

**Lab report**

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

**School of Computer and Information Science**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | | Iterator | | | |
| Date | | November 6，2020 | Type | | √ Confirmatory  √ Design  □Comprehensive |
| 1. **Objective & Requirements**    1. Know the use of iterator and understand its implementation details    2. Can use iterator to traverse a list to finish a certain task | | | | | |
| 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)   Task 1  Using the source code sent to you about the linked list template with iterator, implement the method in the company class   * + - findBestPaid()   and test your implementation in the main function.  Task 2   * + - 1. Based on the source code of the container template using array storage, implement an iterator inner class for it       2. Based on the implemented iterator, implement the findBestPaid() method for the Company class, just as what you do in Task 1   Task 1 code   1. **void** Company::findBestPaid() 2. { 3. ListTemp<Employee>::Iterator itr; 4. **if** (container.isEmpty()) 5. cout << "No employee exists!" << endl; 6. **else** 7. { 8. bestPaid = \*(container.Begin()); 9. **for** (itr = container.Begin(); !(itr == container.End()); itr++) 10. { 11. **if** (\*itr > bestPaid) 12. { 13. bestPaid = \*itr; 14. } 15. } 16. } 17. }   Task 2 code   * 1. Inner iterator class  1. #ifndef CONTTEMP\_H 2. #define CONTTEMP\_H 4. #include<iostream> 5. //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*the following is the class delcaration\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*// 6. **template**<**class** T> 7. **class** ContTemp 8. { 9. **private**: 10. **int** capacity; 11. **int** size; 12. T\* elemArray; 14. **void** Extend(); 15. **bool** isFull() **const**; 17. **public**: 19. ContTemp(); 20. ~ContTemp(); //destructor 22. **int** getLength() **const**; 24. **void** Add(**const** T& newElem); 25. T& Access(**const** **int** i) **const**; 27. **class** Iterator 28. { 29. **friend** **class** ContTemp<T>; 30. **private**: 31. T\* curr; 32. Iterator(T\* ptr); 34. **public**: 35. Iterator(); 37. Iterator operator++(**int**); 39. Iterator operator++(); 41. **bool** operator ==(**const** Iterator other) **const**; 43. T& operator\*() **const**; 44. }; 45. Iterator Begin(**int** i) **const**; 47. Iterator End() **const**; 49. //please implement this! 50. **void** Remove(**int** index, ContTemp<T> container);  53. }; 54. //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*the following is the iterator inner class implementation\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*// 55. **template**<**class** T> 56. ContTemp<T>::Iterator::Iterator() 57. { 58. curr = nullptr; 59. }  62. **template**<**class** T> 63. ContTemp<T>::Iterator::Iterator(T\* ptr) 64. { 65. curr = ptr; 66. }  69. **template**<**class** T> 70. **typename** ContTemp<T>::Iterator ContTemp<T>::Iterator::operator++(**int**) 71. { 72. Iterator temp = \***this**; 73. curr++; 74. **return** temp; 75. } 77. **template**<**class** T> 78. **typename** ContTemp<T>::Iterator ContTemp<T>::Iterator::operator++() 79. { 80. curr++; 81. **return** \***this**; 82. }  85. **template**<**class** T> 86. **bool** ContTemp<T>::Iterator::operator==(**const** Iterator other) **const** 87. { 88. **return** curr == other.curr; 89. } 90. **template**<**class** T> 91. T& ContTemp<T>::Iterator::operator\*() **const** 92. { 93. **return** \*(curr); 94. } 95. //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*the following is the method implementation\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*// 97. **template**<**class** T> 98. **typename** ContTemp<T>::Iterator ContTemp<T>::Begin(**int** i) **const** 99. { 100. **return** Iterator(elemArray+i); //private of Iterator class 101. } 103. **template**<**class** T> 104. **typename** ContTemp<T>::Iterator ContTemp<T>::End() **const** 105. { 106. **return** Iterator(elemArray+size); 107. } 109. **template**<**class** T> 110. ContTemp<T>::ContTemp() 111. { 112. size = 0; 113. capacity = 5; 114. elemArray = **new** T[capacity]; 115. } 117. **template**<**class** T> 118. **int** ContTemp<T>::getLength() **const** 119. { 120. **return** size; 121. }  124. **template**<**class** T> 125. **void** ContTemp<T>::Add(**const** T& newElem) 126. { 127. **if** (!isFull()) 128. { 129. elemArray[size] = newElem; 130. size++; 131. } 132. **else** 133. { 134. Extend(); 135. Add(newElem); 136. } 137. } 139. **template**<**class** T> 140. **void** ContTemp<T>::Extend() 141. { 142. capacity = capacity \* 2; 143. T\* temp = **new** T[capacity]; 145. **for** (**int** i = 0; i < size; i++) 146. temp[i] = elemArray[i]; 148. **delete**[] elemArray; 150. elemArray = temp; 151. } 153. **template**<**class** T> 154. **bool** ContTemp<T>::isFull() **const** 155. { 156. **return** (size == capacity); 157. } 159. **template**<**class** T> 160. T& ContTemp<T>::Access(**const** **int** i) **const** 161. { 162. **if** (i < 0 || i >= size) 163. std::cout << "Out-of-bound access!" << std::endl; 164. **else** 165. **return** elemArray[i]; 166. } 168. **template**<**class** T> 169. **void** ContTemp<T>::Remove(**int** index,ContTemp<T> container) 170. { 171. T\* temp = **new** T[capacity]; 172. ContTemp<T>::Iterator itr1(temp); 173. ContTemp<T>::Iterator itr2 = container.Begin(0); 174. **for** ( ; !(itr2 == container.End());itr1++, itr2++) 175. { 176. **if**(!(itr2 == container.Begin(index))) 177. \*itr1 = \*itr2; 178. **else** 179. { 180. \*itr1 = \*(++itr2); 181. } 182. } 183. **this**->size--; 184. **delete**[] elemArray; 185. elemArray = temp; 186. }   190. **template**<**class** T> 191. ContTemp<T>::~ContTemp() 192. { 193. **delete**[] elemArray; //garbage collection/ 194. } 195. #endif      1. findBestPaid 196. **void** Company::findBestPaid() 197. { 198. **if** (container.getLength() == 0) 199. cout << "No employee exists!" << endl; 201. **else** 202. { 203. ContTemp<Employee>::Iterator itr = container.Begin(0); 204. bestPaid = \*itr; 205. **for** (; !(itr == container.End()); itr++) 206. { 207. **if** (\*itr > bestPaid) 208. bestPaid = \*itr; 209. } 210. } 212. }   The Task 1 result    The Task 2 result | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results and summing up the harvest and the existing problems）    1. Analysis       1. For the iterator of a link list, I should use operator overloading to overload ++, \* and == to access next node, get the data of this node and determine that whether two nodes are the same. In addition, I should record the first node as the beginning of this link list and the next position of the last node as the ending of this link list.       2. For the iterator of an array, it also should contain operator overloading to overload ++, \* and == to access next element, get the data of this element and determine that whether two elements are the same. What is more, there should define two overloading of ++ for the front self-increasing and rear self-increasing.    2. Summing up the harvest   Iterator: The iterator can traverse the interface on the container object without the designer having to concern himself with the implementation details of the memory allocation of the container object.  End(): Instead of returning a pointer to the last element, it returns a pointer to the position after the last element. Therefore, it is vital to realize its difference compared with the index.   * 1. The existing problems      1. The implement of function “remove ()”is not perfect yet. It still has some problems in the process of destructing. | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |