子
    Polynomial() {
        capacity = 2;
        termArray = new Term[capacity];
        terms = 0;
    }
    //------------------------------------------------------------
```

Figure2.1.1:Polynomial.cpp



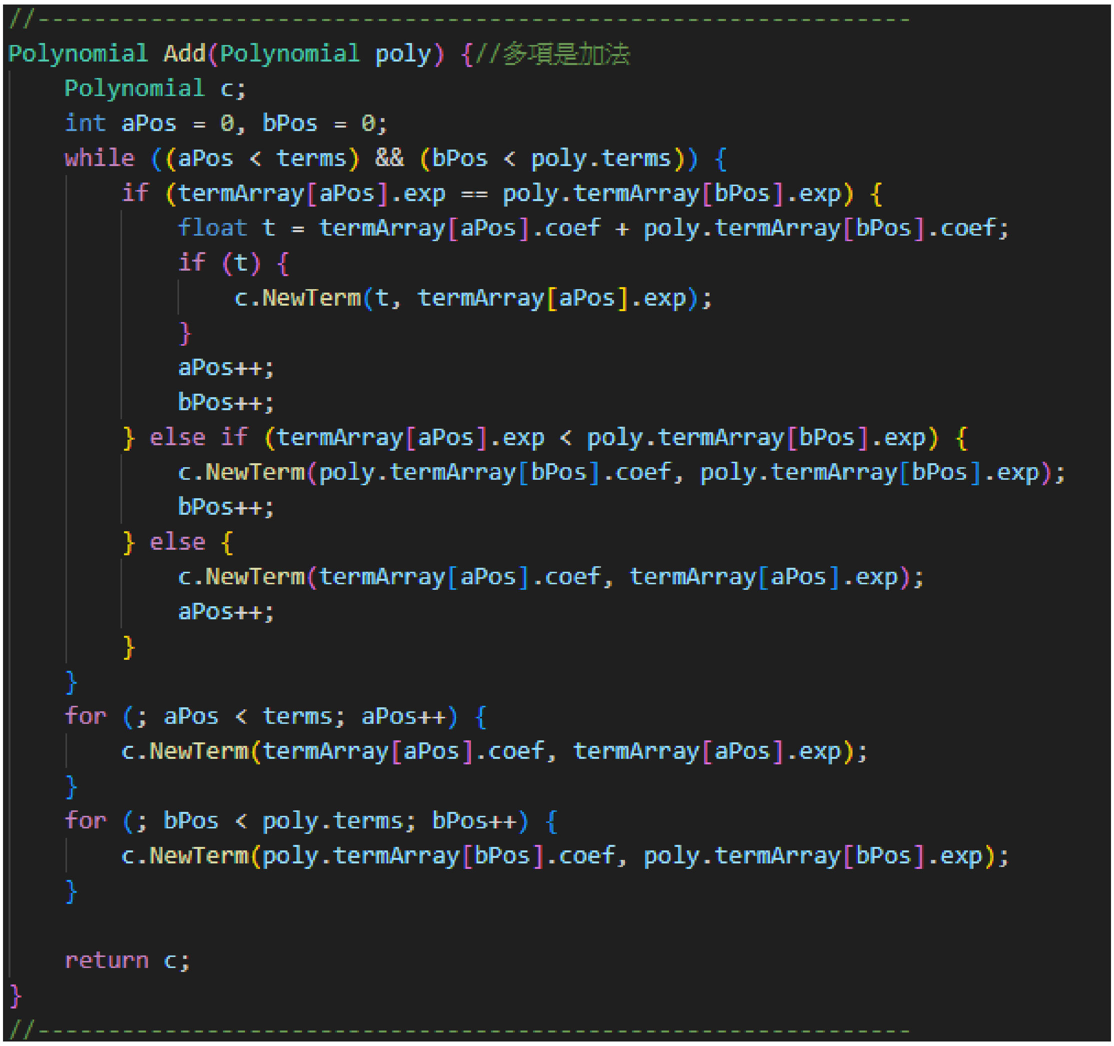
```
//------------------------------------------------------------
// 複製構造函數
Polynomial(const Polynomial& poly) {
    capacity = poly.capacity;
    terms = poly.terms;
    termArray = new Term[capacity];
    copy(poly.termArray, poly.termArray + terms, termArray);
}
//------------------------------------------------------------
//運算子多載
Polynomial& operator=(const Polynomial& poly) {
    if (this == &poly) return *this;
    delete[] termArray;
    capacity = poly.capacity;
    terms = poly.terms;
    termArray = new Term[capacity];
    copy(poly.termArray, poly.termArray + terms, termArray);
    return *this;
}
//------------------------------------------------------------
// 解構子
~Polynomial() {
    delete[] termArray;
}

//------------------------------------------------------------
```

Figure2.1.2:Polynomial.cpp



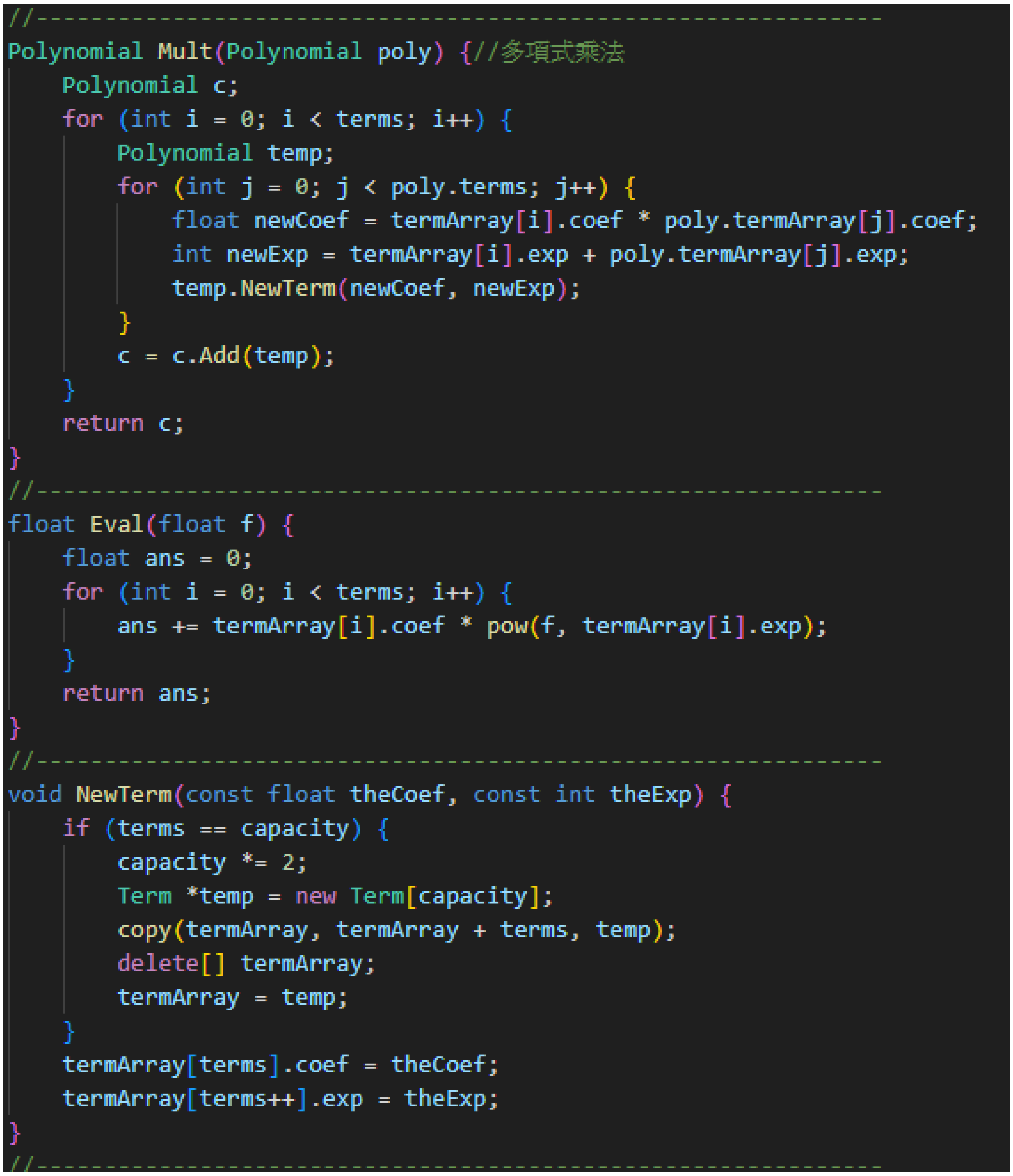
```cpp
//------------------------------------------------------------
void show(){//印出多項式
    for (int i = 0; i < terms; i++) {
        if (i == terms - 1) {
            if (termArray[terms - 1].coef >= 0) {
                cout << " + " << termArray[i].coef << endl;
            }
            else{
                cout << " - " << termArray[i].coef << endl;
            }
        }
        else{
            if (i + terms == terms) {
                if (termArray[i].coef >= 0) {
                    cout << termArray[i].coef << "X^" << termArray[i].exp;
                }
                else{
                    cout << termArray[i].coef << "X^" << termArray[i].exp;
                }
            }
            else{
                if (termArray[i].coef >= 0) {
                    cout << " + " << termArray[i].coef << "X^" << termArray[i].exp;
                }
                else{
                    cout << " - " << termArray[i].coef << "X^" << termArray[i].exp;
                }
            }
        }
    }
}
//------------------------------------------------------------
```

Figure2.1.3:Polynomial.cpp



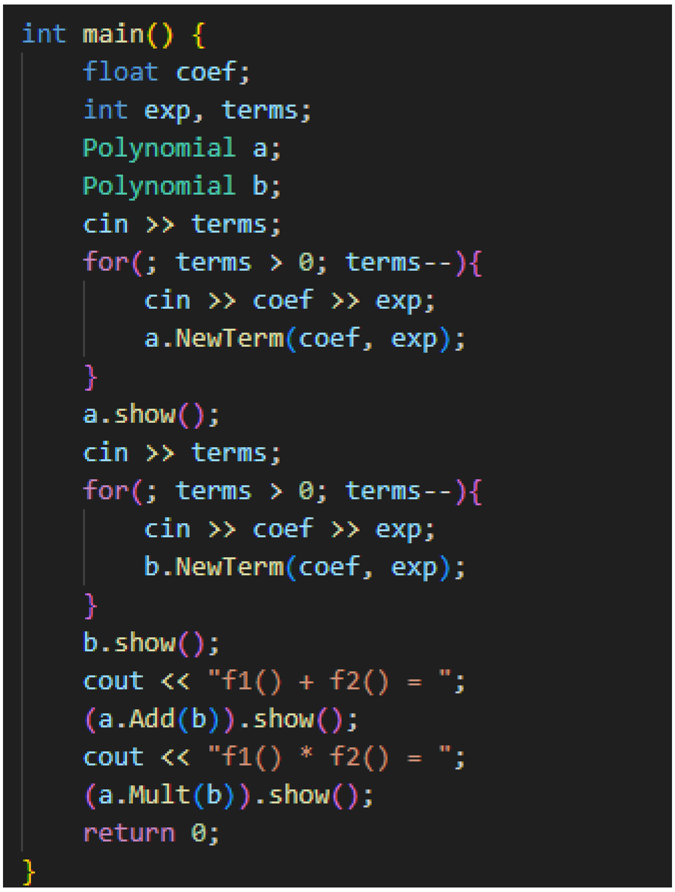
```
//-------------------------------------------------------------
Polynomial Add(Polynomial poly) {//多項是加法
    Polynomial c;
    int aPos = 0, bPos = 0;
    while ((aPos < terms) && (bPos < poly.terms)) {
        if (termArray[aPos].exp == poly.termArray[bPos].exp) {
            float t = termArray[aPos].coef + poly.termArray[bPos].coef;
            if (t) {
                c.NewTerm(t, termArray[aPos].exp);
            }
            aPos++;
            bPos++;
        } else if (termArray[aPos].exp < poly.termArray[bPos].exp) {
            c.NewTerm(poly.termArray[bPos].coef, poly.termArray[bPos].exp);
            bPos++;
        } else {
            c.NewTerm(termArray[aPos].coef, termArray[aPos].exp);
            aPos++;
        }
    }
    for (; aPos < terms; aPos++) {
        c.NewTerm(termArray[aPos].coef, termArray[aPos].exp);
    }
    for (; bPos < poly.terms; bPos++) {
        c.NewTerm(poly.termArray[bPos].coef, poly.termArray[bPos].exp);
    }

    return c;
}
//-------------------------------------------------------------
```

Figure2.1.4:Polynomial.cpp



```cpp
//--------------------------------------------------------------
Polynomial Mult(Polynomial poly) {//多項式乘法
    Polynomial c;
    for (int i = 0; i < terms; i++) {
        Polynomial temp;
        for (int j = 0; j < poly.terms; j++) {
            float newCoef = termArray[i].coef * poly.termArray[j].coef;
            int newExp = termArray[i].exp + poly.termArray[j].exp;
            temp.NewTerm(newCoef, newExp);
        }
        c = c.Add(temp);
    }
    return c;
}
//--------------------------------------------------------------
float Eval(float f) {
    float ans = 0;
    for (int i = 0; i < terms; i++) {
        ans += termArray[i].coef * pow(f, termArray[i].exp);
    }
    return ans;
}
//--------------------------------------------------------------
void NewTerm(const float theCoef, const int theExp) {
    if (terms == capacity) {
        capacity *= 2;
        Term *temp = new Term[capacity];
        copy(termArray, termArray + terms, temp);
        delete[] termArray;
        termArray = temp;
    }
    termArray[terms].coef = theCoef;
    termArray[terms++].exp = theExp;
}
//--------------------------------------------------------------
```

Figure2.1.5:Polynomial.cpp



```cpp
int main() {
    float coef;
    int exp, terms;
    Polynomial a;
    Polynomial b;
    cin >> terms;
    for(; terms > 0; terms--){
        cin >> coef >> exp;
        a.NewTerm(coef, exp);
    }
    a.show();
    cin >> terms;
    for(; terms > 0; terms--){
        cin >> coef >> exp;
        b.NewTerm(coef, exp);
    }
    b.show();
    cout << "f1() + f2() = ";
    (a.Add(b)).show();
    cout << "f1() * f2() = ";
    (a.Mult(b)).show();
    return 0;
}
```

Figure2.1.6:Polynomial.cpp

3. 效能分析

F(n) = O(n)

時間複雜度

Show():O(terms)

Add():O(max(terms))

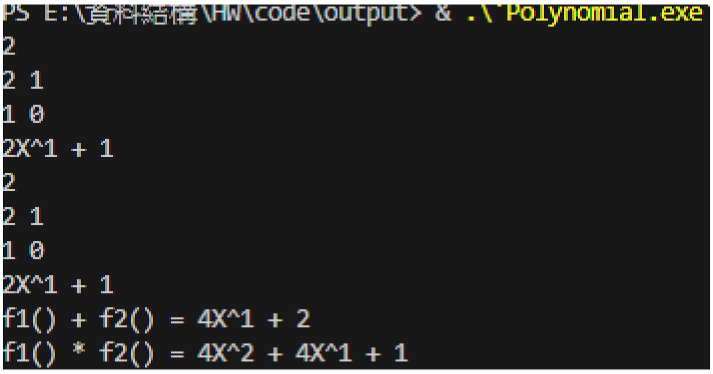
Mult():O(terms^2)

空間複雜度

不知道該怎麼算

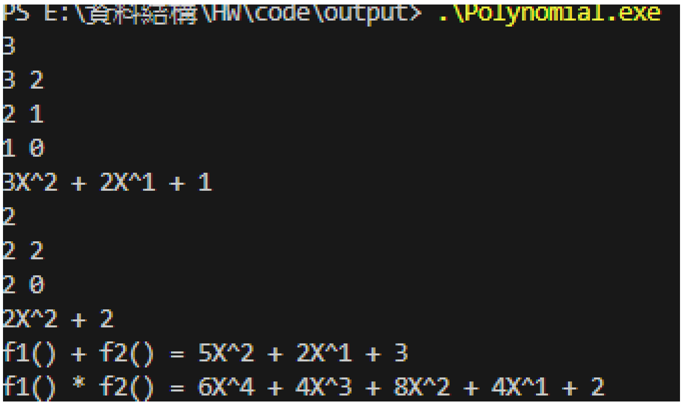
4. 測試與過程

起初是打算自己從 0 到有，但太多 bug 查課本後才好的，後來在寫 Mult 遇到很多問題，多了 Figure 4.3 兩個完整性處理才好的。



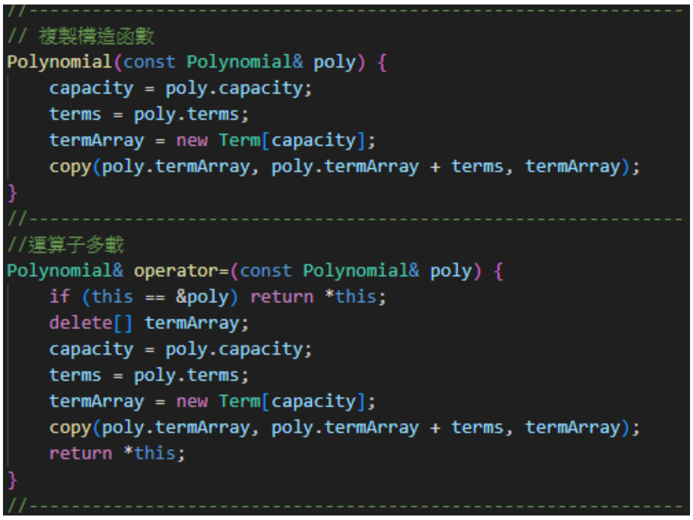
```
PS E:\資料結構\HW\code\output> & .\`Polynomial.exe
2
2 1
1 0
2X^1 + 1
2
2 1
1 0
2X^1 + 1
f1() + f2() = 4X^1 + 2
f1() * f2() = 4X^2 + 4X^1 + 1
```

Figure4.1:Polynomial.cpp



```
PS E:\資料結構\HW\code\output> .\Polynomial.exe
3
3 2
2 1
1 0
3X^2 + 2X^1 + 1
2
2 2
2 0
2X^2 + 2
f1() + f2() = 5X^2 + 2X^1 + 3
f1() * f2() = 6X^4 + 4X^3 + 8X^2 + 4X^1 + 2
```

Figure4.2:Polynomial.cpp



```cpp
//----------------------------------------------------------------
// 複製構造函數
Polynomial(const Polynomial& poly) {
    capacity = poly.capacity;
    terms = poly.terms;
    termArray = new Term[capacity];
    copy(poly.termArray, poly.termArray + terms, termArray);
}
//----------------------------------------------------------------
//運算子多載
Polynomial& operator=(const Polynomial& poly) {
    if (this == &poly) return *this;
    delete[] termArray;
    capacity = poly.capacity;
    terms = poly.terms;
    termArray = new Term[capacity];
    copy(poly.termArray, poly.termArray + terms, termArray);
    return *this;
}
//----------------------------------------------------------------
```

Figure4.3:Polynomial.cpp