HomeWork_2 Polynomial

這段程式碼多項式類別 (polynomial class)，在寫程式時，我們需要先寫出建構函式和其私有成員，這樣我們才能看到程式的部分。

```
class Polynomial {
    private:
        Term *termArray;    //array of nonzero terms
        int capacity;   //size of termArray
        int terms;  //number of nonzero terms
    public:
        Polynomial(int C = 0, int T = 0) : capacity(C), terms(T) {  //constructor.
            termArray = new Term[capacity];
        }
```

可以看到在 Polynomial 類別的私有成員中，有三個成員，其中資料型態 Term 是另一個類別，代表 Polynomial 類別中的一項。termArray 是一個陣列，用來儲存和存取每個項的係數和指數。

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

我們初始化 capacity 和 terms，如果沒有傳入值，則會使用預設值來初始化。接著，會為 termArray 根據 capacity 分配空間。完成了建構函式和私有成員的規範後，接下來是 public 函式成員部分。

```
public:
    Polynomial(int C = 0, int T = 0) : capacity(C), terms(T) {  //constructor.
        termArray = new Term[capacity];
    }
    //construct the polynomial p(x) = 0;
    void showTerm();
    //To show the polynomial.
    void NewTerm(const float theCoeff, const int theExp);
    //Add a new term to the end of termArray.
    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.
```

1. showTerm()：在主程式中顯示多項式。

```
void Polynomial::showTerm() {   //To show the polynomial.
    for(int j=0;j<terms;j++) {
        if((int)termArray[j].coef != 0 && (int)termArray[j].exp > 0) cout<<termArray[j].coef<<"x^"<<termArray[j].exp;
        else if(termArray[j].exp == 0) cout<<termArray[j].coef;
        else continue;
        if(j==terms-1) break;
        cout<<" + ";
    }
}
```

首先，使用 for 迴圈來印出 termArray 陣列中每一項的係數 (coef) 和指數 (exp)。接著，判斷 coef 是否不等於零且 exp 也不等於零。如果條件成立，就輸出 coef x^exp；否則，如果 exp 等於零，則只輸出 coef；如果條件不滿足，則繼續進入下一個迴圈。

2. NewTerm(coef,exp) : Add a new term with coefficient ‘coef’ and exponent ‘exp’ to the end of termArray.

```
void Polynomial::NewTerm(const float theCoeff, const int theExp) {  //Add a new term to the end of termArray.
    if(terms == capacity) { //double capacity of termArray
        capacity*=2;
        Term *temp = new Term[capacity];    //new array
        copy(termArray, termArray + terms, temp);
        delete [] termArray;    //deallocate old memory
        termArray = temp;
    }
    termArray[terms].coef = theCoeff;
    termArray[terms++].exp = theExp;
}
```

如果 termArray 的容量不足以容納新增的項數，則會建立一個更大的 Term 陣列，其容量是原來的兩倍來存取資料。接著，將舊陣列中的資料複製到新陣列，並刪除舊陣列以節省空間。這樣可以動態擴展陣列的容量，以容納更多的項目。

3. Add(poly)：回傳兩個多項式的相加結果。

```
Polynomial Polynomial::Add(Polynomial b) {  //Return the sum of the polynomials *this and b.
    Polynomial c;
    int aPos = 0, bPos = 0;
    while((aPos<terms) && (bPos<b.terms)) {
        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);
            aPos++; bPos++;
        }
        else if((termArray[aPos].exp<b.termArray[bPos].exp)) {
            c.NewTerm(b.termArray[bPos].coef, b.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<b.terms;bPos++) c.NewTerm(b.termArray[bPos].coef, b.termArray[bPos].exp);
    return c;
}
```

第一個 if 條件是當兩個多項式的指數相同時，會將它們的係數相加，並將新的結果作為一個新的項目加入到結果多項式 c 中。

第二個 if 條件是當 poly 的指數比 this（即目前的多項式）大時，會將 poly 的該項加到結果多項式 c 中。第三個 if 條件是當 this 的指數比 poly 的大時，會將 this 的該項加到結果多項式 c 中。最後，將剩餘的項目從兩個多項式加到結果多項式中，然後 return 結果多項式 c。

4. Mult(poly) :return 兩個多項式的乘積。

```
Polynomial Polynomial::Mult(Polynomial b) { //Return the product of the polynomials *this and b.
    Polynomial d;
    int aPos = 0, bPos = 0, dPos = 0, temp, count = 0;
    float temp1;
    for(aPos=0;aPos<terms;aPos++) {
        for(bPos=0;bPos<b.terms;bPos++) {
            float t=termArray[aPos].coef*b.termArray[bPos].coef;
            int r=termArray[aPos].exp+b.termArray[bPos].exp;
            if(r>=0) d.NewTerm(t,r);
        }
    }
    for(int i=0;i<d.terms;i++) {
        for(int j=i+1;j<=d.terms;j++) {
            if(d.termArray[i].exp<d.termArray[j].exp) {
                temp=d.termArray[i].exp;
                d.termArray[i].exp=d.termArray[j].exp;
                d.termArray[j].exp=temp;
                temp1=d.termArray[i].coef;
                d.termArray[i].coef=d.termArray[j].coef;
                d.termArray[j].coef=temp1;
            }
            if(d.termArray[i].exp==d.termArray[j].exp && d.termArray[i].exp>0) {
                temp1=d.termArray[i].coef+d.termArray[j].coef;
                d.termArray[i].coef=temp1;
                for(int k=j;k<d.terms;k++) {
                    d.termArray[k]=d.termArray[++k];
                    k--;
                }
                d.terms--;
            }
        }
    }
    return d;
}
```

第一個 for 迴圈：這個迴圈的作用是將當前多項式 (this) 和 poly 進行相乘，並將乘積儲存到結果多項式 d 中。如果有相同指數的項，它們的係數會被相乘並儲存在 d 中。

第二個 for 迴圈：這個迴圈的作用是對多項式 d 進行選擇排序，將 d 中的項按指數大小排序。排序完成後，會刪除多餘的項目，這是指在 d 中指數相同的項目會合併，將它們的係數相加，並去除冗餘項目。

5. Eval(f) : 當 x 等於某個值 f 時，**評估多項式**的結果是將 x 的每一個指數項替換為 f，並計算出總和

6. 。

```
float Polynomial::Eval(float f) {   //Evaluate the polynomial *this at f and return the result.
    float sum = 0;
    for(int i=0;i<terms;i++) sum+= termArray[i].coef*pow(f,termArray[i].exp);
    return sum;
}
```

例如:

如果多項式 a 是 2x1000+12x^{1000} + 12x1000+1，那麼 a.Eval(1) 將會等於 2×11000+1=32 \times 1^{1000} + 1 = 32×11000+1=3，當 fff 等於 1 時，計算結果為 3。

Time complexity & Space complexity

```
int main() {
    int capacity, terms, exponent;
    float coefficient, f;
    cout<<"Enter the capacity and number of terms for this polynomial(p1): ";
    cin>>capacity>>terms;
    Polynomial p(capacity);
    for(int i=0;i<terms;i++) {
        cout<<"Enter the coefficient and its exponent: ";
        cin>>coefficient>>exponent;
        p.NewTerm(coefficient ,exponent);
    }
    cout<<"p1(x) = ";
    p.showTerm();
    cout<<'\n';
    cout<<"Enter the capacity and number of terms for another polynomial(p2): ";
    cin>>capacity>>terms;
    Polynomial p2(capacity);
    for(int i=0;i<terms;i++) {
        cout<<"Enter the coefficient and its exponent: ";
        cin>>coefficient>>exponent;
        p2.NewTerm(coefficient ,exponent);
    }
    cout<<"p2(x) = ";
    p2.showTerm();
    cout<<'\n';
    cout<<"The sum of p1 and p2 = ";
    p.Add(p2).showTerm();
    cout<<'\n';
    cout<<"The product of p1 and p2 = ";
    p.Mult(p2).showTerm();
    cout<<'\n';
    cout<<"Enter a number(the value of x) to evaluate the polynomial : ";
    cin>>f;
    cout<<"p1(x) = "<<p.Eval(f)<<endl;
    cout<<"p2(x) = "<<p2.Eval(f)<<endl;
    cout<<"p1(x) + p2(x) = "<<p.Eval(f)+p2.Eval(f)<<endl;
    cout<<"p1(x) * p2(x) = "<<p.Eval(f)*p2.Eval(f)<<endl;
    return 0;
}
```

對於空間複雜度，Polynomial 類別會為 termArray 定義一個容量，因此我們可以知道定義一個 Polynomial 類別所需的空間等於容量，這個容量由使用者輸

入。但如果容量小於項目數（terms），則會在 NewTerm 中將容量擴展為原來的兩倍。在 Add 函數中，會創建一個新的多項式 c，因此它的空間最壞情況下會等於 terms(p1) + terms(p2)。在 Mult 函數中，會創建一個新的多項式 d，因此它的空間最壞情況下會等於 terms(p1) * terms(p2)。最後，Eval 函數使用的空間為 2（分別用於 f 和 sum）。

因此，空間複雜度為：

3 (對於 int)+2 (對於 float)+2 (對於 p1 建構子)+2×capacity (對於 p1)+2 (對於 p2 建構子)+2×capacity (對於 p2)+(terms(p1)+terms(p2)) (對於 Add)+terms(p1)×terms(p2) (對於 Mult)+2 (對於 Eval)×4 (重複四次)=17+2×capacity (對於 p1)+2×capacity (對於 p2)+(terms(p1)+terms(p2)) (對於 Add)+terms(p1)×terms(p2) (對於 Mult).3 \ (\text{對於 int}) + 2 \ (\text{對於 float}) + 2 \ (\text{對於 p1 建構子}) + 2 \times \text{capacity} \ (\text{對於 p1}) + 2 \ (\text{對於 p2 建構子}) + 2 \times \text{capacity} \ (\text{對於 p2}) + (\text{terms}(p1) + \text{terms}(p2)) \ (\text{對於 Add}) + \text{terms}(p1) \times \text{terms}(p2) \ (\text{對於 Mult}) + 2 \ (\text{對於 Eval}) \times 4 \ (\text{重複四次}) = 17 + 2 \times \text{capacity} \ (\text{對於 p1}) + 2 \times \text{capacity} \ (\text{對於 p2}) + (\text{terms}(p1) + \text{terms}(p2)) \ (\text{對於 Add}) + \text{terms}(p1) \times \text{terms}(p2) \ (\text{對於 Mult}).3 (對於 int)+2 (對於 float)+2 (對於 p1 建構子)+2×capacity (對於 p1)+2 (對於 p2 建構子)+2×capacity (對於 p2)+(terms(p1)+terms(p2)) (對於 Add)+terms(p1)×terms(p2) (對於 Mult)+2 (對於 Eval)×4 (重複四次)=17+2×capacity (對於 p1)+2×capacity (對於 p2)+(terms(p1)+terms(p2)) (對於 Add)+terms(p1)×terms(p2) (對於 Mult).

接下來是時間複雜度，Polynomial 類別的建構子需要 3。showTerm 函數中的 for 迴圈會遍歷項目數，而不是項目數加 1，因為當 j = terms - 1 時會跳出迴圈。在這個 for 迴圈中有三個條件表達式，如果要考慮最壞情況，則需要計算 (terms - 1) * 3。所以我們可以知道，showTerm 函數的總運行時間為 terms + (terms - 1) * 3。接著，NewTerm 函數會花費 1（對於 if）+ 5（語句）+ 2（語句）= 8。Add 函數一開始需要 2（因為 aPos = 0 和 bPos = 0），然後 while 迴圈需要遍歷 terms + 1 或 b.terms + 1，取較小者。但在最壞情況下，terms 等於 b.terms。因此，它的運行時間為 terms * (1 + 1 + 1 + 8 + 2) = 13 * terms，所以總運行時間為 14 * terms + 1。接著在 Mult 函數中，最初需要 3（aPos = 0, bPos = 0, count = 0）。然後它會花費 (terms + 1) + terms * (b.terms + 1) + b.terms *

(3 + 8)，也就是 2 * terms + 1 + terms * b.terms + 11 * b.terms，這是第一個 for 迴圈的運行時間。第二個 for 迴圈將會進行選擇排序，時間複雜度是 d.terms * d.terms。最後，Eval 函數將花費 1（sum = 0）+ (terms + 1)（對於 for 迴圈）+ terms * 1（語句）= 2 * terms + 2。

所以，總時間複雜度為：

3 (對於Polynomial p)+9×terms+1 (對於for 和語句(8 對於 NewTerm))+terms+(terms−1)×3 (對於p1.showTerm)+3 (對於 Polynomial p2)+9×terms+1 (對於for 和語句(8 對於 NewTerm))+terms+(terms−1)×3 (對於p2.showTerm)+14×terms+1 (對於 Add)+terms+(terms−1)×3 (對於c.showTerm)+2×terms+(terms+11)×b.terms+1+d.terms×d.terms (對於 Mult)+terms+(terms−1)×3 (對於d.showTerm)+4×(2×terms+2) (對於Eval).3 \(\text{對於 Polynomial p}) + 9 \times \text{terms} + 1 \(\text{對於 for 和語句(8 對於 NewTerm)}) + \text{terms} + (\text{terms} - 1) \times 3 \(\text{對於 p1.showTerm}) + 3 \(\text{對於 Polynomial p2}) + 9 \times \text{terms} + 1 \(\text{對於 for 和語句(8 對於 NewTerm)}) + \text{terms} + (\text{terms} - 1) \times 3 \(\text{對於 p2.showTerm}) + 14 \times \text{terms} + 1 \(\text{對於 Add}) + \text{terms} + (\text{terms} - 1) \times 3 \(\text{對於 c.showTerm}) + 2 \times \text{terms} + (\text{terms} + 11) \times b.terms + 1 + d.terms \times d.terms \(\text{對於 Mult}) + \text{terms} + (\text{terms} - 1) \times 3 \(\text{對於 d.showTerm}) + 4 \times (2 \times \text{terms} + 2) \(\text{對於 Eval}).3 (對於 Polynomial p)+9×terms+1 (對於for 和語句(8 對於 NewTerm))+terms+(terms−1)×3 (對於p1.showTerm)+3 (對於 Polynomial p2)+9×terms+1 (對於for 和語句(8 對於 NewTerm))+terms+(terms−1)×3 (對於p2.showTerm)+14×terms+1 (對於 Add)+terms+(terms−1)×3 (對於c.showTerm)+2×terms+(terms+11)×b.terms+1+d.terms×d.terms (對於 Mult)+terms+(terms−1)×3 (對於d.showTerm)+4×(2×terms+2) (對於Eval).

Testing and Proving & Measurement

P1 P2 有一樣 term 的情況。

```
Enter the capacity and number of terms for this polynomial(p1): 2 2
Enter the coefficient and its exponent: 2 1000
Enter the coefficient and its exponent: 1 0
p1(x) = 2x^1000 + 1
Enter the capacity and number of terms for another polynomial(p2): 2 2
Enter the coefficient and its exponent: 3 10
Enter the coefficient and its exponent: 5 0
p2(x) = 3x^10 + 5
The sum of p1 and p2 = 2x^1000 + 3x^10 + 6
The product of p1 and p2 = 6x^1010 + 10x^1000 + 3x^10 + 5
Enter a number(the value of x) to evaluate the polynomial : 1
p1(x) = 3
p2(x) = 8
p1(x) + p2(x) = 11
p1(x) * p2(x) = 24

--------------------------------
Process exited after 17.74 seconds with return value 0
請按任意鍵繼續 . . .
```

P2 terms 比 P1 大的情況。

```
Enter the capacity and number of terms for this polynomial(p1): 2 2
Enter the coefficient and its exponent: 2 1000
Enter the coefficient and its exponent: 1 0
p1(x) = 2x^1000 + 1
Enter the capacity and number of terms for another polynomial(p2): 3 3
Enter the coefficient and its exponent: 6 10
Enter the coefficient and its exponent: 5 5
Enter the coefficient and its exponent: 12 0
p2(x) = 6x^10 + 5x^5 + 12
The sum of p1 and p2 = 2x^1000 + 6x^10 + 5x^5 + 13
The product of p1 and p2 = 12x^1010 + 10x^1005 + 24x^1000 + 6x^10 + 5x^5 + 12
Enter a number(the value of x) to evaluate the polynomial : 1
p1(x) = 3
p2(x) = 23
p1(x) + p2(x) = 26
p1(x) * p2(x) = 69

--------------------------------
Process exited after 35.27 seconds with return value 0
請按任意鍵繼續 . . .
```

P2 terms 比 P1 小的情況。

```
Enter the capacity and number of terms for this polynomial(p1): 3 3
Enter the coefficient and its exponent: 3 20
Enter the coefficient and its exponent: 2 10
Enter the coefficient and its exponent: 1 5
p1(x) = 3x^20 + 2x^10 + 1x^5
Enter the capacity and number of terms for another polynomial(p2): 2 2
Enter the coefficient and its exponent: 2 10
Enter the coefficient and its exponent: 6 0
p2(x) = 2x^10 + 6
The sum of p1 and p2 = 3x^20 + 4x^10 + 1x^5 + 6
The product of p1 and p2 = 6x^30 + 22x^20 + 2x^15 + 12x^10
Enter a number(the value of x) to evaluate the polynomial : 1
p1(x) = 6
p2(x) = 8
p1(x) + p2(x) = 14
p1(x) * p2(x) = 48

--------------------------------
Process exited after 30.73 seconds with return value 0
請按任意鍵繼續 . . .
```

有項數是負數

```
Enter the capacity and number of terms for this polynomial(p1): 2 2
Enter the coefficient and its exponent: -2 3
Enter the coefficient and its exponent: -1 0
p1(x) = -2x^3 + -1
Enter the capacity and number of terms for another polynomial(p2): 3 3
Enter the coefficient and its exponent: -3 5
Enter the coefficient and its exponent: -2 2
Enter the coefficient and its exponent: 1 0
p2(x) = -3x^5 + -2x^2 + 1
The sum of p1 and p2 = -3x^5 + -2x^3 + -2x^2
The product of p1 and p2 = 6x^8 + 7x^5 + -2x^3 + 2x^2 + -1
Enter a number(the value of x) to evaluate the polynomial : 1
p1(x) = -3
p2(x) = -4
p1(x) + p2(x) = -7
p1(x) * p2(x) = 12

--------------------------------
Process exited after 47.22 seconds with return value 0
請按任意鍵繼續 . . .
```

申論及開發報告

類別設計

Term 類別：

儲存每一項的係數與指數，主要為 Polynomial 類別服務。

Polynomial 類別：

管理多項式的所有項目，包含動態陣列、運算邏輯與輸出功能。

包含動態記憶體管理以支持大規模運算。

運算邏輯

加法運算：

使用雙指標法掃描兩個多項式的項目，合併相同指數的項。

未合併的項直接加入結果多項式。

乘法運算：

使用巢狀迴圈計算所有項目的交叉乘積，並對結果多項式進行排序與合併。

記憶體管理：

當多項式的項數超過初始容量時，動態擴展陣列以容納更多的項目。

運算子重載：

>> 運算子：實現多項式的友好輸入，允許用戶輸入多項式的係數與指數。

<< 運算子：實現多項式的友好輸出，清晰顯示加號與指數。

程式碼結構

程式碼分為以下部分：

類別定義與建構：

定義 Polynomial 與 Term，確保每個類別的責任明確。

函數實現：

加法、乘法、求值函數，實現了多項式的基本運算邏輯。

運算子重載：

輸入輸出運算子的重載，提升程式使用的友好性。

主程式：

驗證多項式運算功能，包括加法、乘法與評估。

效能測試

測試案例：

多項式項數：少至 2 項，多至 100 項。

運算測試：加法與乘法正確性，評估運算效率。

結果：

加法與乘法運算在小型多項式上正確無誤。

測試表明程式能動態管理大規模多項式，但當項數過多（超過 1,000 項）時，運算時間會顯著增加，需進一步優化程式碼。