哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称：数据结构与算法课程类型：必修

实验项目：线性表的链式存储结构与应用

实验题目：线性结构及其应用

实验日期：2018年11月7日

班级：1703002

学号：1170500913

姓名：熊健羽

|  |  |  |
| --- | --- | --- |
| 设计成绩 | 报告成绩 | 指导老师 |
|  |  | 张岩 |

1. **实验目的**
   1. 熟悉线性表的链式存储结构
   2. 熟悉线性表的链式存储结构的基本操作
   3. 掌握线性表的链式存储结构的基本算法及其应用
2. **实验要求及实验环境**
   1. 实验要求
      1. 以链表存储一元多项式，在此基础上完成对多项式的代数操作。
      2. 能够输入多项式（可以按各项的任意输入顺序，建立按指数降幂排列的多项式）和输出多项式（按指数降幂排列）,以文件形式输入和输出，并显示。
      3. 能够计算多项式在某一点x=x0的值，其中x0是一个浮点型常量，返回结果为浮点数。
      4. 能够给出计算两个多项式加法、减法、乘法和除法运算的结果多项式，除法运算的结果包括商多项式和余数多项式。
      5. 要求尽量减少乘法和除法运算中间结果的空间占用和结点频繁的分配与回收操作。（提示：利用循环链表结构或者可用空间表的思想，把循环链表表示的多项式返还给系统或者可用空间表，从而解决上述问题）。
   2. 实验环境
      1. 硬件环境

CPU：Intel(R) Core(TM) i5-7200U @ 2.50GHz (64位)

GPU：Intel(R) HD Graphics 620

Nvidia GeForce 940MX

物理内存：8.00GB

磁盘：1TB HDD

128GB SSD

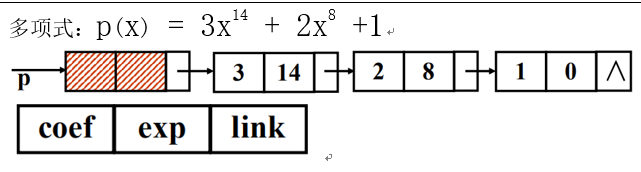
* + 1. 软件环境：

OS: Microsoft Windows 10 家庭中文版

编译环境：MinGW-W64

IDE：VS code、code::blocks

1. **设计思想**（本程序中的用到的所有数据类型的定义，主程序的流程图及各程序模块之间的调用关系）
   1. 逻辑设计
      1. 数据结构:多项式的逻辑结构为线性表，每个表项存储指数（double）、系数（int）.



* + 1. 各个函数的功能及其调用关系：
       1. 链表的基础操作：

插入 :void insert(double x, int y, pos p);

新建 : LIST makeNull(void);

释放整个多项式：void deletePoly(LIST head);

* + - 1. 读入多项式：

函数原型：void readPoly(LIST head);

调用函数：链表插入（insert），文件读字符串（fgets），按格式读入字符串（sscanf）

* + - 1. 输出多项式：

函数原型：void printPoly(LIST head);

调用函数：按格式向文件中写字符（fprintf）

* + - 1. 多项式指数降序排序：

函数原型：void polySort(LIST head);

* + - 1. 多项式排序、合并同类项、去除零项：

函数原型：pos polyClear(LIST head);

调用函数：多项式排序（polySort）

* + - 1. 加法操作:

函数原型： LIST polyAdd(LIST head1, LIST head2);

调用函数：多项式排序、合并同类项、去除零项（polyClear），新建多项式（makeNull），插入（insert）

* + - 1. 减法操作:

函数原型： LIST polyAdd(LIST head1, LIST head2);

调用函数：多项式排序、合并同类项、去除零项（polyClear），新建多项式（makeNull），插入（insert）

* + - 1. 乘法操作:

函数原型：LIST polyMult(LIST head1, LIST head2);

调用函数：新建多项式（makeNull），多项式排序、合并同类项、去除零项（polyClear），插入（insert）

* + - 1. 多项式求值:

函数原型：double polyValue(LIST head, double x);

函数调用：多项式排序、合并同类项、去除零项（polyClear）

* + - 1. 除法操作:

函数原型：LIST \*polyDiv(LIST head1, LIST head2);

调用函数：新建多项式（makeNull），多项式排序、合并同类项、去除零项（polyClear），插入（insert）

* + - 1. 根据运算符输出结果:

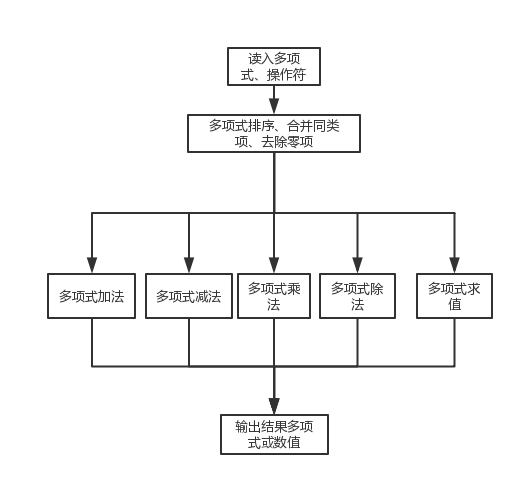
函数原型：void figure(char op, LIST poly1, LIST poly2);

调用函数：多项式加法（polyAdd）， 多项式减法（polySub），多项式除法（polyDiv），多项式乘法（polyMult），文件读字符串（fgets），释放整个多项式（delete），按格式向文件中写字符（fprintf）

* + 1. 主函数：

调用：新建多项式（makeNull），文件打开（fopen），读多项式（readPoly）， 读文件字符（fgetc）， 根据运算符输出结果（figure），释放多项式（deletePoly），文件关闭（fclose）

* + 1. 主程序流程：

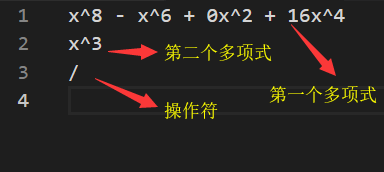


* 1. 物理设计
     1. 多项式的存储结构：链式存储结构，为带表头节点的单链表。

1. **typedef** **struct** POLYNODE
2. {
3. **double** coef;
4. **int** exp;
5. **struct** POLYNODE \*next;
6. } polynode;

定义两种数据类型，分别代表某一多项式、多项式的某一项的位置：

1. **typedef** **struct** POLYNODE \*LIST;
2. **typedef** **struct** POLYNODE \*pos;
   * 1. 输入输出文件为.txt格式，使用FILE指针打开、读取、写入、关闭。
     2. 文件中的输入格式：



* + 1. 函数的实现思路：
       1. 读入多项式：

把文件中的一行，先放入一个字符数组中，依次处理这个数组的每一个字符。先把exp置为0，coef置为1，代表缺省值。设置一个枚举类型flag，可选值为COEF、EXP，分别代表下一次遇到数字时，当做系数/指数读入。比如,遇到 ^ 号，flag置为EXP；遇到+ - 号，flag置为COEF。每次遇到加号/减号，代表某一项读入结束，此时把上一轮读入的系数、指数插入链表中。如此往复。

时间复杂度：需读入每个元素，O(n)

空间复杂度：需一个字符数组暂存，O(n)

* + - 1. 输出多项式：

如果链表为空，输出0；若某项的系数为1且次数不为0，则省略系数；若某项的次数为零，则只显示系数；若系数为负数，则省略该项前面的加号。（具体细节见代码注释）

时间复杂度：O(n)

空间复杂度：O(1)

* + - 1. 多项式指数降序排序：

采用选择排序法，通过改变指针的指向，改变链表中节点的次序。

时间复杂度：O(n^2)

空间复杂度：O(1)

* + - 1. 多项式排序、合并同类项、去除零项：

首先调用排序函数。再设置两个指针，一前一后，当前后两节点指数相等时，则把后一节点的系数加进前面的节点，并删除后一个节点；若有一个节点系数为零，则删除该节点。

时间复杂度：O(n^2)

排序：O(n^2) 合并同类项、去除零项：O(n)

O(n^2) + O(n) = O(n^2)

空间复杂度：O(1)

* + - 1. 加法操作:

比较两个多项式的最高此项的系数，如果指数相同则系数相加后存入目标多项式，不同则把指数高的接到目标多项式中

时间复杂度:O(m + n)

空间复杂度:O(m)或O(n)

* + - 1. 减法操作:

比较两个多项式的最高此项的系数，如果指数相同则系数相减，后存入目标多项式，不同则把指数高的接到目标多项式中（被减多项式接入正值，减数多项式接入负值）

时间复杂度:O(m + n)

空间复杂度:O(m)或O(n)

* + - 1. 乘法操作:

设相乘的两个多项式的次数分别是m，n。则结果多项式的最大长度为m+n+1。建立一个长度为m+n+1的循环链表，将每个节点的系数初始化为0，指数按降幂初始化。依次将每一项相乘，得到一个待插入的项，定义一个位置指针，沿着循环链表移动，当找到与待插入的项的指数相等的节点，就把待插入的项的系数加入其中，如此循环往复。每个项都插入完毕后，解开循环链表，调用polyClear函数，清除所有的系数为零的项，即得到结果。

时间复杂度:O(m \* n)

空间复杂度:O(m + n)

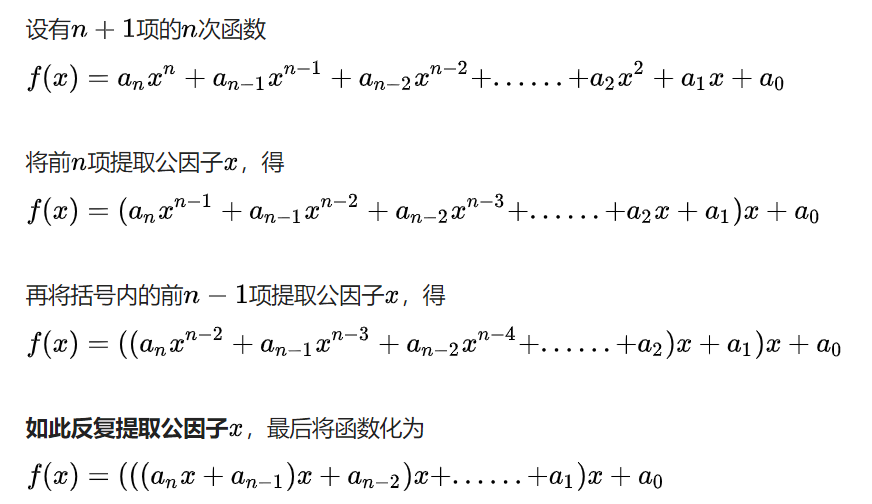
* + - 1. 除法操作:

首先判断除数多项式是否为零，为零则报错。再判断被除多项式是否为零，为零则直接输出商=0，余数=0 。接着判断被除多项式的次数是否小于除数多项式的次数，若是，则输出商 = 0，余数 = 被除多项式。

最后开始一般情况的计算。建立一个长度为m+1的循环链表，将每个节点的系数初始化为0，指数按降幂初始化。把被除多项式先按照次项的对应关系放入其中。设置一个指针highestExp，指向循环链表中系数非零的最高次项。每次先给商多项式加入一项：系数=循环链表中最高次项的系数 / 除数的最高次项的系数，指数=循环链表中最高次项的系数 - 除数的最高次项的系数,再模仿乘法操作，将商的该项与除数多项式相乘，每项取相反数之后，再加入循环链表中，如此循环往复，每次更新highestExp指针的位置，直到其指向的位置的指数小于除数的指数，或者整个循环链表为零。此时，商多项式建立完毕；循环链表即为余数多项式。解开循环链表，调用polyClear函数，清除所有的系数为零的项，即得到余数多项式。

* + - 1. 多项式求值:

采用秦九韶算法：



时间复杂度：O(n)

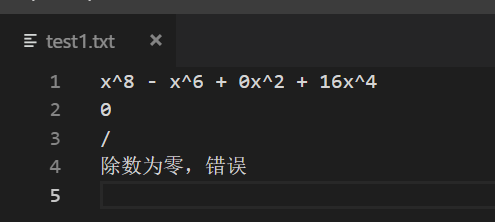
空间复杂度：O(1)

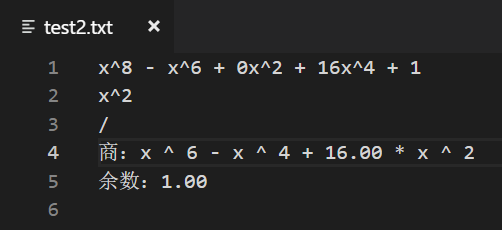
* + - 1. 根据运算符输出结果:

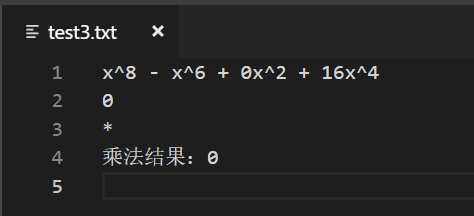
读取文件中的运算符（\* / + - x），判断进行何种操作。

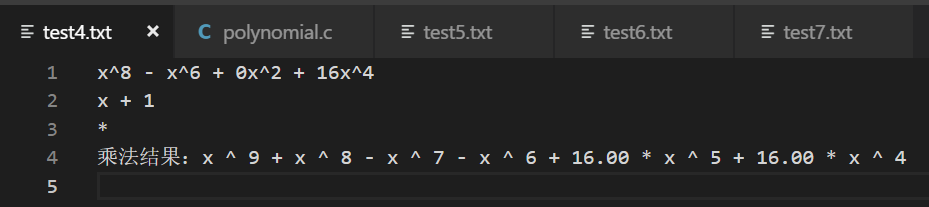
若读到（\* / + -）则进行相应的算数操作。读到x时，继续读后面的数字，作为代入x的数值。

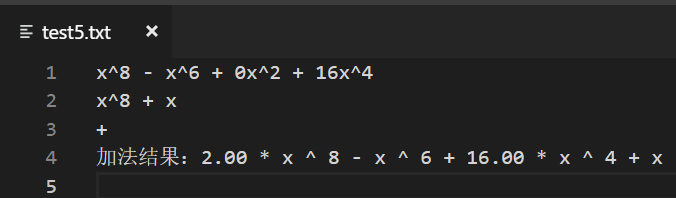
1. **测试结果**

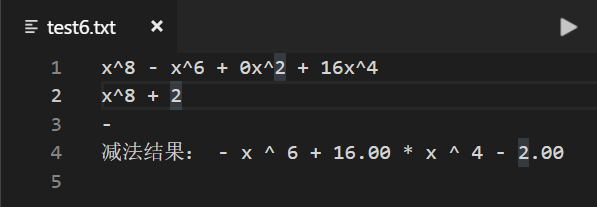


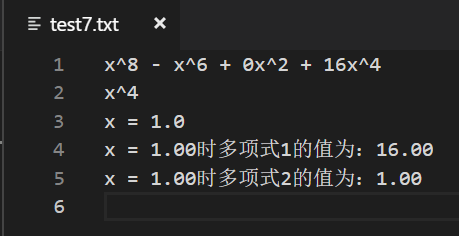












1. **经验体会与不足**

通过本次实验，熟练地掌握了带表头的单链表和循环列表的逻辑结构、存储结构和实现方法。链表是一种典型的动态储存结构，优点是高灵活性和空间节约， 缺点是查找元素较为麻烦，基本操作实现需要合理封装。

在对乘法、除法的优化过程中，体会到了存储结构对算法空间复杂度、时间复杂度的影响作用。

不足：

* + 1. 个别函数的操作过于繁琐，没有想出更为合理的模块化思路。
    2. 编写初期，思路不是很清晰，靠后期的缝缝补补，程序才逐渐满足要求。
    3. 还有一些细节可优化，由于时间关系，并没有充分优化。

1. **附录：源代码（带注释）**
2. #include <stdio.h>
3. #include <stdlib.h>
4. #define QUO 0 //商
5. #define REM 1 //余数
7. **FILE** \*fp = NULL;
9. **typedef** **struct** POLYNODE
10. {
11. **double** coef;
12. **int** exp;
13. **struct** POLYNODE \*next;
14. } polynode;
16. **typedef** **struct** POLYNODE \*LIST;
17. **typedef** **struct** POLYNODE \*pos;
19. **typedef** **enum**
20. {
21. EXP,
22. COEF
23. } read\_what;
25. **void** insert(**double** x, **int** y, pos p) //向p位置插入一个coef = x, exp = y的节点
26. {
27. pos s = (pos)malloc(**sizeof**(polynode));
28. s->coef = x;
29. s->exp = y;
30. s->next = p->next;
31. p->next = s;
32. }
34. **void** deletePoly(LIST head) //删除整个个多项式并释放空间
35. {
36. pos p = head->next;
37. **while** (p != NULL) //删空所有有效节点
38. {
39. head->next = p->next; //删除第一个有效节点
40. free(p);
41. p = head->next;
42. }
43. free(head); //释放头结点
44. }
46. LIST makeNull(**void**) //生成新的多项式，并置空
47. {
48. LIST head = (LIST)malloc(**sizeof**(polynode));
49. head->next = NULL;
50. **return** head;
51. }
53. //从文件中读入一个多项式，
54. //样例: 3x^6 + 2 + 9x^2
55. //     3\*x^7 + 2 + 9\*x^2
56. **void** readPoly(LIST head)
57. {
58. pos poly\_p;       //用来遍历整个多项式，作为插入操作的位置
59. poly\_p = head;    //poly\_p赋初值
60. **char** s[1000];     //存储文件中读取的一行字符串
61. **char** \*c\_readPoly; //对上述的字符串s进行逐项读取的指针
62. **int** t\_exp = 0;
63. **double** t\_coef = 1.0; //缺省值: t\_exp = 0, t\_coef = 1
64. **double** sign = 0;     // + -> 0, - -> 1
65. read\_what flag = COEF;
66. //表示下一次 c\_readPoly指向数字时，当做指数还是系数，缺省值为系数
67. fgets(s, 1000, fp); //读一行字符串，放入s中
69. //下面处理s中的字符，读入各项的知识与系数
70. **for** (c\_readPoly = s; \*c\_readPoly != '\0'; c\_readPoly++) //从头读到尾，直到结束
71. {
72. **if** (\*c\_readPoly <= '9' && \*c\_readPoly >= '0') //遇到数字
73. {
74. **switch** (flag)
75. {
76. **case** COEF: //读入系数
77. sscanf(c\_readPoly, "%lf", &t\_coef);
78. t\_coef = t\_coef - 2 \* sign \* t\_coef; // 带上符号
79. sign = 0;                            //符号重新置为正
80. **break**;
81. **case** EXP: //读入指数
82. sscanf(c\_readPoly, "%d", &t\_exp);
83. t\_exp = t\_exp - 2 \* (**int**)sign \* t\_exp; //带上符号
84. sign = 0;
85. flag = COEF; //下一项应读入系数，也代表这项结束了
86. **break**;
87. }
88. **while** ((\*c\_readPoly <= '9' && \*c\_readPoly >= '0') || \*c\_readPoly == '.')
89. c\_readPoly++; //读完数字之后，跳过之后的数字和小数点
90. }
92. **switch** (\*c\_readPoly)
93. {
94. **case** '+':                          //遇到 + 号，代表一项结束
95. insert(t\_coef, t\_exp, poly\_p); //把一项存入链表
96. poly\_p = poly\_p->next;         //节点指针后移
98. t\_coef = 1.0;
99. t\_exp = 0;
100. flag = COEF; //下一个数字应该读入系数
101. sign = 0;    //符号置为正
102. **break**;
103. **case** '-':             //遇到 - 号
104. **if** (flag == COEF) //如果下面应该读入系数，也说明一项结束
105. {
106. **if** (c\_readPoly >= s + 2)
107. {
108. insert(t\_coef, t\_exp, poly\_p); //把一项存入链表
109. poly\_p = poly\_p->next;         //节点指针后移
110. }
111. t\_coef = -1.0;
112. t\_exp = 0; //coef, exp置为初始值
113. }
114. sign = 1; //符号置为负
115. **break**;
116. **case** '^': //遇到 ^ 号
117. sign = 0;
118. flag = EXP; //接下来应读入指数
119. **break**;
120. **case** 'X':
121. **case** 'x':
122. t\_exp++; //遇到x，代表exp指数至少为1，由零加1
123. **break**;
124. **default**:
125. ;
126. }
127. }
128. insert(t\_coef, t\_exp, poly\_p); //将最后一项放入链表
129. }
131. **void** printPoly(LIST head) //向文件里写入一个多项式
132. {
133. pos p;
134. **if** (head->next == NULL)
135. {
136. fprintf(fp, "0"); //如果链表为空，输出'0'
137. }
138. **for** (p = head->next; p != NULL; p = p->next)
139. {
140. **if** (p->coef != 0) //若这项的系数不为零，才显示该项
141. {
142. **if** (p->coef < 0)
143. fprintf(fp, " - "); //如果系数小于零，输出一个 '-' 号
145. //如果指数等于零、或者系数不为+1、-1，则输出系数部分
146. **if** ((p->coef != 1 && p->coef != -1) || p->exp == 0)
147. {
148. **if** (p->coef > 0)
149. fprintf(fp, "%.2f", p->coef);
150. **if** (p->coef < 0)
151. fprintf(fp, "%.2f", -p->coef);
152. **if** (p->exp != 0)
153. fprintf(fp, " \* ");
154. }
156. //如果指数不为零，才显示指数
157. **if** (p->exp != 0)
158. {
159. fprintf(fp, "x");
160. **if** (p->exp != 1)
161. fprintf(fp, " ^ %d", p->exp);
162. }
164. //如果不是最后一项, 则输出一个加号
165. **if** (p->next != NULL && p->next->coef > 0)
166. fprintf(fp, " + ");
167. }
168. }
169. fprintf(fp, "\n"); //输出换行符
170. }
172. **void** polySort(LIST head) //按照指数降序，选择法排序
173. {
174. pos p0, p, q0, q, r0, r, temp;
175. **int** max\_exp;
176. **for** (p0 = head, p = head->next; p->next != NULL; p0 = p0->next, p = p->next)
177. {
178. max\_exp = p->exp;
179. r0 = p0;
180. r = p;
181. **for** (q0 = p, q = p->next; q != NULL; q0 = q0->next, q = q->next)
182. **if** (max\_exp < q->exp)
183. {
184. r0 = q0;
185. r = q;
186. max\_exp = q->exp;
187. }
189. p0->next = r;
190. r0->next = p;
192. temp = p->next;
193. p->next = r->next;
194. r->next = temp;
196. temp = p;
197. p = r;
198. r = temp;
199. }
200. }
202. pos polyClear(LIST head) //合并同类项, 删除0项, 并排序, 返回指向末尾节点的指针
203. {
204. polySort(head);
205. pos p, q;
206. p = head->next;
207. q = p->next;
208. **while** (p != NULL && q != NULL)
209. {
210. **if** (p->exp == q->exp)
211. {
212. p->coef = p->coef + q->coef;
213. p->next = q->next;
214. free(q);
215. q = p->next;
216. }
217. **else**
218. {
219. p = p->next;
220. q = q->next;
221. }
222. }
223. p = head;
224. q = head->next;
225. **while** (q != NULL)
226. {
227. **if** (q->coef == 0)
228. {
229. p->next = q->next;
230. free(q);
231. q = p->next;
232. }
233. **else**
234. {
235. p = p->next;
236. q = q->next;
237. }
238. }
239. **return** p;
240. }
242. LIST polyAdd(LIST head1, LIST head2) //两多项式相加，合并同类项并排序
243. {
244. polyClear(head1);
245. polyClear(head2); //首先合并同类项并排序，保证两个多项式都是指数降序
246. LIST head = makeNull();
247. pos p = head1->next, q = head2->next, r = head;
248. **while** (p != NULL && q != NULL)
249. {
250. **if** (p->exp == q->exp) //若系数相同，直接相加
251. {
252. insert(p->coef + q->coef, p->exp, r);
253. r = r->next;
254. p = p->next;
255. q = q->next; //指针均向后移
256. }
258. //不相同，把系数较大的插入链表中，其指针向后移
259. **else** **if** (p->exp > q->exp)
260. {
261. insert(p->coef, p->exp, r);
262. r = r->next;
263. p = p->next;
264. }
265. **else**
266. {
267. insert(q->coef, q->exp, r);
268. r = r->next;
269. q = q->next;
270. }
271. }
273. //把剩余项放在多项式末尾
274. **while** (p != NULL)
275. {
276. insert(p->coef, p->exp, r);
277. r = r->next;
278. p = p->next;
279. }
280. **while** (q != NULL)
281. {
282. insert(q->coef, q->exp, r);
283. r = r->next;
284. q = q->next;
285. }
286. **return** head;
287. }
289. LIST polySub(LIST head1, LIST head2) //两多项式相减，合并同类项并排序
290. {
291. // 基本思想同多项式加法
292. polyClear(head1);
293. polyClear(head2);
294. LIST head = makeNull();
295. pos p = head1->next, q = head2->next, r = head;
296. **while** (p != NULL && q != NULL)
297. {
298. **if** (p->exp == q->exp)
299. {
300. insert(p->coef - q->coef, p->exp, r); //改为减运算
301. r = r->next;
302. p = p->next;
303. q = q->next;
304. }
305. **else** **if** (p->exp > q->exp)
306. {
307. insert(p->coef, p->exp, r);
308. r = r->next;
309. p = p->next;
310. }
311. **else**
312. {
313. insert(-q->coef, q->exp, r); //插入减数的相反数
314. r = r->next;
315. q = q->next;
316. }
317. }
318. **while** (p != NULL)
319. {
320. insert(p->coef, p->exp, r);
321. r = r->next;
322. p = p->next;
323. }
324. **while** (q != NULL)
325. {
326. insert(-q->coef, q->exp, r); //插入减数的相反数
327. r = r->next;
328. q = q->next;
329. }
330. **return** head;
331. }
333. **void** polyCopy(LIST head1, LIST head2) //把head2多项式拷贝到head1
334. {
335. pos p1 = head1;
336. pos p2 = head2->next;
337. **for** (; p2 != NULL; p2 = p2->next, p1 = p1->next)
338. insert(p2->coef, p2->exp, p1);
339. }
341. **double** polyValue(LIST head, **double** x) //代入x求值(秦九韶)
342. {
343. polyClear(head);
344. **double** res = 0;
345. **double** temp\_coef;
346. **int** i;
347. pos p = head->next;
348. **for** (i = head->next->exp; i >= 0; i--) //一共迭代 最高次+1 次
349. {
350. **if** (i == p->exp)
351. {
352. temp\_coef = p->coef;
353. **if** (p->next != NULL)
354. p = p->next;
355. }
356. **else**
357. temp\_coef = 0.0;
358. res = res \* x + temp\_coef;
359. }
360. **return** res;
361. }
363. LIST polyMult(LIST head1, LIST head2) //多项式相乘
364. //使用循环链表
365. {
366. LIST head = makeNull(), circle\_head;
367. **int** max\_length = head1->next->exp + head2->next->exp;
368. //循坏链表长度为两个最高次项的指数相加
369. **int** i;
370. **int** new\_exp;
371. **double** new\_coef;
372. pos p, p1, p2, circle\_rear;
373. polyClear(head1);
374. polyClear(head2); //整理两个多项式
376. //构造循环链表，把每个次项的系数先置为0.0
377. **for** (i = 0, p = head; i <= max\_length; i++, p = p->next)
378. {
379. insert(0.0, max\_length - i, p);
380. }
381. p->next = head->next;  //把链表首尾相接，形成循环链表
382. circle\_head = p->next; //标记表头节点
383. circle\_rear = p;       //标记表尾节点
385. **for** (p1 = head1->next; p1 != NULL; p1 = p1->next)
386. {
387. **for** (p2 = head2->next; p2 != NULL; p2 = p2->next)
388. {
389. new\_coef = p1->coef \* p2->coef;
390. new\_exp = p1->exp + p2->exp;
391. //遍历每一项，计算出待相加项的系数与指数
393. **while** (p->exp != new\_exp)
394. p = p->next; //找到该次项在循环链表中的位置
396. p->coef += new\_coef; //相应次项的系数增加
397. }
398. }
399. head->next = circle\_head; //重新将循环链表还原
400. circle\_rear->next = NULL; //表尾节点置为空
401. polyClear(head);          //整理结果
402. **return** head;
403. }
405. LIST \*polyDiv(LIST head1, LIST head2) /\*多项式相除，返回一个数组首地址，
406. 两个元素分别为商多项式和余数多项式的头指针\*/
407. {
408. LIST \*array = (LIST \*)calloc(2, **sizeof**(LIST));
409. array[QUO] = makeNull();
410. array[REM] = makeNull(); //分别为商多项式和余数多项式申请空间
411. pos circleHead\_rem;      //循环链表的头指针
412. pos circleRear\_rem;      //循环链表的尾指针
413. pos highestExp;          //系数不为零的最高次项的位置
414. pos p1, p2;              //辅助位置指针
415. **int** new\_exp, i;
416. **double** new\_coef;
417. polyClear(head1);
418. polyClear(head2);                  //整理待运算的多项式
419. **int** max\_length = head1->next->exp; //余数多项式的最大长度减1 为被除多项式的次数
420. pos p\_quo = array[QUO];
421. pos p\_rem = array[REM];
422. **if** (head2->next == NULL) //如果除数多项式为0，返回空指针，报错
423. {
424. **return** NULL;
425. }
426. **if** (head1->next == NULL) //如果被除多项式为0,则商和余数均为0，直接返回
427. {
428. **return** array;
429. }
431. **if** (head1->next->exp < head2->next->exp) /\*如果被除多项式的次数 大于除数多项式的
432. 次数，则商为0，余数为被除多项式 \*/
433. polyCopy(array[REM], head1);
435. **else**
436. {
437. /\*把余数多项式初始化为被除多项式。并将其变为循环链表\*/
438. **for** (i = 0, p\_rem = array[REM]; i <= max\_length; i++, p\_rem = p\_rem->next)
439. {   // 插入max\_length + 1个元素，指数降序，系数为0.0
440. insert(0.0, max\_length - i, p\_rem);
441. }
442. circleHead\_rem = array[REM]->next;     //保存循环链表的头节点
443. circleRear\_rem = p\_rem;                //保存循环链表的尾节点
444. circleRear\_rem->next = circleHead\_rem; //首尾相接
445. highestExp = circleHead\_rem;           //初始化 系数不为零的最高次项的位置
446. **for** (p1 = head1->next, p\_rem = circleHead\_rem; p1 != NULL; p1 = p1->next)
447. {   //把被除多项式填入循环链表
448. **while** (p\_rem->exp != p1->exp) //找到对应项的位置
449. p\_rem = p\_rem->next;
450. p\_rem->coef += p1->coef;
451. }
453. /\*开始除法过程\*/
454. **while** (highestExp->exp >= head2->next->exp) //余数多项式次数小于除数多项式次数时停止循环
455. {
456. insert(highestExp->coef / head2->next->coef, highestExp->exp - head2->next->exp, p\_quo);
457. //计算出商多项式的一项并插入链表中
458. p\_quo = p\_quo->next;
460. /\*更新余数多项式\*/
461. **for** (p2 = head2->next; p2 != NULL; p2 = p2->next)
462. {
463. new\_coef = p\_quo->coef \* p2->coef;
464. new\_exp = p\_quo->exp + p2->exp;
465. //商多项式与除数多项式相乘
466. **while** (p\_rem->exp != new\_exp)
467. p\_rem = p\_rem->next;
468. p\_rem->coef -= new\_coef; //找到对应次项，更新 (相减)
469. }
471. /\*更新系数不为零的最高次项的位置\*/
472. **while** (highestExp->coef == 0)
473. {
474. highestExp = highestExp->next;
475. **if** (highestExp == circleRear\_rem) //如果转了一圈
476. **if** (highestExp->coef == 0)
477. **goto** jumpout; /\*尾部仍然系数为零，证明整个循环链表为0，
478. 直接结束除法，使用goto跳出多重循环\*/
479. }
480. // polyClear(array[REM]);
481. // if (array[REM]->next == NULL)
482. //     break;
483. }
484. }
485. jumpout:
486. circleRear\_rem->next = NULL; //解除循环链表
487. polyClear(array[QUO]);
488. polyClear(array[REM]); //整理商多项式与余数多项式
489. **return** array;
490. }
492. **void** figure(**char** op, LIST poly1, LIST poly2) //根据读取的操作符，判断运算类别并运算，输出结果
493. {
494. LIST poly3;
495. LIST poly4; //两个结果多项式
496. LIST \*resDiv;
497. **double** val, resVal;
498. **char** temps[100], \*p\_temp;
500. /\*根据操作符，进行不同的运算\*/
501. **switch** (op)
502. {
503. **case** '+':
504. poly3 = polyAdd(poly1, poly2);
505. **break**;
506. **case** '-':
507. poly3 = polySub(poly1, poly2);
508. **break**;
509. **case** '\*':
510. poly3 = polyMult(poly1, poly2);
511. **break**;
512. **case** '/':
513. resDiv = polyDiv(poly1, poly2);
514. **break**;
515. **case** 'x':
516. **case** 'X':      //遇到x，即为求某点多项式的值
517. fgets(temps, 100, fp);
518. **for** (p\_temp = temps; \*p\_temp != '\0'; p\_temp++)
519. {
520. **if** (\*p\_temp >= '0' && \*p\_temp <= '9')
521. {
522. sscanf(p\_temp, "%lf", &val);  //若读取到数字，则写入变量val中
523. **break**;
524. }
525. }
526. **break**;
527. **default**:
528. ;
529. }
530. fclose(fp);  //读数据完毕，关闭文件，开始写数据
532. **if** ((fp = fopen("F:\\data\_structure\\Lab\\lab1\\out.txt", "a")) == NULL)
533. {      // 写入方式：在文件末尾添加文本
534. printf("open failed\n"); //若打开文件失败，则退出程序
535. exit(1);
536. }
538. **if** (op == '/')  //若运算为除法，则需输出两个多项式
539. {
540. **if** (resDiv == NULL)  //除数为零，不合法
541. {
542. fprintf(fp, "除数为零，错误\n");
543. exit(1);
544. }
545. poly3 = resDiv[QUO];
546. poly4 = resDiv[REM];
547. fprintf(fp, "商：");
548. printPoly(poly3);
549. deletePoly(poly3);  //多项式使用完毕，释放其所有空间
550. fprintf(fp, "余数：");
551. printPoly(poly4);
552. deletePoly(poly4); //多项式使用完毕，释放其所有空间
553. }
554. **else**
555. {
556. **switch** (op)
557. {
558. **case** '+':
559. fprintf(fp, "加法结果：");
560. printPoly(poly3);
561. deletePoly(poly3);
562. **break**;
563. **case** '-':
564. fprintf(fp, "减法结果：");
565. printPoly(poly3);
566. deletePoly(poly3);
567. **break**;
568. **case** '\*':
569. fprintf(fp, "乘法结果：");
570. printPoly(poly3);
571. deletePoly(poly3);
572. **break**;
573. **case** 'x':
574. **case** 'X':
575. fprintf(fp, "x = %.2lf时多项式1的值为：%.2lf\n", val, polyValue(poly1, val));
576. fprintf(fp, "x = %.2lf时多项式2的值为：%.2lf\n", val, polyValue(poly2, val));
577. **break**;
578. **default**:
579. fprintf(fp, "符号错误！\n");
580. }
581. }
582. fclose(fp);
583. }
585. **int** main()
586. {
587. LIST poly1 = makeNull();
588. LIST poly2 = makeNull();
589. **char** op;
590. **if** ((fp = fopen("F:\\data\_structure\\Lab\\lab1\\out.txt", "r")) == NULL)
591. {
592. printf("open failed\n"); //若打开文件失败，则退出程序
593. exit(1);
594. }
595. readPoly(poly1);
596. readPoly(poly2);
598. op = fgetc(fp);
600. figure(op, poly1, poly2);
602. deletePoly(poly1);
603. deletePoly(poly2);
604. **return** 0;
605. }