書面報告

功能說明

structure

```
// 多項式非零項
typedef struct
{
    float coef;
    int32_t expo;
}polynomial;

// 多項式名冊
typedef struct
{
    int8_t used;
    char name[32];
    polynomial p[MAX_TERM];
}nameTag;
```

多項式建構相關函式

```
void init_poly();       // 命名新多項式後輸入非零項
void add_delete_term(); // 刪除或新增非零項
```

> 當改變多項式時，都會自動由大到小排列多項式

多項式運算函式

```
void addition();
void subtraction();
void multiplication();
void division();
```

> 由上到下依序為多項式的**加**、**減**、**乘**、**除**，輸入兩個多項式的名稱進行運算

輸出介面說明

```
void menu();          // 輸出主選單
void show_nameBook(); // 輸出多項式名單
void show_poly();     // 顯示指定名稱多項式
void coef();          // 輸入指數輸出對應係數
```

主選單

```
Polynomial Calculator

Menu:
        1) initialize a polynomial
        2) show a polynomial
        3) show particular coefficient
        4) delete/add term
        5) polynomial addition
        6) polynomial subtraction
        7) polynomial multiplication
        8) polynomial division
        0) exit

Please enter option:
```

> 輸入對應的數字來使用功能

舉例使用加法功能

```
Menu:
        1) initialize a polynomial
        2) show a polynomial
        3) show particular coefficient
        4) delete/add term
        5) polynomial addition
        6) polynomial subtraction
        7) polynomial multiplication
        8) polynomial division
        0) exit

Please enter option: 5
Please enter the poly name : p1
2.000x^3 + 2.000x^2 + x^1
Please enter the poly name : p2
3.000x^3 + x^1

5.000x^3 + 2.000x^2 + 2.000x^1
```

> 輸入多項式名稱後自動顯示該多項式
> 結果將顯示於隔一行處

時間複雜度分析

令輸入之多項式 A 有 n 項另一多項式 B 有 m 項

加法和減法

```
while (A[a_idx].coef != 0 && B[b_idx].coef != 0)
{
    if (A[a_idx].expo > B[b_idx].expo)
    {
        D[d_idx] = A[a_idx];
        a_idx++;
    }
```

```
    else if (A[a_idx].expo < B[b_idx].expo)
    {
        D[d_idx] = B[b_idx];
        b_idx++;
    }
    else
    {
        D[d_idx].coef = A[a_idx].coef + B[b_idx].coef;
        D[d_idx].expo = A[a_idx].expo;
        a_idx++;
        b_idx++;
        if (D[d_idx].coef == 0)
        {
            D[d_idx].expo = 0;
            d_idx--;
        }
    }
    d_idx++;
}
```

> 比較指數後填入`D`中

最壞情況指數皆不相同時，時間複雜度為 O(m+n)

乘法

```
for (size_t i = 0; i < MAX_TERM && B[i].coef != 0; i++)
{
    // tempory total
    polynomial tmpT[MAX_TERM];
    initialize(tmpT);
    // previous sum
    polynomial preS[MAX_TERM];
    memcpy(preS, S, sizeof(preS));

    for (size_t j = 0; j < MAX_TERM && A[j].coef != 0; j++)
    {
        tmpT[j].coef = A[j].coef * B[i].coef;
        tmpT[j].expo = A[j].expo + B[i].expo;
    }
    poly_addition(tmpT, preS, S);
}
```

> 以`A`的每一項去乘以`B`的單一項，將每一次的結果暫時存至`tmpT`，最後再加總到`S`

時間複雜度為 O(mn)

除法

```
while (A[0].expo >= B[0].expo)
{
    // record quotient
    Q[q_idx].coef = A[0].coef / B[0].coef;
    Q[q_idx].expo = A[0].expo - B[0].expo;

    // tmpT = B * Q[q_idx]
```

```
    polynomial tmpT[MAX_TERM];
    initialize(tmpT);
    for (size_t i = 0; i < MAX_TERM && B[i].coef != 0; i++)
    {
        tmpT[i].coef = B[i].coef * Q[q_idx].coef;
        tmpT[i].expo = B[i].expo + Q[q_idx].expo;
    }

    polynomial preA[MAX_TERM];
    memcpy(preA, A, sizeof(preA));
    poly_subtraction(preA, tmpT, A);
    q_idx++;
}
// get remainder
for (size_t i = 0; i < MAX_TERM && A[i].coef != 0; i++)
    R[i] = A[i];
```

> 計算`A[0]`(被除數)除以`B[0]`(除數)取得商，當`A[0].expo>B[0]expo`(被除數次方小於除數)將`A`填入`R`(餘數)中

假設最壞情況 n 極大 m 極小 時間複雜度為 O(m/n)