資料結構 — HW2

作者：蔡品辰

學號：41243154

November 28, 2024.

目錄

Homework_2

- 題目說明

```
class Polynomial{
     //p(x) = a₀x^(e₀) + ... + (an)(x^(en));a set of ordered pairs of <ei,ai>,
     //where ai is a nonzero float coefficient and ei is non-negative integer exponent.
     public:
          Polynomial();
          //Construct the ploynomial 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.
};
```

Figure 1. Abstract data type of Polynomial class.

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

//The private data members of Polynomial are deined as follows:
private:
     Term *termArray; //array of nonzero terms.
     int capacity;    //size of termArray.
     int terms;       //number of nonzero terms.
```

Figure 2. The private data members of Polynomial class.

● 問題

1. Implement the Polynomial calss 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.

● 說明

1. 化簡(Simplify)：合併同指數(exp)的項，並移除係數(coef)為零的項。
2. 加法(Add)：回傳二多項式相加，並化簡之結果。
3. 乘法(Mult)：回傳二多項式相乘，並化簡之結果。
4. 求值(Eval)：輸入未知數，並求多項式之值。

✧ 題目要求實現多項式類別 (Polynomial) 及其抽象數據類型（ADT），除了實現多項式的輸入與輸出功能，還需使用運算符重載(operator)來操作。

✧ 在建立輸出多載時，為了作業方便，而追加了 GetCoef()和 GetExp()作使用。

✧ 由於 Polynomial 中的 Term *termArray，是利用位址而非值(value)。
因此除了建構子和解構子外，還需要另外建立複製運算子(copy constructor)。

✧ 在製作乘法(Mult)功能時，注意到了多項式有化簡(Simplify)功能的需求。

- 程式碼

  - 物件 **Term**

```
class Term{

    friend class Polynomial;
private:
    float coef;
    int exp;
};
```

- 物件 Polynomial

```
class Polynomial{
public:
    Polynomial(): termArray(NULL), capacity(0), terms(0){} //constructor
    Polynomial(const Polynomial& other); //copy constructor
    ~Polynomial(){delete[] termArray;} //destructor
    Polynomial Simplify();
    Polynomial Add(Polynomial poly);
    Polynomial Mult(Polynomial poly);
    float Eval(float f);

    float GetCoef(int index)const{return termArray[index].coef;}
    int GetExp(int index)const{return termArray[index].exp;}

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

private:
    Term *termArray;
    int capacity;
    int terms;
    void AddTerm(float coef, int exp);
};
Polynomial::Polynomial(const Polynomial& other){
    terms = other.terms;
    capacity = other.capacity;
    termArray = new Term[capacity];
    for(int i=0;i<terms;i++){
        termArray[i] = other.termArray[i];
    }
}
```

- 複製建構子(copy constructor)

```
Polynomial::Polynomial(const Polynomial& other){
    terms = other.terms;
    capacity = other.capacity;
    termArray = new Term[capacity];
    for(int i=0;i<terms;i++){
        termArray[i] = other.termArray[i];
    }
}
```

- 化簡(Simplify)

```
Polynomial Polynomial::Simplify(){
    Polynomial result;

    for(int i=0;i<terms;i++){
        for(int j=i+1;j<terms;j++){
            if(termArray[i].exp < termArray[j].exp){
                Term temp = termArray[i];
                termArray[i] = termArray[j];
                termArray[j] = temp;
            }
        }
    }
    for(int i=0;i<terms;i++){
        float newCoef = termArray[i].coef;
        int newExp = termArray[i].exp;
        while(i+1 < terms && termArray[i+1].exp == newExp){
            newCoef += termArray[i+1].coef;
            i++;
        }
        if(newCoef != 0){result.AddTerm(newCoef, newExp);}
    }

    return result;
}
```

- **加法(Add)**

```
Polynomial Polynomial::Add(Polynomial poly){
    Polynomial p1 = Simplify();
    Polynomial p2 = poly.Simplify();
    Polynomial result;
    int i = 0, j = 0;

    while(i < p1.terms || j < p2.terms){
        if(i < p1.terms && (j >= p2.terms || p1.termArray[i].exp >
p2.termArray[j].exp)){
            result.AddTerm(p1.termArray[i].coef, p1.termArray[i].exp);
            i++;
        }else if(j < p2.terms && (i >= p1.terms || p2.termArray[j].exp >
p1.termArray[i].exp)){
            result.AddTerm(p2.termArray[j].coef, p2.termArray[j].exp);
            j++;
        }else{
            float newCoef = p1.termArray[i].coef + p2.termArray[j].coef;
            if(newCoef != 0){result.AddTerm(newCoef, p1.termArray[i].exp);}
            i++; j++;
        }
    }

    return result;
}
```

- **乘法(Mult)**

```
Polynomial Polynomial::Mult(Polynomial poly){
    Polynomial result;

    for(int i=0;i<terms;i++){
        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;
            result.AddTerm(newCoef, newExp);
        }
    }

    return result.Simplify();
}
```

- **求值(Eval)**

```
求值(Eval)
C++
複製
float Polynomial::Eval(float f){
    float result = 0;
    for(int i=0;i<terms;i++){
        result += GetCoef(i) * pow(f, GetExp(i));
    }

    return result;
}
```

- **AddTerm()**

```
void Polynomial::AddTerm(float coef, int exp){
    if(coef == 0)return;
    if(terms == capacity){
        capacity = (capacity == 0)?1 :capacity*2;
        Term *newArray = new Term[capacity];
        for(int i=0;i<terms;i++){
            newArray[i] = termArray[i];
        }
        delete[] termArray;
        termArray = newArray;
    }
    termArray[terms].coef = coef;
    termArray[terms].exp = exp;
    terms++;
}
```

- **istream**

```
istream& operator>>(istream& in, Polynomial& poly){
    int numTerms;
    cout<<" Enter the number of terms: ";
    in>>numTerms;

    for(int i=0;i<numTerms;i++){
        float coef;
        int exp;
        cout<<"  Enter coefficient and exponent for term: "<< i+1 <<": ";
        in>>coef>>exp;
        poly.AddTerm(coef,exp);
    }

    return in;
}
```

- **ostream**

```
ostream& operator<<(ostream& out, Polynomial poly){
    if(poly.terms == 0){
        out<<"0";
        return out;
    }

    for(int i=0;i<poly.terms;i++){
        if(i > 0 && poly.GetCoef(i) > 0){out<<" + ";}
        if(i > 0 && poly.GetCoef(i) < 0){out<<" - ";}
        if(poly.GetCoef(i) != 1 || (i+1 == poly.terms)){out<<abs(poly.GetCoef(i));}
        if(poly.GetExp(i) != 0){
            out<<"x";
            if(poly.GetExp(i) != 1)out<<"^"<<poly.GetExp(i);
        }
    }

    return out;
}
```

- **主程式(main)**

```
int main(){
    Polynomial p1,p2;
    cout<<"Enter the first polynomial: "<<endl;
    cin>>p1;
    cout<<"Enter the second polynomial: "<<endl;
    cin>>p2;

    cout<<"Sum: "<<p1.Add(p2)<<endl;
    cout<<"Product: "<<p1.Mult(p2)<<endl;

    float x;
    cout<<"Enter the value to evaluate the first polynomial: ";
    cin>>x;
    cout<<"P1("<<x<<") = "<<p1.Eval(x)<<endl;

    return 0;
}
```

● 效能分析(Analysis)

1. 化簡(Simplify)
   - Time complexity
     - ✧ 排序功能：使用泡沫排序法，時間複雜度為 $O(n^2)$，n 為項的數量。
     - ✧ 合併功能：時間複雜度為 O(n)。
   - Space complexity
     - ✧ 僅用額外變數暫存係數(coef)和指數(exp)，空間複雜度為 O(1)。
2. 加法(Add)
   - Time complexity
     - ✧ 時間複雜度為 O(n+m)，n 和 m 是兩個多項式的項數。
   - Space complexity
     - ✧ 多項式最多包函 n+m 項，空間複雜度為 O(n+m)。
3. 乘法(Mult)
   - Time complexity
     - ✧ 使用兩個迴圈，時間複雜度為 O(n*m)。
   - Space complexity
     - ✧ 最多有 n*m 的結果，時間複雜度為 O(n*m)。
4. 求值(Eval)
   - Time complexity
     - ✧ 時間複雜度為 O(n)，n 是多項式的項數(terms)。
   - Space complexity
     - ✧ 僅用額外變數儲存結果，空間複雜度為 O(1)。
5. AddTerm
   - Time complexity
     - ✧ 時間複雜度為 O(n)，n 是多項式的項數(terms)。
   - Space complexity
     - ✧ 空間複雜度為 O(n)，n 是多項式的項數(terms)。
6. 整體(total)
   - Time complexity
     - ✧ 時間複雜度為乘法的 O(n*m+p)，p 為乘法(Mult)結果的項數。
   - Space complexity
     - ✧ 空間複雜度為 O(n)。

- **執行結果(Testing)**

```
C:\Users\ping9\OneDrive\文件\.aino\虎科\2-1_資料結構\Homework_2\src>Hw_p1.exe
Enter the first polynomial:
 Enter the number of terms: 2
  Enter coefficient and exponent for term: 1: 3 2
  Enter coefficient and exponent for term: 2: 5 0
Enter the second polynomial:
 Enter the number of terms: 3
  Enter coefficient and exponent for term: 1: 1 3
  Enter coefficient and exponent for term: 2: 2 2
  Enter coefficient and exponent for term: 3: -4 0
Sum: x^3 + 5x^2 + 1
Product: 3x^5 + 6x^4 + 5x^3 - 2x^2 - 20
Enter the value to evaluate the first polynomial: 2
P1(2) = 17

C:\Users\ping9\OneDrive\文件\.aino\虎科\2-1_資料結構\Homework_2\src>Hw_p1.exe
Enter the first polynomial:
 Enter the number of terms: 2
  Enter coefficient and exponent for term: 1: 1 1
  Enter coefficient and exponent for term: 2: 2 1
Enter the second polynomial:
 Enter the number of terms: 2
  Enter coefficient and exponent for term: 1: 3 1
  Enter coefficient and exponent for term: 2: 4 1
Sum: 10x
Product: 21x^2
Enter the value to evaluate the first polynomial: 5
P1(5) = 15

C:\Users\ping9\OneDrive\文件\.aino\虎科\2-1_資料結構\Homework_2\src>
```

- **驗證計算(Proving)**

$(3x^2 + 5) + (x^3 + 2x^2 - 4) = x^3 + 5x^2 + 1$。

$(3x^2 + 5) * (x^3 + 2x^2 - 4) = (3x^5 + 6x^4 - 12x^2) + (5x^3 + 10x^2 - 20) = 3x^5 + 6x^4 + 5x^3 - 2x^2 - 20$。

$x = 2$，$3x^2 + 5 = 12 + 5 = 17$。

$x + 2x + 3x + 4x = 10x$。

$x = 5$，$x + 2x = 3*5 = 15$。

- **效能量測(Measuring)**

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\ping9\OneDrive\文件\.aino\虎科\2-1_資料結構\Homework_2\src\Hw_p1.exe
- Output Size: 1.84063625335693 MiB
- Compilation Time: 1.31s
```

- **心得**

這次程式我花了將進 5 天的時間去完成，主要原因在太長時間沒有去使用 C++的物件導向，一些使用上的細節需要重新複習，像是 friend、輸入輸出多載、複製結構子等功能。

程式本身並不算太困難，重點在於資料的調動、讀取和使用，經過這一次的練習後，我又對 C++的物件導向更進一步的了解和如何利用於未來的程式中。