数据结构实验报告 1

学号：2022114467　　**姓名：**林泽宇　　**专业：**软件工程

知识范畴：线性表　　**完成日期：**2023 年 9 月 22 日

实验题目：基于链式存储的一元 n 次多项式相乘

课程目标 1（60%）	课程目标 2（40%）	得分(100 分) 期末成绩占比 10%	批阅人签字
程序代码及测试	写作、排版、代码注释等		

实验内容及要求：

从字符文件输入两个多项式的非零系数及对应的指数，建立多项式的链式存储结构，计算这两个多项式的乘积，输出乘积多项式的全部非零系数及对应的指数到另一字符文件中。

要求输入输出字符文件中的数据格式自拟；编程语言采用 C/C++。

实验目的：掌握单向链表的基本操作以及基于链表的多项式加法与乘法。

数据结构设计简要描述：

采用带附加头结点链表存储多项式,，附加头节点的 ecof 存储多项式项数；

此外其余节点包括双精度浮点型的系数和长整形的指数的数据域和一个指针域。

```
typedef struct Polynomial //多项式结构体
{
    double ecof;
    long long int expo;
    struct Polynomial *next;
} PolynTerm, *PolynPtr;
```

算法设计简要描述：

在实现多项式加法与乘法前，先实现格式化多项式函数，通过链表的插入排序实现逆序排序各个节点。

多项式加法通过先将待相加多项式的项逐个插入目标多项式，后格式化目标多项式实现。

多项式乘法通过先将待相乘多项式逐项相乘的结果插入目标多项式，后格式化目标多项式实现。

输入/输出设计简要描述：

输入共 $n+m+2$ 行，从标准输入读入数据，键盘输入。

输入的第一行包含一个整数 n，代表第一个多项式的项数。

接下来 n 行，每行包含一个实数 $ecof_i$ 和一个整数 $expo_i$。$(1 \le i \le n)$

输入的第一行包含一个整数 m，代表第二个多项式的项数。

接下来 m 行，每行包含一个实数 $ecof_i$ 和一个整数 $expo_i$。$(1 \le i \le m)$

输出的多项式以加减号相连，每一项均由系数，变量以及指数组成，形如 $ax\wedge b$，若系数为 1 则不输出系数，若指数为 1 则形如 ax ，若指数为 0 则只输出系数，若指数为负数则指数带括号。输出的系数均保留两位小数。

数据修约：

$$1 \le n,m \le LONG\_LONG\_MAX$$

$$sqrt(LONG\_LONG\_MIN) \le ecof_i \le sqrt(LONG\_LONG\_MAX)$$

$$LONG\_LONG\_MIN/2 \le expo_i \le LONG\_LONG\_MAX/2$$

编程语言说明：

使用 C++编程。 主要代码采用 C 语言实现 ；动态存储分配采用 C++的 new 和 delete 操作符实现；输入与输出采用 C++的 ifstream 和 ofstream 流；程序注释采用 C/C++规范。变量命名采用 Google 规范，函数名采用驼峰命名法，对象/结构体采用帕斯卡命名法。

主要函数说明：

```
void createPolyn(PolynPtr &head) //新建空多项式
```
时间复杂度: O(1), 空间复杂度: O(1).

```
void formatPolyn(PolynPtr &head) //格式化多项式(合并同次项并按照指数降序排列)
```
时间复杂度: $O(n^2)$, 空间复杂度: O(n).

```
PolynPtr inputPolyn(PolynPtr &head) //通过输入创建一个多项式
```
时间复杂度: O(n), 空间复杂度: O(n).

```
void outputPolyn(PolynPtr p) //输出多项式
```
时间复杂度: O(n), 空间复杂度: O(1).

```
PolynPtr addPolyn(PolynPtr pah, PolynPtr pbh) //多项式加法
```
时间复杂度: O(n), 空间复杂度: O(n).

```
PolynPtr multiplyPolyn(PolynPtr pah, PolynPtr pbh) // 多项式乘法
```
时间复杂度: $O(n^2)$, 空间复杂度: $O(n^2)$.

程序测试简要报告：

(1) 测试样例 1：基本测试

Testcase 1 Passed 17ms

Input: Copy

```
2
1.1 2
3.2 0
2
-9 1
23 -2
```

Received Output: Copy

```
多项式A为:
该多项式有2项
1.10x^2 + 3.20
多项式B为:
该多项式有2项
-9.00x + 23.00x^(-2)
两个多项式相加的结果为:
该多项式有4项
1.10x^2 - 9.00x + 3.20 + 23.00x^(-2)
两个多项式相乘的结果为:
该多项式有4项
-9.90x^3 - 28.80x + 25.30 + 73.60x^(-2)
```

结论

程序可以在小数据范围内输出准确结果

(2) 测试样例 2：长数据测试

Testcase 2 Passed 16ms

Input: Copy

```
2
99999999999999999999999999.99 99999999999999
9999999 0
2
9999999999999999.9 999999999999999
-9999 -999
```

Received Output: Copy

```
多项式A为:
该多项式有2项
10000000000000000004764729344.00x^99999999999999 + 9999999.00
多项式B为:
该多项式有2项
10000000000000000.00x^999999999999999 - 9999.00x^(-999)
两个多项式相加的结果为:
该多项式有4项
10000000000000000.00x^999999999999999 +
10000000000000000004764729344.00x^99999999999999 + 9999999.00 -9999.00x^(-999)
两个多项式相乘的结果为:
该多项式有4项
10000000000000000013937211695941409913071206 4.00x^1099999999999998 +
99999989999999994193510 4.00x^999999999999999 -
99990000000000003398453270937 6.00x^99999999999000 - 99989990001.00x^(-999)
```

结论

程序可以在较大数据范围内输出正确结果

(3) 测试样例 3 ： 复杂数据测试

Testcase 3 Passed 16ms

Input:

```
9
-1.2 9
99.23 8
-100.9999900001 7
222224 6
2 0
1000 23426
92.2 238
-1.11 22
19 0
3
1 200
-99 0
100.000000001 1
```

Received Output:

```
多项式A为:
该多项式有8项
1000.00x^23426 + 92.20x^238 - 1.11x^22 - 1.20x^9 + 99.23x^8 - 101.00x^7 + 222224.00x^6 + 21.00
多项式B为:
该多项式有3项
x^200 + 100.00x - 99.00
两个多项式相加的结果为:
该多项式有10项
1000.00x^23426 + 92.20x^238 + x^200 - 1.11x^22 - 1.20x^9 + 99.23x^8 - 101.00x^7 + 222224.00x^6 +
100.00x - 78.00
两个多项式相乘的结果为:
该多项式有21项
1000.00x^23626 + 100000.00x^23427 - 99000.00x^23426 + 92.20x^438 + 9220.00x^239 - 9127.80x^238 -
1.11x^222 - 1.20x^209 + 99.23x^208 - 101.00x^207 + 222224.00x^206 + 21.00x^200 - 111.00x^23 +
109.89x^22 - 120.00x^10 + 10041.80x^9 - 19923.77x^8 + 22232399.00x^7 - 22000176.00x^6 + 2100.00x -
2079.00
```

结论

程序可以在复杂数据下输出正确结果

源程序代码：

```
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;
ifstream inputFile;
ofstream outputFile;
typedef struct Polynomial //多项式结构体
{
    double ecof;             //双精度浮点型存储系数
    long long int expo;      //整型存储指数
    struct Polynomial *next; //指针域
} PolynTerm, *PolynPtr;
void createPolyn(PolynPtr &head) //新建空多项式
{
    head = new PolynTerm;
}
```

```
void formatPolyn(PolynPtr &head) //格式化多项式(合并同次项并按照指数降序排列)
{
    PolynPtr ohead = new Polynomial;
    ohead->next = head->next; //存储未排序数据
    head->next = nullptr;     //置空原多项式，方便插入
    PolynPtr p = head, pr = ohead->next;
    while (pr) //当仍有数据未插入
    {
        PolynPtr tmp = new PolynTerm;
        tmp->ecof = pr->ecof;
        tmp->expo = pr->expo;
        while (p) //寻找插入位置
        {
            if (!p->next) //表尾处直接插入
            {
                tmp->next = nullptr;
                p->next = tmp;
                break;
            }
            if (p->next->expo == pr->expo) //若指数相同则合并
            {
                p->next->ecof += pr->ecof;
                --head->ecof;       //合并后项数减一
                if (!p->next->ecof) //如果合并后系数为0则删除
                {
                    p->next = p->next->next;
                    --head->ecof; //项数再减一
                }
                break;
            }
            if (p->next->expo < pr->expo) // 满足条件插入
            {
                tmp->next = p->next;
                p->next = tmp;
                break;
            }
            p = p->next;
        }
        p = head, pr = pr->next;
    }
    for (int i = 0; i < head->ecof; ++i) //记得标记表尾
        p = p->next;
    p->next = nullptr;
}
PolynPtr inputPolyn(PolynPtr &head) //通过输入创建一个多项式
{
    head->ecof = 0;
```

```
    while (head->ecof <= 0) //项数为 0 则重新输入
    {
        inputFile >> head->ecof;
        if (!head->ecof)
            outputFile << "项数必须为正数!\n", outputFile.flush();
    }
    PolynPtr p = head; // p 指针作为哨兵
    for (int i = 0; i < head->ecof; ++i)
    {
        PolynPtr tmp = new PolynTerm;
        outputFile.flush();
        inputFile >> tmp->ecof >> tmp->expo;
        if (tmp->ecof) //系数不能为 0
            p->next = tmp, p = p->next;
        else
        {
            outputFile << "该项系数为 0, 请重新输入!\n";
            outputFile.flush();
            --i;
        }
    }
    p->next = nullptr;
    formatPolyn(head);
    return head;
}
void outputPolyn(PolynPtr p) //输出多项式
{
    outputFile << "该多项式有" << p->ecof << "项" << endl;
    p = p->next;
    if (!p)
    {
        outputFile << "该多项式为空\n";
        outputFile.flush();
        return;
    }
    if (p->ecof < 0) //数据和符号分开输出
        outputFile << "-";
    while (p) //已经保证系数不为 0
    {
        if (p->expo == 0) //指数为 0, 只输出系数
        {
            outputFile << abs(p->ecof);
            // printf("%.2lf", abs(p->ecof)); //只输出绝对值
            p = p->next;
            if (p) //处理下一项符号
                if (p->ecof > 0)
                    outputFile << " + ";
```

```
                else if (p->ecof < 0)
                    outputFile << " -";
            continue; //只输出系数
        }
        if (abs(p->ecof) != 1) //除非系数为正负 1，否则输出系数
            outputFile << abs(p->ecof);
        // printf("%.2lf", abs(p->ecof));
        if (p->expo == 1) //输出变量
            outputFile << "x";
        else if (p->expo > 0)
            outputFile << "x^" << p->expo;
        else
            outputFile << "x^(" << p->expo << ')';
        p = p->next;
        if (p) //处理下一项符号
            if (p->ecof > 0)
                outputFile << " + ";
            else if (p->ecof < 0)
                outputFile << " - ";
    }
    outputFile << '\n';
}
PolynPtr addPolyn(PolynPtr pah, PolynPtr pbh) //多项式加法
{
    PolynPtr pa = pah->next, pb = pbh->next;
    PolynPtr res = new PolynTerm, p = res;
    res->ecof = pah->ecof + pbh->ecof; //项数相加
    while (pa)                          //先把 A 表所有项导入
    {
        PolynPtr tmp = new PolynTerm;
        tmp->ecof = pa->ecof;
        tmp->expo = pa->expo;
        p->next = tmp;
        p = p->next, pa = pa->next;
    }
    while (pb) //再导入 B 表
    {
        PolynPtr tmp = new PolynTerm;
        tmp->ecof = pb->ecof;
        tmp->expo = pb->expo;
        p->next = tmp;
        p = p->next, pb = pb->next;
    }
    p->next = nullptr;
    formatPolyn(res); //最后格式化多项式
    return res;
}
```

```
PolynPtr multiplyPolyn(PolynPtr pah, PolynPtr pbh) // 多项式乘法
{
    PolynPtr pa = pah->next, pb = pbh->next;
    PolynPtr res = new PolynTerm, p = res;
    res->ecof = pah->ecof * pbh->ecof; //项数相乘
    while (pa)
    {
        while (pb) //逐项相乘
        {
            PolynPtr tmp = new PolynTerm;
            tmp->ecof = pa->ecof * pb->ecof; //系数相乘
            tmp->expo = pa->expo + pb->expo; //指数相加
            p->next = tmp;
            p = p->next;
            pb = pb->next;
        }
        pa = pa->next;
        pb = pbh->next;
    }
    p->next = nullptr;
    formatPolyn(res); //格式化多项式
    return res;
}
int main()
{
    //设置输入输出文件，控制精度
    inputFile.open("input.txt");
    outputFile.open("output.txt");
    outputFile << fixed << setprecision(2);
    ios_base::sync_with_stdio(0);
    inputFile.tie(0), outputFile.tie(0);

    //先建表
    PolynPtr polyn_A, polyn_B, polyn_C, polyn_D;
    createPolyn(polyn_A), createPolyn(polyn_B), createPolyn(polyn_C),
createPolyn(polyn_D);
    inputPolyn(polyn_A);
    inputPolyn(polyn_B);

    outputFile << "多项式 A 为: \n";
    outputFile.flush();
    outputPolyn(polyn_A);
    outputFile << "多项式 B 为: \n";
    outputFile.flush();
    outputPolyn(polyn_B);

    polyn_C = addPolyn(polyn_A, polyn_B);
```

```
    outputFile << "两个多项式相加的结果为: \n";
    outputFile.flush();
    outputPolyn(polyn_C);

    polyn_D = multiplyPolyn(polyn_A, polyn_B);
    outputFile << "两个多项式相乘的结果为: \n";
    outputFile.flush();
    outputPolyn(polyn_D);

    outputFile.close();
    return 0;
}
```