
# 資料結構

# Data Structure

# Lab 03

姓名：曾致嘉

學號：113AB0014

**Lab03-Ex1**

**According to the lecture slides, three polynomial representations are discussed. Please complete the functions 'Add', 'Mult', 'Eval', and output based on the required print format using ^ to denote exponents.**

**Code**

```
#include <iostream>
#include <array>
#include <string>
#include <cmath>
using namespace std;

const int MAX_TERMS = 10; // 設定最大項數

struct Term {
    double coef; // 係數
    int exp;     // 冪次
};

class Polynomial {
private:
    array<Term, MAX_TERMS> terms;
    int termCount;

    void ParsePoly(const string& poly) {
        termCount = 0;
        int pos = 0;
        while (pos < poly.length() && termCount < MAX_TERMS) {
            size_t nextPos = poly.find_first_of("+-", pos + 2);// 找到下一個符號的位置
            string str_num = poly.substr(pos + 2, nextPos - pos - 3);// 取得係數+次方的字串
            bool str_sign = (poly.at(pos) == '-');// 判斷正負號
            pos = nextPos;// 更新位置
            double coef = 1;
            int exp = 0;
            size_t xPos = str_num.find("X");// 以 X 為分界分開係數與冪次
            if (xPos != string::npos) {// 存在 X 的情況
```

```
                    if (xPos > 0) coef = stod(str_num.substr(0, xPos));
                    size_t expPos = str_num.find("^");// 找到冪次
                    if (expPos != string::npos) exp = stoi(str_num.substr(expPos +
1));// 取得冪次
                    else exp = 1;// 為一次的情況
                } else {
                    coef = stod(str_num);// 常數
                }
                if (str_sign) coef = -coef;// 處理負號
                terms[termCount++] = {coef, exp};
            }
        }

public:
    // 建構函式：從字串初始化多項式
    Polynomial(const string& poly) {
        ParsePoly(poly);
    }
    // 多項式相加
    Polynomial Add_Poly(const Polynomial& other) {
        Polynomial result("");
        result.termCount = 0;
        int i = 0, j = 0;
        while (i < termCount && j < other.termCount) {
            if (terms[i].exp > other.terms[j].exp) {//比較冪次大小，冪次大的先
放入，第一個多項式的冪次較大
                result.terms[result.termCount++] = terms[i++];
            } else if (terms[i].exp < other.terms[j].exp) {//比較冪次大小，冪次
大的先放入，第二個多項式的冪次較大
                result.terms[result.termCount++] = other.terms[j++];
            } else {// 冪次相同，係數相加
                double new_coef = terms[i].coef + other.terms[j].coef;
                if (new_coef != 0) {
                    result.terms[result.termCount++] = {new_coef,
terms[i].exp};
                }
                i++, j++;
            }
```

```
            }
            while (i < termCount) result.terms[result.termCount++] = terms[i++];
            while (j < other.termCount) result.terms[result.termCount++] =
other.terms[j++];
            return result;
      }

      // 多項式相乘
      Polynomial mult_Poly(const Polynomial& other) {
            Polynomial result("");
            result.termCount = 0;
            for (int i = 0; i < termCount; i++) {
                  for (int j = 0; j < other.termCount; j++) {
                        int exp = terms[i].exp + other.terms[j].exp;//冪次相加
                        double coef = terms[i].coef * other.terms[j].coef;//係數相乘
                        bool found = false;
                        for (int k = 0; k < result.termCount; k++) {
                              if (result.terms[k].exp == exp) {
                                    result.terms[k].coef += coef;
                                    found = true;//找到相同冪次，係數相加
                                    break;
                              }
                        }
                        if (!found && result.termCount < MAX_TERMS) {
                              result.terms[result.termCount++] = {coef, exp};
                        }
                  }
            }
            return result;
      }

      // 計算多項式值
      double eval_Poly(int x) {
            double result = 0;
            for (int i = 0; i < termCount; i++) {
                  result += terms[i].coef * pow(x, terms[i].exp);
            }
            return result;
```

```
    }

    // 輸出多項式
    void printPoly() {
        if (termCount == 0) {
            cout << "0";
            return;
        }
        for (int i = 0; i < termCount; i++) {
            if (terms[i].coef > 0 && i != 0) cout << "+ ";
            cout << terms[i].coef;
            if (terms[i].exp != 0) cout << "X^" << terms[i].exp << " ";
        }
        cout << endl;
    }
};

int main() {
    string poly1, poly2;
    getline(cin, poly1);
    getline(cin, poly2);

    // 處理首項正號省略的情況
    if (poly1.at(0) != '-') poly1 = "+ " + poly1;
    if (poly2.at(0) != '-') poly2 = "+ " + poly2;

    // 建立 Polynomial 物件
    Polynomial term1(poly1);
    Polynomial term2(poly2);

    // 計算、輸出相加後結果
    Polynomial add_terms = term1.Add_Poly(term2);
    cout << "Addition: ";
    add_terms.printPoly();

    // 計算、輸出相乘後結果
    Polynomial mult_terms = term1.mult_Poly(term2);
    cout << "Multiplication: ";
```

```
    mult_terms.printPoly();

    // 帶入 x 計算
    int x;
    cout << "Input x: ";
    cin >> x;
    cout << "Eval1: " << term1.eval_Poly(x) << endl;
    cout << "Eval2: " << term2.eval_Poly(x) << endl;

    return 0;
}
```

```
#include <iostream>
#include <vector>
#include <string>
#include <cmath>
using namespace std;

struct Term {
    double coef; // 係數
    int exp;     // 冪次
};

class Polynomial {
private:
    vector<Term> terms;

    // 解析多項式字串
    vector<Term> ParsePoly(const string& poly) {
        vector<Term> parsedTerms;
        int pos = 0;

        while (pos < poly.length()) {
            size_t nextPos = poly.find_first_of("+-", pos + 2); // 找到下一個符號的位置
            string str_num = poly.substr(pos + 2, nextPos - pos - 3); // 取得係數+
```

```
次方的字串
                bool str_sign = (poly.at(pos) == '-'); // 判斷正負號
                pos = nextPos; // 更新位置
                double coef = 1;
                int exp = 0;
                size_t xPos = str_num.find("X"); // 以 X 為分界分開係數與冪次
                if (xPos != string::npos) { // 存在 X 的情況
                    if (xPos > 0) coef = stod(str_num.substr(0, xPos));
                    size_t expPos = str_num.find("^"); // 找到冪次
                    if (expPos != string::npos) exp = stoi(str_num.substr(expPos +
1)); // 取得冪次
                    else exp = 1; // 為一次的情況
                } else {
                    coef = stod(str_num); // 常數
                }
                if (str_sign) coef = -coef; // 處理負號
                parsedTerms.push_back({coef, exp});
            }
            return parsedTerms;
        }

public:
    // 建構函式：從字串初始化多項式
    Polynomial(const string& poly) {
        terms = ParsePoly(poly);
    }

    // 多項式相加
    Polynomial Add_Poly(const Polynomial& other) {
        vector<Term> result;
        int index_1 = 0, index_2 = 0;
        while (index_1 < terms.size() && index_2 < other.terms.size()) {
            if (terms[index_1].exp > other.terms[index_2].exp) {//比較冪次大
小，冪次大的先放入，第一個多項式的冪次較大
                result.push_back(terms[index_1++]);
            } else if (terms[index_1].exp < other.terms[index_2].exp) {//比較冪
次大小，冪次大的先放入，第二個多項式的冪次較大
                result.push_back(other.terms[index_2++]);
```

```
            } else { // 冪次相同，係數相加
                double new_coef = terms[index_1].coef +
other.terms[index_2].coef;
                if (new_coef != 0) result.push_back({new_coef,
terms[index_1].exp});
                index_1++;
                index_2++;
            }
        }
        while (index_1 < terms.size()) result.push_back(terms[index_1++]);
        while (index_2 < other.terms.size())
result.push_back(other.terms[index_2++]);

        return Polynomial(result);
    }

    // 多項式相乘
    Polynomial mult_Poly(const Polynomial& other) {
        vector<Term> result;
        for (int i = 0; i < terms.size(); i++) {
            for (int j = 0; j < other.terms.size(); j++) {
                int exp = terms[i].exp + other.terms[j].exp;
                double coef = terms[i].coef * other.terms[j].coef;
                bool found = false;
                for (int k = 0; k < result.size(); k++) {
                    if (result[k].exp == exp) {
                        result[k].coef += coef;
                        found = true;
                        break;
                    }
                }
                if (!found) {
                    result.push_back({coef, exp});
                }
            }
        }
        return Polynomial(result);
    }
```

```
    // 計算多項式值
    double eval_Poly(int x) {
        double result = 0;
        for (const Term& term : terms) {
            result += term.coef * pow(x, term.exp);
        }
        return result;
    }

    // 輸出多項式
    void printPoly() {
        if (terms.empty()) {
            cout << "0";
            return;
        }
        for (int i = 0; i < terms.size(); i++) {
            if (terms[i].coef > 0 && i != 0) cout << "+ ";
            cout << terms[i].coef;
            if (terms[i].exp != 0) cout << "X^" << terms[i].exp << " ";
        }
        cout << endl;
    }

    // 建構函式：從 vector<Term> 直接初始化
    Polynomial(const vector<Term>& newTerms) {
        terms = newTerms;
    }
};

int main() {
    string poly1, poly2;
    getline(cin, poly1);
    getline(cin, poly2);

    // 處理首項正號省略的情況
    if (poly1.at(0) != '-') poly1 = "+ " + poly1;
    if (poly2.at(0) != '-') poly2 = "+ " + poly2;
```

```
    // 建立 Polynomial 物件
    Polynomial term1(poly1);
    Polynomial term2(poly2);

    // 計算、輸出相加後結果
    Polynomial add_terms = term1.Add_Poly(term2);
    cout << "Addition: ";
    add_terms.printPoly();

    // 計算、輸出相乘後結果
    Polynomial mult_terms = term1.mult_Poly(term2);
    cout << "Multiplication: ";
    mult_terms.printPoly();

    // 帶入 x 計算
    int x;
    cout << "Input x: ";
    cin >> x;
    cout << "Eval1: " << term1.eval_Poly(x) << endl;
    cout << "Eval2: " << term2.eval_Poly(x) << endl;

    return 0;
}
```

**Discussion Section**

通常靜態的程式碼基本上就是將動態原本使用 vector 改成用 array 實現，由於 C++的 array 必須在宣告前先規劃出大小，導致靜態式有項數的上限。時間複雜度大致相等