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：一元多项式计算器

实验日期：2020.10.20

班级：1903002

学号：1190200526

姓名：沈城有

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

**一、实验目的**

设计线性表的动态或者静态链式存储结构，并实现一个一元多项式的计算器。

**二、实验要求及实验环境**

实验要求：

以动态或者静态链表存储一元多项式，在此基础上按要求完成对一元多项式

的运算（为保证多项式的值的准确性，多项式的系数可以用分数表示，涉及到

两个分数相除时，结果也可以用分数表示）。

1.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。

2.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。

3.能够计算一元多项式的k阶导函数。

4.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。

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）。

实验环境：Windows 10 Visual Studio 2019

**三、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．物理设计（数据类型简要说明）

（1）动态链表存储多项式，结点包括数据域和指针域两部分，数据域存储分子、分母和指数（均为int类型），指针域指向下一个结点（若为末尾结点则指向NULL）。创建不存储信息的表头以便操作；

下图为链表结点直观示意图。



相关基本操作：创建新链表（含表头）、添加结点、删除结点等。

本次实验的主要要求均基于这些基本操作来实现。

（2）文件读写，保存原始数据和运算结果。

文件路径：FileDir1:"D:\\poly.txt" 读入文件路径

FileDir2:"D:\\output.txt" 输出文件路径

注：具体数据类型定义及实现请查看附录源代码开头部分。

2．逻辑设计

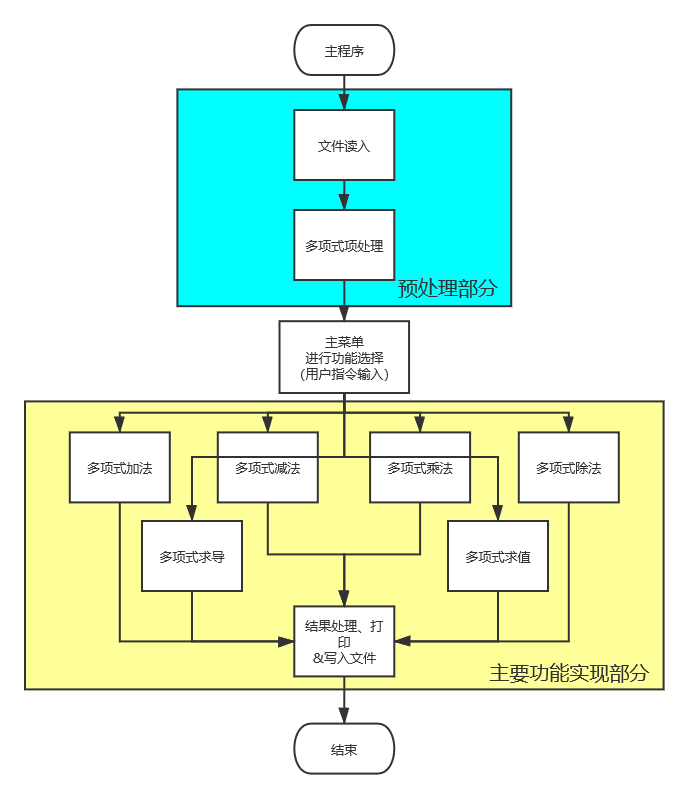
（1）主程序流程图

图1 主程序流程图

（2）各程序模块间调用关系：

a.多项式四则运算的四个子函数调用分数系数化简函数、链表逆置函数和多项式排列处理函数对运算完成的多项式进行处理；

b.多项式排列处理函数调用分数系数化简函数和元素交换函数实现按幂降序排列、合并同幂项、删除无效或非法项；

c.多项式除法函数调用多项式乘法和多项式减法函数实现；

d.多项式求导函数调用分数系数化简函数进行结果简化；

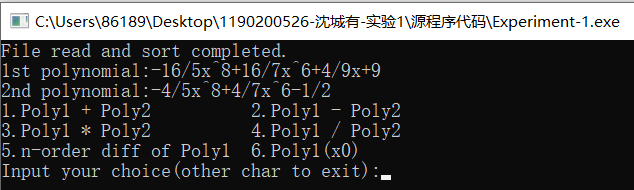
e.多项式排列处理函数调用多项式添加、删除项函数;

f.主函数调用模块化的各子函数执行预处理及运算、显示、保存功能。

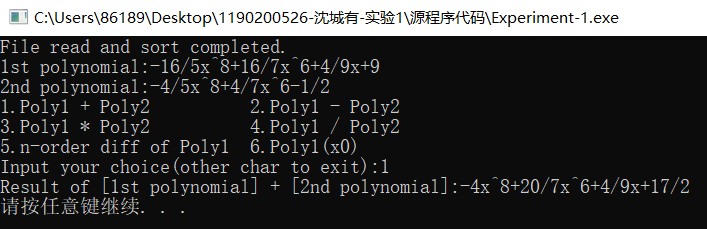
注：具体函数定义、实现及传入参数请查看附录源代码。

**四、测试结果**

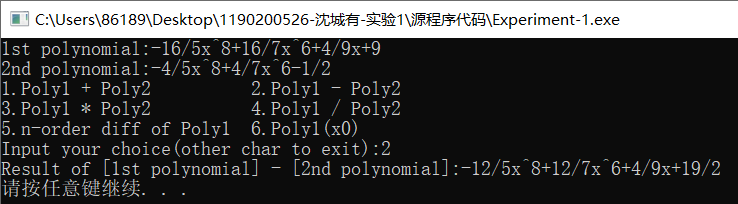
1、文件读入及屏幕显示



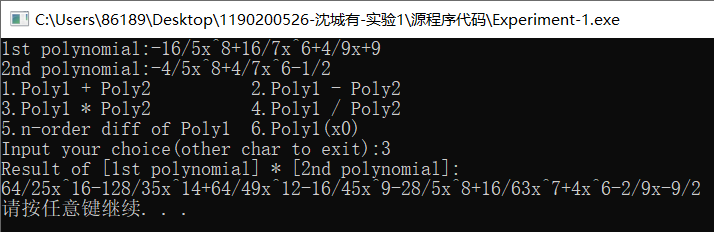
2、多项式加法



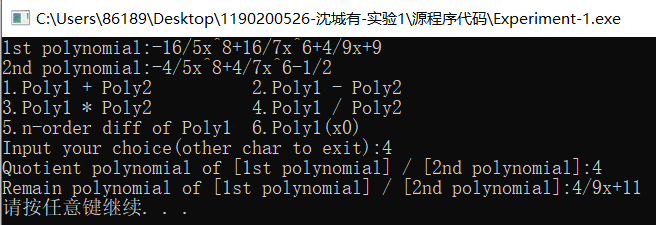
3、多项式减法



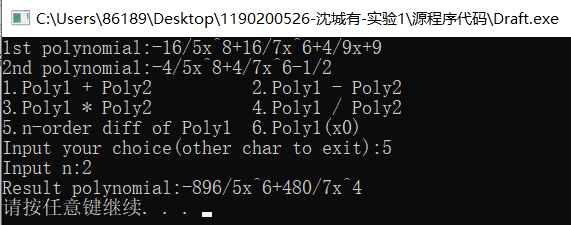
4、多项式乘法



5、多项式除法

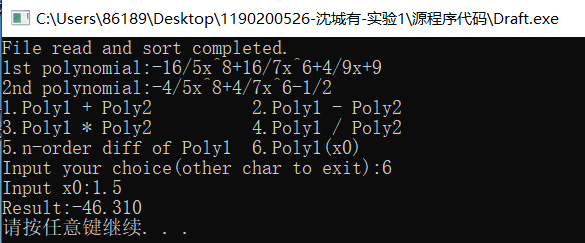


6、多项式求导

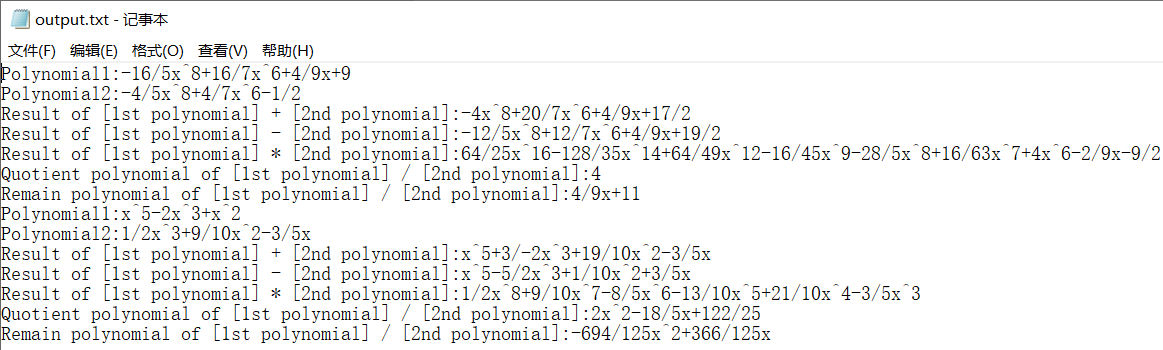


（此处是对第一个多项式求2阶导数）

7、多项式求值



8、文件输出展示

****

输出在D:\output.txt中，程序的多次运行输出的信息可连续保存，而不用清空文件。

**五、经验体会与不足**

经验体会：

（1）通过编写和调试熟练掌握了链式存储结构的实现及相关操作；

（2）在编写多项式运算算法的过程中更加深刻地理解了循环链表、可用空间表的思想，提高了算法的效率和空间利用率；

（3）复习了文件操作相关库函数的使用；

（4）练习了使用VS进行调试和变量监视的技巧。

不足：

（1）程序中存在几处操作可能导致内存泄漏；

（2）部分功能实现不够简洁，存在不必要的操作。

**六、附录：源代码（带注释）**

|  |
| --- |
| **程序的全部源代码：** |
| 1. /\* 2. \* 实验1 线性结构及应用 3. \* 实验项目：线性表的链式存储结构与应用 4. \* 实验题目：一元多项式计算器 5. \* 学号：1190200526 6. \* 姓名：沈城有 7. \*/ 8. #include <stdio.h> 9. #include <stdlib.h> 10. #include <malloc.h> 11. #include <math.h> 12. #define FileDir1 "D:\\poly.txt" //读入文件路径 13. #define FileDir2 "D:\\output.txt" //输出文件路径 14. struct LinkNode 15. { 16. int numer; //分子 17. int deno; //分母 18. int expo; //指数 19. LinkNode\* next; 20. }; 21. typedef LinkNode\* Poly; 22. Poly CreateNewPoly(void); //建立新多项式（包括表头） 23. void PolyReverse(Poly& L); //多项式链表逆置函数 24. void AddNode(Poly& L, int numer, int deno, int expo); //多项式添加项 25. void DeleteNode(Poly& L, Poly& p); //多项式删除项 26. int PolyFRead(Poly& L1, Poly& L2); //多项式文件读入函数 27. int PolyFWrite(Poly OUT, int n); //多项式文件输出函数 28. void Swap(int& a, int& b); //元素交换 29. void PolySort(Poly& L); //多项式指数降幂排列处理函数 30. void PolyPrint(Poly& L); //多项式显示函数 31. void PolyAdd(Poly L1, Poly L2, Poly& OUT); //多项式加法函数 32. void PolySub(Poly L1, Poly L2, Poly& OUT); //多项式减法函数 33. void PolyMuti(Poly L1, Poly L2, Poly& OUT); //多项式乘法函数 34. void PolyDiv(Poly L1, Poly L2, Poly& OUT, Poly& REM); //多项式除法函数 35. void PolyDiff(Poly& L); //多项式求导函数 36. float PolyValue(Poly L, float x); //多项式求值函数 37. void FracSimp(int& numer, int& deno); //分数系数化简函数 38. //建立新多项式（包括表头） 39. Poly CreateNewPoly(void) 40. { 41. Poly L; 42. L = (LinkNode\*)malloc(sizeof(LinkNode)); 43. if (L == NULL) 44. { 45. printf("Memory allocation failed!\n"); 46. return NULL; 47. } 48. L->next = NULL; 49. return L; 50. } 51. //多项式链表逆置函数 52. void PolyReverse(Poly& L) 53. { 54. Poly p, q; 55. p = L->next; 56. if (p) 57. { 58. q = p->next; 59. p->next = NULL; 60. while (q != NULL) 61. { 62. p = q; 63. q = q->next; 64. p->next = L->next; 65. L->next = p; 66. } 67. } 68. } 69. //多项式添加项 70. void AddNode(Poly& L, int numer, int deno, int expo) 71. { 72. Poly temp; 73. temp = L->next; 74. L->next = (LinkNode\*)malloc(sizeof(LinkNode)); 75. if (L->next == NULL) 76. { 77. printf("Memory allocation failed!\n"); 78. return; 79. } 80. L->next->numer = numer; 81. L->next->deno = deno; 82. L->next->expo = expo; 83. L->next->next = temp; 84. } 85. //多项式删除项 86. void DeleteNode(Poly& L, Poly& p) 87. { 88. Poly q; 89. if (p->next != NULL) 90. { 91. q = p->next; 92. p->next = q->next; 93. free(q); 94. } 95. } 96. //多项式文件读入函数 97. int PolyFRead(Poly& L1, Poly& L2) 98. { 99. FILE\* fp; 100. char ch; 101. int value[3] = { 0, 1, 0 }; 102. fp = fopen(FileDir1, "r"); 103. if (fp == NULL) 104. { 105. printf("File open failed!\n"); 106. return 0; 107. } 108. do { 109. fscanf(fp, "%d %d %d", &value[0], &value[1], &value[2]); 110. AddNode(L1, value[0], value[1], value[2]); 111. ch = fgetc(fp); 112. }while(ch != '#'); 113. do { 114. fscanf(fp, "%d %d %d", &value[0], &value[1], &value[2]); 115. AddNode(L2, value[0], value[1], value[2]); 116. ch = fgetc(fp); 117. } while (ch != '#'); 118. fclose(fp); 119. return 1; 120. } 121. //多项式文件输出主函数 122. int PolyFWrite(Poly OUT, int n) 123. { 124. FILE\* fp; 125. fp = fopen(FileDir2, "a"); 126. Poly p = OUT; 127. int test = 1; 128. if (fp == NULL) 129. { 130. printf("File write failure!\n"); 131. return 0; 132. } 133. switch (n) 134. { 135. case 1: 136. fprintf(fp, "Polynomial1:"); 137. break; 138. case 2: 139. fprintf(fp, "Polynomial2:"); 140. break; 141. case 3: 142. fprintf(fp, "Result of [1st polynomial] + [2nd polynomial]:"); 143. break; 144. case 4: 145. fprintf(fp, "Result of [1st polynomial] - [2nd polynomial]:"); 146. break; 147. case 5: 148. fprintf(fp, "Result of [1st polynomial] \* [2nd polynomial]:"); 149. break; 150. case 6: 151. fprintf(fp, "Quotient polynomial of [1st polynomial] / [2nd polynomial]:"); 152. break; 153. case 7: 154. fprintf(fp, "Remain polynomial of [1st polynomial] / [2nd polynomial]:"); 155. break; 156. default: 157. return 0; 158. } 159. while (p->next != NULL) 160. { 161. p = p->next; 162. if (test) 163. test = 0; 164. else if (p->numer > 0) 165. fprintf(fp, "+"); 166. if (p->numer != 1 || p->deno != 1) 167. fprintf(fp, "%d", p->numer); 168. if (p->deno != 1) 169. fprintf(fp, "/%d", p->deno); 170. if (p->expo != 0) 171. { 172. fprintf(fp, "x"); 173. if (p->expo != 1) 174. fprintf(fp, "^%d", p->expo); 175. } 176. } 177. fprintf(fp, "\n"); 178. fclose(fp); 179. return 1; 180. } 181. //元素交换 182. void Swap(int& a, int& b) 183. { 184. int temp; 185. temp = a; 186. a = b; 187. b = temp; 188. } 189. //多项式指数降幂排列处理函数 190. void PolySort(Poly& L) 191. { 192. Poly p = NULL; 193. Poly q = NULL; 194. Poly t = NULL; 195. if (L == NULL || L->next == NULL) 196. return; 197. for (p = L->next; p != NULL; p = p->next) 198. { 199. t = p; 200. for (q = p->next; q != NULL; q = q->next) 201. { 202. if (q->expo > t->expo) 203. t = q; 204. } 205. if (t != p) 206. { 207. Swap(t->expo, p->expo); 208. Swap(t->numer, p->numer); 209. Swap(t->deno, p->deno); 210. } 211. } 212. for (p = L->next; p != NULL; p = p->next) 213. { 214. FracSimp(p->numer, p->deno); 215. } 216. t = L; 217. for (p = L->next; p != NULL; ) 218. { 219. if (p->next != NULL) 220. { 221. if (p->expo == p->next->expo) 222. { 223. p->numer = p->numer \* p->next->deno + p->next->numer \* p->deno; 224. p->deno = p->deno \* p->next->deno; 225. FracSimp(p->numer, p->deno); 226. DeleteNode(L, p); 227. } 228. } 229. if (p->numer == 0) 230. DeleteNode(L, t); 231. else if (p->deno < 0) 232. { 233. p->deno = 0 - p->deno; 234. p->numer = 0 - p->numer; 235. } 236. t = t->next; 237. p = t->next; 238. } 239. } 240. //分数系数化简函数 241. void FracSimp(int& numer, int& deno) 242. { 243. int temp, r; 244. int n = numer; 245. int m = deno; 246. if (n < m) 247. { 248. temp = n; 249. n = m; 250. m = temp; 251. } 252. while (m != 0) 253. { 254. r = n % m; 255. n = m; 256. m = r; 257. } 258. numer /= n; 259. deno /= n; 260. } 261. //多项式显示函数 262. void PolyPrint(Poly& L) 263. { 264. Poly p = L; 265. int test = 1; 266. while (p->next != NULL) 267. { 268. p = p->next; 269. if (test) 270. test = 0; 271. else if(p->numer > 0) 272. printf("+"); 273. if (p->numer != 1 || p->deno != 1) 274. printf("%d", p->numer); 275. if (p->deno != 1) 276. printf("/%d", p->deno); 277. if (p->expo != 0) 278. { 279. printf("x"); 280. if (p->expo != 1) 281. printf("^%d", p->expo); 282. } 283. } 284. } 285. //多项式加法函数 286. void PolyAdd(Poly L1, Poly L2, Poly& OUT) 287. { 288. Poly p = L1->next; 289. Poly q = L2->next; 290. Poly temp; 291. temp = CreateNewPoly(); 292. int numer, deno; 293. while (p != NULL && q != NULL) 294. { 295. if (p->expo == q->expo) 296. { 297. numer = p->numer \* q->deno + q->numer \* p->deno; 298. deno = p->deno \* q->deno; 299. FracSimp(numer, deno); 300. AddNode(temp, numer, deno, p->expo); 301. p = p->next; 302. q = q->next; 303. } 304. else if (p->expo > q->expo) 305. { 306. AddNode(temp, p->numer, p->deno, p->expo); 307. p = p->next; 308. } 309. else if (p->expo < q->expo) 310. { 311. AddNode(temp, q->numer, q->deno, q->expo); 312. q = q->next; 313. } 314. } 315. while (p != NULL) 316. { 317. AddNode(temp, p->numer, p->deno, p->expo); 318. p = p->next; 319. } 320. while (q != NULL) 321. { 322. AddNode(temp, q->numer, q->deno, q->expo); 323. q = q->next; 324. } 325. OUT = temp; 326. PolySort(OUT); 327. } 328. //多项式减法函数 329. void PolySub(Poly L1, Poly L2, Poly& OUT) 330. { 331. Poly p = L1->next; 332. Poly q = L2->next; 333. Poly temp; 334. temp = CreateNewPoly(); 335. int numer, deno; 336. while (p != NULL && q != NULL) 337. { 338. if (p->expo == q->expo) 339. { 340. numer = p->numer \* q->deno - q->numer \* p->deno; 341. deno = p->deno \* q->deno; 342. FracSimp(numer, deno); 343. AddNode(temp, numer, deno, p->expo); 344. p = p->next; 345. q = q->next; 346. } 347. else if (p->expo > q->expo) 348. { 349. AddNode(temp, p->numer, p->deno, p->expo); 350. p = p->next; 351. } 352. else if (p->expo < q->expo) 353. { 354. AddNode(temp, 0 - q->numer, q->deno, q->expo); 355. q = q->next; 356. } 357. } 358. while (p != NULL) 359. { 360. AddNode(temp, p->numer, p->deno, p->expo); 361. p = p->next; 362. } 363. while (q != NULL) 364. { 365. AddNode(temp, 0 - q->numer, q->deno, q->expo); 366. q = q->next; 367. } 368. OUT = temp; 369. PolySort(OUT); 370. } 371. //多项式乘法函数 372. void PolyMuti(Poly L1, Poly L2, Poly& OUT) 373. { 374. Poly p, q; 375. Poly C1, C2; //C1是表头，C2用于暂时保存结点待链接至链表 376. Poly C3 = (Poly)malloc(sizeof(LinkNode)); //用于暂时保存链表 377. Poly temp; //用于链表链接 378. C3->next = NULL; 379. int expo, numer, deno; 380. int count = 0; 381. OUT = (Poly)malloc(sizeof(LinkNode)); 382. OUT->next = NULL; 383. p = L1->next; 384. while (p != NULL) 385. { 386. OUT = (Poly)malloc(sizeof(LinkNode)); 387. OUT->next = NULL; 388. C1 = (Poly)malloc(sizeof(LinkNode)); 389. C1->next = NULL; 390. temp = C1; 391. q = L2->next; 392. while (q != NULL) 393. { 394. C2 = (Poly)malloc(sizeof(LinkNode)); 395. expo = p->expo + q->expo; 396. numer = p->numer \* q->numer; 397. deno = p->deno \* q->deno; 398. FracSimp(numer, deno); 399. C2->expo = expo; 400. C2->numer = numer; 401. C2->deno = deno; 402. temp->next = C2; 403. temp = temp->next; 404. q = q->next; 405. } 406. p = p->next; 407. ++count; 408. temp->next = NULL; 409. PolyAdd(C1, C3, OUT); 410. C3 = OUT; 411. } 412. PolySort(OUT); 413. } 414. //多项式除法函数 415. void PolyDiv(Poly L1, Poly L2, Poly& OUT, Poly& REM) 416. { 417. Poly p = L1->next; 418. Poly q = L2->next; 419. Poly temp1, temp2; 420. OUT = CreateNewPoly(); 421. REM = CreateNewPoly(); 422. temp1 = CreateNewPoly(); 423. temp2 = CreateNewPoly(); 424. PolyAdd(L1, temp1, temp1); 425. while (p->expo >= q->expo) 426. { 427. temp2->next = (Poly)malloc(sizeof(LinkNode)); 428. temp2->next->numer = p->numer \* q->deno; 429. temp2->next->deno = p->deno \* q->numer; 430. FracSimp(temp2->next->numer, temp2->next->deno); 431. temp2->next->expo = p->expo - q->expo; 432. temp2->next->next = NULL; 433. AddNode(OUT, temp2->next->numer, temp2->next->deno, temp2->next->expo); 434. PolyMuti(L2, temp2, temp2); 435. PolySub(L1, temp2, L1); 436. p = L1->next; 437. temp2->next = NULL; 438. } 439. PolyMuti(L2, OUT, temp2); //调用乘法 440. PolySub(temp1, temp2, REM); //调用减法得余数 441. PolySort(OUT); 442. PolySort(REM); 443. } 444. //多项式求导函数 445. void PolyDiff(Poly& L) 446. { 447. Poly p, q, pt; 448. p = L; 449. q = L->next; 450. while(q != NULL) 451. { 452. if (q->expo == 0) 453. { 454. pt = q; 455. q = q->next; 456. p->next = q; 457. free(pt); 458. } 459. else 460. { 461. q->numer \*= q->expo; 462. FracSimp(q->numer, q->deno); 463. --q->expo; 464. p = q; 465. q = q->next; 466. } 467. } 468. PolySort(L); 469. } 470. //多项式求值函数 471. float PolyValue(Poly L, float x) 472. { 473. float result = 0; 474. Poly p = L; 475. while (p->next != NULL) 476. { 477. p = p->next; 478. result += ((float)p->numer / (float)p->deno) \* (float)(pow(x, p->expo)); 479. } 480. return result; 481. } 482. int main(void) 483. { 484. Poly L1, L2, OUT, REM; 485. char ch; 486. int n; 487. float value; 488. L1 = CreateNewPoly(); 489. L2 = CreateNewPoly(); 490. if (PolyFRead(L1, L2)) 491. { 492. PolySort(L1); 493. PolySort(L2); 494. printf("File read and sort completed.\n"); 495. PolyFWrite(L1, 1); 496. PolyFWrite(L2, 2); 497. do { 498. OUT = CreateNewPoly(); 499. printf("1st polynomial:"); 500. PolyPrint(L1); 501. printf("\n2nd polynomial:"); 502. PolyPrint(L2); 503. printf("\n"); 504. printf("1.Poly1 + Poly2 2.Poly1 - Poly2\n"); 505. printf("3.Poly1 \* Poly2 4.Poly1 / Poly2\n"); 506. printf("5.n-order diff of Poly1 6.Poly1(x0)\n"); 507. printf("Input your choice(other char to exit):"); 508. scanf("%c", &ch); 509. switch (ch) 510. { 511. case '1': { 512. PolyAdd(L1, L2, OUT); 513. printf("Result of [1st polynomial] + [2nd polynomial]:"); 514. PolyPrint(OUT); 515. printf("\n"); 516. PolyFWrite(OUT, 3); 517. } 518. break; 519. case '2': { 520. PolySub(L1, L2, OUT); 521. printf("Result of [1st polynomial] - [2nd polynomial]:"); 522. PolyPrint(OUT); 523. printf("\n"); 524. PolyFWrite(OUT, 4); 525. } 526. break; 527. case '3': { 528. PolyMuti(L1, L2, OUT); 529. printf("Result of [1st polynomial] \* [2nd polynomial]:\n"); 530. PolyPrint(OUT); 531. printf("\n"); 532. PolyFWrite(OUT, 5); 533. } 534. break; 535. case '4': { 536. PolyDiv(L1, L2, OUT, REM); 537. printf("Quotient polynomial of [1st polynomial] / [2nd polynomial]:"); 538. PolyPrint(OUT); 539. printf("\n"); 540. printf("Remain polynomial of [1st polynomial] / [2nd polynomial]:"); 541. PolyPrint(REM); 542. printf("\n"); 543. PolyFWrite(OUT, 6); 544. PolyFWrite(REM, 7); 545. } 546. break; 547. case '5': { 548. printf("Input n:"); 549. scanf("%d", &n); 550. if (n <= 0) //检测 551. printf("Illegal input(n > 0)!\n"); 552. else 553. { 554. for (int i = 1; i <= n; ++i) 555. PolyDiff(L1); 556. printf("Result polynomial:"); 557. PolyPrint(L1); 558. printf("\n"); 559. } 560. } 561. break; 562. case '6': { 563. printf("Input x0:"); 564. scanf("%f", &value); 565. value = PolyValue(L1, value); 566. printf("Result:%.3f\n", value); 567. } 568. break; 569. default:return 0; 570. } 571. free(OUT); 572. getchar(); 573. system("pause"); 574. system("cls"); 575. } while (1); 576. } 577. else 578. printf("File read failed or illegal input!\n"); 579. return 0; 580. } |