数据结构与算法实验一

实验题目：多项式乘法问题

实验目的：

设计一个一元稀疏多项式简单计算器。

实验内容与要求：

一元稀疏多项式简单计算器的基本功能是：

（1)输入并建立多项式；

（2）输出多项式，输出形式为整数序列：n,c1,e1,c2,e2,...,cn,en,其中n是多项式的项数，ci和ei分别是第i项的系数和指数，序列按指数降序排列。

（3）多项式a与多项式b相乘，建立多项式。

实验内容和实验步骤：

1．需求分析：

（1）输入的形式和输入值的范围：一元稀疏多项式的项数以及各项的系数和指数，系数为浮点数，指数为整数，各项的输入顺序随机。

（2）输出的形式：按各项的指数由大到小倒序排列，分别输出两个多项式和相乘结果。

（3）程序所能实现的功能：实现多项式的排序以及两个多项式相乘。

2．概要设计：

整体设计采用单向链表、链式存储。首先参照课本第42页例2-4建立抽象数据类型polynomial用来实现多项式的建立、输入等基本操作。乘法可以分解为一系列的加法运算，故两个多项式乘法可以利用多项式加法实现。参照课本43页完成实现两个多项式加法的函数，再完善多项式乘法以及输出等其他函数，实现多项式乘法器。

3．详细设计：

```
#include <iostream>

using namespace std;

typedef struct{
    float coef; //系数
    int expn; //指数
}term, ElemType;

typedef struct LNode{
    ElemType data;
    struct LNode *next;
}*Link, *Position;

typedef struct{
    Link head, tail;
    int len;
}LinkList;

typedef LinkList polynomial;

typedef int Status;

void InitList(LinkList &L);
        //初始化空链表
Position GetHead(LinkList L);
        //返回头结点位置
void SetCurElem(Link &p, ElemType e);
        //用e更新p结点的值
void SetCurElem_c(Link &p, float s);
        //更新系数
void InsFirst(LinkList &L, Link &h, Link &s);
        //插入元素
void InsEnd(LinkList &L, Link &s);
        //尾插
Position NextPos(LinkList L, Link p);
        //返回p的后继结点位置
ElemType GetCurElem(Link p);
        //返回p结点元素的expn值
void DelFirst(Link &h, Link &q);
        //删除头结点
void FreeNode(Link &p);
        //释放p结点
Status ListEmpty(LinkList L);
        //是否空表
void Append(LinkList &L, Link s);
        // 链接s及之后的结点到L尾部
void CreatPolyn(polynomial &p, int m);
          //输入m项的系数和指数，建立表示一元多项式的有序链表P
void PrintPolyn(polynomial p);
```

```
          //打印输出一元多项式P
Status PolynLenght(polynomial p);
          //返回一元多项式P中的项数
void AddPolyn(polynomial &Pa, polynomial &Pb);
          //完成多项式相加运算,即Pa=Pa+Pb
void MultiplyPolyn(polynomial &Pa, polynomial &Pb);
          //完成多项式相乘运算,即Pa=Pa×Pb
Status cmp(term a, term b);
          //依a的指数值<(或=)(或>)b的指数值，分别返回-1、0和+1
void sort_exp(LinkList &L);
        //按exp值排序
void InitList(LinkList &L){
    L.head = (Link)malloc(sizeof(LNode));
    L.tail = L.head;
    L.head -> next = NULL;
    L.tail -> next = NULL;
    L.len = 0;
}

Position GetHead(LinkList L){
    return L.head;
}

void SetCurElem(Link &p, ElemType e){
    p->data = e;
}

void SetCurElem_c(Link &p, float s){
    p->data.coef = s;
}

void InsFirst(LinkList &L, Link &h, Link &s) {
    if(h == L.head) {
        s->next = h->next;
        h->next = s;
    }
    else {
        Link t = L.head;
        while(t->next != h) {
            t = t->next;
        }
        t->next = s;
        s->next = h;
    }
}

void InsEnd(LinkList &L, Link &s) {
    L.tail->next = s;
    L.tail = s;
    s->next = NULL;
}
```

```
Position NextPos(LinkList L, Link p) {
    Position t = L.head;
    while(t != p->next) {
        t = t->next;
    }
    return t;
}

ElemType GetCurElem(Link p) {
    return p->data;
}

void DelFirst(Link &h, Link &q) {
    h->next = q->next;
}

void FreeNode(Link &p) {
    free(p);
}

Status ListEmpty(LinkList L) {
    if(L.head->next == NULL) return true;
    else return false;
}

void Append(LinkList &L, Link s) {
    L.tail->next = s;
    Link &t = s;
    while(t->next != NULL) t = t->next;
    L.tail = t; //将s及之后的结点依次并入L的尾部
}

void CreatPolyn(polynomial &p, int m){
    InitList(p);
    Link h;
    term e;
    h = GetHead(p);
    e.coef = 0.0;
    e.expn = -1;
    SetCurElem(h, e);
    for(int i = 0; i < m; i++){
        cin>>e.coef>>e.expn;
        Link s = (Link)malloc(sizeof(LNode)); //分配空间
        s->data.coef = e.coef;
        s->data.expn = e.expn;
        InsEnd(p, s); //将生成的结点插入链表
    }
}

void AddPolyn (polynomial &Pa, polynomial &Pb){
```

```
    sort_exp(Pa);
    sort_exp(Pb); //先将两个多项式按照指数倒序的顺序排列
    Link ha = GetHead(Pa);
    Link hb = GetHead(Pb);
    Link qa = NextPos(Pa, ha);
    Link qb = NextPos(Pb, hb);
    while(qa && qb){
        term a = GetCurElem(qa);
        term b = GetCurElem(qb);
        switch(cmp(a, b)){
            case -1:{      // 多项式PA中当前结点的指数值小
                DelFirst(hb, qb);
                InsFirst(Pa, ha, qb);
                qb = NextPos(Pb, hb);
                ha = NextPos(Pa, ha);
                break;
            }
            case 0:{       // 两者的指数值相等
                float sum= a.coef + b.coef;
                if(sum != 0.0){         //修改多项式PA中当前结点的系数值
                    SetCurElem_c(qa, sum);
                    ha=qa;
                }
                else{                   //删除多项式PA中当前结点
                    DelFirst(ha, qa);
                    FreeNode(qa);
                }
                DelFirst(hb, qb);
                FreeNode(qb);
                qb = NextPos(Pb, hb);
                qa = NextPos(Pa, ha);
                break;
            }
            case 1:{         //多项式PB中当前结点的指数值小
                ha = qa;
                qa = NextPos(Pa,qa);
                break;
            }
        }
    }
    if(!ListEmpty(Pb)) Append(Pa, qb);         //链接Pb中剩余结点
    FreeNode(hb);                  // 释放Pb的头结点
}

Status cmp(term a, term b) {
    if(a.expn < b.expn) return -1;
    else if(a.expn == b.expn) return 0;
    else return 1;
}

Status PolynLenght(polynomial p) {
```

```
    int i = 0;
    Link t = p.head->next;
    while(t != NULL) {
        if(t->data.coef != 0) i++;
        t = t->next;
    }
    return i;
}

void MultiplyPolyn(polynomial Pa, polynomial Pb, polynomial &Pc) {
    Link t1, t2;
    t1 = Pa.head->next;
    while(t1 != NULL) {
        t2 = Pb.head->next;
        while(t2 != NULL) {
            if(t2->data.coef != 0 && t1->data.coef != 0) {
                polynomial Pd;
                CreatPolyn(Pd, 0);
                Link t = (Link)malloc(sizeof(LNode));
                t->data.coef = t1->data.coef * t2->data.coef;
                t->data.expn = t1->data.expn + t2->data.expn;
                InsEnd(Pd, t);
                AddPolyn(Pc, Pd);
            }
            t2 = t2->next;
        }
        t1 = t1->next;
    }
}

void sort_exp(LinkList &L) {
    Link t1, t2;
    t1 = L.head->next;
    while(t1 != NULL) {
        t2 = t1->next;
        while(t2 != NULL) {
            if(t1->data.expn < t2->data.expn) {
                int t3 = t1->data.expn;
                float t4 = t1->data.coef;
                t1->data.expn = t2->data.expn;
                t1->data.coef = t2->data.coef;
                t2->data.expn = t3;
                t2->data.coef = t4;
            }
            else if(t1->data.expn == t2->data.expn) { //指数相同则两项合并，后一项的系数
置0
                t1->data.coef += t2->data.coef;
                t2->data.coef = 0.0;
            }
            t2 = t2->next;
        }
```

```
        t1 = t1->next;
    }
}

void PrintPolyn(polynomial p) {
    Link t;
    t = p.head->next;
    sort_exp(p);
    cout<<PolynLenght(p)<<" ";
    while(t != NULL) {
        if(t->data.coef != 0) cout<<t->data.coef<<"x^"<<t->data.expn<<" ";
        t = t->next;
    }
    cout<<endl;
}

int main()
{
    polynomial pa, pb, pc;
    int m;
    cout<<"输入第一个多项式的项数以及各项系数和指数："<<endl;
    cin>>m;
    CreatPolyn(pa, m);
    cout<<"输入第二个多项式的项数以及各项系数和指数："<<endl;
    cin>>m;
    CreatPolyn(pb, m);
    cout<<"第一个多项式："<<endl;
    PrintPolyn(pa);
    cout<<"第二个多项式："<<endl;
    PrintPolyn(pb);
    CreatPolyn(pc, 0);
    MultiplyPolyn(pa, pb, pc);
    cout<<"相乘结果："<<endl;
    PrintPolyn(pc);
    return 0;
}
```

4．调试分析：

（1）调试过程中所遇到的问题及解决方法：

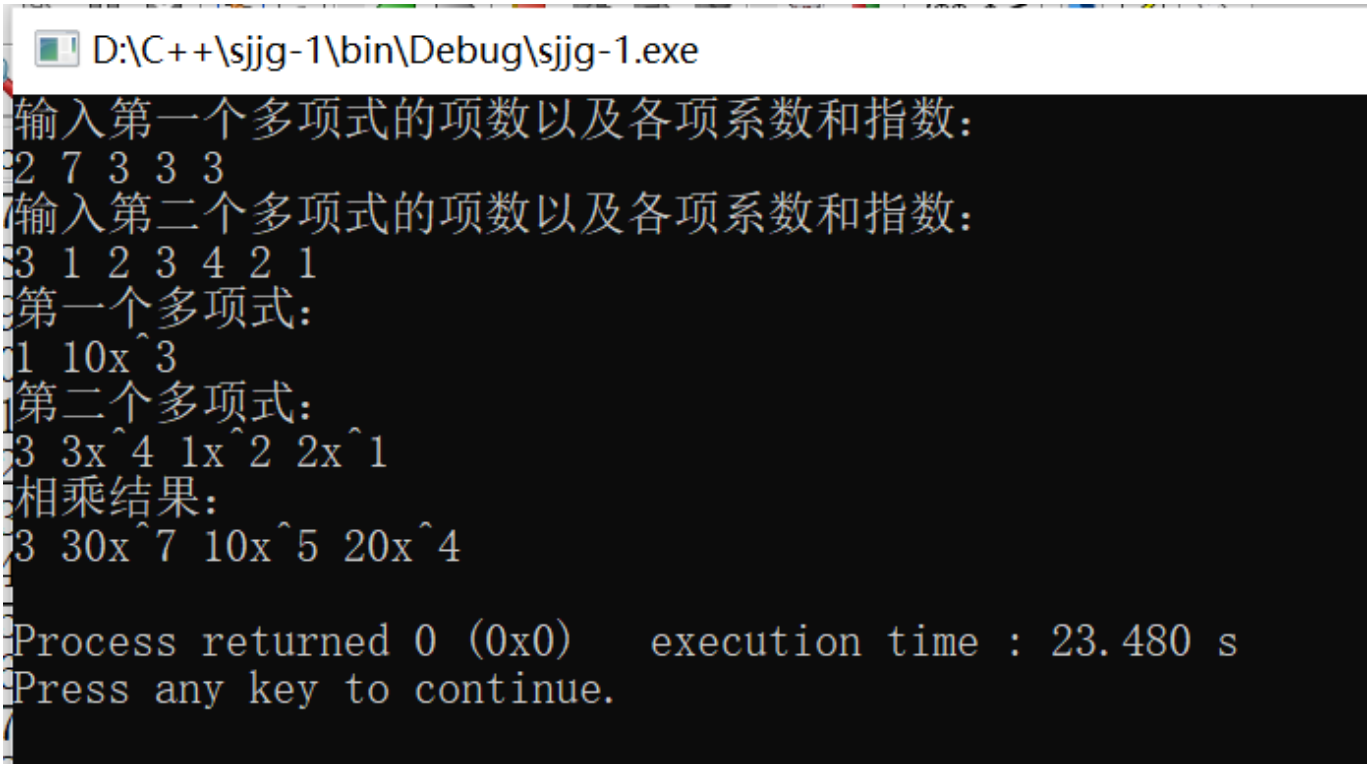
问题：输入中若存在指数相同的几项，则不会合并成一项，导致输出存在多项指数相同的项

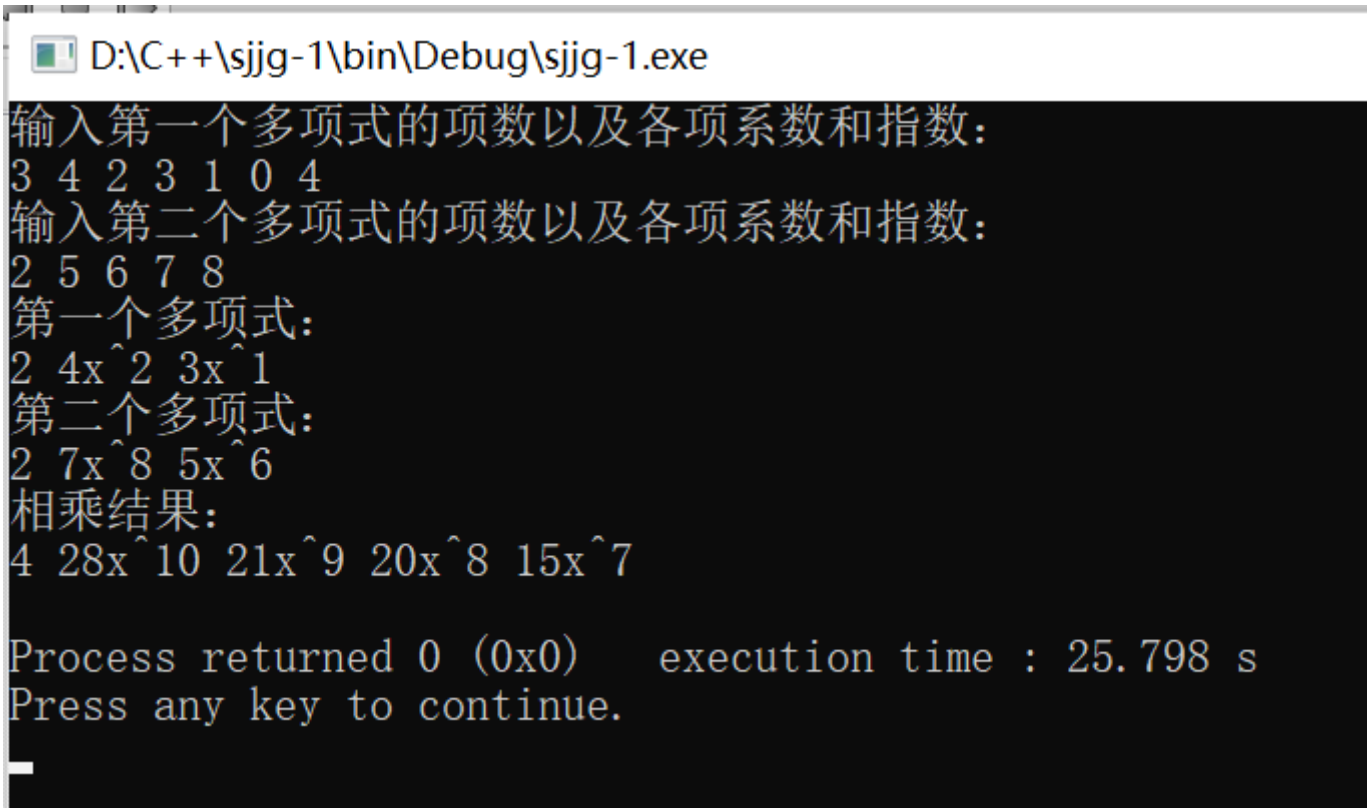
解决方法：在sort_exp函数中针对指数相同项进行处理，若扫描到两项指数相同，则将两项合并在一起。这样通过sort_exp函数既能使各项按照指数由大到小倒序排列又同时完成了合并指数相同项的功能。

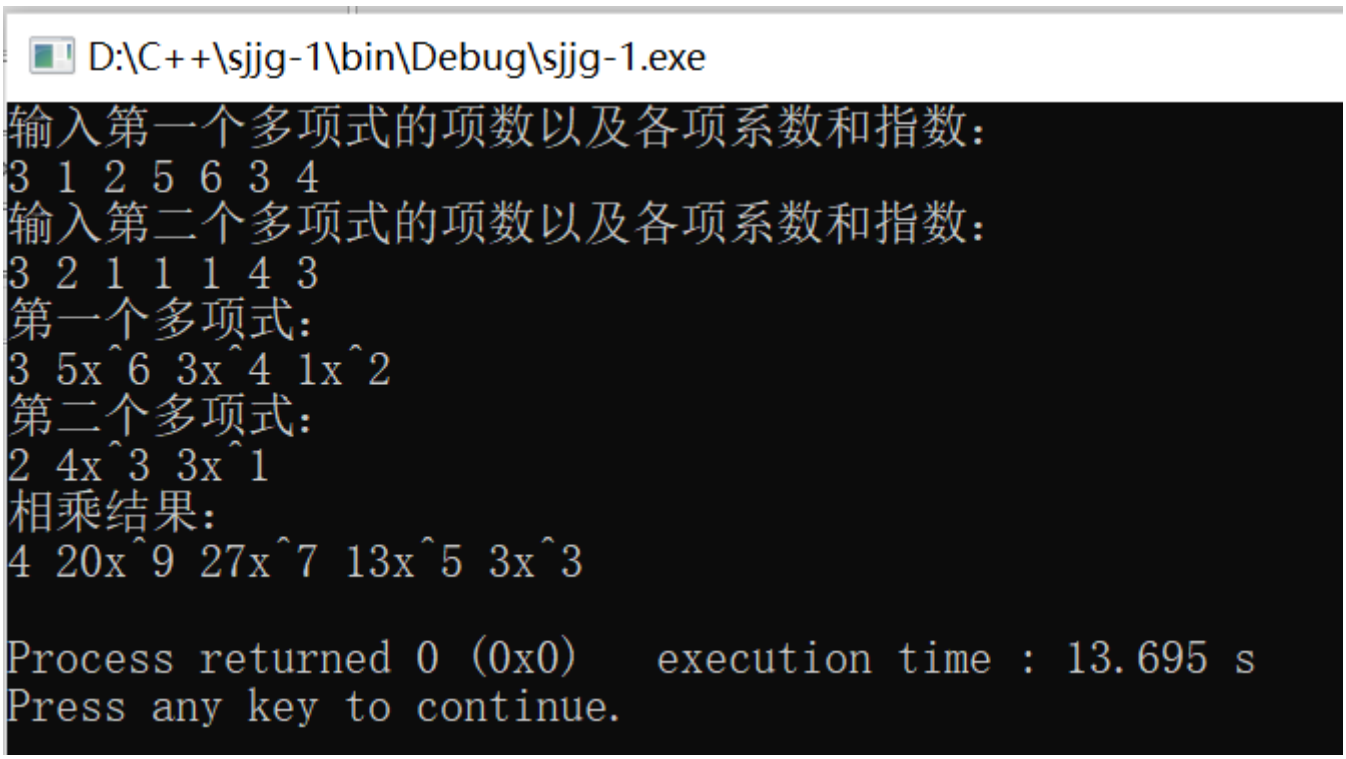
（2）算法的时空分析：

设输入项数(即遍历结点数)为n，CreatPolyn函数复杂度O(n)，PrintPolyn函数复杂度O(n^2)，MultiplyPolyn函数复杂度O(n^3)，总复杂度O(n^3)。

实验测试结果：







实验总结:

本实验要求编写程序实现两个一元稀疏多项式的乘法计算。输入一元稀疏多项式的项数以及各项的系数和指数，建立多项式，进行计算，最后按照各项指数由大到小倒序的排列输出两个多项式以及相乘结果。

由于多项式乘法可以分解为一系列的加法运算，故两个多项式乘法可以利用多项式加法实现。实验中有两点需要注意，一是当系数为0时该项不存在不计入项数，二是当同一多项式有多个指数相同的项时要合并成一项。