資料結構 hw2

許家瑋

2024/11/09

1. 實作 Polynomial 類別的 ADT 實作 完整程式碼放在.cpp 檔

```cpp
class Polynomial
{
    friend ostream& operator<<(ostream& os, const Polynomial &p);
    friend istream& operator>>(istream& is, Polynomial &p);
    class Term {
        friend Polynomial;
        friend ostream& operator<<(ostream& os, const Polynomial &p);
    private:
        float coef;
        int exp;
    };
private:
    Term *termArray;
    int capacity;
    int terms;
public:
    Polynomial();
    Polynomial Add(Polynomial poly);
    Polynomial Mult(Polynomial poly);
    float Eval(float f);
    void Setup(int a, int b,float c[],int d[]);
};
```

```cpp
Polynomial Polynomial::Add(Polynomial poly)
{
    Polynomial result;
    int maxDegree = max(this->capacity - 1, poly.capacity - 1);
    result.Setup(maxDegree + 1, 0, nullptr, nullptr);

    for (int i = 0; i <= maxDegree; i++)
    {
        result.termArray[i].coef = (i < this->capacity) ? this->termArray[i].coef : 0;
        result.termArray[i].coef += (i < poly.capacity) ? poly.termArray[i].coef : 0;
        result.termArray[i].exp = i;
    }

    return result;
}
```

```cpp
Polynomial Polynomial::Mult(Polynomial poly)
{
    Polynomial result;
    int maxCapacity = this->capacity + poly.capacity - 1;
    result.Setup(maxCapacity, 0, nullptr, nullptr);

    for (int i = 0; i < this->capacity; i++)
    {
        for (int j = 0; j < poly.capacity; j++)
        {
            int newExp = this->termArray[i].exp + poly.termArray[j].exp;
            float newCoef = this->termArray[i].coef * poly.termArray[j].coef;
            result.termArray[newExp].coef += newCoef;
            result.termArray[newExp].exp = newExp;
        }
    }

    return result;
}
```

```cpp
float Polynomial::Eval(float x)
{
    float result = 0;

    for (int i = 0; i < this->capacity; i++)
    {
        result += this->termArray[i].coef * pow(x, this->termArray[i].exp);
    }
    return result;
}
```

```cpp
void Polynomial::Setup(int a,int b,float c[],int d[]) {
    this->capacity = a + 1;
    this->terms = b;
    this->termArray = new Term[capacity];
    for (int i = 0; i < capacity; i++) {
        termArray[i].coef = 0;
        termArray[i].exp = i;
    }
    for (int i = 0; i < terms; i++) {
        termArray[d[i]].coef = c[i];
        termArray[d[i]].exp = d[i];
    }
    /*
    for (int i = 0; i < capacity; i++) {
        cout << termArray[i].coef <<" "<< termArray[i].exp << ' '<< i << endl;
    }
    */
}
```

2. 主程式

hw2 (全域範圍)

```cpp
int main() {
    Polynomial a, b;
    float x;
    cout << "請輸入A(x)：";
    cin >> a;
    cout << "請輸入B(x)：";
    cin >> b;
    cout << a << endl;
    cout << b << endl;
    cout << "A(x) + B(x) = " << a.Add(b) << endl;
    cout << "A(x) * B(x) = " << a.Mult(b) << endl;
    cout << "請輸入x的值：";
    cin >> x;
    cout << "A(x) = " << a.Eval(x) << endl;
    cout << "B(x) = " << b.Eval(x) << endl;
    system("pause");
    return 0;
}
```

3.

n,m 為兩多項式的次方數

時間複雜度：

Add: O(n)O(n)O(n)

Mult: O(n×m)O(n \times m)O(n×m)

Eval: O(n)O(n)O(n)

Setup: O(n)O(n)O(n)

空間複雜度：

Add: O(n)O(n)O(n)

Mult: O(n+m)O(n + m)O(n+m)

Eval: O(1)O(1)O(1)

Setup: O(n)O(n)O(n)

4.心得

這次的作業有用到類別封裝數學運算，如加法、乘法和評估……等，時間複雜度和空間複雜度的分析則幫助我們理解該演算法的效率，適合處理不同次方的多項式運算需求。這次的作業有比上次的困難些，但我認為這種實作可以運用到老師上課教的理論，做完這次更印象深刻了。