```
////////////////////////////////////////////////////////////////////////////////
///////////////////////////
//   链表一元多项式的类实现
//   Author：Melissa M.CAO
//   Belong：Section of software theory，School of Computer Engineering & Science，
Shanghai University
//   Version： 1.0
////////////////////////////////////////////////////////////////////////////////
///////////////////////////
#include "StdAfx.h"
#include "LinkList.cpp"//因为包含该项，故测试函数亦放在此文件中，若置于
MCAOTest.cpp中，则对LinkList的定义是重复的
#include "polynomial.h"

////////////////////////////////////////////////////////////////////////////////
///////////////////////////
//   链表一元多项式+重载的类实现
////////////////////////////////////////////////////////////////////////////////
///////////////////////////
polynomial & operator + (polynomial &a , polynomial &b)
{

    node<term> * pa,*pb,*pc,*p,*c;
    term at,bt;
    pc = a.poly.Reset(0);
    c = pc;
    p = b.poly.Reset(0);
    pa = a.poly.Reset(1);
    pb = b.poly.Reset(1);
    //delete p; //如果此处删除b链表头结点，则析沟函数需要重写，否则，无法析构b
    while (!a.poly.EndofList() && !b.poly.EndofList())
    {
        int i=0;
        at = pa->data;
        bt = pb->data;
        if ( at.exp > bt.exp)  i = 1;
        if (at.exp < bt.exp)    i = -1;
        switch (i)
        {
            case 0:  //  case'=='
                at.coef = at.coef + bt.coef;
                p = pb;
                pb = b.poly.Next();
                delete  p;
                if (abs(at.coef) < 0.0001)
                {
                     p = pa;
                     pa = a.poly.Next();
                     delete p;
                }
                else
                {
                     pa->data = at;
                     pc->SetNext(pa);
                     pc = pa;
                     pa = a.poly.Next();
                }
            break;

            case 1:  // case '>'
                pc->SetNext(pa);
                pc = pa;
                pa = a.poly.Next();
            break;

            case -1:  // case '<'
               pc->SetNext(pb);
               pc = pb;
               pb = b.poly.Next();
            break;
        }
    }
    if ( !a.poly.EndofList())
        pc->SetNext(pa);
    else
        pc->SetNext(pb);

    p = b.poly.Reset(0);//因为合并的结果置于a链表，故b链表头指针的next域置空，
否则，析沟时无法判断b链表为空
    p->SetNext(NULL);

    return a;
}

////////////////////////////////////////////////////////////////////////////////
///////////////////////////
//    链表一元多项式-重载的类实现
//    算法思想：第二个多项式每项系统取反，与第一个多项式相加
//    思考题：此算法先将多项式b的每一项系数取反，再调用多项式加法来实现减法。代码
简单好写，效率如何？
//            能否重复加法的代码？何处进行修改？
////////////////////////////////////////////////////////////////////////////////
///////////////////////////
polynomial & operator - (polynomial &a , polynomial &b)
{
    b.change();
    return a + b;
}

////////////////////////////////////////////////////////////////////////////////
///////////////////////////
//  链表一元多项式*重载的类实现
//  算法思想：第一个多项式的每一项乘以第二个多项式，累加
////////////////////////////////////////////////////////////////////////////////
///////////////////////////
polynomial & operator * (polynomial &a , polynomial &b)
{
    polynomial c(0);
    node<term> *p;
    p = a.poly.Reset(1);

    while (p)
    {
        polynomial d(0);
        d.Copy(b,0);
        d.MultipleOneTerm(p);
        c = c + d;
        p = a.poly.Next();
    }
    a.Copy(c,1);
    return a;
}

////////////////////////////////////////////////////////////////////////////////
///////////////////////////
//    链表一元多项式复制操作的类实现
//    tt：为0时多项式为空，直接复制；为1时先把多项式清0，再复制copy多项式的内
容
////////////////////////////////////////////////////////////////////////////////
///////////////////////////
void polynomial::Copy(polynomial &copy, int tt)
{
    node<term> *p;
    if (tt == 1)
    {
        poly.Clear();
    }
    p = copy.poly.Reset(1);
    while (p)
    {
        poly.Append(p->data);
        p = copy.poly.Next();
    }
}

////////////////////////////////////////////////////////////////////////////////
///////////////////////////
//   链表一元多项式输出<<重载的类实现
////////////////////////////////////////////////////////////////////////////////
///////////////////////////
ostream & operator << (ostream & output,polynomial & c)
{
    node<term> *pa;
    term at;
    pa = c.poly.Reset(1);
    while (!c.poly.EndofList())
    {
        at = pa->data;
        output << at.coef << ″ X^″ << at.exp;
        pa = c.poly.Next();
        if (pa && pa->data.coef > 0)
            output << ″+″;
    }
    return output;
}

////////////////////////////////////////////////////////////////////////////////
///////////////////////////
//   链表一元多项式构造函数的实现
////////////////////////////////////////////////////////////////////////////////
///////////////////////////
polynomial :: polynomial()
{
    term c;
    double x,x1;
    int y, z = 100000;
    cout << "Please input a float which means polynomial(coef) is end：";
    cin >> x1;
    cout << "Please input every term of the polynomial(coef, exp)：";
    cin >> x;
    while (x != 0 && x!= x1)
    {
        cin >> y;
        while (y >= z || y < 0)
        {
                cout << "输入不合理，要求指数从大到小有序，且暂不考虑 x 的负次方！
请重新输入：" << endl;
                cin >> x >> y;
        }
        c.init(x,y);
        poly.InsertAfter(c);
        z = y;
        cin >> x;
    }
}

////////////////////////////////////////////////////////////////////////////////
///////////////////////////
//   链表一元多项式构造函数的实现
//  后面补充函数：n 用来控制类型，为 0 时初始化一个空链表/空多项式即可；不为 0 则由
用户输入系数和指数来建立多项式
////////////////////////////////////////////////////////////////////////////////
///////////////////////////
polynomial :: polynomial(int n)
{
    term temp;
    temp.init(0,0);

    if (n == 0)
    {
        poly.head = new node<term>(temp);
    }
    else
    {
        term c;
        double x,x1;
        int y, z = 100000;
        cout << "Please input a float which means polynomial(coef) is end：";
        cin >> x1;
        cout << "Please input every term of the polynomial(coef, exp)：";
        cin >> x;
        while (x != 0 && x!= x1)
        {
            cin >> y;
            while (y >= z || y < 0)
            {
                    cout <<"输入不合理，要求指数从大到小有序，且暂不考虑x的负次
方！请重新输入：" << endl;
                    cin >> x >> y;
            }
            c.init(x,y);
            poly.InsertAfter(c);
            z = y;
            cin >> x;
        }
    }// end of else
}

////////////////////////////////////////////////////////////////////////////////
///////////////////////////
//    链表一元多项式每一项系数取反操作的实现
////////////////////////////////////////////////////////////////////////////////
///////////////////////////
void polynomial :: change()
{
    node<term> * p;
    p = poly.Reset(1);
    while (p)
    {
        p->data.coef = -1 * (p->data.coef);
        p = poly.Next();
    }
}

////////////////////////////////////////////////////////////////////////////////
///////////////////////////
//    链表一元多项式每一项与固定一项p相乘操作的实现
////////////////////////////////////////////////////////////////////////////////
///////////////////////////
void polynomial :: MultipleOneTerm(node<term> *q)
{
    node<term> * p;
    p = poly.Reset(1);
    while (p)
    {
        p->data.coef = q->data.coef * (p->data.coef);
        p->data.exp = q->data.exp + p->data.exp;
        p = poly.Next();
    }
}

////////////////////////////////////////////////////////////////////////////////
///////////////////////////
//   链表一元多项式求值函数的实现
////////////////////////////////////////////////////////////////////////////////
///////////////////////////
double polynomial :: calculation(double x)
{
    double f = 0;
    double temp;
    term t;

    node<term> * p;
    p = poly.Reset(1);
    while (p)
    {
        //请同学们自己完成
    }

    return f;
}

////////////////////////////////////////////////////////////////////////////////
///////////////////////////
//   链表一元多项式的测试函数
////////////////////////////////////////////////////////////////////////////////
///////////////////////////
void polynomialTest()
{
    int select;
    double x;
    char con = 'Y';
    //构造第一个多项式
    cout << "构造第一个多项式：";
    polynomial a;
    //构造第二个多项式
    cout << "构造第二个多项式：";
    polynomial b;
```

```
cout << "第一个多项式：" << a << endl;
cout << "第二个多项式：" << b << endl;

while (con != 'n' && con != 'N')
{
    cout << "输入操作选择：1--求和；" << endl;
    cout << "              2--分别求两个多项式的值；" << endl;
    cout << "              3--求差；" << endl;
    cout << "              4--求积；" << endl;
    cout << "              5--求导；" << endl;
    cout << "请输入您的选择：";
    cin >> select;
    switch (select)
    {
        case 1:
            cout << "两个多项式的和为：" << a+b << endl;
        break;

        case 2:
            cout << "请输入多项式中参数 x 的值，计算多项式：";
            cin >> x;
            cout << "第一个多项式的值为：" << a.calculation(x) << endl;
            cout << "第二个多项式的值为：" << b.calculation(x) << endl;
        break;

        case 3:
            cout << "两个多项式的差为：" << a-b << endl;
        break;

        case 4:
            cout << "两个多项式的积为：" << a*b << endl;
        break;

        case 5:
            cout << "此功能尚未实现，请有兴趣的同学自行补充！" << endl;
            //cout << "第一个多项式求导结果为：" << a.calculation(x) << endl;
            //cout << "第二个多项式求导结果为：" << b.calculation(x) << endl;
        break;

        default:
            cout << "选择的操作不存在！";
        break;
    }
```

```
        cout << "还需要继续测试吗（Y/N）？";
        cin >> con;
    }
} /**/
```