

**Lab report**

|  |  |
| --- | --- |
| **Course**: | Class Libraries and Data Structures |
| **Semester**: | 1st semester of the academic year **2020-2021** |
| **Major**: | Software Engineering |
| **Class**: | 2019.3 |
| **Student Name**: | 冯春霖 |
| **Student ID:** | 222019321062074 |
| **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. task 1   In employee.cpp:  bool Employee::operator > (const Employee& otherEmployee)  {  return this->grossPay > otherEmployee.grossPay;  }  In company.cpp:  while (employee.getName() != "\*")  {  if (employee > bestPaid)  bestPaid = employee;  employee.input();  } // while  2）task 2  In empCont.cpp:  void EmpCont::remove(int i)  {  for (int j = i; j < size - 1; j++)  empArray[j] = empArray[j + 1];  Employee temp;  empArray[size-1] = temp;  size--;  }  In compCont.cpp:  void CompCont::removeEmployee()  {  cout << "please input the number of employee you want to remove: ";  int order;  cin >> order;  container.remove(order);  }  In empCout.cpp:  void EmpCont::remove(int i)  {  for (int j = i; j < size - 1; j++)  {  empArray[j] = empArray[j + 1];  }  Employee temp;  empArray[size-1] = temp;  size--;  }  Screenshot of the results of the program: | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results and summing up the harvest and the existing problems）   The results of the experiment fully met the expectations and fulfilled all the requirements in the purpose of the experiment. Through this experiment, I have gained an understanding of the principles and applications of operator overloading and containers in c++. In the process of designing the *remove* function, the initial design could not remove the object at the end of the container, but it was improved to accomplish the purpose. There were no lingering problems with the experiment. | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |