

一、实验目的：

设计一个一元稀疏多项式简单计算器。

二、实验要求：

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： $n, c_1, e_1, c_2, e_2, \dots, c_n, e_n$ ，其中 n 是多项式的项数， c_i 和 e_i 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。

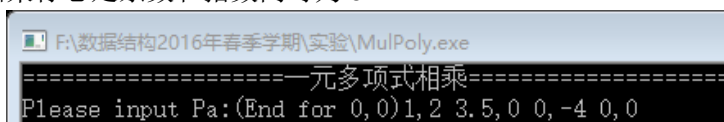
三、实验内容和实验步骤：

1、需求分析：

(1) 输入形式： $c_1, e_1 \ c_2, e_2 \ \dots \ 0, 0$

- a) 直接输入多项式每一项的系数和指数，不需输入有多少项。
- b) 同一项的系数和指数用逗号隔开，项与项之间用空格隔开。
- c) 并且规定系数可以是小数，但指数只能是整数。
- d) 系数和指数都可以为 0，但不能同时为 0。
- e) 输入结束标志是系数和指数同时为 0。

f) 截图：

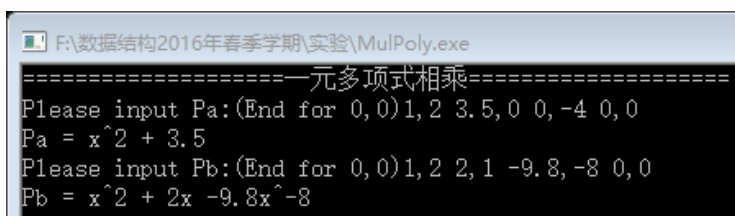


```
F:\数据结构2016年春季学期\实验\MulPoly.exe
=====一元多项式相乘=====
Please input Pa: (End for 0,0)1, 2 3.5, 0 0, -4 0, 0
```

(2) 输出形式： $P = c_1x^{e_1} + c_2x^{e_2} + \dots$

- a) 系数为 0 不输出，指数为 0 直接输出数字。
- b) 系数为 1 则省略系数，指数为 1 则省略指数。
- c) 系数为负数则省略加号。

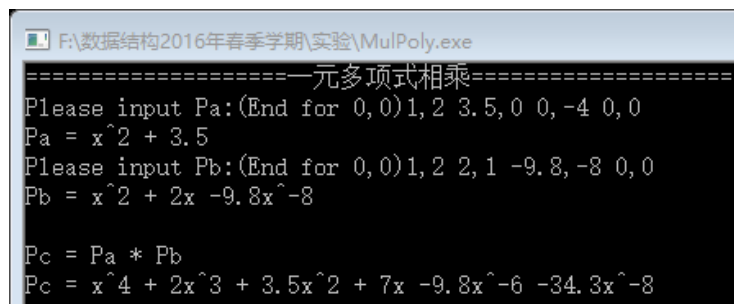
d) 截图：



```
F:\数据结构2016年春季学期\实验\MulPoly.exe
=====一元多项式相乘=====
Please input Pa: (End for 0,0)1, 2 3.5, 0 0, -4 0, 0
Pa = x^2 + 3.5
Please input Pb: (End for 0,0)1, 2 2, 1 -9.8, -8 0, 0
Pb = x^2 + 2x -9.8x^-8
```

(3) 实现功能：创建两个多项式链表，并分别输出多项式的内容。最后对他们进行多项式乘法。

截图：



```
F:\数据结构2016年春季学期\实验\MulPoly.exe
=====一元多项式相乘=====
Please input Pa: (End for 0,0)1, 2 3.5, 0 0, -4 0, 0
Pa = x^2 + 3.5
Please input Pb: (End for 0,0)1, 2 2, 1 -9.8, -8 0, 0
Pb = x^2 + 2x -9.8x^-8

Pc = Pa * Pb
Pc = x^4 + 2x^3 + 3.5x^2 + 7x -9.8x^-6 -34.3x^-8
```

2、概要设计：

(1) 数据结构：不带头结点的链表，每个结点表示一个项，其中 `coef` 表示项的系数，为浮点型；`expn` 表示项的指数，为整型；`next` 表示下一项的位置。按指数的降序排列。

```
typedef struct Poly{
    float coef; //系数
    int expn;   //指数
    struct Poly *next;
}ElemType;
```

(2) 主函数流程：

- 接收数据并创建链表（多项式）Pa，打印 Pa；
- 接收数据并创建链表（多项式）Pb，打印 Pb；
- 创建链表 Pc，进行 $Pa \cdot Pb$ 的运算，将结果输入 Pc，并打印 Pc。

(3) 各程序模块之间的调用关系

```
ElemType *LinkList(); //建立空链表
void DestroyPolyn(ElemType *p); //销毁一元多项式p
void PrintPolyn(ElemType *p); //打印输出一元多项式p
ElemType *CreatPolyn(ElemType *head); //输入系数和指数，建立表示一元多项式的有序链表p
ElemType *ListInsert_L(ElemType *head, ElemType *e); //处理并插入元素e
ElemType *MultiplyPolyn(ElemType *Pa, ElemType *Pb); //相乘
```

其中 `CreatPolyn` 函数的功能只是接受用户系数和指数，其建立链表的功能是通过调用 `ListInsert_L` 函数实现。`ListInsert_L` 函数插入时已经按照指数降序插入。

3、详细设计（主要展示 `ListInsert_L` 函数和相乘 `MultiplyPolyn` 的函数）

(1) `ListInsert_L` 函数

- 按照指数降序插入结点；
- 分插入位置在表头表中表尾三种情况；
- 若查询插入位置时发现指数相同则让指数相加，若相加后指数不为零才插入。

```
ElemType *ListInsert_L(ElemType *head, ElemType *e){
    //处理并插入元素e
    ElemType *p0, *p1, *p2;
    p1 = head;
    p0 = e;

    if(head == NULL){
        head = p0;
        p0->next = NULL;
    } //if
    else{
        while((p0->expn < p1->expn) && (p1->next != NULL)) { //寻找要插入的地方，降序
            p2 = p1; //p2指向的是比插入结点小的结点
            p1 = p1->next; //p1指向的是比插入结点大的结点，综上，p2 > p0 > p1
        } //while
    }
}
```

```

        if(p0->expn > p1->expn){
            if(head == p1) //要插入的地方是开头
                head = p0;
            else
                p2->next = p0;
            p0->next=p1;
        }//if
        else if(p0->expn == p1->expn){
            p1->coef += p0->coef;
            if(p1->coef == 0) { //系数为0, 删除结点
                p2->next = p1->next;
                free(p1);
            }//if
        }//else if
        else{//要插入的地方是结尾
            p1->next = p0;
            p0->next = NULL;
        }
    }//else

    return head;
} //ListInsert_L

```

(2) MultiplyPolyn 函数

```

ElemType *MultiplyPolyn(ElemType *Pa, ElemType *Pb){
    //多项式乘法: Pc = Pa * Pb
    ElemType *ha,*hb,*qa,*qb,*Pc,*c; //qa指向Pa中的当前结点, qb同理
    float ccoef;
    int cexpn;
    ha = Pa;
    hb = Pb;

    if(!Pa || !Pb)
        return NULL;
    Pc = LinkList();
    for(qa = ha; qa; qa = qa->next) { //当qa非空, 逐个检索qa
        for(qb = hb; qb; qb = qb->next) {
            ccoef = qa->coef * qb->coef;
            cexpn = qa->expn + qb->expn;
            if(ccoef != 0) {
                c = (ElemType*)malloc(LEN);
                c->coef = ccoef;
                c->expn = cexpn;
                Pc = ListInsert_L(Pc, c);
            }
        } //for(qb)
    } //for(qa)

    //计算完毕, 释放Pa和Pb
    DestroyPolyn(ha);
    DestroyPolyn(hb);

    return Pc;
} //MultiplyPolyn

```

4、调试分析:

(1) 容错性考虑:

- a. 如打印链表的函数, 若发现传参时空链表, 而没对其进行特殊对待, 那么程序出错。为此输出时先对链表进行非空判断, 对空链表直接输出 0 处理。

```

void PrintPolyn(ElemType *p) { //打印输出一元多项式p
    ElemType *q;
    q = p;

    if (!p) //空表即表达式为0
        printf("0");
}

```

- b. 如相乘函数，若发现 Pa 或 Pb 有一个是空的，那么相乘结果必为 NULL，则直接返回 NULL（在本程序多项式为0视为多项式结果为0），见上面相乘函数的截图。

(2) 对于插入函数，一开始并没有分插入位置在表头表尾的情况，但是调试时发现若链表一开始为空，则应当先建立头结点，这样若不分插入位置的情况，则遇到新建头结点的情况程序就会出错。由于和书上的举例的结构有所不同：书上的多项式的头结点用来存放该多项式的项数，而我没有，主要考虑到程序实现时不必要通过项数来作为循环结束的条件，可以用判断 $p \rightarrow next == NULL$ 进行替代，换言之把头结点用来存放链表信息的做法在本程序中并没有降低算法的复杂度反而增加了空间复杂度，增加了内存的开销。

(3) 时间复杂度分析

- a. void DestroyPolyn(ElemType *p); //销毁一元多项式 P
O (p_length)
- b. void PrintPolyn(ElemType *p); //打印输出一元多项式 P
O (p_length)
- c. ElemType *ListInsert_L(ElemType *head, ElemType *e); //处理并插入元素 e
O (p_length)
- d. ElemType *MultiplyPolyn(ElemType *Pa, ElemType *Pb); //相乘
O (Pa_length * Pb_length)

四、实验结果：

1、一般情况： $(x^2 + 3.5)(x^2 + 2x - 9.8x^{-8}) = x^4 + 2x^3 + 3.5x^2 + 7x - 9.8x^{-6} - 34.3x^{-8}$

```

F:\数据结构2016年春季学期\实验\MulPoly.exe
=====一元多项式相乘=====
Please input Pa:(End for 0,0)1,2 3.5,0 0,-4 0,0
Pa = x^2 + 3.5
Please input Pb:(End for 0,0)1,2 2,1 -9.8,-8 0,0
Pb = x^2 + 2x -9.8x^-8

Pc = Pa * Pb
Pc = x^4 + 2x^3 + 3.5x^2 + 7x -9.8x^-6 -34.3x^-8

```

2、有一个多项式为0： $0 \times (2x^3 + x^2) = 0$

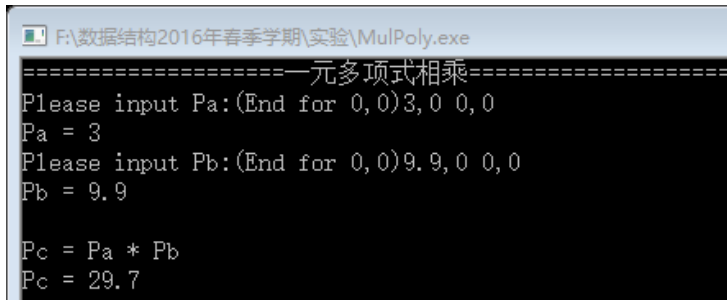
```

F:\数据结构2016年春季学期\实验\MulPoly.exe
=====一元多项式相乘=====
Please input Pa:(End for 0,0)0,3 0,0
Pa = 0
Please input Pb:(End for 0,0)1,2 2,3 0,0
Pb = 2x^3 + x^2

Pc = Pa * Pb
Pc = 0

```

3、常数乘常数：3*9.9=29.7



```
F:\数据结构2016年春季学期\实验\MulPoly.exe
=====一元多项式相乘=====
Please input Pa:(End for 0,0)3,0 0,0
Pa = 3
Please input Pb:(End for 0,0)9.9,0 0,0
Pb = 9.9

Pc = Pa * Pb
Pc = 29.7
```

五、实验总结：

- 1、链表头结点存储链表信息的优点除了可以不需要遍历整个链表来获取链表长度等信息外，还有一个优点就是为插入删除结点等算法的实现提供思路上的便利。当一个结点所消耗的空间不大，程序所需要的链表不多的情况下，采用链表头结点存放链表信息的方式还是值得提倡的。
- 2、这次实验用到的数据结构是链表，链表会经常使用指针，实验时要格外注意指针是如何移动的，搞清楚每个指针变量究竟指向哪里，除了防止指针指向程序以外的内存，更重要的是避免指向其他数据。
- 3、这次实验大部分函数的返回值都是多项式的头指针，实际上可以让返回类型是空类型，传参时多加入一个头指针的参数即可。