**《数据结构》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2019级计算机科学与技术卓越02班** | | | **姓名** | **李燕琴** |
| **实验题目** | **双向链表及其应用** | | | | | |
| **实验时间** | **2020年10月20日** | | **实验地点** | **DS1401** | | |
| **实验成绩** |  | | **实验性质** | **□验证性 □设计性 □综合性** | | |
| 教师评价：  □算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  其他：  评价教师签名： | | | | | | |
| 一、实验目的  1、理解双向链表的逻辑特征以及双向链表的优点。  2、理解头结点、尾结点以及设置头结点、尾结点的优点。   1. 熟练掌握带头结点、尾结点的双向链表的基本操作。 | | | | | | |
| 二、实验项目内容  1、编写使用freelist 带头、尾结点的双向链表类的定义，实现双向链表的基本操作。  2、利用双向链表实现2个一元多项式的加法和乘法运算，运算结果得到的链表要求按照指数降序排列的多项式。  **输入格式:**  3 2 //第一行，两个正整数分别表示多项式的项数  5 4 //输入第一个多项式各项的系数和指数，按指数降序输入  -3 2  1 0 //第一个多项式：5x4-3x2+1  6 2 //输入第二个多项式各项的系数和指数，按指数降序输入  -3 1 //第二个多项式：6x2-3x  **输出格式：**  4 //相加得到的多项式的项数  5 4 //每一项的系数与指数，按指数降序排列输出  3 2  -3 1  1 0 //和：5x4+3x2-3x+1  6 //相乘得到的多项式的项数  30 6  -15 5  -18 4  9 3  6 2  -3 1 //乘积：30x6-15x5-13x4+9x3+6x2-3x  3、最后提交完整的实验报告和源程序。 | | | | | | |
| 三、实验过程或算法（源程序）  **1、双链表的构建：**  按照书上的List ADT程序、freelist的Link结点程序、DLList的双链表程序，完整实现双链表这一数据结构的基本功能，如插入、增加、删除、跳至链首、跳至链尾、跳至指定位置、前驱、后继、长度、现位置读取、读值等。  **2、具体算法分析：**  **①sort()函数：**虽然在所给实验测试题目中，输入的A、B多项式，均是按照指数降序输入，但是在本程序中，设置了sort()函数，实现指数降序、系数去0、系数去重等功能，以增强其实用性，对于任意一个指数无序或系数重复的多项式，均可进行计算。因为sort()设计到具体的对象Nomial（链表结点对应的element对象）和具体的功能，故没有将之作为DLList的内置函数。源程序如下：   1. **void** sort(DLList<Nomial<**int**,**int**>>& A){ // 自测通过+去0功能+去重功能 2. A.moveToStart(); 3. Nomial<**int**, **int**>\* sign; 4. Nomial<**int**, **int**> temp; 5. **for** (**int** i = 0; i < A.length()-1; i++){ 6. A.moveToPos(i); 7. sign = & A.getValue(); 8. **if**(A.getValue().multi == 0){ 9. A.remove(); 10. } 11. **for** (**int** j = i + 1; j < A.length();j++){ 12. A.moveToPos(j);  //类似数组a[j]，实际所指的值是第j+1的数值 13. **if**(A.getValue().power > sign->power){ 14. temp = A.getValue(); 15. A.getValue() = \*sign; 16. sign->power = temp.power; 17. sign->multi = temp.multi; 18. } 19. **else** **if**(A.getValue().power == sign->power){ 20. sign->multi += A.getValue().multi; 21. A.getValue().multi = 0; 22. } 23. } 24. } 25. }   **②add()函数：**先将排好序的A多项式，拷贝到addres链表中，再依次遍历排好序的B多项式，与addres的结点指数值进行对比，并作相应处理，用以实现两个多项式的和。源程序如下：   1. **void** add(DLList<Nomial<**int**,**int**>>& addres, DLList<Nomial<**int**,**int**>>& A, DLList<Nomial<**int**,**int**>>& B){ 2. Nomial<**int**, **int**> temp; 3. A.moveToStart(); 4. **for** (**int** i = 0; i < A.length();){ 5. addres.append(A.getValue()); 6. A.next(); 7. i++; 8. } 9. addres.moveToStart(); 10. B.moveToStart(); 11. **for** (**int** i = 0; i < B.length();){ 12. **if**(B.getValue().power > addres.getValue().power){ 13. addres.insert(B.getValue()); // address.getValue指的就是后面一个结点的数值，直接insert插入在后面就行 14. B.next(); 15. i++; 16. addres.next(); // 回到判断时的address位置 17. } 18. **else** **if**(B.getValue().power == addres.getValue().power){ 19. addres.getValue().multi += B.getValue().multi; 20. B.next(); 21. i++; 22. } 23. **else** { 24. addres.next(); 25. } 26. } 27. }   **③mul()函数：**依次遍历A多项式的每一项，与B多项式中的每一项的乘积结果，附加到mulres链表中，两层循环，求得两个多项式的乘积，再对mulres链表进行排序、去重、去0操作即可。源程序如下：   1. **void** mul(DLList<Nomial<**int**,**int**>>& mulres, DLList<Nomial<**int**,**int**>>& A, DLList<Nomial<**int**,**int**>>& B){ 2. Nomial<**int**, **int**> temp; 3. A.moveToStart(); 5. **for** (**int** i = 0; i < A.length();){ 6. B.moveToStart(); 7. **for** (**int** j = 0; j < B.length();){ 8. temp.multi = A.getValue().multi \* B.getValue().multi; 9. temp.power = A.getValue().power + B.getValue().power; 10. mulres.append(temp); 11. B.next(); 12. j++; 13. } 14. A.next(); 15. i++; 16. } 17. }   **④print()函数：**按照要求，一层循环遍历输出即可，源程序如下：   1. **void** print(DLList<Nomial<**int**,**int**>>& A){ //自测通过 2. A.moveToStart(); 3. **for** (**int** i = 0; i < A.length();){ 4. cout << A.getValue().multi << " " << A.getValue().power << endl; 5. A.next(); 6. i++; 7. } 8. }   **3、文件源程序：**   1. #include<iostream> 2. **using** **namespace** std; 4. **template**<**typename** E>**class** List{ // List ADT 5. **private**: 6. **void** operator=(**const** List&){} 7. List(**const** List&){} 8. **public**: 9. List(){} 10. **virtual** ~List(){} 12. **virtual** **void** clear() = 0; //纯虚函数，抽象类，定义统一函数功能接口 13. **virtual** **void** insert(**const** E &item) = 0; 14. **virtual** **void** append(**const** E &item) = 0; 15. **virtual** E remove() = 0; //移除curr的结点，并返回结点值 16. **virtual** **void** moveToStart() = 0; //set the curr to the head(start of the list) 17. **virtual** **void** moveToEnd() = 0; //set the curr to the tail(end of the list) 18. **virtual** **void** prev() = 0; //move the current position one step left 19. **virtual** **void** next() = 0; //move the current position one step right 20. **virtual** **int** length() **const** = 0; 21. **virtual** **int** currPos() **const** = 0; // the index of the current position 22. **virtual** **void** moveToPos(**int** pos) = 0; 23. **virtual** E &getValue() **const** = 0; 24. }; 26. **template**<**typename** Multi, **typename** Power> **class** Nomial{ 27. **public**: 28. Multi multi; 29. Power power; 30. Nomial(){} // 无参构造 31. Nomial(Multi mm,Power pp){ // 参数构造 32. multi = mm; 33. power = pp; 34. } 35. Nomial(**const** Nomial& it){ // 拷贝函数构造 36. multi = it.multi; 37. power = it.power; 38. } 39. **void** operator =(**const** Nomial& it){ // 重载"="赋值运算符 40. multi = it.multi; 41. power = it.power; 42. } 43. }; 45. **template**<**typename** E> **class** Link{ // 结点类 46. **private**: 47. **static** Link<E> \*freelist; //一个结点模块 48. **public**: 49. E element; 50. Link \*prev; 51. Link \*next; 52. Link(**const** E& it,Link \*pr,Link \*ne){ // 有参构造函数 53. element = it; // 对应等号重载运算符 54. prev = pr; 55. next = ne; 56. } 57. Link(Link \*pr=NULL,Link \*ne=NULL){ // 无参&有参构造函数（初始化） 58. prev = pr; 59. next = ne; 60. } 61. **void**\* operator **new**(**size\_t**){ // new运算重载 62. **if**(freelist==NULL) 63. **return** ::**new** Link; // creat place 64. Link<E> \*temp = freelist; 65. freelist = freelist->next; 66. **return** temp; 67. } 68. **void** operator **delete**(**void** \*ptr){  //delete运算重载 69. ((Link<E> \*)ptr)->next = freelist; 70. freelist = (Link<E> \*)ptr; 71. } 72. }; 73. **template**<**typename** E> Link<E>\* Link<E>::freelist = NULL; // freelist静态变量初始化 75. **template**<**typename** E> **class** DLList: **public** List<E>{ 76. **private**: 77. Link<E> \*head; 78. Link<E> \*curr; // Behind the current element,is what we do with 79. Link<E> \*tail; 80. **int** len; 81. **void** init(){ 82. head = **new** Link<E>; 83. tail = **new** Link<E>; 84. head->next = tail; 85. tail->prev = head; 86. curr = head; 87. len = 0; 88. } 90. **void** removeall(){ 91. **while**(head!=NULL){ 92. curr = head; 93. head = head->next; 94. **delete** curr; // delete one by one from the strat of the list 95. } 96. } 98. **public**: 99. DLList() { init(); } // Constructors 100. ~DLList() { removeall(); } // Destructor 101. **void** clear() { removeall(); init(); } 102. **void** insert(**const** E& it) { // add it after the current point 103. curr->next = curr->next->prev = **new** Link<E>(it,curr,curr->next); 104. len++; 105. } 106. **void** append(**const** E& it){ // add it to the end of the list 107. tail->prev = tail->prev->next = **new** Link<E>(it,tail->prev,tail); 108. len++; 109. } 110. E remove(){ // remove the curr->next node 111. E it = curr->next->element; 112. Link<E> \*temp = curr->next; 113. curr->next->next->prev = curr; 114. curr->next = curr->next->next; 115. **delete** temp; 116. len--; 117. **return** it; 118. } 119. **void** moveToStart(){ 120. curr = head; 121. } 122. **void** moveToEnd(){ 123. curr = tail->prev; 124. } 125. **void** prev(){ 126. **if** (curr != head) 127. curr = curr->prev; 128. } 129. **void** next(){ 130. **if** (curr->next != tail) 131. curr = curr->next; 132. } 133. **int** length() **const**{ **return** len; } 134. **int** currPos()**const**{ 135. Link<E> \*temp = head; 136. **int** i; 137. **for** (i = 0; curr != temp;i++){ 138. temp = temp->next; 139. } 140. **return** i; 141. } 142. **void** moveToPos(**int** pos){ 143. **if**(pos<0 || pos>=len) { 144. cout << "Position out of range"; 145. **return**; 146. } 147. curr = head; 148. **for** (**int** i = 0; i < pos;i++) 149. curr = curr->next; 150. } 151. E& getValue() **const**{ 152. **if**(curr->next != tail) 153. **return** curr->next->element;  //这是给的curr后面的值！！！ 154. } 155. }; 157. **void** add(DLList<Nomial<**int**,**int**>>& addres, DLList<Nomial<**int**,**int**>>& A, DLList<Nomial<**int**,**int**>>& B){ 158. Nomial<**int**, **int**> temp; 159. A.moveToStart(); 160. **for** (**int** i = 0; i < A.length();){ 161. addres.append(A.getValue()); 162. A.next(); 163. i++; 164. } 165. addres.moveToStart(); 166. B.moveToStart(); 167. **for** (**int** i = 0; i < B.length();){ 168. **if**(B.getValue().power > addres.getValue().power){ 169. addres.insert(B.getValue()); // address.getValue指的就是后面一个结点的数值，直接insert插入在后面就行 170. B.next(); 171. i++; 172. addres.next(); // 回到判断时的address位置 173. } 174. **else** **if**(B.getValue().power == addres.getValue().power){ 175. addres.getValue().multi += B.getValue().multi; 176. B.next(); 177. i++; 178. } 179. **else** { 180. addres.next(); 181. } 182. } 183. } 184. **void** mul(DLList<Nomial<**int**,**int**>>& mulres, DLList<Nomial<**int**,**int**>>& A, DLList<Nomial<**int**,**int**>>& B){ 185. Nomial<**int**, **int**> temp; 186. A.moveToStart(); 188. **for** (**int** i = 0; i < A.length();){ 189. B.moveToStart(); 190. **for** (**int** j = 0; j < B.length();){ 191. temp.multi = A.getValue().multi \* B.getValue().multi; 192. temp.power = A.getValue().power + B.getValue().power; 193. mulres.append(temp); 194. B.next(); 195. j++; 196. } 197. A.next(); 198. i++; 199. } 200. } 202. **void** print(DLList<Nomial<**int**,**int**>>& A){ //自测通过 203. A.moveToStart(); 204. **for** (**int** i = 0; i < A.length();){ 205. cout << A.getValue().multi << " " << A.getValue().power << endl; 206. A.next(); 207. i++; 208. } 209. } 210. **void** sort(DLList<Nomial<**int**,**int**>>& A){ // 自测通过+去0功能+去重功能 211. A.moveToStart(); 212. Nomial<**int**, **int**>\* sign; 213. Nomial<**int**, **int**> temp; 214. **for** (**int** i = 0; i < A.length()-1; i++){ 215. A.moveToPos(i); 216. sign = & A.getValue(); 217. **if**(A.getValue().multi == 0){ 218. A.remove(); 219. } 220. **for** (**int** j = i + 1; j < A.length();j++){ 221. A.moveToPos(j);  //类似数组a[j]，实际上所指的数值是第j+1的数值 222. **if**(A.getValue().power > sign->power){ 223. temp = A.getValue(); 224. A.getValue() = \*sign; 225. sign->power = temp.power; 226. sign->multi = temp.multi; 227. } 228. **else** **if**(A.getValue().power == sign->power){ 229. sign->multi += A.getValue().multi; 230. A.getValue().multi = 0; 231. } 232. } 233. } 234. } 235. **int** main(){ 236. **int** Na, Nb; 237. cin >> Na >> Nb; 238. DLList<Nomial<**int**,**int**>> A; 239. DLList<Nomial<**int**,**int**>> B; 240. DLList<Nomial<**int**,**int**>> addres; 241. DLList<Nomial<**int**,**int**>> mulres; 242. Nomial<**int**,**int**> it; 243. **for** (**int** i = 0; i < Na;i++){ 244. **int** multi, power; 245. cin >> multi >> power; 246. it.multi=multi; 247. it.power=power; 248. A.append(it); 249. } 250. **for** (**int** i = 0; i < Nb;i++){ 251. **int** multi, power; 252. cin >> multi >> power; 253. it.multi=multi; 254. it.power=power; 255. B.append(it); 256. } 257. sort(A); 258. sort(B); 259. add(addres,A, B); 260. cout << addres.length() << endl; 261. print(addres); 263. mul(mulres, A, B); 264. sort(mulres); 265. cout << mulres.length() << endl; 266. print(mulres); 267. } | | | | | | |
| 四、实验结果及分析和（或）源程序调试过程  **1、实验结果及分析**  （1）输入测试样例，结果符合题意  （2）预期功能测试，结果符合题意（无序、有重复指数项的多项式）  **2、过程总结**  （1）通过本实验深入理解了链表中curr的具体含义，在写程序的过程中，将curr指向结点当作操作结点来进行读值处理，但总是出现和索引不一致的读取数据结果，导致Debug程序的时候花了很多时间。在书上的数据结构代码中，是将curr的后继结点当作读值操作点，虽然在双链表中，前者这样的规定也能轻松实现链表功能，但是后面这种，更容易理解，保证了数组、单链表、双链表的一致性，在处理A[i]时，实际操作的是第i+1个结点。  （2）注重代码每一模块的准确性，特别是很长的代码，可以每做好一个模块，写一个小的测试程序进行功能验证，避免之后查错麻烦费时。  （3）双链表的构建，源程序是抄写书上的写好的程序，自己独立写ADT的能力还需要加强练习。虽然之前对C++的接触比较少，在理解书上的ADT方面也很吃力，但是发现将书上的代码在软件上写一遍，理解起来就不会停留在抽象的概念上（如上述的curr），对于面向对象的设计，也有了更全面的理解，所以以后还是要多多实练。  （4）对于数据结构的设计上，也要多注意它的封装和多态，增强其实用性。对于其他函数的设计上，也需要逐步考虑多种情况的可能性，是其更全面完整。 | | | | | | |