國立臺北科技大學
National Taipei University of Technology

資料結構

Data Structure

Lab 03

姓名：曾致嘉

學號：113AB0014

// 解析多項式字串

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 <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

通常靜態的程式碼相較於動態的程式碼，有較低的複雜度，但相對的稀疏處理的部分較弱。

Lab01-Ex2. 1480
Add clear comments to the program to describe its actions and functions effectively.
Code

```
class Solution {
    public:
    vector<int> runningSum(vector<int>& nums) {
        int n = nums.size(); // 取得陣列長度
        vector<int> result(n);  // 建立一個大小為 n 的 vector
        result[0]=nums[0]; // 第一個元素直接存入

        // 第二個元素之後，為前項累計
        for(int i=1; i<n; i++){
            result[i]=result[i-1]+nums[i];
        }

        // 回傳答案
        return result;
    }
};
```

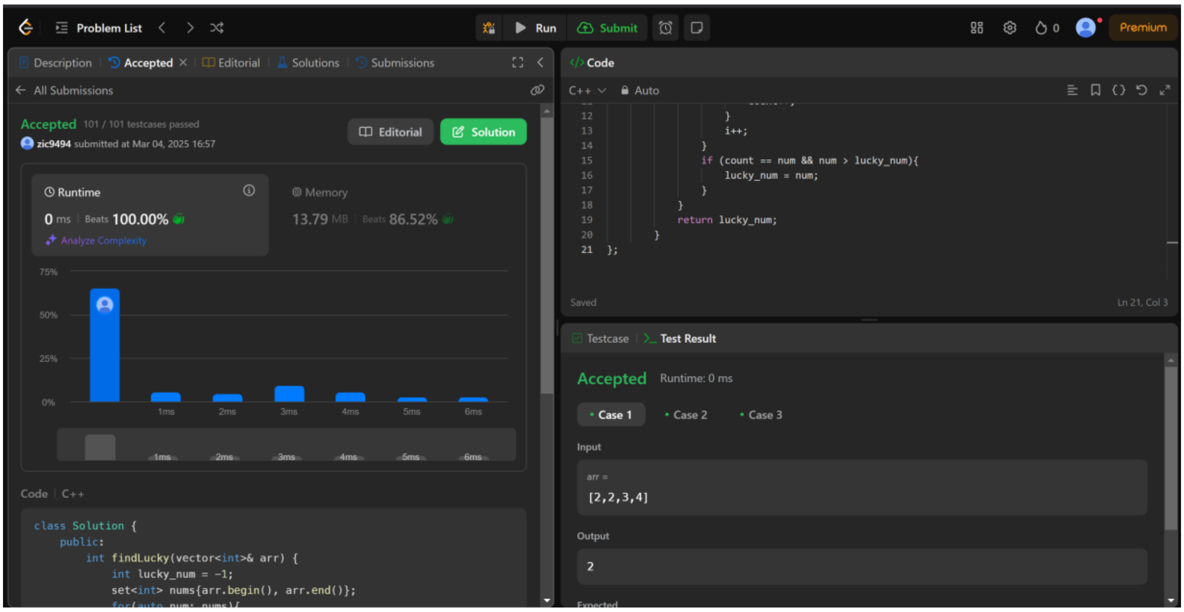
Lab01-Q1. 1394
the largest number where the digit appears as many times as its value.
Code

```
class Solution {
    public:
        int findLucky(vector<int>& arr) {
            int lucky_num = -1; // 設定初始值 1
            set<int> nums{arr.begin(), arr.end()}; // 將 arr 轉換為 set，一次插入 set
是 logn= +nlogn

            for(auto num: nums){
                // 指派變數*2 = +2n
                int i = 0;
                int count =0;
                while(i<arr.size()){
                    // 迴圈判斷*1、i 遞增*2 = 3n^2
                    if(arr[i] == num){
                        count++;
                    }
                    i++;
                }

                //判斷*3、指派*1 = +4n
                if (count == num && num > lucky_num){
                    lucky_num = num;
                }
            }
            //返回結果*1 = +1
            return lucky_num;
            //f(n) = nlogn + 2n + 3n^2 + 4n + 1 = 3n^2 + 4n + nlogn + 2 => O(n^2)
        }
};
```

Result



Discussion

3. The time complexity is known to be O(n^2) from the comments in the code.