山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202300130183 | 姓名： 宋浩宇 | | 班级：23级人工智能班 |
| 实验题目：2024数据结构--数据/智能 实验3 数组描述线性表 | | | |
| 实验学时：2 | | 实验日期： 2024/9/25 | |
| 实验目的：  1.掌握线性表结构、数组描述方法（顺序存储结构）、数组描述线性表的实现。  2.掌握线性表应用。 | | | |
| 软件开发工具：   1. visual studio code 2022（使用C/C++、C/C++ Extension Pack、C/C++ Themes插件） 2. mingw64工具包 | | | |
| 1. 实验内容   完成2024数据结构--数据/智能 实验3 数组描述线性表中的题目A 通讯录实验。   1. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）   我们首先创建一个结构体PersonalData来存储通讯录中结构化的数据。  我们由此来编写arraylist类。其中PersonalData的指针datas用于指向PersonalData的数组，count用于计量数组的元素个数。我们需要完成增删查改的功能，我们以此来描述实现的方式，其中增操作是创建一个count+1大小的PersonalData数组，将原本数组的数据拷贝过去，再将原本datas指向的内存空间释放掉，再指向新的空间，并将count自增。这样就可以完成数组形式的增操作。删操作的实现方式与增操作类似，我们创建一个count-1大小的数组，将原本数组中除了要删除的元素以外的元素都拷贝过来，再将原本datas指向的内存空间释放掉，再让datas指向新的空间，并将count自减，这样就可以完成删的操作。查操作比较简单，只需遍历datas指向的数组，如果其中的某个PersonalData的name元素与要查找的相同，则输出1，否则输出0。而改的操作也很简单，只需对datas指向的数组的对应元素进行重新赋值即可。但是存在这么一个问题，删操作和改操作是需要要被操作的元素的下标的，因此我们在查操作的基础上在进行修改，获得新的查找函数，该函数会返回需要查找的元素的下标。至此该类内部可以进行正常运行，增删查改操作也都可以进行。题目要求我们再写一个找出同一个班级的学生，实现起来也很简单只需要遍历数组即可。另外，因为实验要求我们自己来实现所有需要的算法，因此再自己手写一个strcmp函数和strcpy函数，因为题目说所有的数据都是合法的，因此不需要考虑这两个函数操作C风格字符串时的异常，实际实现也是很简单的。至此，本题结束。   1. 测试结果（测试输入，测试输出）   测试输入为：  28  0 Evan 57298577609 1 65  0 WINNIE 37367348390 4 1  3 Evan  4 6  3 WINNIE  1 Evan  4 7  1 WINNIE  3 MARYAM  3 CAMERON  3 TZIVIA  0 OMAR 16447001130 6 55  4 8  4 2  3 JADEN  3 ELIZABETH  2 OMAR 1 79409905568  3 JOSHUA  2 OMAR 1 8978214817  1 OMAR  3 Azaan  3 MARIA  0 HANNAH 94060479192 5 98  3 HEIDY  1 HANNAH  0 Axel 92066832927 3 70  1 Axel  3 TIFFANY  输出结果为：  1  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   从结果来看，我们这个题目是成功解决了的，但实际上因为我们本身在设计时，是以解决这个问题为导向的，并没有编写arraylist的类模板，实际上将增删查改更改为抽象操作，即不处理具体数据而是只进行声明的模板T的拷贝，以及引用的返回，再加上运算符==的重载，该代码可重构出arraylist类模板。  2024/9/25补充，经过修改，成功将代码中的arraylist类写成了一个可用的类模板，其中大部分运算与原本代码逻辑一致，主要区别在于，为了解决本题还需要书写一个运算符重载，以及将PersonalData结构体改写为类，并单独书写修改其中元素的函数（详见附录）   1. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释） 2. /\*2024数据结构--数据智能 实验3 数组描述线性表 A  通讯录.cpp\*/ 3. #include<iostream> 4. **using** **namespace** std; 6. **struct** PersonalData 7. { 8. **char** name[20]; 9. **char** telenum[20]; 10. **char** classnum[10]; 11. **int** house; 12. }; 13. **class** arraylist 14. { 15. **private**: 16. PersonalData\* datas = nullptr; 17. **int** count = 0; 18. **bool** equal(**const** **char**\* a, **const** **char**\* b) 19. { 20. **int** count = 0; 21. **while** (**true**) 22. { 23. **if** (a[count]=='\0' && b[count] == '\0') 24. { 25. **return** **true**; 26. } 27. **if** (a[count]!= b[count]) 28. { 29. **return** **false**; 30. } 31. count++; 32. } 33. }; 34. **void** cpy(**char**\* a, **const** **char**\* b) 35. { 36. **int** count = 0; 37. **while** (**true**) 38. { 39. a[count] = b[count]; 40. **if** (b[count] =='\0') 41. { 42. **return**; 43. } 44. count++; 45. } 46. } 47. **public**: 48. **void** insert(); 49. **void** disp(); 50. **void** del(**int** index); 51. **bool** check(**const** **char**\* name); 52. **void** edit(**int** index, **int** num); 53. **int** find(**const** **char**\* name); 54. **void** operation(); 55. **void** XOR(**const** **char**\* num); 56. }; 58. **class** Solution 59. { 60. **public**: 61. **void** solute() 62. { 63. arraylist line; 64. **int** n; 65. cin >> n; 66. **for** (**int** i = 0; i < n; i++) 67. { 68. line.operation(); 69. } 70. } 71. }; 73. **void** arraylist::insert() 74. { 75. PersonalData newdata; 76. cin >> newdata.name >> newdata.telenum >> newdata.classnum >> newdata.house; 77. PersonalData\* newlist = **new** PersonalData[count + 1]; 78. **for** (**int** i = 0;i < count;i++) 79. { 80. newlist[i] = datas[i]; 81. } 82. count++; 83. newlist[count-1] = newdata; 84. **delete** [] datas; 85. datas = newlist; 86. } 88. **void** arraylist::disp() 89. { 90. **for** (**int** i = 0; i < count; i++) 91. { 92. cout << datas[i].name << " " << datas[i].telenum << " " << datas[i].classnum << " " << datas[i].house << endl; 93. } 94. } 96. **void** arraylist::del(**int** index) 97. { 98. PersonalData\* newlist = **new** PersonalData[count - 1]; 99. **for** (**int** i = 0; i < count; i++) 100. { 101. **if** (i<index) 102. { 103. newlist[i] = datas[i]; 104. } 105. **if** (i == index) 106. { 107. **continue**; 108. } 109. **if** (i>index) 110. { 111. newlist[i - 1] = datas[i]; 112. } 113. } 114. **delete**[] datas; 115. count--; 116. datas = newlist; 117. } 119. **bool** arraylist::check(**const** **char**\* name) 120. { 121. **for** (**int** i = 0; i < count; i++) 122. { 123. **if** (equal(datas[i].name , name)) 124. { 125. cout << 1 << endl; 126. **return** **true**; 127. } 128. } 129. cout << 0 << endl; 130. **return** **false**; 131. } 133. **void** arraylist::edit(**int** index,**int** num) 134. { 135. **if** (num == 1) 136. { 137. **char** newdata[20]; 138. cin >> newdata; 139. cpy(datas[index].telenum, newdata); 140. } 141. **if** (num == 2) 142. { 143. **char** newdata[10]; 144. cin >> newdata; 145. cpy(datas[index].classnum, newdata); 146. } 147. **if** (num == 3) 148. { 149. **int** newdata; 150. cin >> newdata; 151. datas[index].house = newdata; 152. } 153. } 155. **int** arraylist::find(**const** **char**\* name) 156. { 157. **for** (**int** i = 0; i < count; i++) 158. { 159. **if** (equal(datas[i].name,name)) 160. { 161. **return** i; 162. } 163. } 165. } 167. **void** arraylist::XOR(**const** **char**\* num) 168. { 169. **int** ans = 0; 170. // bool mark = 1; 171. **for** (**int** i = 0; i < count; i++) 172. { 173. **if** (equal(datas[i].classnum,num)) 174. { 175. ans = ans ^ datas[i].house; 176. } 177. } 178. cout << ans << endl; 179. } 180. **void** arraylist::operation() 181. { 182. **int** mark; 183. cin >> mark; 184. **if** (mark==0) 185. { 186. insert(); 187. } 189. **if** (mark == 1) 190. { 191. **char** name[20]; 192. cin >> name; 193. del(find(name)); 194. } 195. **if** (mark==2) 196. { 197. **char** name[20]; 198. cin >> name; 199. **int** a; 200. cin >> a; 201. edit(find(name), a); 202. } 203. **if** (mark==3) 204. { 205. **char** name[20]; 206. cin >> name; 207. check(name); 208. } 209. **if** (mark==4) 210. { 211. **char** classnum[10]; 212. cin >> classnum; 213. XOR(classnum); 214. } 215. } 217. **int** main() 218. { 219. Solution ans; 220. ans.solute(); 221. } 222. /\*2024数据结构--数据智能 实验3 数组描述线性表 A  通讯录.cpp\*/ 223. /\*类模板版本\*/ 224. #include<iostream> 225. **using** **namespace** std;   229. **class** PersonalData 230. { 231. **private**: 232. **bool** strequal(**const** **char**\* a, **const** **char**\* b) **const** 233. { 234. **int** count = 0; 235. **while** (**true**) 236. { 237. **if** (a[count]=='\0' && b[count] == '\0') 238. { 239. **return** **true**; 240. } 241. **if** (a[count]!= b[count]) 242. { 243. **return** **false**; 244. } 245. count++; 246. } 247. }; 248. **void** strcpy(**char**\* a, **const** **char**\* b) 249. { 250. **int** count = 0; 251. **while** (**true**) 252. { 253. a[count] = b[count]; 254. **if** (b[count] =='\0') 255. { 256. **return**; 257. } 258. count++; 259. } 260. }   264. **public**: 266. **char** name[20]; 267. **char** telenum[20]; 268. **char** classnum[10]; 269. **int** house; 270. **bool** operator==(**const** PersonalData& compared) **const**; 271. PersonalData(**const** PersonalData& copyed); 272. PersonalData(**int** mark); 273. PersonalData() {}; 274. PersonalData(**const** **char**\* name) { strcpy(**this**->name, name);strcpy(**this**->classnum, "\0"); strcpy(**this**->telenum, "\0"); **this**->house = 0; }; 275. **void** edit(); 276. **int** getHouse() { **return** **this**->house; }; 277. **bool** checkClassnum(**const** **char**\* target) { **return** **this**->strequal(**this**->classnum, target); }; 278. **void** disp() { cout << name << endl; } 279. }; 281. **bool** PersonalData::operator==(**const** PersonalData& compared) **const** 282. { 283. **return** **this**->strequal(**this**->name, compared.name); 284. } 286. PersonalData::PersonalData(**const** PersonalData& copyed) 287. { 288. **this**->strcpy(**this**->name, copyed.name); 289. **this**->strcpy(**this**->classnum, copyed.classnum); 290. **this**->strcpy(**this**->telenum, copyed.telenum); 291. **this**->house = copyed.house; 292. } 294. PersonalData::PersonalData(**int** mark) 295. { 296. cin >> **this**->name; 297. cin >> **this**->telenum; 298. cin >> **this**->classnum; 299. cin >> **this**->house; 300. } 302. **void** PersonalData::edit() 303. { 304. **int** operation; 305. cin >> operation; 306. **if** (operation == 1) 307. { 308. **char** tele[20]; 309. cin >> tele; 310. **this**->strcpy(**this**->telenum, tele); 311. } 312. **if** (operation == 2) 313. { 314. **char** clas[10]; 315. cin >> clas; 316. **this**->strcpy(**this**->classnum, clas); 317. } 318. **if** (operation == 3) 319. { 320. **int** new\_house; 321. cin >> new\_house; 322. **this**->house = new\_house; 323. } 324. } 326. **template**<**class** T> 327. **class** arraylist 328. { 329. **private**: 330. T\* datas; 331. **int** count; 332. **void** copy(T\* copyed, T\* to); 334. **public**: 335. T& operator[](**const** unsigned **int** subscript) { **return** datas[subscript]; }; 336. **int** find\_first(**const** T& target); 337. **void** push\_back(**const** T& element); 338. **void** erase(**const** T& target); 339. **bool** is\_in(**const** T& target); 340. **int** size() { **return** count; }; 341. arraylist(); 342. }; 344. **template**<**class** T> 345. **void** arraylist<T>::copy(T\* copyed, T\* to) 346. { 347. **for** (**size\_t** i = 0; i < count; i++) 348. { 349. to[i] = copyed[i]; 350. } 351. } 353. **template**<**class** T> 354. **void** arraylist<T>::push\_back(**const** T& element) 355. { 356. T\* new\_datas = **new** T[count + 1]; 357. copy(datas, new\_datas); 358. new\_datas[count] = element; 359. count++; 360. **delete**[] datas; 361. datas = new\_datas; 362. } 364. **template**<**class** T> 365. **int** arraylist<T>::find\_first(**const** T& target) 366. { 367. **for** (**size\_t** i = 0; i < count; i++) 368. { 369. **if** (target == datas[i]) 370. { 371. **return** i; 372. } 373. } 374. **return** -1; 375. } 377. **template**<**class** T> 378. **void** arraylist<T>::erase(**const** T& target) 379. { 380. **int** index = find\_first(target); 382. T\* new\_datas = **new** T[count - 1]; 384. **for** (**size\_t** i = 0; i < count; i++) 385. { 386. **if** (i < index) 387. { 388. new\_datas[i] = datas[i]; 389. } 390. **if** (i == index) 391. { 392. **continue**; 393. } 394. **if** (i > index) 395. { 396. new\_datas[i - 1] = datas[i]; 397. } 398. } 400. **delete**[] datas; 402. count--; 404. datas = new\_datas; 405. } 407. **template**<**class** T> 408. **bool** arraylist<T>::is\_in(**const** T& target) 409. { 410. **int** index = **this**->find\_first(target); 412. **if** (index != -1) 413. { 414. **return** **true**; 415. } 416. **if** (index == -1) 417. { 418. **return** **false**; 419. } 420. **return** **false**; 421. } 423. **template**<**class** T> 424. arraylist<T>::arraylist() 425. { 426. datas = nullptr; 427. count = 0; 428. } 430. **class** Solution 431. { 432. **public**: 433. **void** solute(); 434. **void** test(); 435. }; 437. **void** Solution::solute() 438. { 439. **int** n; 440. cin >> n; 441. **int** operation; 442. arraylist<PersonalData> data; 444. **for** (**size\_t** i = 0; i < n; i++) 445. { 446. cin >> operation; 447. **if** (operation == 0) 448. { 449. PersonalData newdata(1); 450. data.push\_back(newdata); 451. } 453. **if** (operation == 1) 454. { 455. **char** name[20]; 456. cin >> name; 457. data.erase(name); 458. } 459. **if** (operation == 2) 460. { 461. **char** name[20]; 462. cin >> name; 463. data[data.find\_first(name)].edit(); 464. } 465. **if** (operation == 3) 466. { 467. **char** name[20]; 468. cin >> name; 469. cout << data.is\_in(name); 470. } 471. **if** (operation == 4) 472. { 473. **int** ans = 0; 474. **char** target[10]; 475. cin >> target; 476. **for** (**size\_t** i = 0; i < data.size(); i++) 477. { 478. **if** (data[i].checkClassnum(target)) 479. { 480. ans = ans ^ data[i].getHouse(); 481. } 482. } 483. cout << ans; 484. } 485. } 487. } 489. **void** Solution::test() 490. { 491. arraylist<**int**> list; 492. list.push\_back(1); 493. list.push\_back(1); 494. list.push\_back(1); 495. **for** (**int** i = 0;i < list.size();i++) 496. { 497. cout << list[i]; 498. } 499. list.erase(1); 500. **for** (**int** i = 0;i < list.size();i++) 501. { 502. cout << list[i]; 503. } 504. }   508. **int** main() 509. { 510. Solution ans; 511. ans.solute(); 512. // ans.test(); 513. // system("pause"); 514. } | | | |