# 题目：一元稀疏多项式计算器

班级：19184115

姓名：

学号：19063140

日期：2020.10.26

### 需求分析：

设计一个一元稀疏多项式简单计算器。

一元稀疏多项式简单计算器的基本功能是:

(1)输人并建立多项式;

(2)输出多项式,输出形式为整数序列:n,c1,e1, c2,e2...., cn,en,其中n是多项式的项数,ci和ei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。

### 概要设计：

LNode 为链表

LinkList 为链表指针

Status ListInit(LinkList\*) 链表初始化

Status ListInsert(LinkList L, int i, CType c, PType p) 在链表第i位插入(表头为0)

LinkList StringToList(char \*s) 把字符串多项式转化为链表多项式,返回表头指针

LinkList PolynomialAddition(LinkList lhs, LinkList rhs) 多项式加法

LinkList PolynomialSubtraction(LinkList lhs, LinkList rhs) 多项式减法

int LNodePrint(LNode L) 输出单个链表结点

void ListPrint(LinkList L) 按指数升序输出

int ListReversePrint(LinkList L, int is\_head) 按指数降序输出

### 调试分析：

1. 遇到的问题
   1. 问题：字符串转化为链表时候难以处理多种特殊情况的输入

解决方法：暴力枚举分类讨论各种情况

* 1. 问题：难以处理多种特殊情况的输出

解决方法：暴力枚举分类讨论各种情况

* 1. 问题：如果计算得到结果为0，没有输出

解决方法：特判

1. 时空分析

设两个多项式的项数分别为n,m

时间复杂度 O(n^2+m^2) 瓶颈在构建链表时的插入操作

空间复杂度 O(n+m)

### 用户使用说明：

输入为一行,两个由括号包裹的多项式,多项式的输入形式为类数学表达式,按指数升序输入

### 测试结果：

输入

(2x+5x^8—3.1x^11)+(7-5x^8+11x^9)

(6x^-3-x+4.4x^2-1.2x^9)-(-6x^-3+5.4x^2-x^2+7.8x^15)

(1+x+x^2+x^3+x^4+x^5)+(-x^3-x^4)

(x+x^3)+(-x-x^3)

(x+x^100)+(x^100+x^200)

(x+x^2+x^3)+(0)

输出

-3.1x^11+11.0x^9+2.0x+7.0

-7.8x^15-1.2x^9+x^2-x^2-x+12.0x^-3

+x^5+x^2+x+1.0

0

+x^200+2.0x^100+x

+x^3+x^2+x

### 附录：

1. #include <math.h>
2. #include <stdio.h>
3. #include <string.h>
4. #include <stdlib.h>
6. // #define DEBUG // 调试用
8. /\* 状态码识别类型 \*/
9. **typedef** **int** Status;
10. /\* 状态码 \*/
11. #define TRUE        1           //真
12. #define FALSE       0           //假
13. #define YES         1           //是
14. #define NO          0           //否
15. #define OK          1           //通过
16. #define ERROR       0           //错误
17. #define SUCCESS     1           //成功
18. #define UNSUCCESS   0           //失败
19. #define INFEASIBLE  -1          //不可行
21. #ifndef \_MATH\_H\_                //系统中已有此状态码定义，要避免冲突
22. #define OVERFLOW    -2          //堆栈上溢
23. #define UNDERFLOW   -3          //堆栈下溢
24. #endif
26. #ifndef NULL
27. #define NULL ((void\*)0)
28. #endif
30. /\* 数据类型 \*/
31. **typedef** **double** CType;
32. **typedef** **int** PType;
33. /\* 数据大小 \*/
34. #define N 100
36. **struct** LNode {
37. CType c;
38. PType p; // c\*x^p
39. **struct** LNode \*next;
40. };
41. **typedef** **struct** LNode LNode;
42. **typedef** LNode\* LinkList;
44. /\* 链表初始化 \*/
45. Status ListInit(LinkList \*L) {
46. \*L =  (LinkList)malloc(**sizeof**(LNode));
47. **if** (!(\*L)) exit(OVERFLOW);
48. (\*L)->next = NULL;
49. **return** OK;
50. }
52. /\* 在链表第i位插入(表头为0) \*/
53. Status ListInsert(LinkList L, **int** i, CType c, PType p) {
54. **if** (!L || i <= 0) **return** ERROR;
55. **int** cur = 0;
56. **while** (L && cur < i-1) {
57. L = L->next;
58. ++cur;
59. }
60. **if** (!L || cur != i-1) **return** ERROR;
61. LinkList s = (LinkList)malloc(**sizeof**(LNode));
62. **if** (!s) exit(OVERFLOW);
63. s->c = c;
64. s->p = p;
65. s->next = L->next;
66. L->next = s;
67. **return** OK;
68. }
70. /\* 把字符串多项式转化为链表多项式 \*/
71. LinkList StringToList(**char** \*s) {
72. LinkList L;
73. CType c, tag;
74. PType p;
75. ListInit(&L);
76. **int** i = 0;
77. **do** {
78. // 系数符号
79. **if** (\*s == '-') {
80. tag = -1;
81. ++s; // 读入'-'
82. } **else** {
83. tag = 1;
84. **if** (\*s == '+') ++s; // 读入'+'
85. }
86. // 系数
87. **if** (\*s == 'x') {
88. c = 1;
89. } **else** {
90. sscanf(s, "%lf", &c);
91. **while** (\*s == '.' || (\*s >= '0' && \*s <= '9')) ++s;
92. }
93. // 指数
94. **if** (\*s == 'x') {
95. ++s; // 读入'x'
96. **if** (\*s == '^') {
97. ++s; // 读入'^'
98. sscanf(s, "%d", &p);
99. **if** (\*s == '-') ++s;
100. **while** (\*s >= '0' && \*s <= '9') ++s;
101. } **else** {
102. p = 1;
103. }
104. } **else** {
105. p = 0;
106. }
107. ListInsert(L, ++i, tag\*c, p);
108. } **while** (\*s);
109. **return** L;
110. }
112. /\* 多项式加法 \*/
113. LinkList PolynomialAddition(LinkList lhs, LinkList rhs) {
114. LinkList sum;
115. **int** i = 0;
116. ListInit(&sum);
117. lhs = lhs->next;
118. rhs = rhs->next;
119. **while** (lhs || rhs) {
120. **if** (lhs && rhs && lhs->p == rhs->p) {
121. ListInsert(sum, ++i, lhs->c+rhs->c, lhs->p);
122. lhs = lhs->next;
123. rhs = rhs->next;
124. } **else** **if** (!rhs || (lhs && lhs->p < rhs->p)) {
125. ListInsert(sum, ++i, lhs->c, lhs->p);
126. lhs = lhs->next;
127. } **else** **if** (!lhs || (rhs && lhs->p > rhs->p)) {
128. ListInsert(sum, ++i, rhs->c, rhs->p);
129. rhs = rhs->next;
130. }
131. }
132. **return** sum;
133. }
135. /\* 多项式减法 \*/
136. LinkList PolynomialSubtraction(LinkList lhs, LinkList rhs) {
137. LinkList sum;
138. **int** i = 0;
139. ListInit(&sum);
140. lhs = lhs->next;
141. rhs = rhs->next;
142. **while** (lhs || rhs) {
143. **if** (lhs && rhs && lhs->p == rhs->p) {
144. ListInsert(sum, ++i, lhs->c-rhs->c, lhs->p);
145. lhs = lhs->next;
146. rhs = rhs->next;
147. } **else** **if** (!rhs || (lhs && lhs->p < rhs->p)) {
148. ListInsert(sum, ++i, lhs->c, lhs->p);
149. lhs = lhs->next;
150. } **else** **if** (!lhs || (rhs && lhs->p > rhs->p)) {
151. ListInsert(sum, ++i, -rhs->c, rhs->p);
152. rhs = rhs->next;
153. }
154. }
155. **return** sum;
156. }
158. **int** LNodePrint(LNode L) {
159. **if** (L.c == 0) **return** 0;
160. putchar(L.c > 0 ? '+' : '-');
161. **if** (!L.p || fabs(L.c) != 1) printf("%.1f", fabs(L.c));
162. **if** (L.p) putchar('x');
163. **if** (L.p != 0 && L.p != 1) printf("^%d", L.p);
164. **return** 1;
165. }
167. **void** ListPrint(LinkList L) {
168. **int** not\_zero = 0;
169. L = L->next;
170. **while** (L) {
171. not\_zero += LNodePrint(\*L);
172. L = L->next;
173. }
174. **if** (!not\_zero) putchar('0');
175. putchar('\n');
176. }
178. **int** ListReversePrint(LinkList L, **int** is\_head) {
179. **if** (!L) **return** 0;
180. **int** not\_zero = ListReversePrint(L->next, 0);
181. **if** (is\_head) {
182. **if** (!not\_zero) putchar('0');
183. putchar('\n');
184. **return** not\_zero;
185. } **else** {
186. **return** not\_zero+LNodePrint(\*L);
187. }
188. }
190. **char** CutString(**char** \*str, **char** \*lstr, **char** \*rstr) {
191. **char** add\_or\_minus;
192. **for** (**int** i = 0, l = 0, flag = 0; str[i]; ++i) {
193. **if** (str[i] == '(') {
194. l = i+1;
195. **if** (i) add\_or\_minus = str[i-1];
196. } **else** **if** (str[i] == ')') {
197. **char** \*cur = flag ? rstr : lstr;
198. flag = 1;
199. strncpy(cur, str+l, i-l);
200. cur[i-l] = '\0';
201. }
202. }
203. **return** add\_or\_minus;
204. }
206. **int** main() {
207. **static** **char** str[N], lstr[N], rstr[N];
208. LinkList lhs, rhs, sum;
209. **while** (~scanf("%s", str)) {
210. **char** add\_or\_minus = CutString(str, lstr, rstr);
211. #ifdef DEBUG
212. printf("lhs:%s\n", lstr);
213. printf("rhs:%s\n", rstr);
214. #endif
215. lhs = StringToList(lstr);
216. rhs = StringToList(rstr);
217. #ifdef DEBUG
218. printf("lhs:"); ListPrint(lhs);
219. printf("rhs:"); ListPrint(rhs);
220. #endif
221. sum = add\_or\_minus == '+' ?
222. PolynomialAddition(lhs, rhs) :
223. PolynomialSubtraction(lhs, rhs);
224. ListReversePrint(sum, 1);
225. }
226. **return** 0;
227. }