# 解題想法：

這次要實現一個用圓形鏈結與標頭節點表示的一元多項式類。多項式的每一項都由一個節點表示，節點包含係數 (coef)、指數 (exp) 和指向下一個節點的指針 (link)。

# 演算法設計與程式：

#include <iostream>

#include <cmath>

#include <stdexcept>

using namespace std;

struct Node {

int coef; // 項的係數

int exp; // 項的指數

Node\* link; // 指向下一個節點的指針

Node(int c = 0, int e = 0, Node\* l = nullptr) : coef(c), exp(e), link(l) {}

};

class Polynomial {

private:

Node\* header;

void deleteAll() {

Node\* current = header->link;

while (current != header) {

Node\* nextNode = current->link;

delete current;

current = nextNode;

}

}

void copyFrom(const Polynomial& a) {

Node\* current = a.header->link;

Node\* last = header;

while (current != a.header) {

last->link = new Node(current->coef, current->exp);

last = last->link;

current = current->link;

}

last->link = header;

}

public:

Polynomial() {

header = new Node(); // 初始化標頭節點

header->link = header;

}

Polynomial(const Polynomial& a) {

header = new Node(); // 初始化標頭節點

header->link = header;

copyFrom(a);

}

const Polynomial& operator=(const Polynomial& a) {

if (this != &a) {

deleteAll();

copyFrom(a);

}

return \*this;

}

~Polynomial() {

deleteAll();

delete header;

}

friend istream& operator>>(istream& is, Polynomial& x);

friend ostream& operator<<(ostream& os, const Polynomial& x);

Polynomial operator+(const Polynomial& b) const {

Polynomial result;

Node\* current = header->link;

while (current != header) {

result.addTerm(current->coef, current->exp);

current = current->link;

}

current = b.header->link;

while (current != b.header) {

result.addTerm(current->coef, current->exp);

current = current->link;

}

return result;

}

Polynomial operator-(const Polynomial& b) const {

Polynomial result;

Node\* current = header->link;

while (current != header) {

result.addTerm(current->coef, current->exp);

current = current->link;

}

current = b.header->link;

while (current != b.header) {

result.addTerm(-current->coef, current->exp);

current = current->link;

}

return result;

}

Polynomial operator\*(const Polynomial& b) const {

Polynomial result;

Node\* current1 = header->link;

while (current1 != header) {

Node\* current2 = b.header->link;

while (current2 != b.header) {

int newCoef = current1->coef \* current2->coef;

int newExp = current1->exp + current2->exp;

result.addTerm(newCoef, newExp);

current2 = current2->link;

}

current1 = current1->link;

}

return result;

}

float Evaluate(float x) const {

float result = 0.0;

Node\* current = header->link;

while (current != header) {

result += current->coef \* std::pow(x, current->exp);

current = current->link;

}

return result;

}

void addTerm(int coef, int exp) {

if (coef == 0) return; // 忽略零係數

Node\* newNode = new Node(coef, exp);

Node\* current = header;

while (current->link != header && current->link->exp > exp) {

current = current->link;

}

if (current->link != header && current->link->exp == exp) {

current->link->coef += coef;

if (current->link->coef == 0) {

// 移除係數為零的項

Node\* temp = current->link;

current->link = current->link->link;

delete temp;

}

delete newNode;

} else {

newNode->link = current->link;

current->link = newNode;

}

}

};

istream& operator>>(istream& is, Polynomial& x) {

int coef, exp;

char op;

while (is >> coef >> op >> exp) {

if (op != 'x') {

throw std::invalid\_argument("無效的輸入格式");

}

x.addTerm(coef, exp);

if (is.peek() == '\n' || is.peek() == EOF) {

break;

}

}

return is;

}

ostream& operator<<(ostream& os, const Polynomial& x) {

Node\* current = x.header->link;

if (current == x.header) {

os << "0";

return os;

}

bool first = true;

while (current != x.header) {

if (current->coef != 0) {

if (!first && current->coef > 0) {

os << " + ";

}

if (current->coef < 0) {

os << " - ";

if (current->coef != -1 || current->exp == 0) {

os << -current->coef;

}

} else if (current->coef != 1 || current->exp == 0) {

os << current->coef;

}

if (current->exp > 0) {

os << "x";

if (current->exp > 1) {

os << "^" << current->exp;

}

}

first = false;

}

current = current->link;

}

return os;

}

int main() {

Polynomial p1, p2;

cout << "請輸入第一個多項式 (格式: 係數 x 指數): ";

cin >> p1;

cout << "請輸入第二個多項式 (格式: 係數 x 指數): ";

cin >> p2;

cout << "多項式 1: " << p1 << endl;

cout << "多項式 2: " << p2 << endl;

Polynomial sum = p1 + p2;

Polynomial difference = p1 - p2;

Polynomial product = p1 \* p2;

cout << "和: " << sum << endl;

cout << "差: " << difference << endl;

cout << "積: " << product << endl;

float x;

cout << "請輸入 x 的值來評估第一個多項式: ";

cin >> x;

cout << "p1 在 x = " << x << 時的值: " << p1.Evaluate(x) << endl;

return 0;}

# 時間複雜度與空間複雜度

### 時間複雜度：

**加法與減法**：O(m + n)

**乘法**：O(m \* n)

**整體評估：**O(m)

### 時間複雜度：

**加法與減法**：O(m + n)

**乘法**：O(m \* n)

**整體評估：**O(m)

# 心得

這次所使用的圓形鏈結讓我覺得非常的困難，雖然在課堂上理論理解起來感覺還行，但到了實作程式碼以後讓我發現有很多麻煩的點，需要多花一些時間來去理解。