

1. 解題說明：想法(How to do?)陳述，並舉例說明

這段程式碼建立了一個 Polynomial 類別，用於表示多項式。在類別中，私有部分包含了 termarray (儲存多項式項目)、capacity (目前的陣列容量) 和 terms (項目數量)。公開部分定義了建構子和解構子，建構子初始化容量為 2，項目數為 0，分配對應的動態陣列空間，解構子是釋放原先陣列的記憶體空間。

```
class Polynomial {  
private:  
    Term *termarray;  
    int capacity; // 陣列容量  
    int terms;    // 項數  
  
public:  
    Polynomial() : capacity(2), terms(0) { // 預設空間 2 項數 0  
        termarray = new Term[capacity];  
    }  
    ~Polynomial() {  
        delete[] termarray;  
    }  
};
```

這段程式碼是 addcapacity 函式，用來擴充多項式的儲存空間。當現有的陣列容量不夠再加入新的項目時，就會把原來容量擴增到原先的 2 倍，再把原先陣列的內容複製到新的陣列中。最後釋放舊陣列的記憶體，然後 termarray 再指向新的陣列。

```

void addcapacity() {           //增加新的陣列容量
    capacity *= 2;             //增加原先的 2 倍
    Term *newarray = new Term[capacity];
    for (int i = 0; i < terms; i++) {    //複製原先的元素到新的
        newarray[i] = termarray[i];
    }
    delete[] termarray;         //刪除原先的陣列
    termarray = newarray;       //指向新陣列
}

```

newterm 函式負責將新的項目新增或合併到多項式中。當傳入的係數為 0 時，直接忽略該項目。若係數不為 0，就會逐一檢查陣列中是否有相同次方的項目。如果找到相同次方的項目，將兩者的係數相加，如果合併後的係數變為 0，就把該項目移到下一個項目。如果當前的空間不足以存放新項目，就會呼叫 addcapacity 函式擴充陣列容量至原來的兩倍。最後將新項目的係數和次方存入陣列的下一個可用位置，並更新項目數。

```

void newterm(float coef, int exp) {    //合併多項式
    if (coef == 0) return;             // 忽略係數為 0 的項目
    for (int i = 0; i < terms; i++) {
        if (termarray[i].exp == exp) { //如果次方一樣
            termarray[i].coef += coef; // 把次方一樣的系數加起來
            if (termarray[i].coef == 0) { // 如果合併後係數為 0
                for (int j = i; j < terms - 1; j++) {
                    termarray[j] = termarray[j + 1];
                }
            }
        }
        // 到下一個項目
    }
    terms--; // 項目數-1
}

```

```

    }

    return; // 完成後返回

}

}

if (terms >= capacity) {    // 如果項目>=陣列容量 陣列容量不
夠了

    addcapacity();          // 增加陣列容量

}

termarray[terms].exp = exp;

termarray[terms].coef = coef;

terms++ }

```

add 成員函式負責執行多項式的相加，並將結果回傳。傳入的參數 `const Polynomial& p` 保證多項式 `p` 不會被修改，而函式結尾的 `const` 保證當前物件不會改變。函式開始時建立一個新的 `Polynomial` 物件 `result` 用來存放相加後的多項式。然後再使用兩個 `for` 迴圈，第一個 `for` 迴圈(`p1`)是在做當前多項式的係數和次方都加入到 `result`，用 `newterm` 是確保有正確的合併或新增，下一個 `for` 迴圈(`p2`)跟剛剛是一樣的但是是換成第二個多項式傳入 `result` 中去和 `p1` 做合併，最後回傳相加後的結果 `result`。

```

Polynomial add(const Polynomial& p) const { //相加

    Polynomial result;                      // 建立一個新對象

    for (int i = 0; i < terms; i++) {

        result.newterm(termarray[i].coef, termarray[i].exp); // 複製
當前的多項式

    }

    for (int i = 0; i < p.terms; i++) {

        result.newterm(p.termarray[i].coef, p.termarray[i].exp); //
加入另一個多項式到 newterm 合併

    } return result; //把答案回傳回去    }

```

mult 成員函式負責執行多項式的相乘，並將結果回傳。傳入的參數 const Polynomial& p 保證多項式 p 不會被修改，而函式結尾的 const 保證當前物件不會改變。函式開始時，建立一個新的 Polynomial 物件 result，用來存放相乘後的多項式。2 個 for 迴圈都是從 0 開始做，但是要小於 terms，for 迴圈內容先宣告新的係數，存放 2 組的多項式的係數相乘，在宣告一個新的次方，存放 2 組多項式的次方相加，最後把新的係數和新的次方傳到 newterm 去合併多項式，最後回傳 result 就是兩個多項式相乘的結果。

```
Polynomial mult(const Polynomial& p) const {    // 相乘

    Polynomial result;                        // 建立一個新對象

    for (int i = 0; i < terms; i++) {

        for (int j = 0; j < p.terms; j++) {

            float newcoef = termarray[i].coef * p.termarray[j].coef;
            // 係數相乘

            int newexp = termarray[i].exp + p.termarray[j].exp;
            // 指數相加

            result.newterm(newcoef, newexp);    // 新多項式到
                                                newterm 合併

        }

    }

    return result;//把答案回傳回去

}
```

eval 函式的作用是将傳入的值 f 代入多項式(pl)中進行計算。函式內部首先宣告一個 double 型別的 result 來儲存運算結果，並初始化為 0.0。接著使用 for 迴圈一項一項的把 f 代入(pl)的 x 中做計算，將結果累加到 result 中。最後回傳 result 就是計算後的總和。

```

double eval(double f) { // 帶數字進去
    double result = 0.0f;
    for (int i = 0; i < terms; i++) {
        result += termarray[i].coef * pow(f, termarray[i].exp); //
        把傳入的數字帶到多項式中 運算每個 x 結果加起來
    }
    return result; // 把答案回傳回去
}

```

input 函式用來接收輸入多項式的各個項。首先寫一個 while 迴圈執行輸入的部分，先宣告係數、次方、字元("[^]、**x**")，首先輸入係數，再輸入字元，如果字元是"**x**"的話在看下一個輸入是"[^]"嗎，如果是再輸入次方，如果有"**x**"但沒有"[^]"代表次方=1，然後如果連字元"**x**"都沒有，代表是常數，所以次方=0，輸入完後把剛剛輸入的係數和次方傳到 newterm 函式進行多項式項目的合併。最後當遇到換行符號時輸入過程結束。

```

void input() { // 輸入多項式
    while (true) {
        float coef;
        int exp;
        char a, b;
        cin >> coef; // 輸入係數
        cin.get(a); // 輸入 'x' 或其他字元
        if (a == 'x') { // 如果是 'x'
            cin.get(b); // 輸入 '^' 或其他字元
            if (b == '^') {
                cin >> exp; // 如果是 '^' 輸入次方
            } else {
                exp = 1; // 如果有 'x' 但沒有 '^' 次方就是 1
            }
        }
    }
}

```

```

    }
} else {
    exp = 0; // 如果沒有 'x' 指數= 0
}

newterm(coef, exp); //把係數指數傳到 newterm 去合併

if (cin.peek() == '\n') break; // 遇到換行符結束輸入
}
}

```

output 函式用來輸出多項式的函式。首先，如果多項式的項數為 0，則直接輸出 "0" 來表示沒有多項式。再來對於每個項目，會先檢查係數是否大於 0，如果是且不是第一項，則輸出 "+" 來連接項目。然後輸出係數值，如果次方大於 0，會顯示 "x"，如果次方大於 1，則會加上 "^" 和對應的次方數字。所有項目處理完後，輸出換行符號。

```

void output() const { //輸出多項式

    if (terms == 0) cout << "0"; //如果項目=0 代表沒有多項式顯示 0
    else {
        for (int i = 0; i < terms; i++) {
            if (i > 0 && termarray[i].coef > 0) { //如果係數和項數
                //都>0 代表多項式還沒顯示完要輸出 '+'
                cout << "+";
            }

            cout << termarray[i].coef; //輸出係數

            if (termarray[i].exp > 0) { //次方>0 就輸出 'x'
                cout << "x";
            }

            if (termarray[i].exp > 1) { //次方>1 就輸出 '^' 和次方

```

```

        cout << "^" << termarray[i].exp;
    }
}
}

cout << endl;
}

```

主程式開始時要建立計算時間的，因為計算效能量測需要用到做某一段程式需要多少時間 start, finish 計算的開始和結束，再創建兩個多項式物件 p1 和 p2，並讓使用者輸入多項式的係數和次方。接著可以在 p1 多項式中新增一個項目，這個新增的項目會通過 newterm 函式加入，並在更新後顯示新的 p1 多項式，之後程式會做兩個多項式運算。第一個是 p1 和 p2 的相加 (add 函式)，並計算這個加法操作所需的時間。第二個是 p1 和 p2 的相乘 (mult 函式)，並同樣計算乘法所需的時間。在每次加法和乘法操作後，程式會輸出結果還要顯示運算所需的時間。最後 eval 函式要使用者輸入一個 f，並把 f 帶入 p1 多項式中計算，最後輸出 p1(f) 的結果，這裡也是一樣要計算帶值進去多程式做運算所需的時間。整個過程中的每個操作 (add、mult、eval) 都會測量並顯示執行時間，幫助分析多項式運算的效能。

```

int main() {
    clock_t start, finish; // 計算時間的

    Polynomial p1, p2; // 2 個多項式

    cout << "p1="; // 3x^2+2x+1
    p1.input();

    cout << "p2="; // 2x^3+4
    p2.input();

    cout << "新增新的項目" << endl;

    cout << "項目的係數:";

    int exp;
}

```

```

float coef;

cin >> coef;           // 5

cout << "項目的指數:";

cin >> exp;             // 2

p1.newterm(coef, exp);  //把新加的項目加到 p1 中


cout << "\n 更新後的 p1:";

p1.outtput();           //  $8x^2+2x+1$ 


cout << "\nadd" << endl;

start=clock();           //計算時間開始

Polynomial sum = p1.add(p2);  //p1+p2

cout << "p1+p2=";

sum.outtput();           //  $8x^2+2x+5+2x^3$ 

finish=clock();          //計算時間結束

cout << "add() 需時: " << (double)(finish - start) /
CLOCKS_PER_SEC << " s" << endl; //計算 add 所需的時間


cout << "\nmult" << endl;

cout << "p1*p2=";

start=clock();           //計算時間開始

Polynomial xx = p1.mult(p2);  //p1*p2

xx.outtput();            //  $16x^5+32x^2+4x^4+8x+2x^3+4$ 

finish=clock();          //計算時間結束

cout << "mult() 需時: " << (double)(finish - start) /
CLOCKS_PER_SEC << " s" << endl; //計算 mult 所需的時間

```



```

double f = 0.0f;

cout << "\neval" << endl;

cout << "f 的值:";           //輸入要帶進 p1 中的值

start = clock();             //計算時間開始

cin >> f;                     //1

cout << "p1(" << f << ")=" << p1.eval(f) << endl;    // 11

finish = clock();             //計算時間結束

    cout << "eval() 需時: " << (double)(finish - start) /
CLOCKS_PER_SEC << " s" << endl; //計算 eval 所需的時間

return 0;

}

```

2. Algorithm Design & Programming

```

1  #include <iostream>
2  #include <cmath>
3  #include <ctime>
4  #include <iomanip>
5  using namespace std;
6
7  class Term {
8      friend class Polynomial;
9  private:
10     float coef; // 係數
11     int exp;    // 次方
12 };
13
14 class Polynomial {
15 private:
16     Term *termarray;
17     int capacity; // 陣列容量
18     int terms;    // 項數
19
20 public:
21     Polynomial() : capacity(2), terms(0) { // 預設陣列容量2 項數0
22         termarray = new Term[capacity];
23     }
24     ~Polynomial() {
25         delete[] termarray;
26     }
27
28     void addcapacity() { // 增加新的陣列容量
29         capacity *= 2; // 增加原先的2倍
30         Term *newarray = new Term[capacity];
31         for (int i = 0; i < terms; i++) { // 複製舊有的元素到新的那裏

```

```

28 void addcapacity() { //增加新的陣列容量
29     capacity *= 2; //增加原先的2倍
30     Term *newarray = new Term[capacity];
31     for (int i = 0; i < terms; i++) { //複製原先的元素到新的那裏
32         newarray[i] = termarray[i];
33     }
34     delete[] termarray; //刪除原先的陣列
35     termarray = newarray; //指向新陣列
36 }
37
38 void newterm(float coef, int exp) { //合併多項式
39     if (coef == 0) return; //忽略係數為0的項目
40     for (int i = 0; i < terms; i++) {
41         if (termarray[i].exp == exp) { //如果次方一樣
42             termarray[i].coef += coef; //把次方一樣的係數加起來
43             if (termarray[i].coef == 0) { //如果合併後係數為0
44                 for (int j = i; j < terms - 1; j++) {
45                     termarray[j] = termarray[j + 1]; //到下一個項目
46                 }
47                 terms--; //項目數-1
48             }
49             return; //完成後返回
50         }
51     }
52     if (terms >= capacity) { //如果項目>=陣列容量 陣列容量不夠了
53         addcapacity(); //增加陣列容量
54     }
55     termarray[terms].exp = exp;
56     termarray[terms].coef = coef;
57     terms++;

```

```

58 }
59
60 Polynomial add(const Polynomial& p) const { //相加
61     Polynomial result; //建立一個新對象
62     for (int i = 0; i < terms; i++) {
63         result.newterm(termarray[i].coef, termarray[i].exp); //複製當前的多項式
64     }
65     for (int i = 0; i < p.terms; i++) {
66         result.newterm(p.termarray[i].coef, p.termarray[i].exp); //加入另一個多項式到newterm合併
67     }
68     return result; //把答案回傳回去
69 }
70
71
72 Polynomial mult(const Polynomial& p) const { //相乘
73     Polynomial result; //建立一個新對象
74     for (int i = 0; i < terms; i++) {
75         for (int j = 0; j < p.terms; j++) {
76             float newcoef = termarray[i].coef * p.termarray[j].coef; //係數相乘
77             int newexp = termarray[i].exp + p.termarray[j].exp; //指數相加
78             result.newterm(newcoef, newexp); //新多項式到newterm合併
79         }
80     }
81     return result; //把答案回傳回去
82 }
83
84 double eval(double f) { //帶數字進去
85     double result = 0.0f;
86     for (int i = 0; i < terms; i++) {
87         result += termarray[i].coef * pow(f, termarray[i].exp); //把傳入的數字帶到多項式中 運算每個結果加起來
88     }
89     return result; //把答案回傳回去
90 }
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115

```

```

85 double result = 0.0f;
86 for (int i = 0; i < terms; i++) {
87     result += termarray[i].coef * pow(f, termarray[i].exp); //把傳入的數字帶到多項式中 運算每個結果加起來
88 }
89 return result; //把答案回傳回去
90 }
91
92 void input() { //輸入多項式
93     while (true) {
94         float coef;
95         int exp;
96         char a, b;
97         cin >> coef; //輸入係數
98         cin.get(a); //輸入 'x' 或其他字元
99         if (a == 'x') { //如果是 'x'
100             cin.get(b); //輸入 '^' 或其他字元
101             if (b == '^') {
102                 cin >> exp; //如果是 '^' 輸入次方
103             } else {
104                 exp = 1; //如果有 'x' 但沒有 '^' 次方就是1
105             }
106         } else {
107             exp = 0; //如果沒有 'x' 指數= 0
108         }
109         newterm(coef, exp); //把係數指數傳到newterm去合併
110         if (cin.peek() == '\n') break; //遇到換行符結束輸入
111     }
112 }
113
114 void output() const { //輸出多項式
115     if (terms == 0) cout << "0" << endl; //如果項目=0 輸出沒有多項式顯示

```

```

112     }
113
114     void output() const { //輸出多項式
115         if (terms == 0) cout << "0"; //如果項目=0代表沒有多項式顯示0
116         else {
117             for (int i = 0; i < terms; i++) {
118                 if (i > 0 && termarray[i].coef > 0) { //如果係數和項數都>0代表多項式還沒顯示完要輸出 '+'
119                     cout << "+";
120                 }
121                 cout << termarray[i].coef; //輸出係數
122                 if (termarray[i].exp > 0) { //次方>0就輸出 'x'
123                     cout << "x";
124                 }
125                 if (termarray[i].exp > 1) { //次方>1就輸出 '^' 和次方
126                     cout << "^" << termarray[i].exp;
127                 }
128             }
129             cout << endl;
130         }
131     }
132 };
133
134 int main() {
135     clock_t start, finish; //計算時間的
136     Polynomial p1, p2; //2個多項式
137     cout << "p1="; // 3x^2+2x+1
138     p1.input();
139     cout << "p2="; // 2x^3+4
140     p2.input();
141     cout << "新增新的項目" << endl;
142 }

```

```

139     cout << "p2="; // 2x^3+4
140     p2.input();
141
142     cout << "新增新的項目" << endl;
143     cout << "項目的係數:";
144     int exp;
145     float coef;
146     cin >> coef; // 5
147     cout << "項目的指數:";
148     cin >> exp; // 2
149     p1.newterm(coef, exp); //把新加的項目加到p1中
150
151     cout << "\n更新後的p1:";
152     p1.output(); // 8x^2+2x+1
153
154     cout << "\nadd" << endl;
155     start=clock(); //計算時間開始
156     Polynomial sum = p1.add(p2); //p1+p2
157     cout << "p1+p2=";
158     sum.output(); // 8x^2+2x+5+2x^3
159     finish=clock(); //計算時間結束
160     cout << "add() 需時: " << (double)(finish - start) / CLOCKS_PER_SEC << " s" << endl; //計算add所需的時間
161
162     cout << "\nmult" << endl;
163     cout << "p1*p2=";
164     start=clock(); //計算時間開始
165     Polynomial xx = p1.mult(p2); //p1*p2
166     xx.output(); // 16x^5+32x^2+4x^4+8x+2x^3+4
167     finish=clock(); //計算時間結束
168     cout << "mult() 需時: " << (double)(finish - start) / CLOCKS_PER_SEC << " s" << endl; //計算mult所需的時間
169
170
171     double f = 0.0f;
172     cout << "\neval" << endl;
173     cout << "f的值:"; //輸入要帶進p1中的值
174     start = clock(); //計算時間開始
175     cin >> f; //1
176     cout << "p1(" << f << ")=" << p1.eval(f) << endl; // 11
177     finish = clock(); //計算時間結束
178     cout << "eval() 需時: " << (double)(finish - start) / CLOCKS_PER_SEC << " s" << endl; //計算eval所需的時間
179     return 0;
180 }

```

3. 效能分析 Time complexity & Space complexity

時間複雜度：

- newterm：檢查和新增項目最壞 $O(n)$ 。
- add：處理兩個多項式，最壞 $O(n_1^2 + n_2^2)$ 。
- mult：項數相乘後合併，最壞 $O(n_1 \times n_2)$ 。
- eval：逐項計算，複雜度 $O(n)$ 。

空間複雜度：

- 動態陣列存儲多項式項數 $O(n)$ 。
- add 結果需要 $O(n_1 + n_2)$ 空間。
- mult 結果需要 $O(n_1 \times n_2)$ 空間。

4. 測試與驗證

Input:

$3x^2 + 2x + 1$

$2x^3 + 4$

5

2

1

Output:

```
C:\Users\Lenovo\OneDrive\Desktop\Untitled1.exe
p1=3x^2+2x+1
p2=2x^3+4
新增新的項目
項目的系數:5
項目的指數:2
更新後的p1:8x^2+2x+1
add
p1+p2=8x^2+2x+5+2x^3
add() 需時: 0.002 s
mult
p1*p2=16x^5+32x^2+4x^4+8x+2x^3+4
mult() 需時: 0.001 s
eval
f的值:1
p1(1)=11
eval() 需時: 2.451 s
-----
Process exited after 12.72 seconds with return value 0
請按任意鍵繼續 . . .
```

5. 效能量測

Add 相加的過程用了 0.002 秒

Mult 相乘的過程用了 0.001 秒

Eval 把值帶進 p1 運算的過程用了 2.451 秒

6. 心得討論

這次撰寫程式作業是我目前寫過最長且最具挑戰性的一次經歷。相較於一年級時，老師在教比較困難的程式時，不會要求我們寫完完整的程式，只寫部分的程式，這次卻需要獨立撰寫整段完整的程式，過程中一直出現錯誤，然後在多次的調整與修改。透過這次的作業，我學到了許多新的程式設計技巧。其中一個重要的學習是 `const` 的使用。`const` 可以用來保證函式在執行時不會改變傳入的多項式物件，像是 `p1` 和 `p2` 這樣的多項式即使有多次運算，也能保持原本的多項式不被修改。在設計輸入和輸出的程式邏輯也是一個思考的點。例如，在輸入多項式時，需要特別處理字元 "`x`" 和次方的輸入。當偵測到 "`x`" 時，還需要檢查是否有次方符號 "`^`"，如果沒有默認次方為 1，如果有 "`^`" 還要在輸入次方。這些細節讓我更加了解如何處理混合數字與字元的輸入邏輯。另一個收穫是效能量測的學習。我學會了記錄程式運行的開始與結束時間，並透過 `CLOCKS_PER_SEC` 將時間換算成秒來輸出。這讓我對程式效能的分析有了初步的概念。整體來說這次作業對我的程式設計能力有很大的提升，特別是在類別與函式的設計、邏輯的處理，以及效能測試方面。這次作業的經驗對於在未來撰寫更長、更複雜的專題或實際工作時，會帶來很大的幫助，也讓我在程式設計上有了一大的進步。