程式碼:

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

// 節點結構
struct Node {
    double coef;   // 係數
    int exp;       // 指數
    Node* link;
    Node(double c = 0, int e = 0, Node* l = nullptr) : coef(c), exp(e), link(l) {}
};

class Polynomial {
private:
    Node* head;
    static Node* avail;


    Node* getNode(double coef = 0, int exp = 0) {
        Node* x;
        if (avail == nullptr) {
            x = new Node(coef, exp);
        }
        else {
            x = avail;
            avail = avail->link;
            x->coef = coef;
            x->exp = exp;
        }
        return x;
    }

    // 回收
    void returnNode(Node* p) {
        p->link = avail;
        avail = p;
```

```
    }

public:
    // 建構函數
    Polynomial() {
        head = getNode();
        head->link = head;
    }

    // copy 建構函數
    Polynomial(const Polynomial& a) {
        head = getNode();
        head->link = head;

        Node* current = head;
        Node* aCurrent = a.head->link;

        while (aCurrent != a.head) {
            current->link = getNode(aCurrent->coef, aCurrent->exp);
            current = current->link;
            aCurrent = aCurrent->link;
        }
        current->link = head;
    }

    // 解構函數
    ~Polynomial() {
        Node* current = head->link;
        while (current != head) {
            Node* temp = current;
            current = current->link;
            returnNode(temp);
        }
        returnNode(head);
    }


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

```
    if (this != &a) {

        Node* current = head->link;
        while (current != head) {
            Node* temp = current;
            current = current->link;
            returnNode(temp);
        }

        // 複製新的多項式
        current = head;
        Node* aCurrent = a.head->link;
        while (aCurrent != a.head) {
            current->link = getNode(aCurrent->coef, aCurrent->exp);
            current = current->link;
            aCurrent = aCurrent->link;
        }
        current->link = head;
    }
    return *this;
}

// 輸入運算
friend istream& operator>>(istream& is, Polynomial& x) {
    int n;
    cout << "請輸入多項式的項數: ";
    is >> n;

    // 清除現有的多項式
    Node* current = x.head->link;
    while (current != x.head) {
        Node* temp = current;
        current = current->link;
        x.returnNode(temp);
    }
    x.head->link = x.head;

    // 輸入新的多項式
```

```
        Node* last = x.head;
        for (int i = 0; i < n; i++) {
            double coef;
            int exp;
            cout << "請輸入第 " << (i + 1) << " 項的係數和指數: ";
            is >> coef >> exp;

            if (coef != 0) {
                Node* newNode = x.getNode(coef, exp);
                newNode->link = x.head;
                last->link = newNode;
                last = newNode;
            }
        }
        return is;
    }

    // 輸出運算
    friend ostream& operator<<(ostream& os, const Polynomial& x) {
        if (x.head->link == x.head) {
            os << "0";
            return os;
        }

        Node* current = x.head->link;
        bool isFirst = true;

        while (current != x.head) {
            if (current->coef > 0 && !isFirst) {
                os << "+";
            }

            if (abs(current->coef + 1.0) > 1e-6 && abs(current->coef - 1.0) > 1e-6 || current->exp == 0)
{
                double coef = current->coef;
                int intCoef = (int)coef;
                if (abs(coef - intCoef) < 1e-6) {
                    os << intCoef;
```

```
            }
            else {
                os << coef;
            }
        }
        else if (current->coef < 0) {
            os << "-";
        }

        if (current->exp > 0) {
            os << "x";
            if (current->exp > 1) {
                os << "^" << current->exp;
            }
        }

        isFirst = false;
        current = current->link;
    }
    return os;
}

// 加法運算
Polynomial operator+(const Polynomial& b) const {
    Polynomial c;
    Node* aPos = head->link;
    Node* bPos = b.head->link;
    Node* last = c.head;

    while (aPos != head && bPos != b.head) {
        if (aPos->exp == bPos->exp) {
            double sum = aPos->coef + bPos->coef;
            if (abs(sum) > 1e-10) {
                Node* newNode = c.getNode(sum, aPos->exp);
                newNode->link = c.head;
                last->link = newNode;
                last = newNode;
            }
```

```
            aPos = aPos->link;
            bPos = bPos->link;
        }
        else if (aPos->exp > bPos->exp) {
            Node* newNode = c.getNode(aPos->coef, aPos->exp);
            newNode->link = c.head;
            last->link = newNode;
            last = newNode;
            aPos = aPos->link;
        }
        else {
            Node* newNode = c.getNode(bPos->coef, bPos->exp);
            newNode->link = c.head;
            last->link = newNode;
            last = newNode;
            bPos = bPos->link;
        }
    }

    while (aPos != head) {
        Node* newNode = c.getNode(aPos->coef, aPos->exp);
        newNode->link = c.head;
        last->link = newNode;
        last = newNode;
        aPos = aPos->link;
    }

    while (bPos != b.head) {
        Node* newNode = c.getNode(bPos->coef, bPos->exp);
        newNode->link = c.head;
        last->link = newNode;
        last = newNode;
        bPos = bPos->link;
    }

    return c;
}
```

```
// 減法運算
Polynomial operator-(const Polynomial& b) const {
    Polynomial c;
    Node* last = c.head;
    Node* bPos = b.head->link;

    while (bPos != b.head) {
        Node* newNode = c.getNode(-bPos->coef, bPos->exp);
        newNode->link = c.head;
        last->link = newNode;
        last = newNode;
        bPos = bPos->link;
    }

    return *this + c;
}

// 乘法運算
Polynomial operator*(const Polynomial& b) const {
    Polynomial result;
    Node* aPos = head->link;

    while (aPos != head) {
        Polynomial temp;
        Node* last = temp.head;
        Node* bPos = b.head->link;

        while (bPos != b.head) {
            Node* newNode = temp.getNode(aPos->coef * bPos->coef, aPos->exp + bPos->exp);
            newNode->link = temp.head;
            last->link = newNode;
            last = newNode;
            bPos = bPos->link;
        }

        result = result + temp;
        aPos = aPos->link;
    }
```

```
        return result;
    }

    // 求值
    double evaluate(double x) const {
        double result = 0;
        Node* current = head->link;

        while (current != head) {
            result += current->coef * pow(x, current->exp);
            current = current->link;
        }

        return result;
    }
};

// 初始化靜態成員
Node* Polynomial::avail = nullptr;

int main() {
    cout << "多項式計算程式\n";
    cout << "==================\n\n";

    Polynomial p1, p2;

    // 輸入多項式
    cout << "輸入第一個多項式:\n";
    cin >> p1;
    cout << "\n 輸入第二個多項式:\n";
    cin >> p2;

    // 顯示輸入的多項式
    cout << "\n========== 輸入的多項式 ==========\n";
    cout << "P1(x) = " << p1 << endl;
    cout << "P2(x) = " << p2 << endl;
```

```
    // 計算並顯示運算結果
    Polynomial sum = p1 + p2;
    Polynomial diff = p1 - p2;
    Polynomial prod = p1 * p2;

    cout << "\n========== 運算結果 ==========\n";
    cout << "P1(x) + P2(x) = " << sum << endl;
    cout << "P1(x) - P2(x) = " << diff << endl;
    cout << "P1(x) * P2(x) = " << prod << endl;

    // 求值運算
    cout << "\n========== 求值運算 ==========\n";
    double x_val;
    cout << "\n 請輸入要計算的 x 值: ";
    cin >> x_val;

    cout << "\n 當 x = " << x_val << " 時：\n";
    cout << "P1(" << x_val << ") = " << p1.evaluate(x_val) << endl;
    cout << "P2(" << x_val << ") = " << p2.evaluate(x_val) << endl;
    cout << "P1 + P2 = " << sum.evaluate(x_val) << endl;
    cout << "P1 - P2 = " << diff.evaluate(x_val) << endl;
    cout << "P1 * P2 = " << prod.evaluate(x_val) << endl;

    return 0;
}
```

輸出:

```
多項式計算程式
================

輸入第一個多項式:
請輸入多項式的項數: 3
請輸入第 1 項的係數和指數: 4.2
3
請輸入第 2 項的係數和指數: 2.32
1
請輸入第 3 項的係數和指數: 1.01
0

輸入第二個多項式:
請輸入多項式的項數: 3
請輸入第 1 項的係數和指數: 2.5
3
請輸入第 2 項的係數和指數: 1.2
2
請輸入第 3 項的係數和指數: 4.21
1

========== 輸入的多項式 ==========
P1(x) = 4.2x^3+2.32x+1.01
P2(x) = 2.5x^3+1.2x^2+4.21x

========== 運算結果 ==========
P1(x) + P2(x) = 6.7x^3+1.2x^2+6.53x+1.01
P1(x) - P2(x) = 1.7x^3-1.2x^2-1.89x+1.01
P1(x) * P2(x) = 10.5x^6+5.04x^5+23.482x^4+5.309x^3+10.9792x^2+4.2521x

========== 求值運算 ==========

請輸入要計算的 x 值: 1.5

當 x = 1.5 時:
P1(1.5) = 18.665
P2(1.5) = 17.4525
P1 + P2 = 36.1175
P1 - P2 = 1.2125
P1 * P2 = 325.751
```

解題說明:

```
friend istream& operator>>(istream& is, Polynomial& x) {
    int n;
    cout << "請輸入多項式的項數: ";
    is >> n;

    // 清除現有的多項式
    Node* current = x.head->link;
    while (current != x.head) {
        Node* temp = current;
        current = current->link;
        x.returnNode(temp);
    }
    x.head->link = x.head;

    // 輸入新的多項式
    Node* last = x.head;
    for (int i = 0; i < n; i++) {
        double coef;
        int exp;
        cout << "請輸入第 " << (i + 1) << " 項的係數和指數: ";
        is >> coef >> exp;

        if (coef != 0) {
            Node* newNode = x.getNode(coef, exp);
            newNode->link = x.head;
            last->link = newNode;
            last = newNode;
        }
    }
    return is;
}
```

輸入運算

```
friend ostream& operator<<(ostream& os, const Polynomial& x) {
    if (x.head->link == x.head) {
        os << "0";
        return os;
    }

    Node* current = x.head->link;
    bool isFirst = true;

    while (current != x.head) {
        if (current->coef > 0 && !isFirst) {
            os << "+";
        }

        if (abs(current->coef + 1.0) > 1e-6 && abs(current->coef - 1.0) > 1e-6 || current->exp == 0) {
            double coef = current->coef;
            int intCoef = (int)coef;
            if (abs(coef - intCoef) < 1e-6) {
                os << intCoef;
            }
            else {
                os << coef;
            }
        }
        else if (current->coef < 0) {
            os << "-";
        }

        if (current->exp > 0) {
            os << "x";
            if (current->exp > 1) {
                os << "^" << current->exp;
            }
        }

        isFirst = false;
        current = current->link;
    }
    return os;
}
```

輸出運算

```
Polynomial operator+(const Polynomial& b) const {
    Polynomial c;
    Node* aPos = head->link;
    Node* bPos = b.head->link;
    Node* last = c.head;

    while (aPos != head && bPos != b.head) {
        if (aPos->exp == bPos->exp) {
            double sum = aPos->coef + bPos->coef;
            if (abs(sum) > 1e-10) {
                Node* newNode = c.getNode(sum, aPos->exp);
                newNode->link = c.head;
                last->link = newNode;
                last = newNode;
            }
            aPos = aPos->link;
            bPos = bPos->link;
        }
        else if (aPos->exp > bPos->exp) {
            Node* newNode = c.getNode(aPos->coef, aPos->exp);
            newNode->link = c.head;
            last->link = newNode;
            last = newNode;
            aPos = aPos->link;
        }
        else {
            Node* newNode = c.getNode(bPos->coef, bPos->exp);
            newNode->link = c.head;
            last->link = newNode;
            last = newNode;
            bPos = bPos->link;
        }
    }
```

```
    while (aPos != head) {
        Node* newNode = c.getNode(aPos->coef, aPos->exp);
        newNode->link = c.head;
        last->link = newNode;
        last = newNode;
        aPos = aPos->link;
    }

    while (bPos != b.head) {
        Node* newNode = c.getNode(bPos->coef, bPos->exp);
        newNode->link = c.head;
        last->link = newNode;
        last = newNode;
        bPos = bPos->link;
    }

    return c;
}
```

加法運算

```
Polynomial operator-(const Polynomial& b) const {
    Polynomial c;
    Node* last = c.head;
    Node* bPos = b.head->link;

    while (bPos != b.head) {
        Node* newNode = c.getNode(-bPos->coef, bPos->exp);
        newNode->link = c.head;
        last->link = newNode;
        last = newNode;
        bPos = bPos->link;
    }

    return *this + c;
}
```

減法運算

```
Polynomial operator*(const Polynomial& b) const {
   Polynomial result;
   Node* aPos = head->link;

   while (aPos != head) {
      Polynomial temp;
      Node* last = temp.head;
      Node* bPos = b.head->link;

      while (bPos != b.head) {
         Node* newNode = temp.getNode(aPos->coef * bPos->coef, aPos->exp + bPos->exp);
         newNode->link = temp.head;
         last->link = newNode;
         last = newNode;
         bPos = bPos->link;
      }

      result = result + temp;
      aPos = aPos->link;
   }

   return result;
}
```

乘法運算

```
    double evaluate(double x) const {
        double result = 0;
        Node* current = head->link;

        while (current != head) {
            result += current->coef * pow(x, current->exp);
            current = current->link;
        }

        return result;
    }
};
```

求值

效能分析:

Copy Constructor O(n)
Destructor O(n)
加法 O(n+m)
減法 O(n+m)
乘法 O(n*m)
求值 O(n)

開發報告:

這次實作了多項式的基本運算功能，包含串列的使用和基礎數學運算的實現。未來或許可以加入多項式的除法運算以及更高效的資料結構來提升運算速度