

**Lab Report**

**实验报告**

|  |  |
| --- | --- |
| **Course**: | Class Libraries and Data Structures |
| **Semester**: | 1st semester of the academic year **2023-2024** |
| **Major**: | Software Engineering |
| **Class**: | 2022 |
| **Student Name**: |  |
| **Student ID:** |  |
| **Teacher:** | ZHAO, Hengjun (赵恒军) |

**School of Computer and Information Science**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | | C++ Template Containers and Iterators  C++模板容器和迭代器 | | | |
| Date | | Oct，2023 | Type | | ☑Confirmatory （验证确认型）  ☑Design（设计型）  🗆Comprehensive（综合型） |
| 1. **Objective & Requirements（实验目的）**    1. Understand the concept of containers; can use template to define generic containers   理解C++容器的概念，学会使用模板定义通用容器   * 1. Understand the difference between contiguous memory allocation and linked memory allocation; grasp the implementation of linked storage   理解链式和连续式两种容器底层的存储方式及其区别; 掌握链式存储的实现方式   * 1. Understand the use of iterator and grasp its implementation   理解迭代器的概念和用途，掌握其实现原理   * 1. Can use iterator to traverse a list to finish a certain task;   Understand the concept of generic algorithms  能够使用容器和迭代器编写泛型算法 | | | | | |
| 1. **Experimental environment (**platform and software**)（实验环境）**   Windows 7 (or higher versions) + Visual Studio 2010 (or higher versions) | | | | | |
| 1. **Experimental content and design** (Main Content, Procedure, Codes and Results)（此部分应包含每一个实验内容的详细设计，含实验思路、详细实验步骤、核心代码说明等） 2. Task 1 （任务1）   For this task, you are provided with a template container with linked storage. Based on the source codes, implement the addHead() method which adds a new element at the head of the linked list of the container. Then, based on the addHead() method, implement the inputEmployee() method for the Company class, which takes users’ inputs and stores the input Employee into the Company’s container. Test your implementation in the main function.  基于所给代码，为链式容器类实现addHead()，用以在容器内链表的头部添加元素；基于addHead(), 为公司类实现inputEmployee()方法，用以接收用户的输入并将其存入公司的容器中。在main函数中测试你的实现。、  答：  **实验思路**：  addHead()函数中，传入的是一个Employee对象，那么首先需要new一个Node对象，将数据存入Node对象。然后要调整新节点的指针域和head指针，让新节点指针域指向原来的第一个结点，让头指针head指向新节点，最后size自增。  inputEmployee()函数中，在循环之外先用临时employee对象接受第一个数据作为第一次判断的依据，之后循环将接收的数据放入容器内。直到输入name为\*时停止读取。代码及详细解释如下       1. Task 2（任务2）   Read and comprehend the source code provided to you on the implementation of the inner iterator class for the linked container. Especially, try to understand the functionality of the public methods of the iterator class. Then, based on your implementations in Task 1 and the public interfaces of the iterator, implement the findBestPaid() method for the Company class to get the best paid employee. Test your implementation in the main() funtion.  阅读并理解链式容器类的内部迭代器类，掌握其公共接口的功能及其实现原理。基于迭代器类和任务1，进一步为公司类编写findBestPaid()方法以实现查找收入最高的员工。在main函数中测试你的实现。  **实验思路：**  使用迭代器从Begin到End遍历一遍，寻找最大收入员工。注意使用重载的运算符：++、\*、==、>等。寻找最大值的方法也很简单，就是首先将第一个员工赋值给bestpaid，之后遍历判断，找到比当前bestpaid大的就直接赋值给bestpaid。  Company.h中的findBestPaid函数：    Main函数：    运行结果：     1. Task 3（任务3）      1. Based on the source codes in Task 1 and Task 2, design a container with doubly linked list storage that has a structure similar to the one in the above picture. 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. Then implement the container and its associated iterator inner class.   基于任务1和任务2，设计一个底层存储为双向链表的容器及其相应的迭代器，该双向链表的架构应和上图中所示类似。为此，需要设计链式存储的节点结构、容器类的数据成员和方法接口、对应迭代器内部类的数据成员和方法接口。最终实现完整的双向链式容器及其迭代器。   1. Based on your implementation of the container with doubly linked storage, design and implement the method findBestPaidReverse() method for the Company class, which traverses the container 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--. Besides, this also means that you should define two more special iterators, i.e. the reverse versions of Begin() and End(), to support the reverse traversal.   基于所实现的双向链式容器和迭代器，改写公司类，设计实现findBestPaidReverse()方法，以从尾至头的方式查找收入最高的员工。这意味着你实现的迭代器应支持--运算（重载operator --）；此外，这还意味着你应该实现Begin()和End()这两个特殊迭代器的反向版本，以配合operator--支持反向遍历。   1. Test your implementation in the main function.   在主函数中测试你的实现。  **思路：**  首先结点结构要增加一个pre指针，指向前一个结点；这时如果要添加一个节点，就要新增一个功能使后一个节点的pre指向新增的节点；迭代器类中需要重载operator --，指向最后一个节点的迭代器reverse\_Begin()（可以通过先用一个临时遍历到最后一个节点，再用这个节点的指针初始化迭代器）和指向第一个节点的前一个节点的reverse\_End()，至于findBestPaidReverse()方法，只需要将之前写过的正向遍历的方法稍作修改为反向遍历即可  **代码：**  双链表头文件listTemp.h  #ifndef LISTTEMP\_H  #define LISTTEMP\_H  #define NULL 0  template<class T>  class doubleListTemp  {  private:  struct Node  {  T data;  Node \*next;  Node\* pre;  };  Node \*head;  int size;  public:  doubleListTemp(); //constructor  ~doubleListTemp(); //destructor  int getLength() const; //get the number of elements  bool isEmpty() const; //check whether the container is empty  // Postcondition: A node with newData has been inserted at the  // head of the Linked container.  void addHead(const T& newData);  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*declaration of the inner iterator class\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  class Iterator  {  friend class doubleListTemp<T>; //friend class of the iterator class  private:  Node \*curr;  Iterator(Node \*ptr); //constructor with Node parameter, defined as private  public:  Iterator(); //default constructor  Iterator operator++(int); //post-increment of ++  Iterator operator--(int); //反向迭代  T& operator\*() const;  bool operator==(const Iterator other) const;  };//class Iterator  Iterator Begin() const;  Iterator reverse\_Begin() const;  Iterator End() const;  Iterator reverse\_End() const;  };  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*implementation of the iterator inner class\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  template<class T>  doubleListTemp<T>::Iterator::Iterator()  {  curr = NULL;  }  template<class T>  doubleListTemp<T>::Iterator::Iterator(Node \*ptr)  {  curr = ptr;  }  template<class T>  typename doubleListTemp<T>::Iterator doubleListTemp<T>::Iterator::operator++(int)  {  Iterator temp = \*this; //default copy constructor  this->curr = curr->next;  return temp; //return iterator object  }  template<class T>  typename doubleListTemp<T>::Iterator doubleListTemp<T>::Iterator::operator--(int)  {  Iterator temp = \*this; //default copy constructor  this->curr = curr->pre; //让当前指针指向前一个结点  return temp; //return iterator object  }  template<class T>  T& doubleListTemp<T>::Iterator::operator\*() const  {  return curr->data; //return data reference  }  template<class T>  bool doubleListTemp<T>::Iterator::operator==(const Iterator other) const  {  return curr == other.curr;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*implementation of the Begin and End position\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  template<class T>  typename doubleListTemp<T>::Iterator doubleListTemp<T>::Begin() const  {  return Iterator(head);  }  template<class T>  typename doubleListTemp<T>::Iterator doubleListTemp<T>::reverse\_Begin() const  {  Node\* endNode = head;  while(endNode->next != NULL) {  endNode = endNode->next;  }  return Iterator(endNode); //用最后一个元素的地址来初始化  }  template<class T>  typename doubleListTemp<T>::Iterator doubleListTemp<T>::End() const  {  return Iterator(); //NULL pointer  }  template<class T>  typename doubleListTemp<T>::Iterator doubleListTemp<T>::reverse\_End() const  {  return Iterator(head->pre);  //friend class is required  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*implementation of the linked list class template\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  template<class T>  doubleListTemp<T>::doubleListTemp()  {  head = NULL; //empty list  size = 0;  }  template<class T>  doubleListTemp<T>::~doubleListTemp()  {  Node \*current = head;  Node \*temp = NULL;  while (current != NULL)  {  temp = current;  current = current->next;  delete temp; //release  }  }  template<class T>  int doubleListTemp<T>::getLength() const  {  return size;  }  template<class T>  bool doubleListTemp<T>::isEmpty() const  {  return size == 0;  }  template<class T>  void doubleListTemp<T>::addHead(const T& newData)  {  //please implement this  Node\* newHead = new Node;  if (head != NULL) {  head->pre = newHead; //反向迭代  }  newHead->data = newData;  newHead->next = head;  head = newHead;  size++;  }  #endif  Company.cpp文件中的findBestPaidReverse()方法：    Main函数：    测试及运行结果： | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results and summing up the harvest and the existing problems）（此部分应包含实验结果，对实验结果的分析，实验收获的总结，实验中存在问题的讨论等；另外，需要回应一下如下思考题：1. 在链式容器中，为什么将容器类定义为其内部迭代器类的友元类？2. 双向链表是否需要头指针Head，还需要一个尾指针Tail，才能实现双向遍历？如果只有头指针，如何逆序遍历整个链表？）   从本次实验中，我初步认识了链表容器的基本性质以及一些对容器的操作，学习到了迭代器基本特性以及其是如何在容器中充当指针发挥作用的，还有关于嵌套类友元类的访问关系等等。实现双向链表的过程又是对链表的一个重新认识。迭代器的构造和方法很巧妙，重载了很多运算符，封装了很多复杂的操作，可以抽象地当作一个泛型指针来看，便于使用。  有很多小细节需要注意，比如头节点和尾结点中pre和next指针的处理  思考题：  （1）迭代器类虽然在容器类内定义，但是它无法直接访问容器类的私有成员，即head、size和Node，因为内部类是一个独立的类，默认内部类是外部类的友元类，但是外部类不是内部类的友元类。而我们在迭代器中的各种方法需要调用到容器类的私有成员，所以将容器类定义为其内部迭代器类的友元类。  （2）其实不需要尾指针Tail就可以完成双向遍历，方法是直接让reverse\_Begin()返回一个指向最后一个结点的迭代器。  法①：我们先创建一个临时的Node指针，用head来初始化它，此时它指向第一个结点，然后利用循环每次将它指向结点的next赋值给它，直到它指向结点的下一个结点为空，此时指向的就是最后一个结点，最后用这个指针来初始化迭代器即可得到一个指向最后一个结点的迭代器。    法②：我们可以先在函数内创建一个临时迭代器，用Begin()复制构造，之后我们用之前重载的++运算符迭代临时迭代器，判断到最后一个节点停止迭代，这样就使得临时迭代器指向最后一个结点。    我觉得方法①更好一些，因为不依赖于++和Begin()。  关键在于如何构建begin和end两个迭代器，剩下的都基本相同。 | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |