

实验报告

1. 实验题目：一元稀疏多项式计算器

2. 需求分析：

1). 实现目的：

设计一个一元稀疏多项式简单计算器，并使其能实现对于多项式的一定计算。

2). 输入规范：

依据用户手册按指令输入，对于多项式输入以指数（空格）系数（空格）指数（空格）系数……的形式输入，最后以换行符终止该多项式输入。

3). 实现基本要求：

(a) 输入多项式，并实现多项式的建立，并以指数降序排列输出，同时输出形式为类数学表达式，如 $-3x^8 + 6x^3 - 18$ （注意：系数值为 1 或 -1 的非零次项的输出形式略去系数 1，如项 $1x^8$ 输出为 x^8 ，项 $-1x^3$ 输出为 $-x^3$ ）；

(b) 多项式 a 和 b 相加，建立多项式 a+b；

(c) 多项式 a 和 b 相减，建立多项式 a-b；

(d) 计算多项式在 x 处的值；

(e) 求多项式 a 的导函数 a'；

(f) 多项式 a 和 b 相乘，建立多项式 a*b；

4). 测试数据：

$$(a)(2x + 5x^8 - 3.1x^{11}) + (7 - 5x^8 + 11x^9) = (-3.1x^{11} + 11x^9 + 2x + 7)$$

$$(b)(6x^{-3} - x + 4.4x^2 - 1.2x^9) - (-6x^{-3} + 5.4x^2 + 7.8x^{15}) = (-7.8x^{15} - 1.2x^9 + 12x^{-3} - x)$$

$$(c)(1 + x + x^2 + x^3 + x^4 + x^5) + (-x^3 - x^4) = (1 + x + x^2 + x^5)$$

$$(d)(x + x^3) + (-x - x^3) = 0$$

$$(e)(x + x^{100}) + (x^{100} + x^{200}) = (x + 2x^{100} + x^{200})$$

$$(f)(x + x^2 + x^3) + 0 = (x + x^2 + x^3)$$

(g) 互换上述测试数据中的前后两项

3. 概要设计：

1). 抽象数据类型多项式的定义如下：

ADT Polynomial{

数据对象：D={ $a_i | a_i \in TermSet, i = 1, 2, \dots, m, m \geq 0$ TermSet 中的每个元素包含一个表示系数的实数和表示指数的整数}

数据关系：R1={ $\langle a_{i-1}, a_i \rangle | a_{i-1}, a_i \in D, \text{且 } a_{i-1} \text{ 中的指数值} < a_i \text{ 中的指数值}, i = 2, 3, \dots, n$ }

基本操作：

CreatPolyn(&List);

操作结果：创建多项式并进行输入排序处理

```
AddPolyn(&Pa,&Pb,&Pab);
```

初始条件：存在多项式 Pa, Pb;

操作结果：实现多项式 Pa 与 Pb 相加并返回 Pab;

```
SubtractPolyn(&Pa,&Pb,&Pab);
```

初始条件：存在多项式 Pa, Pb;

操作结果：实现多项式 Pa 与 Pb 相减并返回 Pab;

```
MultiplyPolyn(&Pa,&Pb,&Pab);
```

初始条件：存在多项式 Pa, Pb;

操作结果：实现多项式 Pa 与 Pb 相乘并返回 Pab;

```
CalcuX(&List,x,result);
```

初始条件：存在多项式 List 并在 x 处有结果值;

操作结果：计算多项式 List 在 x 处的值并返回 result;

```
CalcuDeriva(&List,&D_List);
```

初始条件：存在多项式 List 并有导函数;

操作结果：计算多项式 List 的导函数并返回多项式 D_List;

```
PrinPolyn(&List);
```

初始条件：存在多项式 List;

操作说明：进行多项式的输出操作，并规范相关输出格式;

}ADT Polynomial

2). 主函数:

```
int main(){  
    操作说明;  
    分别输入多项式 Pa,Pb;  
    while(接收操作指令){  
        选择操作并执行;  
    }  
}
```

4. 详细设计:

1). 链表节点类型:

```
typedef float CoefType; //定义系数数据类型  
typedef int ExpnType; //定义指数数据类型  
typedef struct LNode{  
    CoefType coef;  
    ExpnType expn;  
    struct LNode *next;
```

```
} * Polynomial;
```

2). 多项式的基本操作设定:

```
void CreatPolyn(Polynomial List);
    //接收输入的多项式系数及指数，以降序储存到链表 List 中
Polynomial AddPolyn(Polynomial Pa, Polynomial Pb)
    //对多项式 Pa, Pb 实现相加操作，并返回和多项式
Polynomial SubtractPolyn(Polynomial Pa, Polynomial Pb)
    //对多项式 Pa, Pb 实现相减操作，并返回差多项式
Polynomial MultiplyPolyn(Polynomial Pa, Polynomial Pb)
    //对多项式 Pa, Pb 实现相乘操作，并返回积多项式
CoefType CalcuX(Polynomial List, CoefType x)
    //计算多项式 List 在 x 处的值并返回
Polynomial CalcuDeriva(Polynomial List)
    //计算多项式 List 的导函数并返回
void PrinPolyn(Polynomial List)
    //进行多项式的输出操作，并按降序排列
```

其中输入算法如下:

```
void CreatPolyn(Polynomial List){
    while(OK){
        q->coef = x;q->expn = y;//输入并接收系数指数;
        //从已存在的链表节点中，从前向后查找比较，按降序插入;
        p = List->next;p_pre = List;
        if(p == NULL){List->next = q;q->next = p;}
        while(p_pre != NULL){
            if(p == NULL){p_pre->next = q;q->next = p;break;}
            if(q->expn > p->expn){p->coef += q->coef;break;}
            else if(q->expn == p->expn){p->coef += q->coef;break;}
            else{p_pre = p;p = p->next;}}
        //获取输入字符，便于吸收空格符，当接收换行符时多项式输入结束;
        c = getchar();if(c == ' \n ')break;
    }
}
```

多项式相加伪码如下:

```
Polynomial AddPolyn(Polynomial Pa, Polynomial Pb){
    p = Pa->next;q = Pb->next;m = Pab;
```

```

//采取双指针同时历遍，比较指数并相加
while(p || q){
    m->next = n;m = m->next;
    //首先判断一方链表历遍
    if(p == NULL && q != NULL){m->coef = q->coef;m->expn = q->expn;q = q->next;}
    else if(p != NULL && q == NULL){m->coef = p->coef;m->expn = p->expn;p = p->next;}
    else{
        if(p->expn == q->expn)m->coef = p->coef + q->coef;m->expn = p->expn;p = p->next;q = q->next;
        else if(p->expn > q->expn){m->coef = p->coef;m->expn = p->expn;p = p->next;}
        else if(p->expn < q->expn){m->coef = q->coef;m->expn = q->expn;q = q->next;}
    }
}
m->next = NULL;return Pab;
}

```

多项式相减伪码如下：

```

Polynomial SubtractPolyn(Polynomial Pa,Polynomial Pb){
    p = Pa->next;q = Pb->next;m = Pab;
    //同多项式相加的操作
    while(p || q){
        m->next = n;m = m->next;
        //首先判断一方链表历遍
        if(p == NULL && q != NULL){m->coef = -q->coef;m->expn = q->expn;q = q->next;}
        else if(p != NULL && q == NULL){m->coef = p->coef;m->expn = p->expn;p = p->next;}
        else{
            if(p->expn == q->expn){m->coef = p->coef - q->coef;m->expn = p->expn;p = p->next;q = q->next;}
            else if(p->expn > q->expn){m->coef = p->coef;m->expn = p->expn;p = p->next;}
            else if(p->expn < q->expn){m->coef = -q->coef;m->expn = q->expn;q = q->next;}
        }
    }
    m->next = NULL;return Pab;
}

```

多项式相乘伪码如下：

```

Polynomial MultiplyPolyn(Polynomial Pa, Polynomial Pb){

```

//采取两个链表作为主辅，辅链表中不断存储 Pb 中每一项与 Pa 相乘的多项式，并加到主链表上然后释放内存

```

m = Pab;m->next = n;m = m->next;mf = PabF;//主辅链表
m->coef = 0;m->expn = 0;m->next = NULL;
p = Pa->next;q = Pb->next;
while(q){
    while(p){
        mf->next = n;mf = mf->next;
        mf->coef = p->coef * q->coef;mf->expn = p->expn + q->expn;p = p->next;
    }
    mf->next = NULL;p = Pa->next;Pab = AddPolyn(Pab, PabF);
    //初始化 PabF 为空，释放内存
    mf = PabF->next;mf_next = mf->next;
    while (mf_next){
        free(mf);mf = mf_next;mf_next = mf_next->next;
    }
}return Pab;
}

```

计算多项式在 x 处的值伪码如下：

```

CoefType CalcuX(Polynomial List, CoefType x){
    result = 0;p = List->next;
    while (p != NULL){
        result += p->coef * pow(x, p->expn);
        p = p->next;
    }return result;
}

```

计算多项式的导函数伪码如下：

```

Polynomial CalcuDeriva(Polynomial List){
    p = List->next;q = D_List;
    while(p != NULL){
        q->next = m;q = q->next;
        if(p->expn == 0){q->coef = 0;q->expn = 0}
        else{q->coef = p->coef * p->expn;q->expn = p->expn - 1;}
        q->next = NULL;
    }
    q->next = NULL;return D_List;
}

```

输出操作伪码如下：

```
void PrinPolyn(Polynomial List){
    p = List->next;
    //清除链表前面系数为零的节点，便于首位无符号输出，同时考虑结果为零的链表
    while(p->coef == 0 && p->next != NULL) p = p->next;

    //考虑首位输出
    if(p->expn == 0 || p->coef == 0) printf("%g", p->coef);
    else if (p->expn == 1 && p->coef != 1) printf("%gx", p->coef);
    else if (p->coef == 1 && p->expn != 1) printf("x^ %d", p->expn);
    else if (p->coef == 1 && p->expn == 1) printf("x");
    else if (p->coef == -1 && p->expn != 1) printf("-x^%d", p->expn);
    else if (p->coef == -1 && p->expn == 1) printf("-x");
    else printf("%gx%d", p->coef, p->expn);
    p = p->next;
    while (p != NULL){
        if (p->coef == 0){p = p->next;continue;}
        else{
            if (p->expn == 0) printf("%+g", p->coef);
            else if (p->expn == 1 && p->coef != 1 && p->coef != -1) printf("%+gx", p->coef);
            else if (p->coef == 1 && p->expn != 1) printf("+x^%d", p->expn);
            else if (p->coef == 1 && p->expn == 1) printf("+x");
            else if (p->coef == -1 && p->expn != 1) printf("-x^%d", p->expn);
            else if (p->coef == -1 && p->expn == 1) printf("-x");
            else printf("%+gx^%d", p->coef, p->expn);
        }
        p = p->next;
    }printf("\n");
}
```

主程序伪码如下：

```
int main(){
    指令操作说明
    CreatPolyn(Pa);PrinPolyn(Pa);
    CreatPolyn(Pb);PrinPolyn(Pb);
    while(((choose = getchar()) != '#')){
        switch(choose){
            case 'a': P = AddPolyn(Pa, Pb);
            printf("Pa+Pb 结果多项式表示为:\n");PrinPolyn(P);break;
```

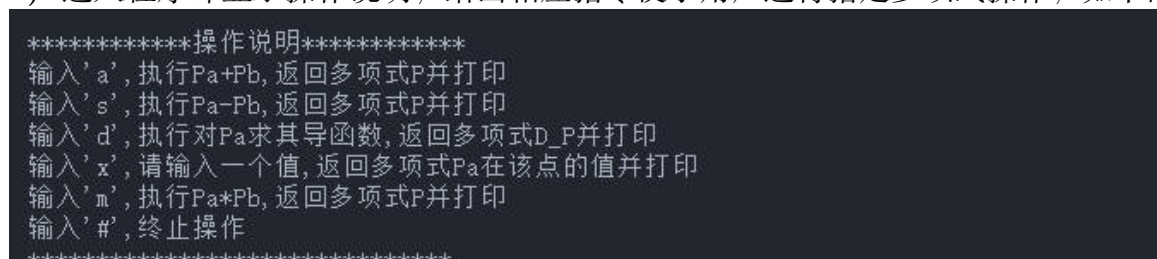
```
case 's': P = SubtractPolyn(Pa, Pb);
printf("Pa-Pb 结果多项式表示为:\n");PrinPolyn(P);break;
case 'd': D_P = CalcuDeriva(Pa);
printf("Pa 导函数多项式表示为:\n");PrinPolyn(D_P);break;
case 'x': printf(" 请输入需要计算处 x0 的值 \n");scanf("%d", &x0);X = CalcuX(Pa,
x0);
printf(" 多项式 Pa 在%d 处的值为%d\n", x0, X);break;
case 'm': P = MultiplyPolyn(Pa, Pb);
printf("Pa*Pb 结果多项式表示为:\n");PrinPolyn(P);break;
default: break;
}
}
free(Pa);free(Pb);
return 0;
}
```

5. 调试分析：

- 1). 本次作业主要是采取链表形式储存多项式，并实现相关的多项式运算。其中调试中出现的问题往往在于对于链表表头表尾特殊情况的考虑不周，其余问题相较而言较为简单。
- 2). 程序对于多项式的输出形式采取分类讨论，以便满足输出形式；
- 3). 其中主要算法 CreatPolyn 与 MultiplyPolyn 的时间复杂度为 $O(n^2)$ ，算法 AddPolyn 与 SubtractPolyn 时间复杂度为 $O(n)$ ；
- 4). 经验体会：借助调试功能可以清楚了解程序运行过程中的变量值便于校准；

6. 用户手册：

- 1). 本程序的运行环境为 DOS 操作系统，执行文件为 Polynomial.exe;
- 2). 进入程序即显示操作说明，给出相应指令便于用户进行指定多项式操作，如下图所示



```
*****操作说明*****
输入'a', 执行Pa+Pb, 返回多项式P并打印
输入's', 执行Pa-Pb, 返回多项式P并打印
输入'd', 执行对Pa求其导函数, 返回多项式D_P并打印
输入'x', 请输入一个值, 返回多项式Pa在该点的值并打印
输入'm', 执行Pa*Pb, 返回多项式P并打印
输入'#', 终止操作
*****
```

之后显示“请以系数 指数 系数 指数 的形式输入多项式 Pa/Pb”，提示用户输入多项式；输入后即可打印多项式，如下图所示

```

请以系数 指数 系数 指数~的形式输入多项式Pa
1 1
打印多项式Pa如下：
x
请以系数 指数 系数 指数~的形式输入多项式Pb
2 2
打印多项式Pb如下：
2x^2

```

然后提示用户“请输入操作指令”，并给出打印相应的计算结果

```

请输入操作指令
a
Pa+Pb结果多项式表示为：
2x^2+x
s
Pa-Pb结果多项式表示为：
-2x^2+x
d
Pa导函数多项式表示为：
1
x
请输入需要计算处x0的值
2
多项式Pa在2处的值为2
m
Pa*Pb结果多项式表示为：
2x^3
#
*****
* 按任意键关闭终端。

```

7. 测试结果：

1). 第一项测试数据如下：

```

请以系数 指数 系数 指数~的形式输入多项式Pa
2 1 5 8 -3.1 11
打印多项式Pa如下：
-3.1x^11+5x^8+2x
请以系数 指数 系数 指数~的形式输入多项式Pb
7 0 -5 8 11 9
打印多项式Pb如下：
11x^9-5x^8+7
请输入操作指令
a
Pa+Pb结果多项式表示为：
-3.1x^11+11x^9+2x+7

```

2). 第二项测试数据如下：


```

请以系数 指数 系数 指数~的形式输入多项式Pa
6 -3 -1 1 4.4 2 -1.2 9
打印多项式Pa如下：
-1.2x^9+4.4x^2-x+6x^-3
请以系数 指数 系数 指数~的形式输入多项式Pb
-6 -3 5.4 2 -1 2 7.8 15
打印多项式Pb如下：
7.8x^15+4.4x^2-6x^-3
请输入操作指令
s
Pa-Pb结果多项式表示为：
-7.8x^15-1.2x^9-x+12x^-3

```

3). 第三项测试数据如下：

```

请以系数 指数 系数 指数~的形式输入多项式Pa
1 0 1 1 1 2 1 3 1 4 1 5
打印多项式Pa如下：
x^5+x^4+x^3+x^2+x+1
请以系数 指数 系数 指数~的形式输入多项式Pb
-1 3 -1 4
打印多项式Pb如下：
-x^4-x^3
请输入操作指令
a
Pa+Pb结果多项式表示为：
x^5+x^2+x+1

```

4). 第四项测试数据如下：

```

请以系数 指数 系数 指数~的形式输入多项式Pa
1 1 1 3
打印多项式Pa如下：
x^3+x
请以系数 指数 系数 指数~的形式输入多项式Pb
-1 1 -1 3
打印多项式Pb如下：
-x^3-x
请输入操作指令
a
Pa+Pb结果多项式表示为：
0

```

5). 第五项测试数据如下：

```

请以系数 指数 系数 指数~的形式输入多项式Pa
1 1 1 100
打印多项式Pa如下：
x^100+x
请以系数 指数 系数 指数~的形式输入多项式Pb
1 100 1 200
打印多项式Pb如下：
x^200+x^100
请输入操作指令
a
Pa+Pb结果多项式表示为：
x^200+2x^100+x

```

6). 第六项测试数据如下：

```
请以系数 指数 系数 指数~的形式输入多项式Pa
1 1 1 2 1 3
打印多项式Pa如下：
x^3+x^2+x
请以系数 指数 系数 指数~的形式输入多项式Pb
0 0
打印多项式Pb如下：
0
请输入操作指令
a
Pa+Pb结果多项式表示为：
x^3+x^2+x
```

7). 第七项测试，在数据互换后，经检验运行结果正确，不做赘述；

8. 附录：

源程序文件名清单：

Polynomial.c //主程序

Polynomial.h //多项式结点类型