**《数据结构与算法》实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验名称** | 一元多项式的加法和乘法 | | | | |
| **姓名** | 叶鹏 | **学号** | 20020007095 | **日期** | 2022/3/10 |
| **实验内容** | 1、通过键盘随机输入两个多项式P（x）和Q（x）的内容。  2、输出结果要有P（x）、Q（x）以及他们的和。  3、输入输出多项式的格式可自行定义。 | | | | |
| **实验目的** | 实现一元多项式的加法和乘法 | | | | |
| **实验步骤** | 1. 一元多项式的加法 2. 考虑到使用链表结构实现一元多项式，格式为 coef x^expn   X作为未知量保留，只需用户输入 coef 系数，与 expn 指数即可，  建立链表    picture 1节点    picture 2链表   1. 对于多项式的加法，我们考虑 Pa + Pb，将加和结果整理到Pa的空间，接着删除Pb，于是 Pa 的结果便是加法的和。 2. 加法的运算逻辑为，相同系数的项相加，如果 Pb 中存在 Pa 中未有的项，便按照大小顺序插入到 Pa 中，如果Pa 中的某一项与 Pb 中的某一项系数加和结果为 0 ，则在 Pa 中删除该项，依照这个思路，完成代码的编写     picture 3多项式加法算法   1. 一元多项式的乘法 2. 与加法类似，乘法中所用到的诸多函数在加法中已经实现 3. 与加法不同的是，乘法需要将 Pa 中的每一项与 Pb 中的每一项进行乘法运算，即系数相乘，指数相加，在此过程中会产生几个边界条件 4. 系数为0，输出时忽略此项 5. 系数为1，输出时忽略系数 6. 指数为0，输出时忽略指数 7. 指数为1，输出时忽略指数上标   为这些条件添加特殊判定，将相乘结果逐个相加生成一个新的多项式 Pc，将 Pc 作为函数结果返回    picture 4多项式乘法   1. 运行结果 2. 多项式加法 3. (x^2 + 2x^3) + (3x^4 + 4x^5 + 5x^6)      1. (x^2 + 2x^3) + (-x^2 - 2x^3 + 3x^4)      1. (x^2 + 2x^3) + (x^2 + 2x^3 + 3x^4)      1. 多项式乘法 2. (x^2 + 2x^3) \* (3x^4 + 4x^5 + 5x^6)      1. (x^2 + 2x^3) \* (x^-2 + 2x^-3 + 3x^4)      1. (3x^4 + 4x^5) \* (6x^-3 + 7x^-4 + 8x^10)      1. 源代码 2. #pragma once 3. #include <iostream> 4. **using** **namespace** std; 6. **struct** LinkList 7. { 8. **float** coef; 9. **int** expn; 10. LinkList\* next; 11. LinkList(**float** coef, **int** expn, LinkList\* next) { 12. **this**->coef = coef; 13. **this**->expn = expn; 14. **this**->next = next; 15. } 16. }; 18. **class** polynomial 19. { 20. **public**: 21. polynomial(); 22. ~polynomial(); 24. LinkList\* getDump() { **return** dumpNode; } 25. LinkList\* getHead(); 26. **void** setHead(LinkList\* node) { dumpNode->next = node; } 27. LinkList\* getTail(); 28. **int** getLength() { **return** getTail()->expn; } 30. **private**: 31. LinkList\* dumpNode; 32. }; 34. polynomial::polynomial() 35. { 36. dumpNode = **new** LinkList(0, 0, nullptr); 37. } 39. polynomial::~polynomial() 40. { 41. } 43. **inline** LinkList\* polynomial::getHead() 44. { 45. **if** (dumpNode->next == nullptr) { 46. **return** nullptr; 47. } 48. **return** dumpNode->next; 49. } 51. **inline** LinkList\* polynomial::getTail() 52. { 53. LinkList\* curNode = getDump(); 54. **while** (curNode->next != nullptr) { 55. curNode = curNode->next; 56. } 57. **return** curNode; 58. }   62. LinkList\* GetHead(polynomial& p) { 63. **return** p.getHead(); 64. } 66. LinkList\* LocateElem(polynomial& p, LinkList\* node) { 67. LinkList\* cur = p.getHead(); 68. **while** (cur != nullptr) { 69. **if** (node->expn == cur->expn) { 70. **return** cur; 71. } 72. cur = cur->next; 73. } 74. **return** nullptr; 75. } 77. //输入m项的系数和指数，建立表示一元多项式的有序链表P 78. **void** CreatPolyn(polynomial& p, **int** m) { 79. **float** coef; 80. **int** expn; 81. **while** (m) 82. { 83. cin >> coef >> expn; 84. LinkList\* newNode = **new** LinkList(coef, expn, nullptr); 85. LinkList\* point = LocateElem(p, newNode); 86. **if** (!point) { 87. **if** (p.getHead() == nullptr) 88. p.setHead(newNode); 89. **else** 90. p.getTail()->next = newNode; 91. } 92. m--; 93. } 94. } 96. //返回一元多项式P中的项数 97. **int** PolynLenght(polynomial p) { 98. **return** p.getLength(); 99. } 101. **void** DestoryHelper(LinkList\* curNode) { 102. **if** (curNode->next == nullptr) { 103. **delete** curNode; 104. **return**; 105. } 106. DestoryHelper(curNode->next); 107. **delete** curNode; 108. **return**; 109. } 111. //销毁一元多项式P 112. **void** DestoryPolyn(polynomial& p) { 113. LinkList\* curNode = p.getHead(); 114. **if** (curNode == nullptr) 115. **return**; 116. DestoryHelper(curNode); 117. **return**; 118. } 120. //打印输出一元多项式P 121. **void** PrintPolyn(polynomial p) { 122. cout << "P" /\*<< p.getLength()\*/ << "(x) = "; 123. **if** (p.getHead() == nullptr) { 124. cout << "多项式不存在！请先创建！" << endl; 125. **return**; 126. } 127. LinkList\* curNode = p.getHead(); 128. **while** (curNode != nullptr) { 129. **if** (curNode != p.getHead()) { 130. **if** (curNode->coef > 0) 131. cout << " + "; 132. **else** 133. cout << " - "; 134. } 136. **if** (fabs(curNode->coef) != 1 || curNode == p.getHead()) { 137. **if** (curNode == p.getHead()) { 138. **if** (curNode->coef == -1) cout << "-"; 139. **else** **if** (curNode->coef != 1) cout << curNode->coef; 140. } 141. **else** 142. cout << fabs(curNode->coef); 143. } 145. **if** (curNode->expn != 0) { 146. cout << "x"; 147. **if** (curNode->expn != 1) { 148. cout << "^" << curNode->expn; 149. } 150. } 152. curNode = curNode->next; 153. } 154. cout << endl; 155. } 157. **void** insertNode(polynomial& p, LinkList\* node) { 158. **if** (p.getDump()->next == nullptr) { 159. p.getDump()->next = node; 160. **return**; 161. } 163. LinkList\* curNode = p.getDump(); 164. **while** (curNode != nullptr) 165. { 166. **if** (curNode->next == nullptr && node->expn > curNode->expn) { 167. curNode->next = node; 168. **return**; 169. } 170. **if** (node->expn > curNode->expn && node->expn < curNode->next->expn) { 171. LinkList\* newNode = **new** LinkList(node->coef, node->expn, nullptr); 172. newNode->next = curNode->next; 173. curNode->next = newNode; 174. **return**; 175. } 176. curNode = curNode->next; 177. } 178. } 180. **bool** ListEmpty(polynomial p) { 181. **return** p.getHead() == nullptr; 182. } 184. **void** Append(polynomial& pa, polynomial& pb) { 185. LinkList\* curNode = pa.getDump(); 186. **while** (curNode->next != nullptr) { 187. curNode = curNode->next; 188. } 189. LinkList\* bNode = pb.getHead(); 190. **while** (bNode != nullptr) { 191. LinkList\* newNode = **new** LinkList(bNode->coef, bNode->expn, nullptr); 192. curNode->next = newNode; 193. curNode = curNode->next; 194. bNode = bNode->next; 195. } 196. } 198. //删除节点 199. **void** deleteNode(polynomial& p, LinkList\* node) { 200. LinkList\* curNode = p.getDump(); 201. **while** (curNode != nullptr) { 202. **if** (curNode->next == node) { 203. curNode->next = node->next; 204. **delete**(node); 205. **return**; 206. } 207. curNode = curNode->next; 208. } 209. cout << "未找到节点！" << endl; 210. **return**; 211. } 213. //完成多项式相加运算,即Pa=Pa+Pb,并销毁一元多项式Pb 214. **void** AddPolyn(polynomial& Pa, polynomial& Pb) { 216. LinkList\* nodeA = Pa.getHead(); 217. LinkList\* nodeB = Pb.getHead(); 219. cout << "多项式："; 220. PrintPolyn(Pa); 221. cout << "多项式："; 222. PrintPolyn(Pb); 224. **while** (nodeA && nodeB) { 225. **if** (nodeA->expn < nodeB->expn) { 226. nodeA = nodeA->next; 227. } 228. **else** **if** (nodeA->expn == nodeB->expn) { 229. **float** coef = nodeA->coef + nodeB->coef; 230. **if** (coef == 0.0) { 231. LinkList\* delNode = nodeA; 232. nodeA = nodeA->next; 233. deleteNode(Pa, delNode); 234. } 235. **else** { 236. nodeA->coef = coef; 237. nodeA = nodeA->next; 238. } 239. LinkList\* delNode = nodeB; 240. nodeB = nodeB->next; 241. deleteNode(Pb, delNode); 243. } 244. **else** **if** (nodeA->expn > nodeB->expn) { 245. insertNode(Pa, nodeB); 246. LinkList\* delNode = nodeB; 247. nodeB = nodeB->next; 248. deleteNode(Pb, delNode); 249. } 250. } 252. **if** (!ListEmpty(Pb)) 253. Append(Pa, Pb); 255. cout << "加和结果："; 256. PrintPolyn(Pa); 258. DestoryPolyn(Pb); 260. **return**; 261. } 263. polynomial\* MutiPolyn(polynomial& Pa, polynomial& Pb) { 264. polynomial\* ans = **new** polynomial(); 266. LinkList\* cur = Pa.getHead(); 268. **while** (cur) { 269. LinkList\* target = Pb.getHead(); 270. **while**(target){ 271. LinkList\* newNode = **new** LinkList(cur->coef \* target->coef, 272. cur->expn + target->expn, nullptr); 273. LinkList\* point = LocateElem(\*ans, newNode); 274. **if** (point) { 275. **if** (point->coef + newNode->coef == 0) { 276. deleteNode(\*ans, point); 277. } 278. **else** { 279. point->coef += newNode->coef; 280. } 281. } 282. **else** { 283. insertNode(\*ans, newNode); 284. } 285. target = target->next; 286. } 288. cur = cur->next; 289. } 291. **return** ans; 292. } | | | | |
| **实验总结** | 本次实验通过使用链表模拟多项式的加法与乘法，熟悉了链表这一数据结构，对链表的简单操作加以掌握，并尝试使用此数据结构来解决实际问题，使我们了解到链表的强大之处 | | | | |