

**Lab report**

|  |  |
| --- | --- |
| **Course**: | Class Libraries and Data Structures |
| **Semester**: | 1st semester of the academic year **2020-2021** |
| **Major**: | Software Engineering |
| **Class**: | 2019-2 |
| **Student Name**: | Fu Ruoxuan |
| **Student ID:** | 222019321062060 |
| **Teacher:** | ZHAO, Hengjun (赵恒军) |

**School of Computer and Information Science**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | | Introduction to C++ Containers | | | |
| Date | | Oct 16，2020 | Type | | √ Confirmatory  √ Design  □Comprehensive |
| 1. **Objective & Requirements**    1. Learn operator overloading in C++    2. Understand dynamic memory allocation in C++; grasp the use of new and delete for memory allocation and reclaim; Grasp the use of pointers and arrays    3. Understand the concept of containers | | | | | |
| 1. **Experimental environment (**platform and software**)**   Windows 7 (or higher versions) + Visual Studio 2010 (or higher versions) | | | | | |
| 1. **Experimental content and design** (Main Content, Procedure, Codes and Results) 2. Task 1    1. Overloading the operator ‘>’to rewrite the compare() method in the Employee class you defined in the last lab;    2. Use your overloaded ‘>’to rewrite the findBestPaid method you defined in the last lab; 3. Task 2    1. In the employee container class sent you, implement a method remove(i) that can remove the item of index **i** in the container (Note that the first item is of index 0)    2. Based on remove(i), implement a method removeEmployee() in the company class that allows the user to input an index i and then remove the employee of index i.(Note that the first employee is of index 0)    3. Test your implementation in the main() function 4. compCont.h 5. #ifndef COMPCONT\_H 6. #define COMPCONT\_H 8. #include "employee.h" 9. #include "empCont.h" 11. **class** CompCont 12. { 13. **private**: 14. Employee bestPaid; 15. EmpCont container; 17. **public**: 18. **void** inputEmployee(); //when input employee from keyboard, store the input data in the container for later use; 19. **void** findBestPaid(); //go through the container to find the best paid employee 20. **void** printBestPaid() **const**; 22. //implement the following methods 23. **void** removeEmployee(); 24. }; 26. #endif 27. empCont.h 28. #ifndef EMPCONT\_H 29. #define EMPCONT\_H 30. #include "employee.h" 32. // defines a container that can hold any number of employees 33. **class** EmpCont 34. { 35. **private**: 36. **int** capacity; 37. **int** size; 38. Employee\* empArray; //a pointer, points to the memory space dynamically allocated for the container 40. **void** Extend(); //extend the capacity when the container is full; called after isFull test returns true 41. **bool** isFull() **const**; //check whether the container is full; called in the Add method 43. **public**: 45. EmpCont(); 47. **int** getLength() **const**; //returns the current number of items stored in the container 49. **void** Add(**const** Employee& newEmp); //add an item at the end of the container 51. Employee& Access(**int** i) **const**; //access the ith item in the container 53. ~EmpCont(); //destructor 55. //implement the following methods 56. **void** remove(**int** i); 57. }; 59. #endif 60. employee.h 61. #ifndef EMPLOYEE 62. #define EMPLOYEE 64. #include <string> // declares string class 66. **using** **namespace** std; 68. **class** Employee 69. { 70. **private**: 71. string name; 72. **double** grossPay; 74. **public**: 76. // Postcondition: this employee's name has been set to "" 77. //                and gross pay to 0.00. 78. Employee();  81. // Postcondition: The name and gross pay of this Employee have 82. //                been read in. 83. **void** input();  86. // Postcondition: this Employee's name and gross pay have been 87. //                written out. 88. **void** output() **const**;  91. // Postcondition: this Employee contains a copy of otherEmployee. 92. //void copy(const Employee& otherEmployee); 93. **void** operator=(**const** Employee& otherEmployee);  96. // Postcondition: true has been returned if this Employee's gross 97. //                pay is greater than that of otherEmployee. 98. //                Otherwise, false has been returned. 99. **bool** operator>(**const** Employee& otherEmployee) **const**; 101. string getName() **const**; 103. }; // Employee 105. #endif 106. compCont.cpp 107. #include "compCont.h" 108. #include <iostream> 109. **using** **namespace** std; 111. **void** CompCont::inputEmployee() 112. { 113. Employee employee; 115. employee.input(); 117. **while** (employee.getName() != "\*") 118. { 119. container.Add(employee); // add to the container 120. employee.input(); 121. } 122. } 124. **void** CompCont::findBestPaid() 125. { 126. **if** (container.getLength() == 0) 127. cout << "No employee exists!" << endl; 129. **else** 130. { 131. bestPaid = container.Access(0); 132. **for** (**int** i = 1; i < container.getLength(); i++) 133. { 134. **if** (container.Access(i) > bestPaid) // overloading > 135. bestPaid = container.Access(i); //overloading = 136. } 137. } 139. } 141. **void** CompCont::printBestPaid() **const** 142. { 143. cout << "The best-paid employee (and gross pay) " << endl; 144. bestPaid.output(); 145. } // printBestPaid 147. //implement the following method 148. **void** CompCont::removeEmployee() 149. { 150. **int** index = 0; 151. cout << "Please input the index[starting at 0] of the employee that you want to delete: "; 152. cin >> index; 153. container.remove(index); 154. } 155. empCont.cpp 156. #include <iostream> 157. #include "empCont.h" 158. **using** **namespace** std; 160. EmpCont::EmpCont() 161. { 162. size = 0; 163. capacity = 5; //initial capacity is 5 164. empArray = **new** Employee[capacity]; //allocate memory space 165. } 167. **void** EmpCont::Extend() 168. { 169. capacity = capacity \* 2; 170. Employee\* temp = **new** Employee[capacity]; 172. **for** (**int** i = 0; i < size; i++) 173. temp[i] = empArray[i]; 175. **delete**[] empArray; //memory reclaim 177. empArray = temp; 178. } 180. **bool** EmpCont::isFull() **const** 181. { 182. **return** (size == capacity); 183. }  186. **int** EmpCont::getLength() **const** 187. { 188. **return** size; 189. } 191. **void** EmpCont::Add(**const** Employee& newEmp) 192. { 193. **if** (!isFull()) 194. { 195. empArray[size] = newEmp; 196. size++; 197. } 198. **else** 199. { 200. Extend(); 201. Add(newEmp); //recursive 202. } 203. }  206. Employee& EmpCont::Access(**int** i) **const** 207. { 208. **if** (i < 0 || i >= size) 209. { 210. cout << "Out-of-bound access!" << endl; 211. exit(-1); 212. } 213. **else** 214. **return** empArray[i]; 215. } 217. EmpCont::~EmpCont() 218. { 219. **delete**[] empArray; //memory reclaim 220. }  223. //implement the following method 224. **void** EmpCont::remove(**int** index) 225. { 226. Employee\* temp = **new** Employee[capacity]; 227. **for** (**int** i =0; i < index; i++) 228. { 229. \*(temp+i) = **this**->empArray[i]; 230. } 231. **for** (**int** i = index; i < **this**->capacity; i++) 232. { 233. temp[i] = **this**->empArray[i + 1]; 234. } 235. **this**->capacity--; 236. **delete**[] empArray; 237. empArray = temp; 238. } 239. Employee.cpp 240. #include <iostream> 241. #include "employee.h"  244. Employee::Employee() 245. { 246. name = ""; 247. grossPay = 0; 248. } // default constructor  251. **void** Employee::input() 252. { 253. cout << "Please enter a name and gross pay; to quit, enter \* followed by any number: "; 254. cin >> name >> grossPay; 255. } // input  258. **void** Employee::output() **const** 259. { 260. cout << "is: " << name << ",  $" << grossPay << endl; 261. } // output 263. **void** Employee::operator=(**const** Employee &otherEmployee) 264. { 265. name = otherEmployee.name; 266. grossPay = otherEmployee.grossPay; 267. } 269. **bool** Employee::operator>(**const** Employee &otherEmployee) **const** 270. { 271. **return** grossPay > otherEmployee.grossPay; 272. } // compare 274. string Employee::getName() **const** 275. { 276. **return** name; 277. } 278. main.cpp 279. #include <iostream> 280. #include "compCont.h" 282. **using** **namespace** std; 284. **int** main() 285. { 286. CompCont cmp; 287. cmp.inputEmployee(); 288. cmp.findBestPaid(); 289. cmp.printBestPaid(); 291. cmp.removeEmployee(); 292. cmp.removeEmployee(); 293. cmp.removeEmployee(); 294. cmp.findBestPaid(); 295. cmp.printBestPaid(); 296. } 297. Result | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results and summing up the harvest and the existing problems） 2. Result analysis   For the task 1,I should use the overload of the operator > which can simply compare two person’s gross pay. I changes all the compare() to “>”making the code more readable and succinct.  For the task 2, I add a new method in compCont.cpp named remove Employee() give the note that user can delete elements in the array by a new method in empCont.cpp called remove(int index). “remove(int index)” achieves the function to delete a element in the array and reduce the capacity of the array.   1. Harvest   The container consists of the sequential container and the chain container. Unlike link list-the chain container，array which is the sequential container can not simply delete an element. Sequential container’s vice is it is hard to delete a element from the container. As a consequence, we should create a new pointer “Employee\* temp” pointing to a new array and copy elements which is before the element that a user want to delete. What is more, we should write“temp[i] = this ->empArray[i + 1];”, reduce free the memory space that empArray exists and let empArray point to the space of temp. However, arrays are easy to traverse since it do not need pointers.   1. Existing problems    1. Use the link container can operate the data easier. | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |