# 四川师范大学 实验报告

学期： 2025 至 2026 第一学期 实验成绩：

课程名字：《程序设计基础——数据结构(C语言版)》 专业： 信息与计算科学

班级： 2024 级 9 班 实验编号： 02

实验项目： 实验二 指导老师： 冯山

姓名： 杨晨 学号： 2024070646

**一、实验题目**

线性表及其应用

**二、实验目的及要求**

1.熟悉线性表的ADT及其存储结构和基本操作算法实现特点。

2.针对给定问题，运用线性表存储结构和基本操作进行问题求解算法设计和实现。

3.熟练掌握线性表的顺序存储结构和动态存储结构的差异。

**三、实验内容：(类C算法的程序实现，任选其二)**

1.设计并实现一元多项式的加、减、乘、除运算的算法。(加法必做，其它选做)

2.设计并实现有序表的并的算法。

3.设计并实现集合的并、交、补、差运算算法。

4.设计并实现约瑟夫问题的求解算法。

**四、实验准备**

1.计算机设备;

2.程序调试环境的准备，本实验采用**Microsoft Visual Studio code**环境;

3.实验内容的算法分析与代码设计与分析准备；

4.实验源程序准备[experiment](https://github.com/Januariu/data-structures.git)。

5.实验测试用例：

实验内容1：

1. 加法运算

1. 减法运算：
2. 乘法运算：

实验内容3：

例1:

例2

五、实验过程

（一）问题分析

1.实验内容1：

（1）设计数据结构：一元多项式的其中一项包含系数（coef）和指数（exp）两个基本属性，有两种描述一元多项式的方式：

· 用线性顺序表记录在相应下标的次项的系数：这需要一个长度为最高次数的数组，该数据结构在多项式求解的情况存在O(n)的算法，且插入项数较方便。但是在多项式次数高但是项数少的情况空间利用率低。

·用线性链式表记录每一项的系数和次数：链表的长度仅和多项式项数有关，可以证明该记录方式是表示一般多项式的最低空间复杂度的数据结构，但是其插入项数的时间复杂度较高。

由于此问题不涉及多项式求解，故选择单向链表结构。链表有三种实现方式，由于c环境提供指针，故选用基于动态链表实现的线性链式表，其类型定义如下：

typedef struct PolyNode{

double coef;

int exp;

struct PolyNode \*next;

}PolyNode, \*Polyptr;

注：维持该数据结构的算法见算法分析

（2）输入多项式。此处规定以-1 -1为退出条件，然后输入每一项的系数coef和指数exp。由于加减法的算法是基于有序表的，故在此将系数coef和指数exp通过函数降序插入到多项式链表中，以维护其有序性。

（3）确定相关运算。本问题主要实现两个一元多项式的加法、减法、乘法运算。其中，加法和减法使用有序表合并算法，减法此处只考虑第一个多项式减去第二个多项式。

（4）相关运算分析：

**加法运算：**根据有序表合并的思想，类似地，可对两个即将进行相加的多项式poly1和poly2设定两个指针p1和p2，分别指向各自第一项（即最高次项），再设定一个多项式rusult用于存储相加结果。然后按照指数大小对逐项进行比较：若各自指针所指的指数相同，则将对应系数相加，取其中一个指数为最终指数插入result中，同时两个指针移动；若指数不同，则向result插入指数较大的项并移动其指针到下一个节点。若其中一个多项式的所有项比较完毕，而另一个多项式还存在剩余项，则插入剩余项到result中。

**减法运算：**与加法运算的算法类似。

**乘法运算：**根据两个一元多项式相乘的性质可知，其运算过程为**“逐项相乘再求和”**，故设置两个循环使得poly1中的每一个元素都与poly2到全部元素相乘，并将结果插入result中。

**除法运算：**由于多项式长除法不可避免的要遍历到全部次数项，故考虑将链式多项式系数提取出来存入数组中（可以用滚动数组优化）。将除数多项式的系数数组按照长除法的规则依次遍历被除多项式的系数数组，遍历一边后剩下的系数为余数多项式。再按照系数数组分别生成商多项式和余数多项式。

**2.实验内容3：**

（1）确定数据结构：由于一个集合包含其**元素（element）**和**大小（size）**两个基本属性,由于集合中的元素没有相关性，不需要考虑插在俩个元素之间的问题，所以链式表优秀的随机插入优势无法很好的发挥；由于要维持集合元素唯一的特性，在插入元素时需要遍历全部元素，线性表和链式表在此情形下的表现相同，故本问题选择**基于动态数组实现的线性顺序表**结构。其类型定义如下：

typedef struct {

ElemType \*data; //存储元素

int size; //元素数量

int \_capacity; //分配空间大小

}set;

注：维持该数据结构的算法见算法分析

（2）基本分析：

实现集合之间的基本运算，并集、交集、补集。在此仅考虑两个集合**A**与**B**之间的并集、交集、补集运算：

**并集：**将两个集合中的所有元素合并在一起，此处考虑元素重复问题。将并集结果记为**C**，符号表示如下：

**交集**：将两个集合中的共有元素合并在一起，此处考虑元素重复问题。将交集结果记为**D**，符号表示如下：

**补集**：对集合**A**相对于全集**U**的补集，即全集**U**中不包含集合**A**的元素的集合，其中注意审查A是否是U的子集。将补集结果记作E，符号表示如下

（3）确定关键函数：为了维护集合元素的唯一性，可以设计**函数Contain**对集合中的元素进行是否**存在进行判定**。**Contain**包含两个参数，**集合set**和**元素element**，即通过**判断集合set中是否存在该元素element**得到相应的返回值。

**（二）算法分析**

**1.实验内容1：**

（1）**Status init\_poly(****polyptr\* head);**

//本算法用于初始化一个空的多项式链表：将Polyptr\*类型的头指针head指向头节点，表示初始化后的多项式在逻辑结构上没有任何项，数学意义为0多项式。

（2）**void free\_poly(polyptr\* head);**

//本算法用于释放一个多项式链表。

（3）**void** **Insert\_term(Polyptr\* head, int exp, int coef);**

//本算法用于将一个新的多项式项插入到多项式链表中，并按照项的指数大小进行降序插入：其中包含新项的double类型系数coef和int类型指数exp，以及Polynimial类型的指向多项式链表头节点的指针

（4）**void add\_poly(Polyptr \*poly1, Polyptr \*poly2, Polyptr \*result);**

**//**本算法用于两个多项式poly1与poly2的相加运算：给出两个一元多项式poly1与poly2，最后将相加结果存入一个Polyptr\*类型的多项式result中并返回

（5）**void sub\_poly(****Polyptr \*poly1, Polyptr \*poly2, Polyptr \*result);**

**//**本算法用于两个多项式poly1与poly2的相减运算（poly1-poly2）: 给出两个一元多项式poly1与poly2，最后将相减结果存入一个Polyptr\*类型的多项式result中并返回

（6）**void mul\_poly(Polyptr \*poly1, Polyptr \*poly2, Polyptr \*result);**

**//**本算法用于两个多项式poly1与poly2的相乘运算: 给出两个一元多项式poly1与poly2，最后将相乘结果存入一个Polyptr\*类型的多项式result中并返回

（7）**void print\_poly(Polyptr poly);**

//本算法用于输出显示相应的多项式

**2.实验内容3：**

（1）**Status init\_set(Set \*s, ElemType SIZE)；**

**//**本算法用于为集合分配内存空间，并设置集合的大小：s指向要初始化的集合，SIZE为集合初始储存的大小

（2）**Status free\_set(Set \*s)；**

**//**本算法用于释放集合所占的内存：s指向要释放的集合，同时检查集合数据是否为NULL，如果不为NULL则释放内存，同时将集合大小设置为0

（3）**void print\_set(Set s)；**

**//**本算法用于输出打印集合元素：s作为要打印的集合，打印前对集合中的元素进行升序排序，然后打印集合中不重复的元素

1. **status insert2set（set \*s, ElemType element）;**

//本算法用于向集合插入一个元素：s指向待插入集合，element为待插入元素，若有空闲空间则插入元素，若无空闲空间调用相应函数扩容后再插入元素。并更新size

1. **status resize\_set(set \*s);**

//本算法用于扩容集合：s指向待扩容集合，在此约定扩容空间为原先空间的两倍。

（6）**Status contain(Set \*s, ElemType element)；**

**//**本算法用于判断集合中是否包含特定元素：s指向待检查的集合，element为要检查的元素。遍历集合的每个元素，如果找到与element相等的元素则返回OK

（7）**Status Union\_Set(Set \*s1, Set \*s2, Set \*result)；**

**//**本算法用于计算两个集合的并集：s1和s2为待操作的集合，result为存储结果的集合。将集合s1的所有元素添加到集合result中，然后检查集合s2的每个元素，如果集合result中不存在，则将其添加，最后更新集合result的大小

（8）**Status Intersection\_Set(Set \*s1, Set \*s2, Set \*result)；**

**//**本算法用于计算两个集合的交集：s1和s2为待操作的集合，result为存储结果的集合。遍历较大的集合中的每个元素，检查是否在另一个集合中存在，如果存在则将其添加到集合C，并更新集合C的大小

（9）**Status Complement\_Set(Set \*s\_U, Set \*s, Set \*result)；**

**//**本算法用于计算集合的补集：s\_U为全集，s为目标集合，result为存储结果的集合。遍历全集中的每个元素，如果该元素不在目标集合中，则将其添加到集合C，并更新集合C的大小

**（三）程序代码**

**1.实验内容1：**

根据以上**实验内容1**的算法描述，求解本问题的程序代码如下：

// base on dynamic linked list to implement polynomial addition subtraction multiplication and division

#include<stdio.h>

#include<stdlib.h>

typedef struct polynode{

int exp; // exponent

double coef; // coefficient

struct polynode \*next; // next node

}polynode, \*polyptr;

polyptr init\_poly(){

polyptr head = (polyptr)malloc(sizeof(polynode));

head->next = NULL;

return head;

}

void free\_poly(polyptr head){

polyptr temp;

while(head){

temp = head;

head = head->next;

free(temp);

}

}

void insert\_term(polyptr head, int exp, double coef){

if(coef == 0) return; // skip zero coefficient terms

polyptr new\_node = (polyptr)malloc(sizeof(polynode));

if(new\_node == NULL){

perror("malloc failed");

exit(EXIT\_FAILURE);

}

new\_node->exp = exp;

new\_node->coef = coef;

new\_node->next = NULL;

// Insert in descending order of exponent

polyptr current = head;

while(current->next != NULL && current->next->exp > exp){

current = current->next;

}

if(current->next != NULL && current->next->exp == exp){

// If term with same exponent exists, add coefficients

current->next->coef += coef;

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

// Remove term if coefficient becomes zero

// polyptr temp = current->next;

// current->next = temp->next;

// free(temp);

current->next = current->next->next;

}

free(new\_node);

} else {

new\_node->next = current->next;

current->next = new\_node;

}

}

void delete\_term(polyptr head, int exp){

polyptr current = head;

while(current->next != NULL && current->next->exp != exp){

current->next = current->next->next;

}

if(current->next == NULL){

printf("Term with exponent %d not found.\n", exp);

return;

}

current->next = current->next->next;

return;

}

void print\_poly(polyptr head){

polyptr current = head->next; // skip head node

if(current == NULL) printf("0\n");

while(current !=NULL){

if(current->coef > 0 && current != head->next) printf("+");

if(current->exp == 0){

printf("%.2lf", current->coef);

} else if(current->exp == 1){

printf("%.2lfx", current->coef);

} else {

printf("%.2lfx^%d", current->coef, current->exp);

}

current = current->next;

}

}

void add\_poly(polyptr poly1, polyptr poly2, polyptr result){

polyptr p1 = poly1->next; // skip head

polyptr p2 = poly2->next; // skip head

while(p1 != NULL && p2 != NULL){

if(p1->exp > p2->exp){

insert\_term(result, p1->exp, p1->coef);

p1 = p1->next;

} else if(p1->exp < p2->exp){

insert\_term(result, p2->exp, p2->coef);

p2 = p2->next;

} else {

double sum\_coef = p1->coef + p2->coef;

if(sum\_coef != 0){

insert\_term(result, p1->exp, sum\_coef);

}

p1 = p1->next;

p2 = p2->next;

}

}

}

void sub\_poly(polyptr poly1, polyptr poly2, polyptr result){

polyptr p1 = poly1->next; // skip head

polyptr p2 = poly2->next; // skip head

while(p1 != NULL && p2 != NULL){

if(p1->exp > p2->exp){

insert\_term(result, p1->exp, p1->coef);

p1 = p1->next;

} else if(p1->exp < p2->exp){

insert\_term(result, p2->exp, -p2->coef);

p2 = p2->next;

} else {

double diff\_coef = p1->coef - p2->coef;

if(diff\_coef != 0){

insert\_term(result, p1->exp, diff\_coef);

}

p1 = p1->next;

p2 = p2->next;

}

}

}

void mul\_poly(polyptr poly1, polyptr poly2, polyptr result){

polyptr p1 = poly1->next; // skip head

polyptr p2 = poly2->next; // skip head

while(p1 != NULL){

p2 = poly2->next; // reset p2 for each term in p1

while(p2 != NULL){

int new\_exp = p1->exp + p2->exp;

double new\_coef = p1->coef \* p2->coef;

insert\_term(result, new\_exp, new\_coef);

p2 = p2->next;

}

p1 = p1->next;

}

}

void div\_poly(polyptr poly1, polyptr poly2, polyptr quotient, polyptr remainder){

// Polynomial division: poly1 / poly2 = quotient ... remainder

int n1 = poly1->next->exp, n2 = poly2->next->exp;

polyptr temp1 = poly1->next;

polyptr temp2 = poly2->next;

double poly1\_coef[n1 + 1];

double poly2\_coef[n2 + 1];

for(int i = 0; i <= n1; i++) poly1\_coef[i] = 0;

for(int i = 0; i <= n2; i++) poly2\_coef[i] = 0;

while(temp1 != NULL){

poly1\_coef[temp1->exp] = temp1->coef;

temp1 = temp1->next;

}

while(temp2 != NULL){

poly2\_coef[temp2->exp] = temp2->coef;

temp2 = temp2->next;

}

int deg\_q = n1 - n2;

double coef\_q[deg\_q + 1];

for(int i = 0; i <= deg\_q; i++) coef\_q[i] = 0;

for(int i = deg\_q; i >= 0; i--){

coef\_q[i] = poly1\_coef[i + n2] / poly2\_coef[n2];

for(int j = 0; j <= n2; j++){

poly1\_coef[i + j] -= coef\_q[i] \* poly2\_coef[j];

}

}

for(int i = deg\_q; i >= 0; i--){

if(coef\_q[i] != 0)

insert\_term(quotient, i, coef\_q[i]);

}

for(int i = n2 - 1; i >= 0; i--){

if(poly1\_coef[i] != 0)

insert\_term(remainder, i, poly1\_coef[i]);

}

}

void create\_poly(polyptr head){

int exp;

double coef;

printf("Enter terms (exp coef), end with -1 -1:\n");

while(1){

scanf("%d %lf", &exp, &coef);

if(exp == -1 && coef == -1) break;

if(exp < 0){

printf("Exponent must be non-negative.\n");

continue;

}

insert\_term(head, exp, coef);

print\_poly(head);

printf("\n");

}

}

int main(){

polyptr poly1 = init\_poly();

polyptr poly2 = init\_poly();

polyptr result = init\_poly();

while(1){

printf("1. Insert term to polynomial 1\n");

printf("2. Insert term to polynomial 2\n");

printf("3. Print polynomial 1\n");

printf("4. Print polynomial 2\n");

printf("5. Add polynomials\n");

printf("6. Subtract polynomials\n");

printf("7. Multiply polynomials\n");

printf("8. Divide polynomials\n");

printf("9. Exit\n");

int choice;

scanf("%d", &choice);

if(choice == 9) break;

if(choice == 1){

create\_poly(poly1);

} else if(choice == 2){

create\_poly(poly2);

} else if(choice == 3){

print\_poly(poly1);

printf("\n");

} else if(choice == 4){

print\_poly(poly2);

printf("\n");

} else if(choice == 5){

free\_poly(result);

result = init\_poly();

add\_poly(poly1, poly2, result);

print\_poly(result);

printf("\n");

} else if(choice == 6){

free\_poly(result);

result = init\_poly();

sub\_poly(poly1, poly2, result);

print\_poly(result);

printf("\n");

} else if(choice == 7){

free\_poly(result);

result = init\_poly();

mul\_poly(poly1, poly2, result);

print\_poly(result);

printf("\n");

} else if(choice == 8){

free\_poly(result);

result = init\_poly();

polyptr remainder = init\_poly();

div\_poly(poly1, poly2, result, remainder);

printf("Quotient: ");

print\_poly(result);

printf("\n");

printf("Remainder: ");

print\_poly(remainder);

free\_poly(remainder);

printf("\n");

} else {

printf("Invalid choice.\n");

}

}

}

**2.实验内容3：**

根据以上**实验内容3**的算法描述，求解本问题的程序代码如下：

// base on dynamic array to implement set union intersection and complement

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

typedef struct set{

double \*data;

int size;

int \_capacity;

}\*set;

set init\_set(int capacity){

set s = (set)malloc(sizeof(struct set));

s->data = (double\*)malloc(sizeof(double) \* capacity);

s->size = 0;

s->\_capacity = capacity;

return s;

}

void free\_set(set s){

free(s->data);

free(s);

}

void print\_set(set s){

if(s->size == 0){

printf("Set is empty.\n");

return;

}

printf("{ ");

for(int i = 0; i < s->size; i++){

printf("%lf ", s->data[i]);

}

printf("}\n");

}

bool is\_contains(set s, double value){

for(int i = 0; i < s->size; i++){

if(s->data[i] == value) return true;

}

return false;

}

void resize(set s){

s->\_capacity \*= 2;

s->data = (double\*)realloc(s->data, sizeof(double) \* s->\_capacity);

if(s->data == NULL){

perror("realloc failed");

exit(EXIT\_FAILURE);

}

}

void insert\_to\_set(set s,double element){

if(is\_contains(s, element)) return;

if(s->size == s->\_capacity) resize(s);

s->data[s->size++] = element;

return;

}

bool union\_set(set s1, set s2, set result){

for(int i = 0; i < s1->size; i++){

insert\_to\_set(result, s1->data[i]);

}

for(int i = 0; i < s2->size; i++){

insert\_to\_set(result, s2->data[i]);

}

return true;

}

bool intersection\_set(set s1, set s2, set result){

for(int i = 0; i < s1->size; i++){

if(is\_contains(s2, s1->data[i])){

insert\_to\_set(result, s1->data[i]);

}

}

return true;

}

bool complement\_set(set s\_U, set s2, set result){

if(s\_U->size < s2->size) {

printf("Universal set must be larger than the set to complement.\n");

return false;

}

for(int i = 0; i < s\_U->size; i++){

if(!is\_contains(s2, s\_U->data[i])){

insert\_to\_set(result, s\_U->data[i]);

}

}

return true;

}

bool input\_set(set s){

printf("Enter elements of the set, end is -1\n");

double element;

while(1){

scanf("%lf", &element);

if(element == -1) break;

insert\_to\_set(s, element);

printf("Current set: ");

print\_set(s);

}

return true;

}

int main(void){

set A, B, U;

A = B = U = NULL;

A = init\_set(10);

B = init\_set(10);

U = init\_set(10);

while(1){

printf("1. Input Set A\n");

printf("2. Input Set B\n");

printf("3. Input Universal Set U\n");

printf("4. Union A U B\n");

printf("5. Intersection A ∩ B\n");

printf("6. Complement of B in U (U - B)\n");

printf("7.print set\n");

printf("8. Exit\n");

int choice;

scanf("%d", &choice);

switch (choice)

{

case 1:

{

input\_set(A);

break;

}

case 2:

{

input\_set(B);

}

break;

case 3:

{

input\_set(U);

}

break;

case 4:{

if(A == NULL || B == NULL){

printf("Please input sets A and B first.\n");

break;

}

set result = init\_set(10);

union\_set(A, B, result);

printf("A U B = ");

print\_set(result);

free\_set(result);

break;

}

case 5:{

if(A == NULL || B == NULL){

printf("Please input sets A and B first.\n");

break;

}

set result = init\_set(10);

intersection\_set(A, B, result);

printf("A ∩ B = ");

print\_set(result);

free\_set(result);

break;

}

case 6:{

if(U == NULL || B == NULL || A == NULL){

printf("Please input sets U A and B first.\n");

break;

}

set result = init\_set(10);

complement\_set(U, A, result);

printf("U - A = ");

print\_set(result);

complement\_set(U, B, result);

printf("U - B = ");

print\_set(result);

free\_set(result);

break;

}

case 7:

{

if(A) {

printf("Set A: ");

print\_set(A);

} else {

printf("Set A is not initialized.\n");

}

if(B) {

printf("Set B: ");

print\_set(B);

} else {

printf("Set B is not initialized.\n");

}

if(U) {

printf("Universal Set U: ");

print\_set(U);

} else {

printf("Universal Set U is not initialized.\n");

}

}

break;

case 8:

if(A) free\_set(A);

if(B) free\_set(B);

if(U) free\_set(U);

exit(0);

break;

default:

printf("Invalid choice.\n");

break;

}

}

return 0;

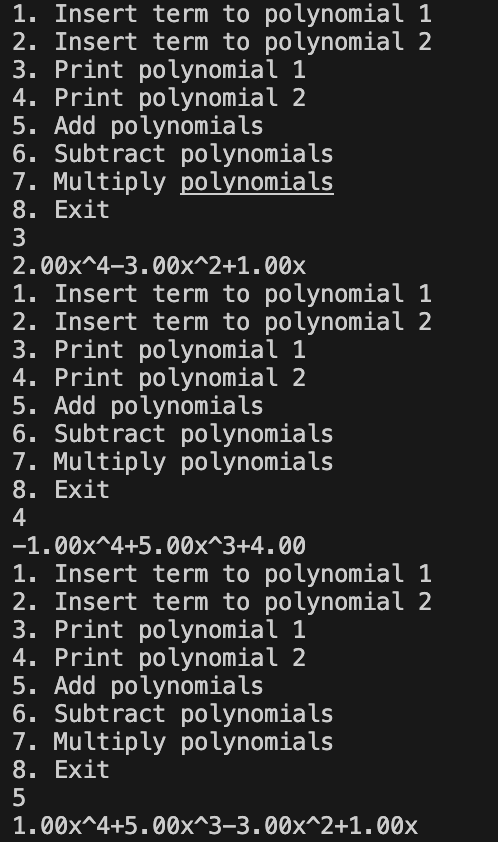
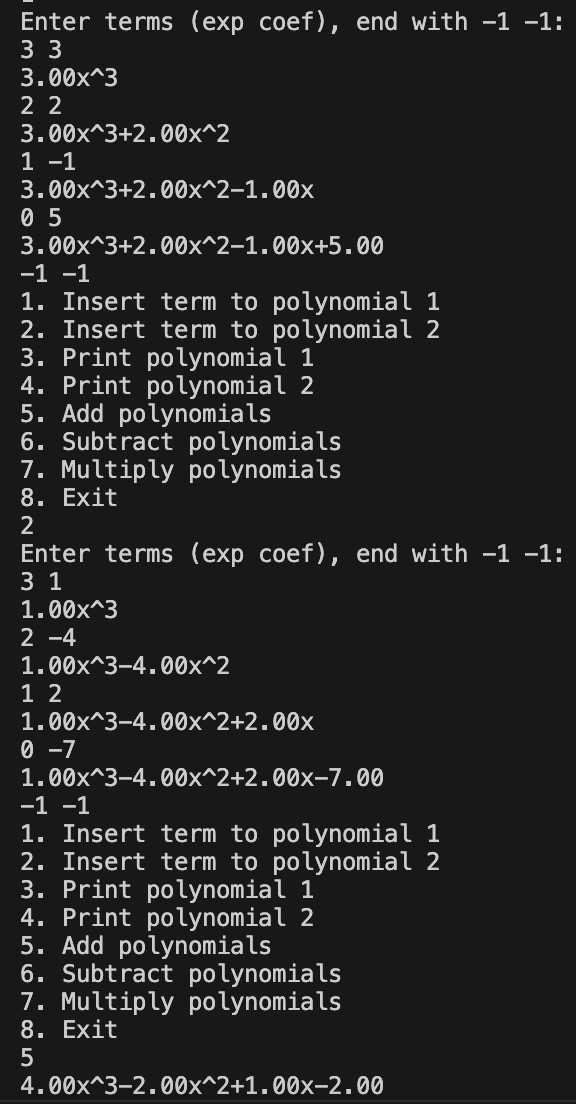
}

**六、结果分析**

**（一）结果呈现：**

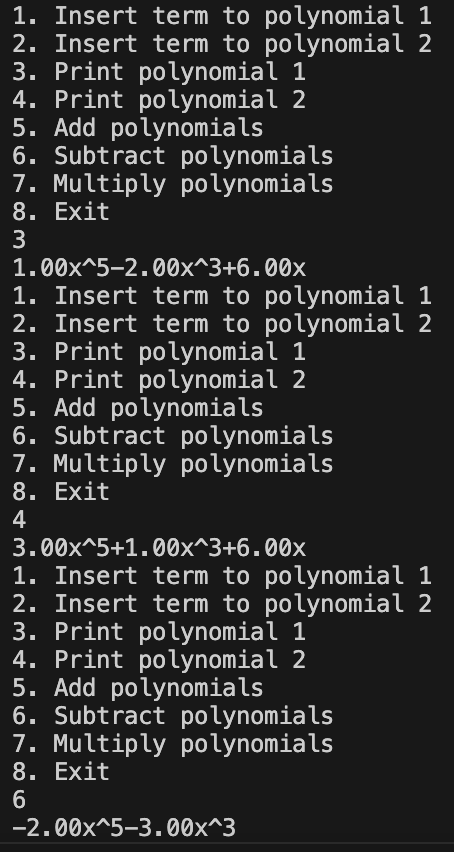
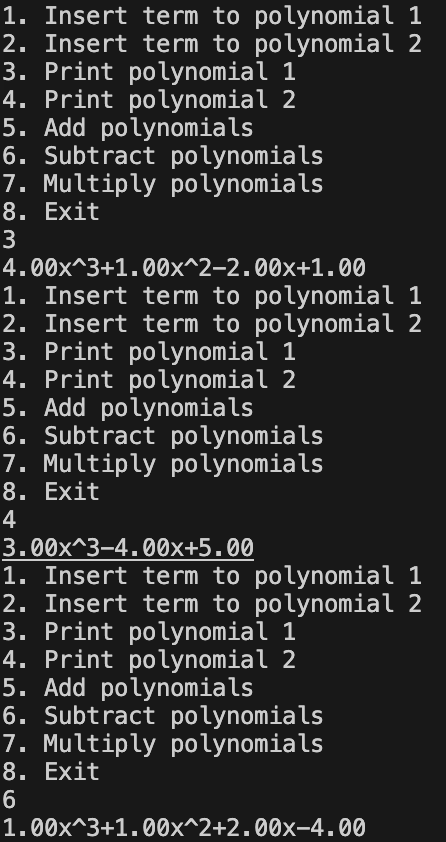
1.根据以上**实验内容1**程序代码运行后进行多次测试，以下是根据**实验内容1的测试用例**进行测试后的结果示意图：

加法：

****

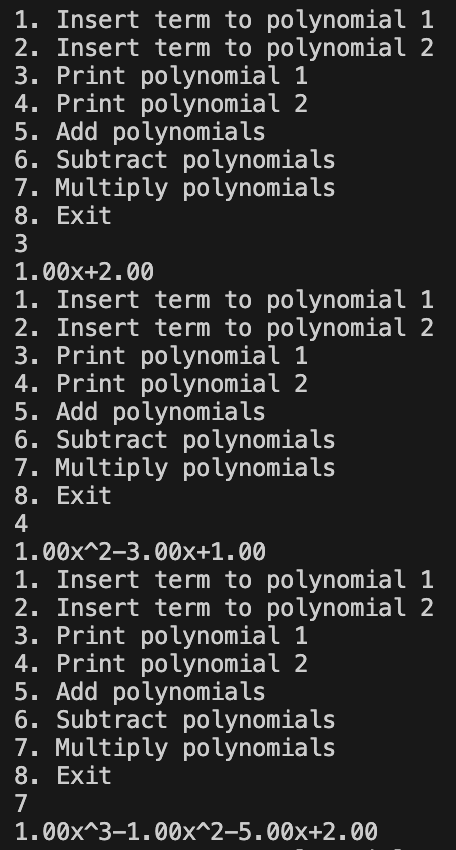
**图1:加法测试结果**

**减法：**

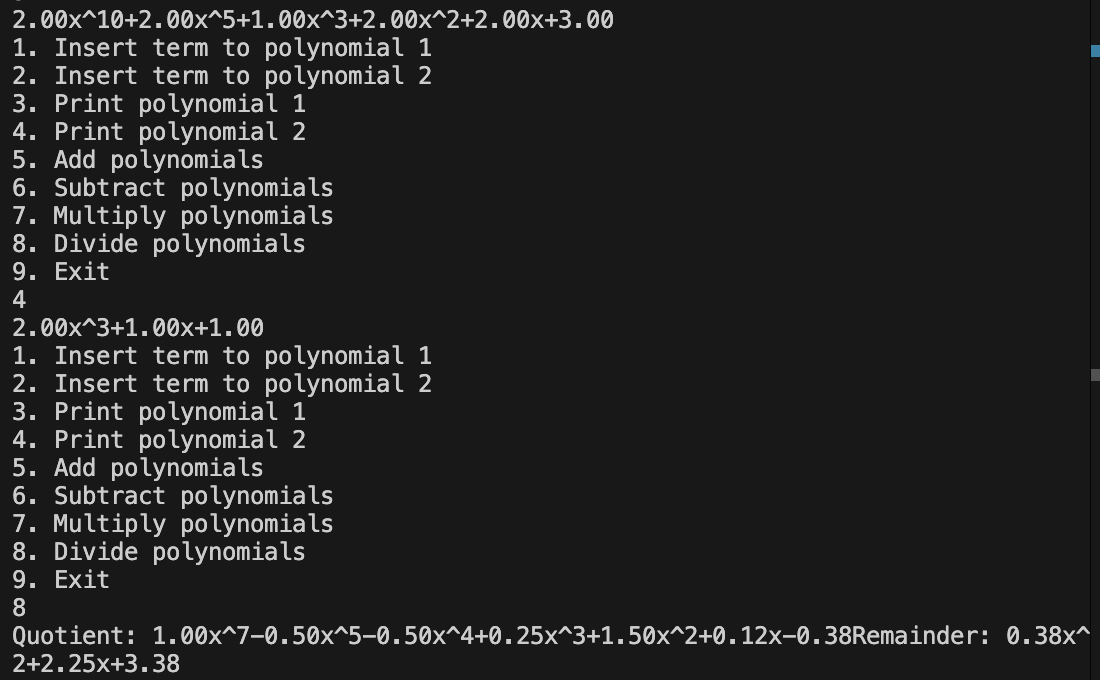
****

**图2：减法测试结果**

**乘法：**

****

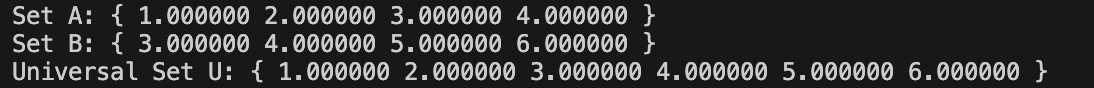
**图3：乘法测试结果**

****

**图4:除法测试结果**

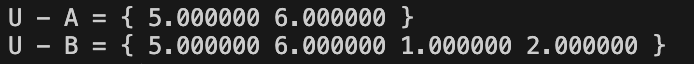
2.根据以上**实验内容3**程序代码运行后进行多次测试，以下是根据**实验内容3的测试用例**进行测试后的结果示意图：

例1:

****

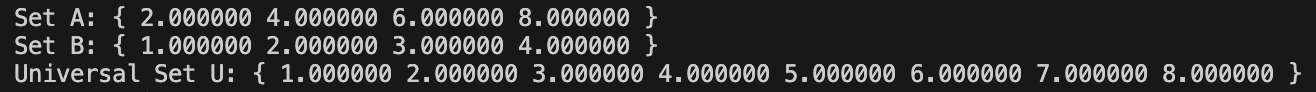
****

****

****

**图5:案例1测试结果**

**例2:**

****

****

****

****

**图6:案例2测试结果**

**（二）结果分析：**

**1.问题1:**

**结果符合预期。**

**2.问题2:**

**结果符合预期。**

**（三）算法分析：**

**1.时间复杂度分析**

**（1）实验内容1：**

**Insert\_term（）：**

该函数在最坏情况下需要遍历整个链表来寻找插入位置，故时间复杂度为**O(n)**，其中**n**是多项式的项数。

**Add\_poly（）, sub\_poly（）:**

在该函数中使用有序表合并算法，即每个多项式的每个项都需要被访问一次,故时间复杂度为**O(m+n)**，其中**m**和**n**分别是两个多项式的项数。

**Mul\_poly（）:**

在该函数中，每个多项式的每个项都需要与另一个多项式的全部项相乘，故时间复杂度为**O(m\*n)**，其中**m**和**n**分别是两个多项式的项数。

**Dev\_poly():**

在该函数中，需要从最高次项遍历到常数项，故时间复杂度为O(n),其中n为两个多项式最高次数。

**print\_poly（）:**

该函数需遍历整个链表打印每个项，故时间复杂度为**O(n)**，其中**n**是多项式的项数。

**（2）实验内容3：**

**Init\_set（）:**

分配内存的操作是常量时间，故时间复杂度为**O(1)**。

**Print\_set（）：**

该函数需遍历整个集合打印每个项，故时间复杂度为**O(n)**，其中**n**是集合的大小。

**Contain（）：**

该函数需要遍历集合，故时间复杂度为**O(n)**。

**Insert2set（）:**

该函数需要调用contain判断元素是否在集合中，在尾部插入的时间复杂度是O(1),故该函数的时间复杂度为**O(n)**。

**Union\_set（）：**

该函数将集合**A，B**的元素用**Insert2set函数**添加到集合**C**中。故时间复杂度O(A.size^2 + B.size^2).

**Intersection\_set（）:**

该函数需要遍历集合**B**并在集合**A**中进行查找是否存在相同的元素，故时间复杂度为**O(A.size\*B.size)**。

**Complement\_set（）：**

该函数需要遍历集合**A**并在全集**U**中进行查找是否存在与集合i中相异的元素，故时间复杂度为**O(U.size\*A.size)**的大小。

**2.空间复杂度分析**

**（1）实验内容1：**

**多项式链表：**

每个多项式的存储表示为一个链表，空间复杂度为**O(n)**，其中， **n**为多项式的项数。

**存储结果多项式：**

结果多项式的相加、相减后会产生一个项数最多为m+n的新多项式，相乘的后最多会产生m\*n的新多相式，故空间复杂度为O(m\*n)。

**综上所述：总体空间复杂度为 O(mn)，其中，m和n分别是两个输入的多项式项数。**

**（2）实验内容3：**

**集合结构体：**

分配的空间为**O(n)**，其中**n**是集合中元素的最大数量，主要体现在**data**指针的动态内存分配。

**额外空间**

在主函数中所创建的集合**C**、**D**、**E**、**F**和**U**，每个都占用**O(n)**的空间，其中**n**是集合中元素的最大数量。

**综上所述：总体空间复杂度为 O(n)，其中n为集合大小。**

**七****、评价**

1.问题一：

优点：对稀疏多项式表现良好；对于频繁修改多项式的情况处理较方便。

缺点：多项式求值的时候，时间复杂度较高；查找速度较慢。

扩展：由于该动态链表实现高度依赖指针，故其移植性并不好

2.问题三：

优点：连续存储，支持随机访问；

缺点：查找时间复杂度较高；

扩展：由于只是在数组实现的时候用了动态分配，很容易就能改成静态数组，故较容易移植到其他语言环境中；可以考虑用哈希表去实现set.

**八、感悟体会**

1.警惕在代码块里面创建副本，有可能会导致segmentation fault