

**Lab report**

|  |  |
| --- | --- |
| **Course**: | Class Libraries and Data Structures |
| **Semester**: | 1st semester of the academic year **2021-2022** |
| **Major**: | Software Engineering |
| **Class**: | 2020 |
| **Student Name**: | 温长锟 |
| **Student ID:** | 222020321062106 |
| **Teacher:** | ZHAO, Hengjun (赵恒军) |

**School of Computer and Information Science**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | | Containers, Templates, and Iterators | | | |
| Date | | September 30，2021 | Type | | √ Confirmatory  √ Design  □Comprehensive |
| 1. **Objective & Requirements**    1. Learn operator overloading in C++    2. Understand the concept of containers    3. Understand the concept of containers; can use template to define generic containers    4. Understand the difference between contiguous memory allocation and linked memory allocation; grasp the use of template to implement a container with linked storage    5. Know the use of iterator and understand its implementation details    6. Can use iterator to traverse a list to finish a certain task    7. Understand the concept of generic algorithms | | | | | |
| 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   1. For this task, you are provided with a container with linked storage. Based on the source codes, design a container with doubly linked list storage. For this you need to design the node structure and its fields for doubly linked list, design the fields and method interfaces of the container class, and design the fields and methods of the iterator inner class for the container. In particular, your design should support efficient insertion of elements both at the head and at the tail of the doubly linked list container. 2. Implement the container and its associated iterator inner class 3. Based on you implementation, implement the findBestPaidReverse() method for the Company class, which traverses the list of Employee from the tail to the head to find the employee with the highest salary. The findBestPaidReverse() method should be implemented using your designed and implemented iterators of the container. This means that your iterator should support the operator-- 4. You can refer to the guidance book (in Chinese) for more details of the lab requirements.   **Codes:**  **doubleList.h(双向链表头文件)**  #ifndef DOUBLELIST\_H  #define DOUBLELIST\_H  #define NULL 0  template<class T>  class DoubleList{  private:  struct Node{  T item;  Node\* next;  Node\* pre;  };  int size;  Node\* head;  public:  DoubleList() {  head = NULL;  size = 0;  }  int getLength() const {  return size;  }  bool isEmpty() const {  return size == 0;  }  class Iterator{  friend class DoubleList;  private:  Node\* cur;  Iterator(Node\* ptr){  cur = ptr;  }  public:  Iterator() {  cur = NULL;  }  Iterator operator++(int){  Iterator temp = \*this;  this->cur = cur->next;  return temp;  }  Iterator operator--(int){  Iterator temp = \*this;  this->cur = cur->pre;  return temp;  }  T& operator\*()const {  return cur->item;  }  bool operator==(const Iterator other) const{  return cur == other.cur;  }  };  void AddHead(const T& newone){  if (isEmpty()==false){  Node\* temp = new Node;  head->pre = temp;  temp->item = newone;  temp->next = head;  temp->pre = NULL;  head = temp;  size++;  }else {  Node\* temp = new Node;  temp->item = newone;  head = temp;  head->pre = NULL;  head->next = NULL;  size++;  }  }  void push\_back(const T& newone) {  Node\* tempPtr;  for (tempPtr = head; tempPtr->next != NULL;)  tempPtr = tempPtr->next;  if (isEmpty() == false){  Node\* temp = new Node;  temp->item = newone;  temp->next = NULL;  tempPtr->next = temp;  temp->pre = tempPtr;  size++;  }  else{  Node\* temp = new Node;  temp->item = newone;  head = temp;  head->pre = NULL;  head->next = NULL;  size++;  }  }  void pop\_front(){  Node\* oldHead = head;  head = head->next;  head->pre = NULL;  delete oldHead;  size--;  }  Iterator start(){  return Iterator(head);  }  Iterator begin(){  return Iterator(NULL);  }  Iterator end(){  Node\* tempPtr;  for (tempPtr = head; tempPtr->next != NULL;)  tempPtr = tempPtr->next;  return Iterator(tempPtr);  }  ~DoubleList(){  while (head != NULL)  pop\_front();  }  };  #endif  company.h  #ifndef COMPCONT\_H  #define COMPCONT\_H  #include "employee.h"  #include "DoubleList.h"  class Company  {  private:  Employee bestPaid;  DoubleList<Employee> container;  public:  void inputEmployee();  void findBestPaidReverse();  void printBestPaid() const;  };  #endif  company.cpp  #include "company.h"  #include <iostream>  using namespace std;  void Company::inputEmployee(){  Employee employee;  employee.input();  while (employee.getName() != "\*"){  container.AddHead(employee);  employee.input();  }  }  void Company::findBestPaidReverse(){  bestPaid = Employee();  DoubleList<Employee>::Iterator cur = container.end();  while (!(cur == container.begin())){  if (\*cur > bestPaid)  bestPaid = \*cur;  cur--;  }  }  void Company::printBestPaid() const{  cout << "The best-paid employee (and gross pay)" << endl;  bestPaid.output();  }  employee.h  #ifndef EMPLOYEE  #define EMPLOYEE  #include <string>  using namespace std;  class Employee{  private:  string name;  double grossPay;  public:  Employee();  void input();  void output() const;  void operator=(const Employee& otherEmployee);  bool operator>(const Employee& otherEmployee) const;  string getName() const;  };  #endif  employee.cpp  #include <iostream>  #include "employee.h"  Employee::Employee(){  name = "";  grossPay = 0;  }  void Employee::input(){  cout << "Please enter a name and gross pay; to quit, enter \* followed by any number: ";  cin >> name >> grossPay;  }  void Employee::output() const{  cout << "is: " << name << ", $" << grossPay << endl;  }  void Employee::operator=(const Employee &otherEmployee){  name = otherEmployee.name;  grossPay = otherEmployee.grossPay;  }  bool Employee::operator>(const Employee &otherEmployee) const{  return grossPay > otherEmployee.grossPay;  }  string Employee::getName() const{  return name;  }  main.cpp  #include "DoubleList.h"  #include "company.h"  #include <iostream>  using namespace std;  int main(){  Company cmp;  cmp.inputEmployee();  cmp.findBestPaidReverse();  cmp.printBestPaid();  return 0;  }  **Output**: | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results and summing up the harvest and the existing problems）   Before doing the experiment, I don't think it's difficult. Just like the previous single linked list experiment, I completed the experiment and immediately completed the experimental report. Until I finished the two-way linked list test experiment, I didn't know it was not easy, but the knowledge I learned was proportional to the difficulty, which benefited me a lot  We must thoroughly understand the knowledge in the textbook, because this is the basis for doing experiments, otherwise I won't understand when the teacher explains, which will make it more difficult for me to do experiments and waste my precious time doing experiments. If I don't know, I should explore it when doing the experiment, which will greatly waste my time and get half the result with twice the effort  When doing the experiment, you must do it yourself. Ensure that each step and detail is clarified and understood. After the experiment, I must review and think. In this way, I will be deeply impressed and firmly remember. Otherwise, I will soon forget everything. You'd better not do that  According to his personal experience, Mr. Zhao taught me some knowledge that I didn't have in the textbook, which broadened my vision and made us realize that this course is so widely used in life. Through the two-way linked list experiment, I learned a lot of practical knowledge. More importantly, the process of doing experiments and the method of thinking are the same as other experiments. It really benefited me a lot | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |