实验报告- 一元稀疏多项式简单计算器

PB21081601 张芷苒

1. 需求分析

一元稀疏多项式简单计算器的基本功能是：

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 $a-b$。

在本程序中加入的附加功能有：

1. 计算器仿真界面；
2. 多项式的输出形式为类数学表达式，系数值为 1 的非零次项的输出形式中略去系数 1。

2. 概要设计

基本思路为：用链表存储多项式信息，在读入时按照指数幂降序把各项依次插入链表，在进行加减法运算时用链表的合并得到结果多项式。

3. 详细设计

下面介绍本程序中几个重要模块的具体实现，语句的具体实现参看注释。

- 处理读入的字符串并将其转换成链表存储多项式信息
  1. 构造 `Get_coef` 函数从输入的字符串里提取x的系数，`Getnum` 函数提取输入字符串中x的系数和指数，分别存入 `coefs` 和 `indexs` 两个数组中。
  2. 使用 `Create_poly` 函数把 `coefs` 和 `indexs` 中的信息转化为链表，在其过程中构造 `Insert_list` 函数，把当前读到结点的指数按照降序存储进链表。同时为了比较系数，构造 `cmp` 函数，以及 `Delete_list` 函数，删除 a) 系数为0；b) 空链表（直接插入）情况下的结点。
  3. 完成以上步骤后，所有输入项的信息都存储在以 `head` 为头指针的 `poly_list*` 型链表 `p` 中。

```
typedef struct                  //结构体，存放单项ax^n的信息
{
    double coef;                //ax^n的系数a
    int index;                  //ax^n的指数n
}term;
```

```
typedef struct poly             //用链表存储多项式
{
    term data;                  //多项式的单项信息
    poly* next;
}poly_list;
```

```
/*函数说明:
    在输入的字符串中提取每一项的系数*/
double Get_coef(char *str)
{
        double s = 0.0;
        double d = 10.0;                                                    //处
理大于10的系数
        bool flag = false;
//flag记录正负，false为正数
        while (*str == ' ')                         //如遇空格则继续往后读
                str++;
        if (*str == '-')
        {
                flag = true;
                str++;
                if (*str == 'x' || *str == 'X')
                        return -1.0;                        //加减符号后没遇到系数，系
数为-1则直接返回-1
        }
        else if ((*str == '+' && (*(str + 1) == 'x' || *(str + 1) == 'X')) || (*str
== 'x'))
                return 1.0;                                         //加减符号后没遇到系
数，系数为1则直接返回1
        if (*str == '+' && (*(str + 1) >= '0' && *(str + 1) <= '9'))
                str++;
        if (!(*str >= '0' && *str <= '9'))              //如果一开始非数字则退出，返回
0.0
                return s;
        while (*str >= '0' && *str <= '9' && *str != '.')
        {                                           //计算小数点前整数部分
                s = s * 10.0 + *str - '0';
                str++;
        }
        if (*str == '.')                            //以后为小数部分
        str++;
        while (*str >= '0'&&*str <= '9')                        //计算小数部分
        {
```

```
                s = s + (*str - '0') / d;
                d *= 10.0;
                str++;
        }
        return s * (flag ? -1.0 : 1.0);             //根据flag值返回相应的正负数
}
```

```
/*GetNums函数说明：
    将输入的字符串中的系数和指数提取存储到数组里*/
void GetNums(int &cnt, double* coefs, int* indexs)
{
        int i = 0;
        cnt = 0;
        double coef;
        int expn = 0;
        char str[80];
        gets (str);
//存放输入的字符串
        while (*(str + i))
        {
                coef = Get_coef(str + i);
                if (*(str + i) != 'x' && *(str + i ) != 'X')
                        i++;
                while ((*(str + i) >= '0' && *(str + i) <= '9') || (*(str + i) ==
'.') || (*(str + i) == '-'))
                        i++;
                if (*(str + i) == '+' || *(str + i) == '\0')      //如果没有x则直接将
指数赋值为零
                        expn = 0;
                else if (*(str + i) == 'x' || *(str + i) == 'X')
                {
                        i++;
                        if (*(str + i) == '+' || *(str + i) == '\0' || *(str + i) ==
'-')    //只有x没有数字说明指数为1
                                expn = 1;
                        else if (*(str + i) == '^')
                        {
                                i++;
                                expn = (int)Get_coef(str + i);
                                if (*(str + i) == '-')
                                        i++;
                                while ((*(str + i) >= '0'&&*(str + i) <= '9') || (*
(str + i) == '.'))
```

```
                                        i++;
                        }
                }
                coefs[cnt] = coef;                          //存入数组，元素序号由cnt确定
                indexs[cnt] = expn;
                cnt++;
        }
}
```

```
/*Create_poly函数说明：
    用通过GetNums函数得到的系数和指数生成多项式链表*/
poly_list* Create_poly()
{
    double coefs[80];                                       //数组，存各项系数
    int indexs[80];                                         //数组，存各项指数
    int count;
    GetNums(count, coefs, indexs);                          //通过GetNums把各项信息导入两个数
组
    poly_list* head;
    head = (poly_list*)malloc(sizeof(poly_list));
    head->next = NULL;
    poly_list* term_list;                                   //多项式链表元素
    for (int i = 0; i < count; i++)
    {
        term_list = (poly_list*)malloc(sizeof(poly_list));
        term_list->next = NULL;
        term_list->data.coef = coefs[i];
        term_list->data.index = indexs[i];
        Insert_list (head, term_list);                      //插入多项式链表
    }
    return head;
}
```

```
/*Insert_list函数说明：
    将新的多项式元素ax^n插入多项式链表，并将链表按指数降序排列*/
void Insert_list (poly_list* head, poly_list* q)
{
    poly_list* p = head;
    poly_list* prior = head;
    p = p->next;
    while (p != NULL)
    {
        if (cmp (p->data, q->data) == 0)                    //若链表中存在的指数相同的项，则直接
```

```
相加
        {
            p->data.coef += q->data.coef;
            if (p->data.coef == 0)                          //遇到系数为0的极端情况删除该
节点
                Delete_list (head, p);
            return;
        }
        else if (cmp (p->data, q->data) > 0)                //指数小于当前节点则向后移
        {
            p = p->next;
            prior = prior->next;
        }
        else                                                //大于则直接在当前位置插入链
表
        {
            prior->next = q;
            q->next = p;
            return;
        }
    }
    if (p == NULL)                                          //空链表直接插入
    {
        prior->next = q;
        q->next = NULL;
        if (q->data.coef == 0)
            Delete_list (head, q);
    }
}
```

```
/*函数说明:
    删除链表指定位置的结点q*/
void Delete_list (poly_list* head, poly_list* q)
{
    poly_list* p;
    p = head;
    while (p->next != q)
        p = p->next;
    p->next = q->next;
}
```

- 多项式加减法的实现

> - 其核心函数为 `Calculate_poly`，用 `flag` 控制计算模式，1时为加法，2时为减法，结果存储在 `result` 指针里。

```
/*Calculate_poly函数说明：
    计算两个多项式的和或差，通过flag的值来控制加法和减法来得到对应的结果*/
poly_list* Calculate_poly (poly_list* p, poly_list* q, int flag)                    //多项式加法运算函数
{
    poly_list* result;      //存储结构的链表
    result = (poly_list*)malloc(sizeof(poly_list));
    result->next = NULL;
    poly_list* term_list;
    poly_list* index1 = p->next;                        //两个指针分别指向加数和被加数
    poly_list* index2 = q->next;
    while (index1 != NULL && index2 != NULL)          //若当前两链表节点指数相同则直接相加
    {
        /*用flag来控制加减法，仅用一个函数即可实现多项式加法和减法*/
                term_list = (poly_list*)malloc(sizeof(poly_list));
        term_list->next = NULL;
        if (cmp (index1->data, index2->data) == 0 && flag == 1)
        {
            term_list->data.coef = index1->data.coef + index2->data.coef;
            term_list->data.index = index1->data.index;
            Insert_list (result,  term_list);
            index1 = index1->next;
            index2 = index2->next;
        }
        else if (cmp (index1->data, index2->data) == 0 && flag == 2)
        {
            term_list->data.coef = index1->data.coef - index2->data.coef;
            term_list->data.index = index1->data.index;
            Insert_list (result,  term_list);
            index1 = index1->next;
            index2 = index2->next;
        }
        else if (cmp(index1->data, index2->data) > 0)   //将当前节点指数大的插入result链表
        {
            term_list->data.coef = index1->data.coef;
            term_list->data.index = index1->data.index;
            Insert_list (result, term_list);
            index1 = index1->next;
```

```
        }
        else
        {
            term_list->data.coef = index2->data.coef;
            term_list->data.index = index2->data.index;
            Insert_list (result,  term_list);
            index2 = index2->next;
        }
    }
    /*接下来判断两链表长度不相同的情况*/
    if (index1 != NULL)             //index1长的情况
        while (index1 != NULL)
        {
                term_list = (poly_list*)malloc(sizeof(poly_list));
                term_list->next = NULL;
            term_list->data.coef = index1->data.coef;
            term_list->data.index = index1->data.index;
            Insert_list (result, term_list);
            index1 = index1->next;
        }
    else if (index2 != NULL)            //index2长的情况
        while (index2 != NULL)
        {
                term_list = (poly_list*)malloc(sizeof(poly_list));
                term_list->next = NULL;
            term_list->data.coef = index2->data.coef;
            term_list->data.index = index2->data.index;
            Insert_list(result, term_list);
            index2 = index2->next;
        }
    return result;
}
```

- 多项式升降序结果显示
    1. `Print_poly` 函数对传入参数-- 指针 `head` 进行多项式输出。
    2. 需要考虑特殊情况，系数为1时不输出1，系数为整数时不按照浮 点型输出，为此构造 `Control_float` 函数。

```
/*函数说明：
    判断一个浮点数是否是整数*/
int Control_float (double a)
{
    if (fabs (a - (int)a) < 0.01)               //近似认为等于0，a = (int)a
```

```
        return 0;
    else return 1;
}
```

```
/*函数说明:
    将计算结果用链表输出*/
void Print_poly (poly_list* head)
{
    poly_list* p = head;
    p = p->next;
    if (p == NULL)
        printf ("0");
    /*各种极端情况的判断*/
    while (p != NULL)
    {
        if (p == head->next)    //第一个节点输出和后续节点输出方式不同,第一个节点不含加减号
        {
                if (p->data.coef == 1 && p->data.index ==0)
                        printf ("%.0f", p->data.coef);          //另外当指数为0时不需要输出x
                if (p->data.coef != 1)
            {
                if (Control_float (p->data.coef) == 0)  //判断系数是否为整数，整数则输出整数，浮点数则输出浮点数
                    printf ("%.0f", p->data.coef);
                    else printf ("%.1f", p->data.coef);
            }
            if (p->data.index != 0)
            {
                        printf ("X^");
                    printf ("%d", p->data.index);
                }
        }
        else
        {
            if (p->data.coef > 0)           //后续节点需要输出加减号
                printf ("+");
            else if (p->data.coef < 0)
                printf ("-");
            if (fabs (p->data.coef) != 1)
            {
                if (Control_float (p->data.coef) == 0)
```

```
                    printf ("%.0f", fabs(p->data.coef));
                else printf ("%.1f", fabs(p->data.coef));
            }
            if (p->data.index != 0)
            {
                        printf ("X^");
                    printf ("%d", p->data.index);
                }
        }
        p = p->next;
    }
    printf ("\n\n\n");
}
```

- 计算器页面仿真
  - 由于本人只会C语言，所以只能用 `windows.h` 库，采用光标移动的方式来实现计算器页面。用 `Goto_xy` 函数控制光标的使用，用 `menu` 函数调用输出页面。

4. 调试以及遇到的问题

由于转系等缘故大一下学期没有进行任何形式的编程，所以对链表、数组等概念很生疏，在开始本实验前需要进行很多背景知识的复习和补充，花费了大量时间。

在实验过程中，主要卡在

1. 某些特殊情况，如：非法输入，系数是否为1，输出整数不能采取浮点数格式等地方，本质上还是因为自身编程知识水平不够，最后通过在网上查阅资料解决。
2. 计算器仿真页面的实现。因为缺少这方面的经验和知识，最后通过在网上查阅资料解决。
3. 写 `Get_coefs` 函数判断并存储输入项的系数时，因为对字符串概念生疏，花了很长时间才搞明白这个函数该怎么写。
4. 截至现在，程序里由一处bug，导致输入整数- x 时减号会被识别为乘号，但是输入 x -整数却能正常计算，这个问题有待上机检查时询问助教解决。

鉴于写代码时间拖得很长，模块分的很细， 除了一些语法错误外没有遇到大型思路上的漏洞。

5. 测试结果

1. $\left(2x+5x^{8}-3.1x^{11}\right)+\left(7-5x^{8}+11x^{9}\right)=\left(-3.1x^{11}+11x^{9}+2x+7\right)$
2. $\left(6x^{-3}-x+4.4x^{2}-1.2x^{9}\right)-\left(-6x^{-3}+5.4x^{2}-x^{2}+7.8x^{15}\right)=\left(-7.8x^{15}-1.2x^{9}+12x^{-3}-x\right)$
3. $\left(1+x+x^{2}+x^{3}+x^{4}+x^{5}\right)+\left(-x^{3}-x^{4}\right)=\left(1+x+x^{2}+x^{5}\right)$
4. $\left(x+x^{3}\right)+\left(-x-x^{3}\right)=0$
5. $\left(x+x^{100}\right)+\left(x^{100}+x^{200}\right)=\left(x+2x^{100}+x^{200}\right)$
6. $\left(x+x^{2}+x^{3}\right)+0=x+x^{2}+x^{3}$