一元稀疏多项式计算器 实验报告

howlingggggg

2022 年 10 月 16 日

1 实验任务

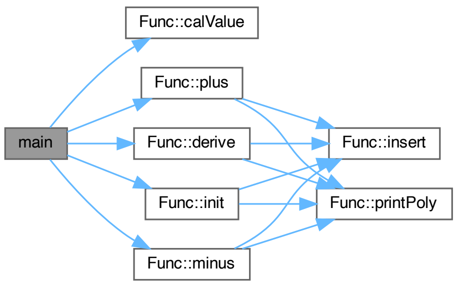
1.1 基本要求

1. 输入并建立多项式;
2. 输出多项式, 输出形式为整数序列; $n, c_1, e_1, c_2, e_2, \ldots, c_n, e_n$, 其中 n 是多项式的项数, c_i 和 e_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 选做内容

1. 计算多项式在 x 处的值。
2. 求多项式 a 的导函数 a'
4. 多项式的输出形式为类数学表达式.

2 概要设计



3 详细设计

3.1 class Func

```
          在class定义了此次实验需要用到的函数
class Func
{
private:
    vector<LNode *> head;

public:
    Func()
    {
        for (int i = 0; i < 6; i++)
        {
            LNode *headx = new (LNode);
            headx->coef = 0;
            headx->next = NULL;
            head.push_back(headx);
        }
    }
    bool init();
    void insert(LNode *node, int n);
    void printPoly(int n);
    void add();
    void minus();
    float calValue(float x);
    void derive();
};
```

3.2 bool init()

初始化多项式 a 和多项式 b，将输入使用 insert() 插入 linklist 中

```
bool Func::init()
{
    cout << "input poly a:";
    float t;
    vector<float> temp_a;
    while (cin >> t)
    {
        temp_a.push_back(t);
    }
    if (temp_a.size() % 2 != 0)
    {
        cout << "error" << endl;
        return false;
    }
    for (size_t i = 0; i < temp_a.size(); i += 2)
    {
        LNode *node = new (LNode);
        node->coef = temp_a[i];
        node->expn = temp_a[i + 1];
        insert(node, 1);
    }
    cin.clear();
    cin.ignore();

    cout << "input poly b:";
    vector<float> temp_b;
    while (cin >> t)
    {
        temp_b.push_back(t);
    }
    if (temp_b.size() % 2 != 0)
    {
        cout << "input error" << endl;
        return false;
    }
    for (size_t i = 0; i < temp_b.size(); i += 2)
    {
        LNode *node = new (LNode);
        node->coef = temp_b[i];
        node->expn = temp_b[i + 1];
        insert(node, 2);
    }
    printPoly(1);
    printPoly(2);
    return true;
}
```

3.3 void insert(LNode *node, int n)

将 LNode 插入第 n 个 linklist 中，其中需要判断是否存在和插入的 LNode 相同的 expn，若存在则将 coef 相加，否则插入到 linklist 中并将后面的节点后移。

```
void Func::insert(LNode *node, int n)
{
    LNode *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;
        }
        else if (node->expn == t->next->expn)
        {
            t->next->coef += node->coef;
            return;
        }
        t = t->next;
    }
    if (!t->next)
    {
        node->next = t->next;
        t->next = node;
    }
    ++head[n - 1]->coef;
}
```

3.4 void printPoly(int n)

实现输出多项式和选做内容 4, 需要特别处理的是若 expn 为 0，则只输出 coef；若 coef 为 ±1 则只输出 expn。在选做 4 中还必须特别考虑第一项，若第一项为 + 则只输出系数。

```
void printPoly(int n)
{
    LNode *t = head[n - 1];
    LNode *t1 = head[n - 1];
    cout << head[n - 1]->coef;
    while (t->next)
    {
        cout << "," << t->next->coef << "," << t->next->
            expn;
        t = t->next;
    }
    cout << endl;

    int temp = 0;

    cout << head[n - 1]->coef << ":";
    while (t1->next)
    {
        if (temp != 0 && t1->next->coef > 0)
            cout << "+";
        if (t1->next->coef == -1)
            cout << "-";
        if (t1->next->coef != 1 && t1->next->coef != -1)
            cout << t1->next->coef;
        else if ((t1->next->coef == 1 || t1->next->coef
            == -1) && t1->next->expn == 0)
            cout << 1;
        if (t1->next->expn != 0 && t1->next->expn != 1)
            cout << "x^" << t1->next->expn;
        else if (t1->next->expn == 1)
            cout << "x";
        t1 = t1->next;
        temp++;
    }
    cout << endl;
}
```

3.5 void plus()

将多项式相加并插入 linklist[2]，若相加后系数为 0 则不插入

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

3.6 void minus()

将多项式相减并插入 linklist[3]，若相减后系数为 0 则不插入

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

3.7 float calValue(float x)

```
float Func::calValue(float x)
{
    float sum = 0;
    LNode *t = head[0];
    while (t->next)
    {
        sum += t->next->coef * pow(x, t->next->expn);
        t = t->next;
    }
    return sum;
}
```

3.8 void derive()

求得 a' 并存入 linklist[4]

```
void Func::derive()
{
    LNode *t = head[0];

    while (t->next)
    {
        LNode *t2 = new (LNode);
        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;
    }
    printPoly(5);
}
```

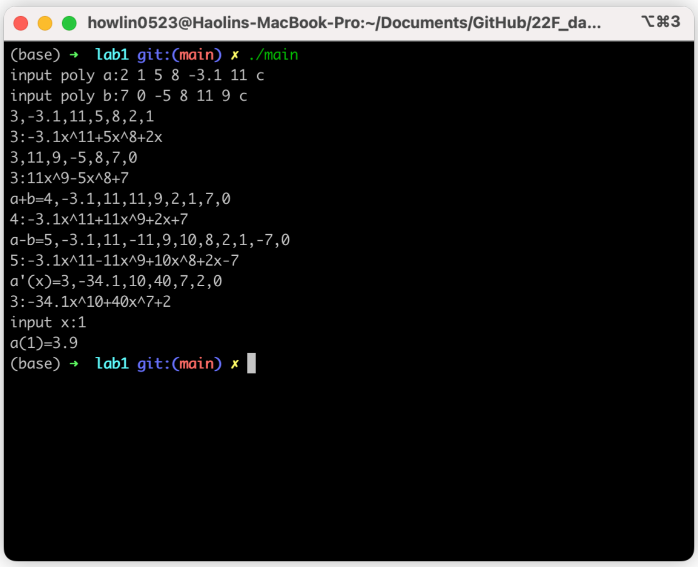
4 测试结果

4.1 输入数据

多项式 a: $\left(2x+5x^8-3.1x^{11}\right)$，多项式 b:$\left(7-5x^8+11x^9\right)$

5 实验总结

在此次实验中熟悉了对单链表的操作



```
(base) ➜  lab1 git:(main) ✗ ./main
input poly a:2 1 5 8 -3.1 11 c
input poly b:7 0 -5 8 11 9 c
3,-3.1,11,5,8,2,1
3:-3.1x^11+5x^8+2x
3,11,9,-5,8,7,0
3:11x^9-5x^8+7
a+b=4,-3.1,11,11,9,2,1,7,0
4:-3.1x^11+11x^9+2x+7
a-b=5,-3.1,11,-11,9,10,8,2,1,-7,0
5:-3.1x^11-11x^9+10x^8+2x-7
a'(x)=3,-34.1,10,40,7,2,0
3:-34.1x^10+40x^7+2
input x:1
a(1)=3.9
(base) ➜  lab1 git:(main) ✗
```