山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201705130113 | 姓名： 黄瑞哲 | | 班级： 计科17.3 |
| 实验题目：链式描述线性表 | | | |
| 实验学时：2 | | 实验日期： 2017.10.18 | |
| 实验目的：   1. 掌握线性表结构、链式描述方法（链式存储结构）、链表的实现 2. 掌握链表迭代器的实现与应用 | | | |
| 软件环境：  G++ 8.1.0  Visual Studio Community 2017 | | | |
| 1. 实验内容（题目内容，输入要求，输出要求） 2. 创建线性表类：线性表的存储结构使用单链表：提供操作：自表首插入元素、删除指定元素、搜索表中是否有指定元素、输出链表 3. 接受键盘录入的一系列整数（例 10,25,8,33,60）作为节点的元素值，创建链 表。输出链表内容 4. 输入一个整数（例33），在链表中进行搜索，输出元素的索引。如果不存在输出-1 5. 设计实现链表迭代器，使用链表迭代器实现链表的正序输出 6. 创建两个有序链表，使用链表迭代器实现链表的合并 7. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法） 8. 在循环体中遍历链表，如果找到元素相等则返回下标，如果循环体运行结束则表示没 有找到，返回-1，时间复杂度为O(n) 9. 链表迭代器实现正序输出：迭代器从begin循环至end，依次输出即可。 10. 有序链表的合并：利用两个迭代器，比较两个迭代器指向元素的大小，将较小的加入新链表的尾部并将对应迭代器自增1，直至有一个迭代器到达链表尾部。然后再将另一个迭代器依次把对应元素加到新链表的尾部，直至到达原链表尾。 11. 测试结果（测试输入，测试输出，结果分析）      1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   在试验中遇到了一个问题：在声明模板类的迭代器的时候需要加上typename 关键字，否则无法通过编译。原因是chain<T>::iterator意义不明确，它依赖于模板参数，称为嵌套依赖名字，为了消除歧义，C++规定，对于嵌套依赖名字，统统不解释为类型，除非使用typename关键字显式声明。   1. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）   /\*linearList.h\*/  #ifndef \_\_LINEARLIST\_H\_  #define \_\_LINEARLIST\_H\_  template<typename T>  class linearList  {  public:  virtual ~linearList() {};  virtual bool empty() const = 0;  virtual int size() const = 0;  virtual int find(const T&) const = 0;  virtual void erase(int) = 0;  virtual void insert(int, const T&) = 0;  virtual void clear() = 0;  virtual void push\_back(const T& ) = 0;  virtual T& operator[](int) = 0;  virtual const T& operator[](int) const = 0;  //virtual  };  #endif //\_\_LINEARLIST\_H\_  /\*chain.h\*/  #ifndef \_\_CHAIN\_H\_  #define \_\_CHAIN\_H\_  #include "linearList.h"  #include <stdexcept>  using namespace std;  template<typename T>  struct chainNode  {  T element;  chainNode<T>\* next;  chainNode(const T& element, chainNode<T>\* next = nullptr)  {  this->element = element;  this->next = next;  }  chainNode(const chainNode<T>\*& c)  {  element = c->element;  next = nullptr;  }  };  template<typename T>  class chain  :virtual public linearList<T>  {  public:  chain(int = 10);  chain(const chain<T>&);  ~chain();  bool empty() const;  int size() const;  int find(const T&) const;  void erase(int);  void insert(int, const T&);  void clear();  void push\_back(const T&);  chain<T>& operator=(const chain<T>&);  T& operator[](int);  const T& operator[](int) const;  class iterator;  class const\_iterator;  iterator begin() { return iterator(pHead->next); }  iterator end() { return iterator(nullptr); }  const\_iterator begin() const { return const\_iterator(pHead->next); }  const\_iterator end() const { return const\_iterator(nullptr); }  class iterator  {  public:  typedef forward\_iterator\_tag iterator\_category;  typedef T value\_type;  typedef ptrdiff\_t difference\_type;  typedef T\* pointer;  typedef T& reference;  iterator(chainNode<T>\* theNode = nullptr) :node(theNode) {};  T& operator\*() { return node->element; }  T\* operator->() { return &node->element; }  iterator& operator++()  {  node = node->next;  return \*this;  }  iterator operator++(int)  {  iterator old = \*this;  node = node->next;  return old;  }  bool operator==(const iterator right) const { return node == right.node; }  bool operator!=(const iterator right) const { return node != right.node; }  protected:  chainNode<T>\* node;  };  class const\_iterator  {  public:  typedef forward\_iterator\_tag iterator\_category;  typedef T value\_type;  typedef ptrdiff\_t difference\_type;  typedef T\* pointer;  typedef T& reference;  const\_iterator(chainNode<T>\* theNode) :node(theNode) {};  const T& operator\*() { return node->element; }  const T\* operator->() { return &node->element; }  const\_iterator& operator++()  {  node = node->next;  return \*this;  }  const\_iterator operator++(int)  {  const\_iterator old = \*this;  node = node->next;  return old;  }  bool operator==(const const\_iterator right) const { return node == right.node; }  bool operator!=(const const\_iterator right) const { return node != right.node; }  protected:  chainNode<T>\* node;  };  protected:  chainNode<T>\* pHead;  chainNode<T>\* pTail;  int listSize;  void checkIndex(int) const;  };  template<typename T>  chain<T>::chain(int initialCapacity)  {  if (initialCapacity < 1) throw out\_of\_range("the initial Capacity of arrayList must > 0");  listSize = 0;  pHead = new chainNode<T>(0);  pTail = pHead;  }  template<typename T>  chain<T>::chain(const chain<T>& c)  {  pHead = new chainNode<T>(c.pHead->element);  pTail = pHead;  chainNode<T>\* sourceNode = c.pHead->next;  chainNode<T>\* currentNode = pHead;  while (sourceNode != nullptr)  {  pTail = currentNode->next = new chainNode<T>(sourceNode->element);  currentNode = currentNode->next;  sourceNode = sourceNode->next;  }  listSize = c.listSize;  }  template<typename T>  chain<T>::~chain()  {  chainNode<T>\* currentNode = pHead->next;  chainNode<T>\* deleteNode;  while (currentNode != nullptr)  {  deleteNode = currentNode;  currentNode = currentNode->next;  delete deleteNode;  }  delete pHead;  }  template<typename T>  bool chain<T>::empty() const { return listSize == 0; }  template<typename T>  int chain<T>::size() const { return listSize; }  template<typename T>  int chain<T>::find(const T& theElement) const  {  int index = 0;  chainNode<T>\* currentNode = pHead->next;  while (currentNode != nullptr)  {  if (currentNode->element == theElement) return index;  currentNode = currentNode->next;  ++index;  }  return -1;  }  template<typename T>  void chain<T>::erase(int theIndex)  {  checkIndex(theIndex);  chainNode<T>\* deleteNode;  chainNode<T>\* pre = pHead;  for (int i = 0; i < theIndex; ++i) pre = pre->next;  if (theIndex == listSize - 1) pTail = pre;  deleteNode = pre->next;  pre->next = pre->next->next;  --listSize;  delete deleteNode;  }  template<typename T>  void chain<T>::insert(int theIndex, const T& theElement)  {  if (theIndex < 0 || theIndex>listSize) throw out\_of\_range("illegalIndex");  chainNode<T>\* pre = pHead;  for (int i = 0; i < theIndex; ++i) pre = pre->next;  pre->next = new chainNode<T>(theElement, pre->next);  if (theIndex == listSize) pTail = pre->next;  ++listSize;  }  template<typename T>  void chain<T>::clear()  {  chainNode<T>\* currentNode = pHead->next;  chainNode<T>\* deleteNode;  while (currentNode != nullptr)  {  deleteNode = currentNode;  currentNode = currentNode->next;  delete deleteNode;  }  listSize = 0;  pHead->next = nullptr;  pTail = pHead;  }  template<typename T>  void chain<T>::push\_back(const T& theElement)  {  pTail->next = new chainNode<T>(theElement, pTail->next);  pTail = pTail->next;  listSize++;  }  template<typename T>  chain<T>& chain<T>::operator=(const chain<T>& c)  {  if (this == &c) return \*this;  clear();  chainNode<T>\* currentNode = pHead;  chainNode<T>\* sourceNode = c.pHead->next;  while (sourceNode != nullptr)  {  pTail = currentNode->next = new chainNode<T>(sourceNode->element);  currentNode = currentNode->next;  sourceNode = sourceNode->next;  }  listSize = c.listSize;  return \*this;  }  template<typename T>  T& chain<T>::operator[](int index)  {  checkIndex(index);  chainNode<T>\* currentNode = pHead->next;  for (int i = 0; i < index; ++i) currentNode = currentNode->next;  return currentNode->element;  }  template<typename T>  const T& chain<T>::operator[](int index) const  {  checkIndex(index);  chainNode<T>\* currentNode = pHead->next;  for (int i = 0; i < index; ++i) currentNode = currentNode->next;  return currentNode->element;  }  template<typename T>  void chain<T>::checkIndex(int theIndex) const  {  if (theIndex < 0 || theIndex >= listSize)  throw out\_of\_range("the index is out of range");  }  #endif //\_\_CHAIN\_H\_  /\*main.cpp\*/  // exe4.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。  //  #include "pch.h"  #include <iostream>  using namespace std;  template<typename T>  void show(const chain<T>& c)  {  for (auto it = c.begin(); it != c.end(); ++it) cout << \*it << ' ';  cout << endl;  }  template<typename T>  chain<T> merge(const chain<T>& a, const chain<T>& b)  {  chain<T> c;  typename chain<T>::const\_iterator it1 = a.begin();//此处需要显式声明  typename chain<T>::const\_iterator it2 = b.begin();  while (it1 != a.end() && it2 != b.end())  {  if (\*it1 <= \*it2) c.push\_back(\*(it1++));  else c.push\_back(\*(it2++));  }  while (it1 != a.end()) c.push\_back(\*(it1++));  while (it2 != b.end()) c.push\_back(\*(it2++));  return c;  }  int main()  {  int n;  chain<int> c;  chain<int> a;  chain<int> b;  cout << "input the length of chain: ";  cin >> n;  cout << "input n numbers: " << endl;  for (int i = 0; i < n; ++i)  {  int x;  cin >> x;  c.push\_back(x);  }  cout << "the number in chain are: " << endl;  for (int i = 0; i < c.size(); ++i) cout << c[i] << ' ';  cout << endl;  cout << "input a number which you want to find:" << endl;  int x;  cin >> x;  cout << c.find(x) << endl;  cout << "reversed:" << endl;  show(c);  cout << "merge: (note that a is equal to c and so is b)" << endl;  a = c;  b = c;  c = merge(a, b);  cout << "after merge: " << endl;  for (auto it = c.begin(); it != c.end(); ++it) cout << \*it << ' ';  cout << endl;  return 0;  }  /\*pch.h\*/  #ifndef PCH\_H  #define PCH\_H  // TODO: 添加要在此处预编译的标头  #include "chain.h"  #endif //PCH\_H | | | |