# 数据结构第一次作业

沈琢乔 16020021031
孙洪超 16020021035

## 一． 问题分析

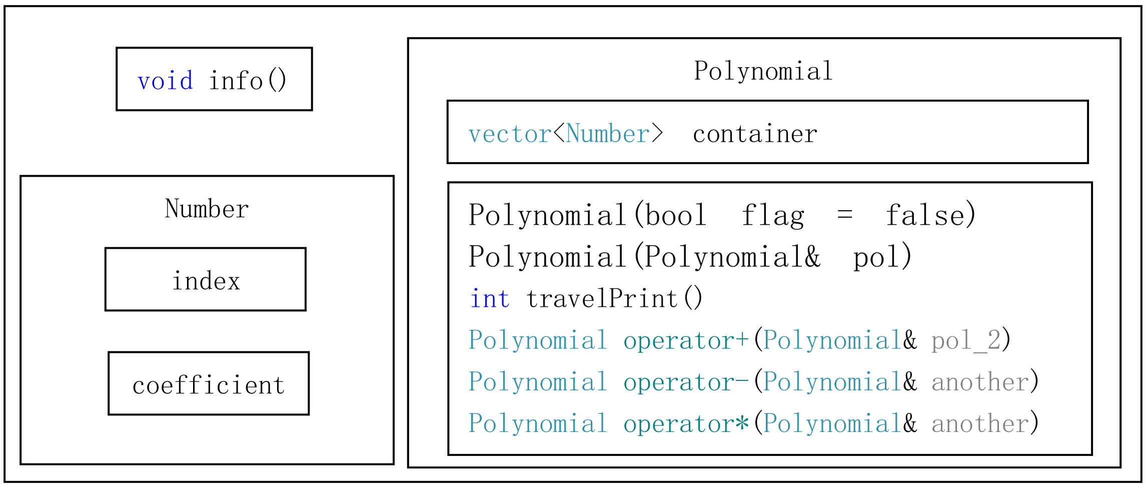
**作业内容：**使用 c++ 标准模版库 list 或 vector 容器，实现一元多项式的初始化、遍历打印、加法、减法及乘法操作。

**分析：**需要使用 list 或 vector 作为类的变量来表示指数系数，根据定义打印输出，做加减法和乘法。

## 二． 解决方法

List 实现(孙洪超) 类中定义两个 list，分别储存指数和系数，按照指数大小排列指数和系数，加法减法把同指数的相加减，不同指数的直接（或乘-1）加到新对象中，乘法按照定义每项系数相乘，指数相加，再次排序。

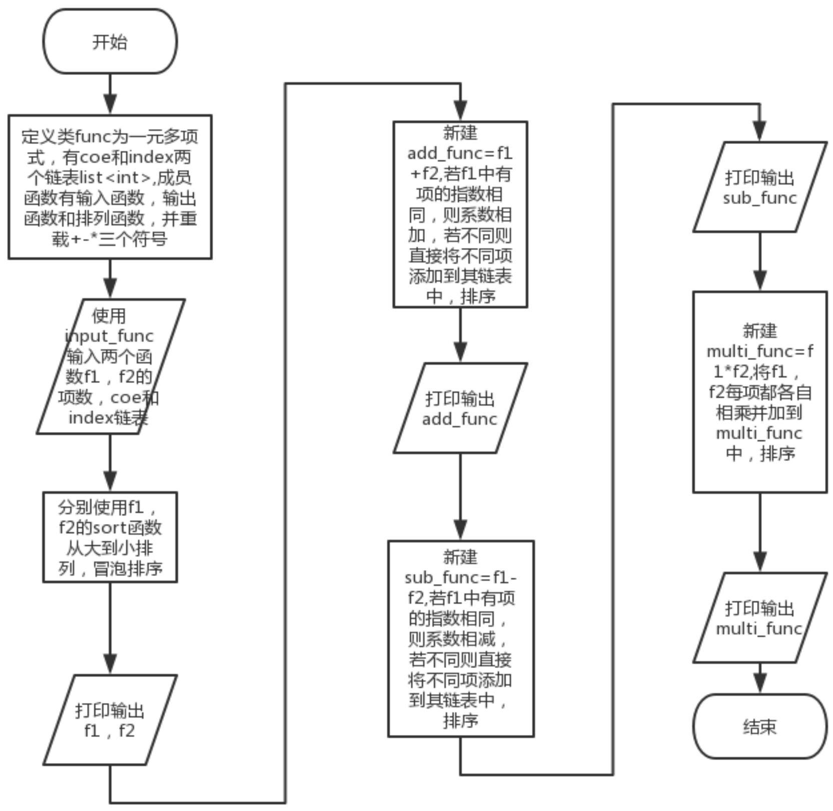
Vector 实现（沈琢乔）：


void info()
Number
index
coefficient
Polynomial
vector<Number> container
Polynomial(bool flag = false)
Polynomial(Polynomial& pol)
int travelPrint()
Polynomial operator+(Polynomial& pol_2)
Polynomial operator-(Polynomial& another)
Polynomial operator*(Polynomial& another)

定义一个 Number 类，并在 Polynomial 内定义 vector<Number> container，通过构造函数初始化 Polynomial，重载+，-，*运算符，定义 travelPrint()函数遍历打印多项式。

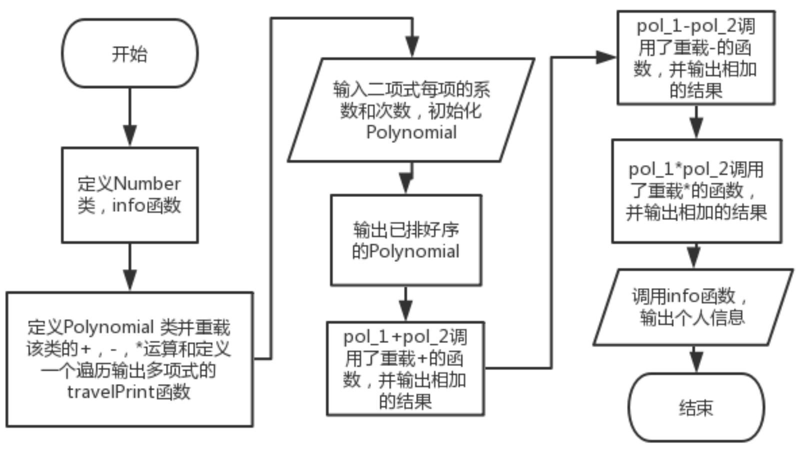
## 三． 算法设计

### List 实现（孙洪超）：



开始
定义类func为一元多项式，有coe和index两个链表list<int>,成员函数有输入函数，输出函数和排列函数，并重载+-*三个符号
使用input_func输入两个函数f1，f2的项数，coe和index链表
分别使用f1，f2的sort函数从大到小排列，冒泡排序
打印输出f1，f2
新建add_func=f1+f2,若f1中有项的指数相同，则系数相加，若不同则直接将不同项添加到其链表中，排序
打印输出add_func
新建sub_func=f1-f2,若f1中有项的指数相同，则系数相减，若不同则直接将不同项添加到其链表中，排序
打印输出sub_func
新建multi_func=f1*f2,将f1，f2每项都各自相乘并加到multi_func中，排序
打印输出multi_func
结束

### Vector 实现（沈琢乔）：



开始
定义Number类，info函数
定义Polynomial 类并重载该类的+，-，*运算和定义一个遍历输出多项式的travelPrint函数
输入二项式每项的系数和次数，初始化Polynomial
输出已排好序的Polynomial
pol_1+pol_2调用了重载+的函数，并输出相加的结果
pol_1-pol_2调用了重载-的函数，并输出相加的结果
pol_1*pol_2调用了重载*的函数，并输出相加的结果
调用info函数，输出个人信息
结束

## 四． 编程实现（程序运行结果截图，运行结果需体现个人信息）

### List 实现（孙洪超）：

```
#include<iostream>
#include<list>
using namespace::std;
class func {
    //使用 c++ 标准模版库 list 或 vector 容器，实现一元多项式的初始化、
    //遍历打印、加法、减法及乘法操作。
public:
    list<int>::iterator index_iter;
    list<int>::iterator coe_iter;
    list<int> coe, index;//定义 int 型 list，分别储存系数，指数；
    int num_of_term;
    void input_func() {//输入系数指数
        cin >> num_of_term;
        for (int i = 0; i < num_of_term; i++) {
            int temp_coe, temp_index;
            cin >> temp_coe >> temp_index;
            coe.push_back(temp_coe);
            index.push_back(temp_index);

        }

    }
    void sort() {//排序函数，将 coe 和 index 链表按照 index 的大小从大到小排列
        index_iter = index.begin();
        coe_iter = coe.begin();
        list<int>::iterator coe_compare, index_compare;
        for (; index_iter != index.end(); index_iter++, coe_iter++) {
            coe_compare = coe_iter, index_compare = index_iter;
                for (coe_compare++, index_compare++; index_compare !=
index.end(); index_compare++, coe_compare++) {
                    if ((*index_iter) < (*index_compare)) {
                        int temp_index=*index_iter, temp_coe=*coe_iter;
                        *index_iter = *index_compare;
                        *index_compare = temp_index;
                        *coe_iter = *coe_compare;
                        *coe_compare = temp_coe;
                    }
                }
            }
            index_iter = index.begin();//为了不影响别的函数两个迭代器的使用
            coe_iter = coe.begin();
        }
        func operator+(func f1) {
            func return_func=f1;
            for (; index_iter!= index.end(); index_iter++,coe_iter++) {
                int repeat_mark=0;//判断是否在 return_func 中存在该指数项
                for (int i=0; i<return_func.index.size(); return_func.index_iter++,
return_func.coe_iter++,i++) {
                    if (*return_func.index_iter == *index_iter) {
                        (*return_func.coe_iter) += *coe_iter;
                        repeat_mark = 1;
                    }
                }
                if (repeat_mark != 1) {//代表无相同，直接插入到该对象链表中。
                    return_func.index.push_back(*index_iter);
                    return_func.coe.push_back(*coe_iter);
                }
                return_func.index_iter = return_func.index.begin();
                return_func.coe_iter = return_func.coe.begin();
            }
            index_iter = index.begin();
            coe_iter = coe.begin();
            return_func.sort();
            return return_func;
        }
        func operator-(func f1) {
            func return_func = f1;
            list<int>::iterator temp = return_func.coe.begin();
            for (int i = 0; i < return_func.index.size(); temp++, i++) {
                *temp = -*temp;
            }
            return_func.coe_iter = return_func.coe.begin();
            for (; index_iter != index.end(); index_iter++, coe_iter++) {
                int repeat_mark = 0;//判断是否在 return_func 中存在该指数项
                for (int i = 0; i<return_func.index.size(); return_func.index_iter++,
return_func.coe_iter++, i++) {

                    if (*return_func.index_iter == *index_iter) {
                        (*return_func.coe_iter) += *coe_iter;
                        repeat_mark = 1;
                    }
                }
                if (repeat_mark != 1) {
                        return_func.index.push_back(*index_iter);
                        return_func.coe.push_back(*coe_iter);
                }
                return_func.index_iter = return_func.index.begin();
                return_func.coe_iter = return_func.coe.begin();
        }
        index_iter = index.begin();
        coe_iter = coe.begin();
        return_func.sort();
        return return_func;
    }
    func operator*(func f1) {
        func return_func=f1;
        int index_size = return_func.index.size();
        for (; index_iter != index.end(); index_iter++, coe_iter++) {//先把所有的都加起
来
                for                (int                i=0;                            i<index_size;
return_func.index_iter++,return_func.coe_iter++, i++) {

                        return_func.index.push_back(*index_iter + *return_func.index_iter);
                        return_func.coe.push_back(*coe_iter**return_func.coe_iter);

                }
                return_func.index_iter = return_func.index.begin();
                return_func.coe_iter = return_func.coe.begin();
        }
        //查重去重
        for (int i = 0;   i < index_size; i++) {
                return_func.index.pop_front();
                return_func.coe.pop_front();
        }
        return_func.sort();
        int before_index = *return_func.index.begin();
        int i = 0;
        for               (return_func.index_iter++,               return_func.coe_iter++;i<
return_func.index.size()-1; return_func.index_iter++, return_func.coe_iter++,i++) {

                if (before_index == *return_func.index_iter) {
                        list<int>::iterator      temp_index_iter      =      return_func.index_iter,
temp_coe_iter= return_func.coe_iter;
                        return_func.index_iter--;
                        return_func.coe_iter--;
```

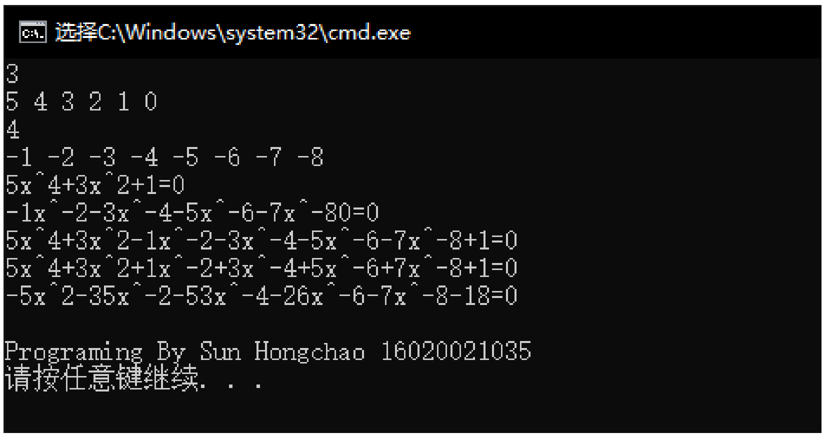
```
                    i--;
                    *return_func.coe_iter = *temp_coe_iter + *return_func.coe_iter;
                    return_func.index.erase(temp_index_iter);
                    return_func.coe.erase(temp_coe_iter);
                }
                else before_index = *return_func.index_iter;
            }
            return return_func;
        }
        void print_func() {//输出，如果下一项大于 0 且该项不为第一项，输出加号，如果
该项 index>1, 输出该项[coe]x^[index]的形式, 如果 index=1, 输出[coe]x,如果 index=0,
仅输出[coe]且在最后一项输出。
            list<int>::iterator index_printer = index.begin();
            list<int>::iterator   coe_printer = coe.begin();
            int index_equal_to_zero = 0;
            for (;index_printer!=index.end() ; index_printer++,coe_printer++) {
                if  (*index_printer  !=  0  &&  *coe_printer  >  0  &&  index_printer  !=
index.begin())
                    cout << "+";
                if (*index_printer != 0 && *index_printer != 1)
                    cout << *coe_printer << "x^" << *index_printer;
                else if (*index_printer == 1)
                    cout << *coe_printer << "x";
                else
                    index_equal_to_zero= *coe_printer;

            }
            if (index_equal_to_zero > 0) cout << "+";
            cout << index_equal_to_zero <<"=0" << endl;

        }
};
int main() {
        func f1, f2, add_func, multi_func, sub_func;
        f1.input_func();
        f2.input_func();
        f1.sort();
        f2.sort();
        f1.print_func();
        f2.print_func();

        add_func = f1 + f2;
        add_func.sort();
        add_func.print_func();
        sub_func = f1 - f2;
        sub_func.sort();
        sub_func.print_func();


        multi_func = f1 * f2;
        multi_func.sort();
        multi_func.print_func();

        cout << endl << "Programing By Sun Hongchao 16020021035" << endl;
        return 0;
    }
```

## Vector 实现（沈琢乔）：

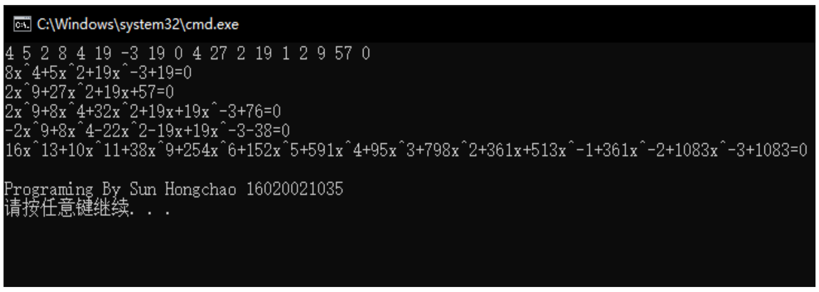
```
//author: 沈琢乔
//email: ouc16020021031@gmail.com
//description:

//使用c++ 标准模版库 vector 容器，
//实现一元多项式的初始化、遍历打印、
//加法、减法及乘法操作。

#define _CRT_SECURE_NO_WARNINGS
#include<iostream>
#include<vector>
#include<ctime>
#include<malloc.h>
using namespace std;

class Number
{
public:
    int coefficient;
    int index;
};

class Polynomial
{
public:
    vector<Number>  container;
    Polynomial(bool flag = false)//初始化多项式
    {
        while (flag)
        {
            vector<Number>::iterator it;
            Number temp;
            cin >> temp.coefficient;
            if (temp.coefficient == 0)break;
            cin >> temp.index;
            for (it = container.begin(); it != container.end(); it++)
                if (it->index > temp.index) { it = container.insert(it, temp);
break; }
            if (it == container.end())container.push_back(temp);
        }
    }//构造函数
    Polynomial(Polynomial& pol) { container = pol.container; }

    int travelPrint()
    {
        string str;
        cout << "f(x) = ";
        for (vector<Number>::iterator it = container.begin(); it !=
container.end(); it++)
        {
            if (it != container.begin())
            {
                if (it->coefficient > 0)cout << " +";
                if (it->coefficient < 0)cout << " ";
            }

            if (it->coefficient == 1);
            else if (it->coefficient == -1)cout << "-";
            else cout << (*it).coefficient;

            if (it->index == 0);
            else if (it->index == 1)cout << "x";
            else cout << "x^" << (*it).index;

            if (it->index == 0 && it->coefficient == 1)cout << "1";
        }
        cout << endl << endl;
        return 0;
    }//遍历打印多项式
    Polynomial operator+(Polynomial& pol_2)
    {
        Polynomial pol_1 = *this;
        vector<Number>::iterator it1, it2;
        for (it2 = pol_2.container.begin(); it2 != pol_2.container.end();
it2++)
        {
            for (it1 = pol_1.container.begin(); it1 != pol_1.container.end();
it1++)
            {
                if (it1->index > it2->index) { it1 =
pol_1.container.insert(it1, *it2); break; }
                else if (it1->index == it2->index) { it1->coefficient +=
it2->coefficient; break; }
            }
            if (it1 == pol_1.container.end())
{ pol_1.container.push_back(*it2); }
        }
        return pol_1;
    }//重载“+”运算符

    Polynomial operator-(Polynomial& another)
    {
        Polynomial pol_1 = *this;
        Polynomial pol_2 = another;
        vector<Number>::iterator it;
        for (it = pol_2.container.begin(); it != pol_2.container.end(); it++)
            it->coefficient = -it->coefficient;
        pol_1 = pol_1 + pol_2;
        return pol_1;
    }//重载“-”运算符

    Polynomial operator*(Polynomial& another)
    {
        Polynomial *polynomials[100], result;
        vector<Number>::iterator it1, it2;
        Number temp;
        int i = 0;
        for (it1 = container.begin(); it1 != container.end(); it1++)
        {
            polynomials[i] = new Polynomial();
            for (it2 = another.container.begin(); it2 !=
another.container.end(); it2++)
            {
                temp.coefficient = it1->coefficient * it2->coefficient;
                temp.index = it1->index + it2->index;
                polynomials[i]->container.push_back(temp);
            }
            i++;
        }
        for (int i = 0; i < container.size(); i++)
            result = result + *polynomials[i];
        return result;
    }//重载“*”运算符
     //f1 * f2 =a1*f2+a2*f2+...+an*f2
     //ai*f2 => n个多项式，再将这n个多项式相加
};

void info()
{
    cout << "沈琢乔" << endl;
    time_t nowtime;
    nowtime = time(NULL); //获取日历时间
    struct tm *local;
    local = localtime(&nowtime);  //获取当前系统时间
    char buf[80];
    strftime(buf, 80, "%Y-%m-%d %H:%M:%S", local);
    cout << buf << endl;
    getchar();//使程序停顿
    getchar();
}

int main()
{
    Polynomial *polynomials[2];
    int n = 2;
    for (int i = 0; i < n; i++)
    {
        cout << "请分别输入第" << i + 1 << "个二次项每项的系数和次数，" <<
endl;
        cout << "输入0并摁回车即可退出输入，" << endl;
        cout << "每行输入其中一项的系数和次数，中间用空格隔开：" << endl;
        polynomials[i] = new Polynomial(true);
        cout << endl;
    }

    Polynomial add, substract, multiply;
    for (int i = 0; i < n; i++)//输出f1，f2
    {
        cout << "f" << i + 1 << ": " << endl;
        polynomials[i]->travelPrint();
    }
    cout << "f1 + f2: " << endl;//输出f1 + f2
    add = *polynomials[0] + *polynomials[1];
    add.travelPrint();
    cout << "f1 - f2: " << endl;//输出f1 - f2
    substract = *polynomials[0] - *polynomials[1];
    substract.travelPrint();
    cout << "f1 * f2: " << endl;//输出f1 * f2
    multiply = *polynomials[0] * *polynomials[1];
    multiply.travelPrint();
    info();
    return 0;
}
```

## 五． 结果分析

### List 实现（孙洪超）：

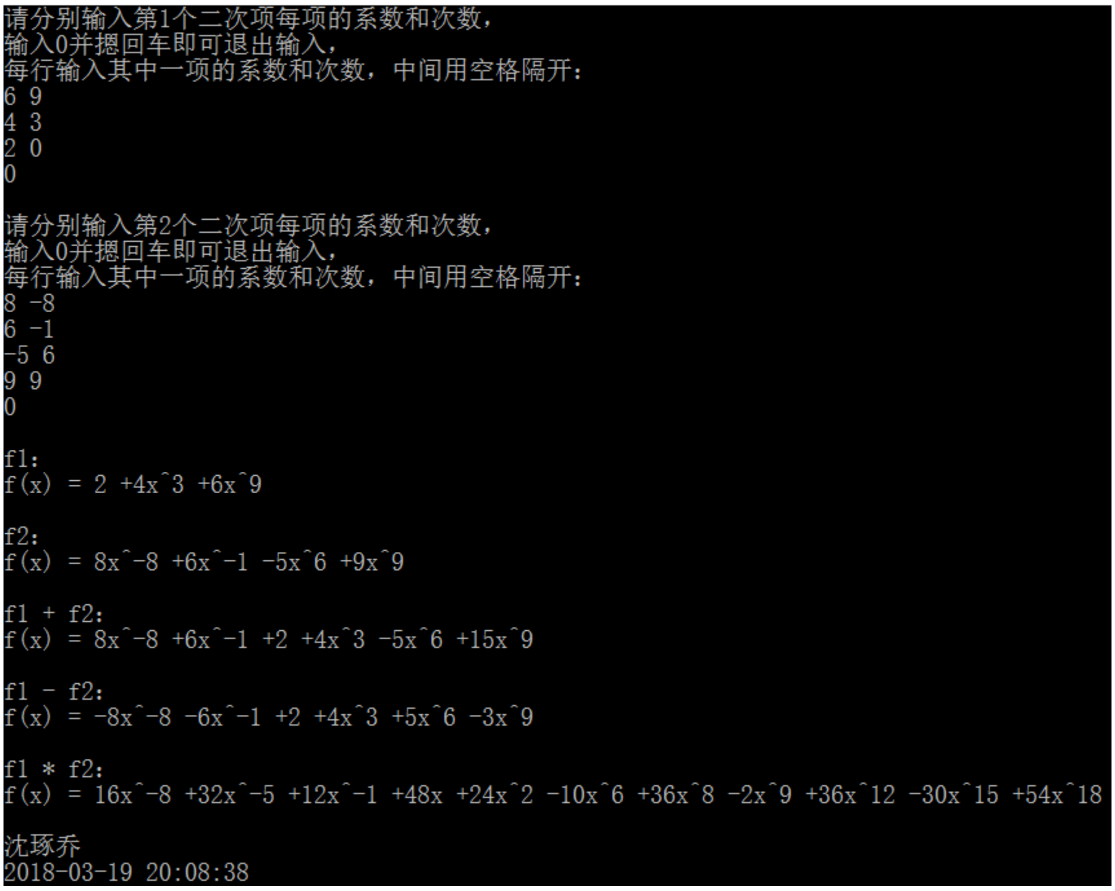


```
选择C:\Windows\system32\cmd.exe
3
5 4 3 2 1 0
4
-1 -2 -3 -4 -5 -6 -7 -8
5x^4+3x^2+1=0
-1x^-2-3x^-4-5x^-6-7x^-80=0
5x^4+3x^2-1x^-2-3x^-4-5x^-6-7x^-8+1=0
5x^4+3x^2+1x^-2+3x^-4+5x^-6+7x^-8+1=0
-5x^2-35x^-2-53x^-4-26x^-6-7x^-8-18=0

Programing By Sun Hongchao 16020021035
请按任意键继续. . .
```



```
C:\Windows\system32\cmd.exe
4 5 2 8 4 19 -3 19 0 4 27 2 19 1 2 9 57 0
8x^4+5x^2+19x^-3+19=0
2x^9+27x^2+19x+57=0
2x^9+8x^4+32x^2+19x+19x^-3+76=0
-2x^9+8x^4-22x^2-19x+19x^-3-38=0
16x^13+10x^11+38x^9+254x^6+152x^5+591x^4+95x^3+798x^2+361x+513x^-1+361x^-2+1083x^-3+1083=0

Programing By Sun Hongchao 16020021035
请按任意键继续. . .
```

Vector 实现（沈琢乔）：



```
请分别输入第1个二次项每项的系数和次数，
输入0并摁回车即可退出输入，
每行输入其中一项的系数和次数，中间用空格隔开：
6 9
4 3
2 0
0

请分别输入第2个二次项每项的系数和次数，
输入0并摁回车即可退出输入，
每行输入其中一项的系数和次数，中间用空格隔开：
8 -8
6 -1
-5 6
9 9
0

f1:
f(x) = 2 +4x^3 +6x^9

f2:
f(x) = 8x^-8 +6x^-1 -5x^6 +9x^9

f1 + f2:
f(x) = 8x^-8 +6x^-1 +2 +4x^3 -5x^6 +15x^9

f1 - f2:
f(x) = -8x^-8 -6x^-1 +2 +4x^3 +5x^6 -3x^9

f1 * f2:
f(x) = 16x^-8 +32x^-5 +12x^-1 +48x +24x^2 -10x^6 +36x^8 -2x^9 +36x^12 -30x^15 +54x^18

沈琢乔
2018-03-19 20:08:38
```

结果正确

## 六． 总结体会

本题目的是为了巩固复习 C++的容器的使用，因此复习是必要的。通过复习，巩固了旧知识，并发现了自己未注意的错误，如将两个迭代器定义为类内成员变量，在复制的时候如果使用默认的复制函数迭代器所指的函数并未改变，仍指向原对象的 list 头，在使用的时候，以迭代器是否等于 list.end()来判断是否跳出循环就会出现越界错误。使用 vector 的 insert 函数，应如此写” it = container.insert(it, temp)”，而非“container.insert(it, temp)”。本次作业最大收获在于重新熟悉了 c++。