# Homework 3

姓名：黃子峻

2024 8 25

# 解題說明：

(a)Polynomial 輸入: 用一個迴圈為不斷循環輸入直到判斷輸入結束跳出，輸入第一個輸入係數，然後用 get 得到係數後的東西，判斷是不是換行如果為+-號代表還有下一項，然後因為正負 號是下一個項的係數的東西，所以如果為負，則把旗標設為 1，表示下一項乘 上(-1)，然後 continue 直接回到迴圈頂端輸入下一項，如果不為換行或+-則 接著輸入指數次方的大小，然後用 get 跟上一個一樣感測。

(b) Polynomial 輸出: 用迭代器、迴圈將方程式的每一項都抓出來，第一個係數正負按照預設輸出即可，如果不是第一個項輸出以及係數是正數就要加+號(5x^3”+2x”)用來連接兩個項，如果係數是負數則不用+號，直接輸出(因為預設，正數不會輸出+號，負數會輸出-號)，然後依指數次方判斷怎麼輸出， 例:5x 輸出 5 跟 x 不用特別輸出^1，5 則輸出 5 就好不用 x 跟次方，依模式判 斷輸出型態。

(c) Polynomial 的 CopyConstructor:必須用迭代器把要複製每個項都挑出來， 並用 NewTerm 加入現在

這個方程式，不然直接複製會使位址一同被複製過去。

(d)Polynomial 的建構子:建構一個環狀串列並將串列頭設為-1，方便走訪個元素。

(e) Polynomial 的加法多載:先用兩個迭代器代表兩個方程式各自執行到的項，同時比較項指數大小，如果 A 方程式的指數大於 B 方程式的指數，則把 A 方程式的項放入 C 方程式，如果 A 方程式的指數等於 B 方程式的指數，則兩者係數相加次方放入 C 方程式，如果 A 方程式的指數小於 B 方程式的指數，則把 B 方程式的項 放入 C 方程式，最後當 A 或 B 方程式其中一個全放完後，則把另一個還沒放完的全放進 C 方程式。

(f) Polynomial 的減法多載:與加法只差在如果是 B 項要放入 C 方程式時，要加負號。

(g) Polynomial 的乘法多載:第一個迴圈用迭代器跑 A 方程式的每一項，裡面再放一個迴圈跑 B 方程式，使得 A 方程式每次抓住一個項乘上 B 方程式的每個 項全部加總起來，得出兩者相加的方程式。

(h) Polynomial 的代入 Evaluate:用迭代器把方程式每

個項次抓出來，把代入數字依照指數值做次方運算乘上係數加總，然後回傳。

(i) Polynomial 的解構子：直接呼叫成員 poly 的 CircularList 的解構子。

## 演算法設計與實作：

```
#include <iostream>
#include <cstdlib>
using namespace std;
template <class T>
class ChainNode {
public:
    T data; //資料
    ChainNode* link; //下一個的位址
    ChainNode() {} //建構子，什麼都沒有
    ChainNode(const T& data)//建構子,有資料 沒下一個的位址
    {
        this->data = data;
    }
    ChainNode(const T& data, ChainNode* link) //建構子,有資料 有下一個的位址
    {
        this->data = data;
        this->link = link;
    }
};
template <class T>
class CircularListWithHeader { //有含頭的環形鏈結串列
public:
    CircularListWithHeader()
    {
        last = new ChainNode<T>(); last->link = last; //建構子初始一個含頭的環形鏈結串列
    }
    void InsertBack(const T& e); //用於插入新的 x 項
    bool isEmpty() //用於判斷是否為空的環形鏈結串列
    {
        return last == last->link;
    }
    ChainNode<T>*begin() const //提供第一個資料的位址
    {
        return (last->link)->link;
    }
    ChainNode<T>* first() const //提供頭的位址
    {
        return last->link;
    }
    class Iterator { //迭代器用於幫我們更好走訪每個資料
    public:
        Iterator(ChainNode<T>* startNode = 0) //建構子用於存迭代器現在位址
        {
            current = startNode;
        }
        T& operator*()const //可存取現在位址的資料
        {
            return current->data;
        }
        T* operator->()const //方便直接存取 Term 的 Coef 跟 Exp
        {
            return &current->data;
        }
        Iterator& operator++() //讓鏈結串列能像數字+1 一樣前往下一個位址,回傳下一個位址
        {
            current = current->link;
            return *this;
        }
        Iterator& operator++(int) //讓鏈結串列能像數字+1 一樣前往下一個位址,回傳現在位址
        {
```

```
            Iterator old = *this;
            current = current->link;
            return old;
        }
        bool operator!=(const Iterator right) //判斷不是存到相同位址
        {
            return current != right.current;
        }
        bool operator==(const Iterator right) //判斷是存到相同位址
        {
            return current == right.current;
        }
    private:
        ChainNode<T>* current;
    };
private:
    ChainNode<T>* last;
};
template <class T>
void CircularListWithHeader<T>::InsertBack(const T& e)
{
    if (last) //如果本來就有一個以上(含)的資料
    {
        last = last->link = new ChainNode<T>(e, last->link);
    }
    else //如果沒有資料
    {
        last->link = last = new ChainNode<T>(e);
    }
}
```

```
struct Term //用於表示每個項資料的結構
{
    float coef;
    int exp;
    Term Set(float c, int e) { coef = c; exp = e; return *this; }
};
class Polynomial { //用於代表多項式的類別
public:
    Polynomial()
    {
        CircularListWithHeader<Term>::Iterator set = poly.first();
        set->exp = -1; //設定環狀串列的頭的次方為-1，以方便我們知道我們走訪一遍
    }
    Polynomial(const Polynomial& a); //Copy Constructor 複製建構子
    ~Polynomial();
        void NewTerm(float c, int e) //用於加入新的項
    {
        Term temp;
        if (c != 0)
            poly.InsertBack(temp.Set(c, e));
    }
    Polynomial& operator+(const Polynomial& b); //多項式物件相加運算子多載
    Polynomial& operator*(const Polynomial& b); //多項式物件相乘運算子多載
    Polynomial& operator-(const Polynomial& b); //多項式物件相減運算子多載
    const Polynomial& operator=(const Polynomial& b);//多項式物件等於運算子多載
    float Evaluate(float x);
    friend ostream& operator<<(ostream& a, Polynomial& b);
    friend istream& operator>>(istream& a, Polynomial& B);
private:
    CircularListWithHeader<Term> poly;
```

```
};
ostream& operator<<(ostream& a, Polynomial& b)
{
    CircularListWithHeader<Term>::Iterator bi = b.poly.begin();
    while (bi->exp != -1)
    {
        if (bi->coef > 0 && bi != b.poly.begin())
            a << "+";
        if (!(bi->coef == 1 && bi->exp > 0))
            a << bi->coef;
        if (bi->exp == 1)
            a << "x";
        else if (bi->exp != 0)
            a << "x^" << bi->exp;
        bi++;
    }
    return a;
}
istream& operator>>(istream& a, Polynomial& B)
{
    float b = 0;//輸入的係數
    int c = 0, flag = 0;//輸入的指數次方跟下一項是否為 0 的旗標
        while (1)//一次迴圈代表輸入一個項
        {
            char n = ' ';
            //輸入時略過^符號或判斷是結束了?
            a >> b;
            if (flag)//判斷上一次迴圈最後讀進的是不是負號,是則這次係數為負
            {
                b *= -1;
```

```
151                 flag = 0;//變回預設
152             }
153             a.get(n);//5x^2 的 x 位置，如為換行則此項為 0 次方輸入結束
154             if (n == '\n')
155             {
156                 B.NewTerm(b, 0);
157                 break;
158             }
159             a.get(n); //5x^2 的^位置，如為換行或 + 或 - 號則此項為 1 次方，然後換行則結束， + 或 - 號則表示下一項是正還負
160                 if (n == '\n')
161                 {
162                     B.NewTerm(b, 1);
163                     break;
164                 }
165                 else if (n == '+' || n == '-')
166                 {
167                     B.NewTerm(b, 1);
168                     if (n == '-')
169                         flag = 1;
170                     continue;
171                 }
172             a >> c; //這一項的指數次方
173                 a.get(n); //5x^2 的最後位置，如為換行則結束， - 號則表示下一項是負繼續輸入
174                 B.NewTerm(b, c);
175             if (n == '-') //判斷 5x^2 的最後位置，如為換行則結束， - 號則表示下一項是負繼續輸入
176                 flag = 1;
177             else if (n == '\n')
178             {
179                 break;
180             }
```

```
181         }
182     return a;
183 }
184 Polynomial::~Polynomial() //多項式解構子
185 {
186     poly.~CircularListWithHeader();
187 }
188 Polynomial::Polynomial(const Polynomial& a)
189 {
190     CircularListWithHeader<Term>::Iterator ai = a.poly.begin();
191     CircularListWithHeader<Term>::Iterator set = poly.first();
192     set->exp = -1; //頭設為-1 辨識是否表示跑完一輪了
193     Term temp;
194     while (ai->exp != -1) //跑過全部項
195     {
196         poly.InsertBack(temp.Set(ai->coef, ai->exp)); //放入多項式 A 的每一項
197         ai++;
198     }
199 }
200 const Polynomial& Polynomial::operator=(const Polynomial& b)//直接等於
201 {
202     poly = b.poly;
203     return *this;
204 }
205 Polynomial& Polynomial::operator+(const Polynomial& b)
206 {
207     Term temp;
208     CircularListWithHeader<Term>::Iterator ai = poly.begin(),
209         bi = b.poly.begin();
210         Polynomial c;
```

```
211     while (1)
212     {
213         if (ai->exp == bi->exp) //兩項的次方相同，係數相加
214         {
215             if (ai->exp == -1) return c; //終止條件，表示兩者皆以跑完加總完，可回傳
216             int sum = ai->coef + bi->coef;
217             if (sum)c.poly.InsertBack(temp.Set(sum, ai->exp));
218             ai++;
219             bi++;
220         }
221         else if (ai->exp < bi->exp) //B 的項的次方較大，B 放入 C 多項式內
222         {
223             c.poly.InsertBack(temp.Set(bi->coef, bi->exp));
224             bi++;
225         }
226         else //A 的項的次方較大，A 放入 C 多項式內
227         {
228             c.poly.InsertBack(temp.Set(ai->coef, ai->exp));
229             ai++;
230         }
231     }
232     return c;
233 }
234 Polynomial& Polynomial::operator*(const Polynomial& b)
235 {
236     Term temp;
237     CircularListWithHeader<Term>::Iterator ai = poly.begin(),
238         bi = b.poly.begin();
239     Polynomial c;
240     while (ai->exp != -1) //外迴圈跑所有 A 方程式項
```

```
    {
        bi = b.poly.begin();
        Polynomial d;
        while (bi->exp != -1) //內迴圈跑所有 B 方程式項
        {
            d.poly.InsertBack(temp.Set(ai->coef * bi->coef, ai->exp + bi->exp)); //每次抓 A多項式的一項乘以 B 多項式的所有項加總
                bi++;
        }
        c = c + d;
        ai++;
    }
    return c;
}
Polynomial& Polynomial::operator-(const Polynomial& b) //整體跟加法一樣只差在是減法
{
    Term temp;
    CircularListWithHeader<Term>::Iterator ai = poly.begin(),
        bi = b.poly.begin();
    Polynomial c;
    while (1)
    {
        if (ai->exp == bi->exp)
        {
            if (ai->exp == -1) return c;
            int sum = ai->coef - bi->coef;
            if (sum)c.poly.InsertBack(temp.Set(sum, ai->exp));
            ai++;
            bi++;
        }
        else if (ai->exp < bi->exp)
        {
            c.poly.InsertBack(temp.Set(-bi->coef, bi->exp)); //由於是被剪的項，且 A 多項式沒有相同次方的, 所以乘上 - 1
                bi++;
        }
        else
        {
            c.poly.InsertBack(temp.Set(ai->coef, ai->exp));
            ai++;
        }
    }
    return c;
}
float Polynomial::Evaluate(float x) //多項式代入功能
{
        float sum = 0;
    CircularListWithHeader<Term>::Iterator ai = poly.begin();
    while (ai->exp != -1)
    {
        sum += ai->coef * pow(x, ai->exp); //走訪每個項代入然後總和
        ai++;
    }
    return sum;
}
int main()
{
    Polynomial a, b;
    cout << "輸入 A 多項式:";
    cin >> a;
    cout << "輸入 B 多項式:";
    cin >> b;
    Polynomial c(a);
    cout << "輸出 C(Copy Constructor A 多項式)多項式:";
    cout << c << endl;
    cout << "輸出 A+B 多項式:";
    c = a + b;
    cout << c << endl;
    cout << "輸出 A*B 多項式:";
    c = a * b;
    cout << c << endl;
    cout << "輸出 A-B 多項式:";
    c = a - b;
    cout << c << endl;
    int n;
    cout << "輸入要代入 A 多項式的數字:";
    cin >> n;
    cout << "輸出 A(n)多項式:";
    cout << a.Evaluate(n) << endl;
}
```

# 效能分析：

## 時間複雜度

Polynomial 建構子 T(p)= O(1)

Polynomial Newterm T(p)= O(1)

Polynomial 加法函式 T(p)= O(m+n) (m 表示 a 方程式的項數，n 代表 b 方程式項數)

Polynomial 乘法函式 T(p)=O(mn)(m 代表 A 方程式的項數, n 代表 B 方程 式的項數)

Polynomial 減法函式 T(p)=O(m+n) (m 表示 a 方程式的項數，n 代表 b 方程式項數)

Polynomial 代入 Eval 函式 T(p)=O(n)(n 表示方程式項數)

Polynomial 輸出運算子<<多載 T(p)=O(n)(n 表示方程式的項數)

Class Polynomial 輸入運算子>>多載 T(p)=O(n)(n 表示輸入方程式項數)

Class Polynomial 主程式 T(p)=O(1)

## 空間複雜度

Polynomial 建構子 S(p)= O(1)

Polynomial Newterm S(p)=3

Polynomial 加法函式 S(p)= 5

Polynomial 乘法函式 S(p)= 5+a(a 表示 A 方程式的項數，d 宣告幾次)

Polynomial 減法函式 S(p)= =5(多項式 B、c、ai、bi、temp)

Polynomial 代入 Eval 函式 S(p)= 3(代入數字 f、回傳輸出 sum、跑全部項 的 ai)

Polynomial 輸出運算子<<多載 S(p)= 3(ostream 跟方程式 B、bi(跑所有項))

Class Polynomial 輸入運算子>>多載 S(p)= 5(istream 跟方程式 B 跟 b、c、 flag)+t(t 表示項數，n 被宣告幾次)

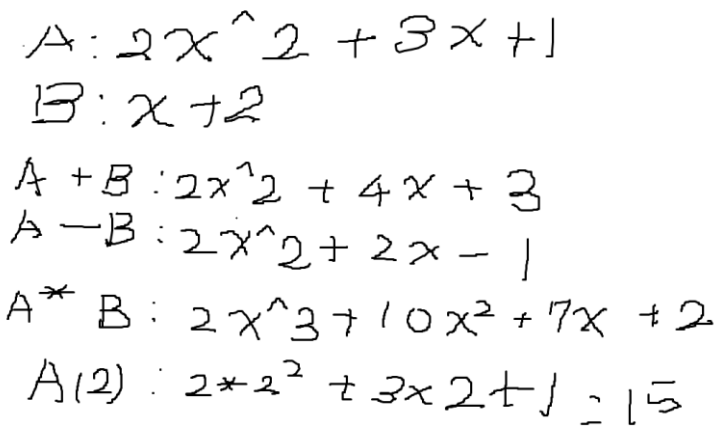
Class Polynomial 主程式 S(p)= 4(方程式 a、b、c 跟代入數字 n)

# 測試與驗證：

## 測試:

```
輸入 A 多項式:2x^2+3x+1
輸入 B 多項式:1x+2
輸出 C(Copy Constructor A 多項式)多項式:2x^2+3x+1
輸出 A+B 多項式:2x^2+4x+3
輸出 A*B 多項式:2x^3+7x^2+7x+2
輸出 A-B 多項式:2x^2+2x-1
輸入要代入 A 多項式的數字:2
輸出 A(n)多項式:15
```

## 驗證:




## 效能量測:

| 識別碼 | 時間 | 配置數 (Diff) | 堆積大小 (差異比對) |
|---|---|---|---|
| 已於 4.66s 啟用原生堆積分析，不包含之前的配置 | | | |
| 1 | 5.90s | 0 (n/a) | 0.00 KB (n/a) |
| 2 | 19.78s | 50 (+50 ↑) | 2.47 KB (+2.47 KB ↑) |

## 程式剛開始(記憶體配置) :5.9

## 程式剛開始(時間) :0s

**程式即將結束(記憶體配置) :50**

**程式即將結束(時間) : 19.78s**

# 心得：

這次的作業不僅讓我更了解環狀串列的使用，也有使用到運算子多載，但我一開始對於運算子多載的使用不太會使用，最後也順利地做出來了。