一元稀疏多项式实验报告

葛淞铂

2022.09.17

实验源码

1 实验任务

1.1 设计一个一元稀疏多项式简单计算器

(1) 输出并建立多项式；

(2) 输出多项式。输出形式为整数序列：$n, c_1, n_1, ..., c_n, a_n$，其中 n 是多项式的项数，$c_i, a_i$ 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 和 b 相加，建立多项式 a+b；

(4) 多项式 a 和 b 相减，建立多项式 a-b；

1.2 选做部分

(1) 计算多项式在 x 处的值；

(2) 计算多项式的导函数；

(4) 多项式的输出形式为类数学表达式；

2 设计思路

2.1 结构部分

(1) 使用 lab1.h, lab1Func.cpp, lab1Main.cpp 作为程序主体，将头文件，函数定义，主函数分别建立在不同的文件中，体现了程序设计的模块化思想。

(2) 建立了 Solution 类用于解决实验问题。

2.2 实现细节

2.2.1 类的初始化

定义抽象数据类型 Solution，private 变量为链表的头结点以增加安全性，public 中为实验所需所有自定义函数。

```
typedef struct ListNode
{
float coef;
float expn;
struct ListNode *next;
}ListNode;
class Solution
{
private:
    vector<ListNode*> head;
public:
    Solution()
    {
        for(int i = 0; i < 6; ++i)
        {
            ListNode* headx = new(ListNode);
            headx -> coef = 0;
            headx -> next = NULL;
            head.push_back(headx);
        }
    }
```

```
        void insert(ListNode* node, int n);
        void printPoly(int n);
        bool init();
        void plus();
        void minus();
        float func_1(int x);
        void func_2();
        /*void func_3();*/
        void func_4(int n);
    };
```

2.2.2 bool init()

为了实现 a，b 多项式的运算，首先令用户输入多项式的系数和指数，依次插入 head[0],head[1] 中初始化多项式 a，b，若用户输入正确将进行进一步操作，若果输入错误则输出错误并结束运行。

```
        bool Solution::init()
        {
            cout<<"Please input Ploynomial A :";
            float t;
            vector<float> temp1;
            while(cin>>t)
            {
                temp1.push_back(t);
            }
            if(temp1.size() % 2 != 0)
            {
                cout<<"Polynomial A input error!"<<endl;
                return false;
            }
            for(size_t i = 0; i < temp1.size(); i += 2)
            {
                ListNode* node = new(ListNode);
                node -> coef = temp1[i];
                node -> expn = temp1[i + 1];
                insert(node, 1);
            }
            cin.clear();
            cin.ignore();
            cout<<"Please input Ploynomial B :";
            vector<float> temp2;
            while (cin>>t)
            {
                temp2.push_back(t);
            }
            if(temp2.size() % 2 != 0)
            {
                cout<<"Polynomial B input error!"<<endl;
                return false;
            }
            for(size_t i = 0; i < temp2.size(); i += 2)
            {
                ListNode* node = new(ListNode);
                node -> coef = temp2[i];
                node -> expn = temp2[i + 1];
                insert(node, 2);
            }
            printPoly(1);
            printPoly(2);
            return true;
        }
```

2.2.3 void insert(ListNode* node, int n)

实验要求生成的多项式按照降序排列故插入并排序的操作需要频繁使用，我们自定义 insert 函数，插入节点的同时保持链表有序。

```
void Solution::insert(ListNode* node, int n)
{

    ListNode* t = head[n - 1];
```

```
        while(t -> next)
        {
            if(node -> expn > t -> next -> expn)
            {
                node -> next = t -> next;
                t -> next = node;
                ++head[n - 1] -> coef;
                return;
            }
            t = t -> next;
        }
        if(!t -> next)
        {
            node -> next = t -> next;;
            t -> next = node;
        }
        ++head[n - 1] -> coef;
    }
```

2.2.4 void printPoly(int n)

完成实验要求 (1) 输出多项式长度并依次输出多项式系数与指数，我们把 a，b，a+b，a-b，求导等生成的多项式以链表的形式存储，将头结点存入 head 数组中，故该函数只需接受所要输出多项式的序号即可完成功能。

```
    void Solution::printPoly(int n)
    {
        ListNode* t = head[n - 1];
        cout<<head[n - 1] -> coef;
        while(t -> next)
        {
            cout<<","<<t -> next -> coef<<","<<t -> next -> expn
                ;
            t = t -> next;
        }
        cout<<endl;
```

```
}
```

2.2.5 void plus()

根据实验要求只需计算 a，b 两个多项式的和，故而不需要额外参数选择多项式，采用双指针的形式，若两多项式中指数相等则将系数相加，若不相等则直接插入新节点，指针后移直至遍历完多项式。

```
void Solution::plus()
{
    ListNode* t1 = head[0];
    ListNode* t2 = head[1];
    while(t1 -> next && t2 -> next)
    {
        if(t1 -> next -> expn == t2 -> next -> expn)
        {
            ListNode* node = new(ListNode);
            node -> coef = t1 -> next -> coef + t2 -> next
               -> coef;
            node -> expn = t1 -> next -> expn;
            insert(node, 3);
            t1 = t1 -> next;
            t2 = t2 -> next;
        }
        else if(t1 -> next -> expn > t2 -> next -> expn)
        {
            ListNode* node = new(ListNode);
            node -> coef = t1 -> next -> coef;
            node -> expn = t1 -> next -> expn;
            insert(node, 3);
            t1 = t1 -> next;
        }
        else
        {
            ListNode* node = new(ListNode);
            node -> coef = t2 -> next -> coef;
```

```
                node -> expn = t2 -> next -> expn;
                insert(node, 3);
                t2 = t2 -> next;
            }
        }
        if(!t1 -> next)
        {
            while(t2 -> next)
            {
                ListNode* node = new(ListNode);
                node -> coef = t2 -> next -> coef;
                node -> expn = t2 -> next -> expn;
                insert(node, 3);
                t2 = t2 -> next;
            }
        }
        else if(!t2 -> next)
        {
            while(t1 -> next)
            {
                ListNode* node = new(ListNode);
                node -> coef = t1 -> next -> coef;
                node -> expn = t1 -> next -> expn;
                insert(node, 3);
                t1 = t1 -> next;
            }
        }
        printPoly(3);
        func_4(3);
    }
```

2.2.6 void minus()

该函数实现方法与 plus 大体相同，需要注意的是在减法情况下，若两系数相减后为 0 则不需将该节点插入结果链表中。

```
void Solution::minus()
{
    ListNode* t1 = head[0];
    ListNode* t2 = head[1];
    while(t1 -> next && t2 -> next)
    {
        if(t1 -> next -> expn == t2 -> next -> expn)
        {
            ListNode* node = new(ListNode);
            node -> coef = t1 -> next -> coef - t2 -> next
               -> coef;
            node -> expn = t1 -> next -> expn;
            if(node -> coef != 0)
                insert(node, 4);
            t1 = t1 -> next;
            t2 = t2 -> next;
        }
        else if(t1 -> next -> expn > t2 -> next -> expn)
        {
            ListNode* node = new(ListNode);
            node -> coef = t1 -> next -> coef;
            node -> expn = t1 -> next -> expn;
            insert(node, 4);
            t1 = t1 -> next;
        }
        else
        {
            ListNode* node = new(ListNode);
            node -> coef = -1*t2 -> next -> coef;
            node -> expn = t2 -> next -> expn;
            insert(node, 4);
            t2 = t2 -> next;
        }
    }
    if(!t1 -> next)
    {
        while(t2 -> next)
        {
            ListNode* node = new(ListNode);
            node -> coef = -1*t2 -> next -> coef;
            node -> expn = t2 -> next -> expn;
            insert(node, 4);
            t2 = t2 -> next;
        }
    }
    else if(!t2 -> next)
    {
        while(t1 -> next)
        {
            ListNode* node = new(ListNode);
            node -> coef = t1 -> next -> coef;
            node -> expn = t1 -> next -> expn;
            insert(node, 4);
            t1 = t1 -> next;
        }
    }
    printPoly(4);
    func_4(4);
}
```

2.2.7 float fun1(int x)

对于选做题 (1)，取得 x 的值，依次计算即可，注意系数为-1，或带根号情况。

```
    float Solution::func_1(int x)
    {
        float sum = 0;
        ListNode* t = head[0];
        while(t -> next)
        {
            if((x == 0 && t -> next -> expn == -1) || (x < 0 &&
                t -> next -> expn > 0 && t -> next -> expn < 1))
            {
                cout<<"Calculation␣error!"<<endl;
                return INT_MIN;
            }
            sum += t -> next -> coef * pow(x, t -> next -> expn)
                ;
            t = t -> next;
        }
        return sum;
    }
```

2.2.8 void func2()

对两个多项式分别求导，额外考虑了指数为 0 的情况。

```
void Solution::func_2()
{
    ListNode* t = head[0];

    while(t -> next)
    {
        ListNode* t2 = new(ListNode);
        if(t -> next -> expn != 0)
        {
            t2 -> coef = t -> next -> coef * t -> next ->
                expn;
            t2 -> expn  = t -> next -> expn - 1;
            insert(t2, 5);
        }
        t = t -> next;
    }
    func_4(5);
}
```

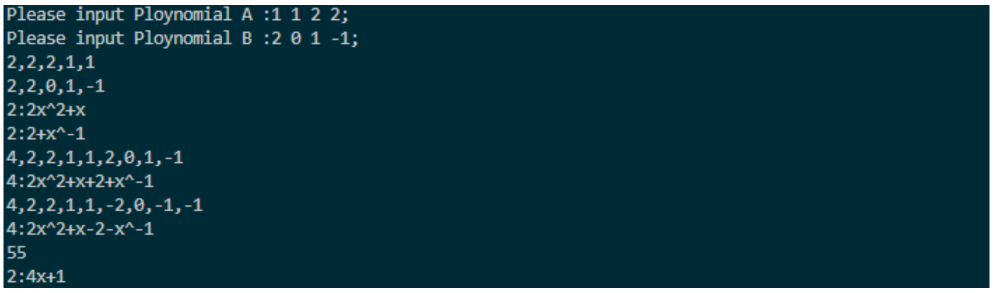
2.2.9 void func4(int n)

类数学公式输出多项式，需要额外考虑当系数为 +，-1 与其他情况的区别，指数为 1,0 与其他形式的区别。

```
void Solution::func_4(int n)
{
    int temp = 0;
    ListNode* t = head[n - 1];
    cout<<head[n - 1] -> coef<<":";
    while(t -> next)
    {
        if(temp != 0 && t -> next -> coef > 0)
            cout<<"+";
        if(t -> next -> coef == -1)
            cout<<"-";
        if(t -> next -> coef != 1 && t -> next -> coef !=
           -1)
            cout<<t -> next -> coef;
        else if((t -> next -> coef == 1 || t -> next ->coef
           == -1) && t -> next -> expn == 0 )
            cout<<1;
        if(t -> next -> expn != 0 && t -> next -> expn != 1)
            cout<<"x^"<<t -> next -> expn;
        else if(t -> next -> expn == 1)
            cout<<"x";
        t = t -> next;
        ++temp;
    }
    cout<<endl;
}
```

3 测试结果

输入与输出格式如图所示，仅列出一个测试用例，其他不一一赘述。



```
Please input Ploynomial A :1 1 2 2;
Please input Ploynomial B :2 0 1 -1;
2,2,2,1,1
2,2,0,1,-1
2:2x^2+x
2:2+x^-1
4,2,2,1,1,2,0,1,-1
4:2x^2+x+2+x^-1
4,2,2,1,1,-2,0,-1,-1
4:2x^2+x-2-x^-1
55
2:4x+1
```

4 实验总结

4.1 实验收获

该实验巩固了对链表插入、删除、排序等操作的深入理解，提升了编程能力与应用能力

4.2 实验的不足之处

由于笔者不熟悉 GUI 的应用，故没有完成 GUI 界面的显示。