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：201700130009 | 姓名：张愈博 | | 班级： 计科3 |
| 实验题目：链式描述线性表 | | | |
| 实验学时：2 | | 实验日期： 2018.10.19 | |
| 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和应用。 | | | |
| 软件环境：  Windows 10  Dev C++ 5.11 | | | |
| 1.内容（题目内容，输入要求，输出要求）  (1)创建线性表类：线性表的存储结构使用单链表，并提供插入、删除、搜索指定元素、输出等操作。  (2)接收键盘输入的一组整数，给线性表初始化。  (3)设计链表迭代器，实现用迭代器正向输出链表和用迭代器实现链表合并。  2.结构与算法描述 （整体思路描述，所需要的数据结构与算法）  要实现链表描述的线性表，首先需设计作为链表节点的ChainNode类，其中包含储存在节点中的元素和和若干种构造函数。然后是Chain类本身，和Arraylist类一样，由linearlist类派生而来。除了要实现linearlist中的成员函数外，还要重载接收数组的构造函数。链表迭代器需设计成Chain的一个子类，其中包含指向链表节点的指针、解引用运算符、自加运算符和判断是否相等的运算符。   1. 测试结果（测试输入，测试输出，结果分析）     图1 图2    图3 图4    图5  测试插入操作：输入1、3、5、7、9,再于索引为1的地方插入数字2,输出线性表（如图1）。  测试删除操作：把刚刚输入的2从线性表中删除（如图2）。  测试查找操作：输入要查找的元素，若表中存在，则返回它的索引（如图3）。  测试迭代器输出：输出指定线性表（如图4）。  测试两个有序线性表的合并：再初始化一个线性表，将两表按顺序合并并输入。（如图5）   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径） 2. 起初我将链表迭代器设为一个单独类，然后使用类的组合，把iterator声明为Chain类中的一个对象。后来在实现过程中产生了种种问题，经过查阅资料，在线性表类中,已经存在一个系统定义的iterator，在其中声明重名对象会报错，但使用类的包含可以解决以上问题，或者把迭代器改名为My\_iterator也可以解决这个问题。 3. 在测试以迭代器方法输出线性表时，出现了错误。经分析，是iterator类中定义前自加运算符时没有返回引用造成的。该函数中，先改变了node指针的值，再将其返回，这种情况下需返回引用。 4. 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）   #include<iostream>  #include<string>  using namespace std;  template<class T>  class linearlist //抽象类线性表  {  public:  virtual ~linearlist() { }  virtual bool empty() const=0;  virtual int size() const=0;  virtual T& get(int theIndex) const=0;  virtual int indexOf(T& theElement) const=0;  virtual void erase(int theIndex)=0;  virtual void insert(int theIndex,const T& theElement)=0;  virtual void output() const = 0;  };  template<class T>  struct ChainNode  {  T element;  ChainNode \*next=NULL;    ChainNode(T ele,ChainNode \*p)  {  element=ele;  next=p;  }  ChainNode(T ele)  {  element=ele;  }  ChainNode() { }  ChainNode& operator=(ChainNode &A)  {//结点赋值  element=A.element;  next=A.next;  return \*this;  }  };  template<class T>  class Chain :public linearlist<T>  {  public:  Chain() { }  Chain(int initialCapacity,T A[])  {  if(initialCapacity>0)  {  firstnode=new ChainNode<T>;  firstnode->element=A[0];  ChainNode<T> \*p1=firstnode,\*p2;  for(int i=1;i<initialCapacity;i++) //等于1的话前面已经创建好了一个节点  {  p2=new ChainNode<T>;  p2->element=A[i];  p1->next=p2;  p1=p2;  }  listSize=initialCapacity;  }  if(initialCapacity<=0) cerr<<"初始长度过小！"<<endl;  }  Chain(int initialCapacity) //相当于默认构造函数  {//这里只开辟节点不给他们赋值  if(initialCapacity>=0)  {  firstnode=new ChainNode<T>;  ChainNode<T> \*p1=firstnode,\*p2;  for(int i=1;i<initialCapacity;i++)  {  p2=new ChainNode<T>;  p1->next=p2;  p1=p2;  }  listSize=initialCapacity;  }  if(initialCapacity<0) cerr<<"初始长度过小！"<<endl;  }  Chain(const Chain<T> &A)  {  listSize=A.listSize;  if(listSize==0)  {//此表为空  firstnode=NULL;  }  else  {//TargetNode指向A的节点，SourceNode指向此链表的节点  //直接节点赋值会变成浅复制，ciao!  ChainNode<T> \*SourceNode=new ChainNode<T>;  ChainNode<T> \*pre\_node,\*TargetNode;  firstnode=SourceNode;  TargetNode=A.firstnode;  SourceNode->element=TargetNode->element;    for(int i=1;i<listSize;i++) //等于1的话前面已经创建好了一个节点  {  TargetNode=TargetNode->next;  pre\_node=new ChainNode<T> (TargetNode->element); //这里给pre\_node创建了一个新空间并初始化  SourceNode->next=pre\_node;  SourceNode=pre\_node;  }  }    }  ~Chain()  {  while(firstnode!=NULL)  {  ChainNode<T> \*nextnode=firstnode->next;  delete firstnode;  firstnode=nextnode;  }  }  bool empty() const  { return listSize==0; }  int size() const  { return listSize; }  T& get(int theIndex) const  {//返回索引是theIndex节点的element内容  checkIndex(theIndex);  ChainNode<T> \*cur\_node=firstnode;  for(int i=0;i<theIndex;i++)  {  cur\_node=cur\_node->next;  }  return cur\_node->element;  }  int indexOf(T& theElement) const  {//返回节点元素内容是theElement的节点索引，若未找到则返回-1  ChainNode<T> \*cur\_node=firstnode;  for(int i=0;i<listSize;i++)  {  if(cur\_node->element==theElement)  return i;  else  cur\_node=cur\_node->next;  }  return -1;  }  void erase(int theIndex)  {//从链表中删除索引为theIndex的节点  checkIndex(theIndex);  ChainNode<T> \*deletenode;  if(theIndex==0)  {//删除头结点  deletenode=firstnode;  firstnode=firstnode->next;  }  else  {  ChainNode<T> \*p=firstnode;  for(int i=0;i<theIndex-1;i++)  {  p=p->next;  }  //此时已经找到要删除的结点的前驱节点  deletenode=p->next;  p->next=p->next->next;  }  listSize--;  delete deletenode;  }  void insert(int theIndex,const T& ele)  {//将ele元素插入索引为theIndex位置的新节点  checkIndex(theIndex);  if(theIndex==0)  {//插入头节点  ChainNode<T> \*p=new ChainNode<T> (ele,firstnode);  firstnode=p;  }  else  {  ChainNode<T> \*p=new ChainNode<T> (ele);//待插入节点  ChainNode<T> \*cur\_node=firstnode;  for(int i=0;i<theIndex-1;i++)  {  cur\_node=cur\_node->next;  }  //此时已经找到要插入的结点的前驱节点  p->next=cur\_node->next;  cur\_node->next=p;  }  listSize++;  }  void output() const  {  if(listSize==0){ cout<<"null"<<endl; return; }  else{  ChainNode<T> \*p=firstnode;  for(int i=0;i<listSize;i++)  {  cout<<p->element<<" ";  p=p->next;  }  }  }  void i\_print() const  {//使用迭代器输出  if(listSize==0){ cout<<"null"<<endl; return; }  else{  iterator I(firstnode);  for(int i=0;i<listSize;i++)  {  cout<<(\*I)<<" ";  I++;  }  }  }  void merge(Chain<T> A,Chain<T> B)  {//把A和B有序合并，放入\*this中  cout<<this->listSize<<endl;  this->clear();  listSize=A.listSize+B.listSize;    firstnode=new ChainNode<T>;  ChainNode<T> \*p1=firstnode,\*p2;  for(int i=1;i<listSize;i++)  {  p2=new ChainNode<T>;  p1->next=p2;  p1=p2;  }    iterator ai(A.firstnode);  iterator bi(B.firstnode);  iterator ti(firstnode);  while(ai!=NULL&&bi!=NULL)  {  if(\*ai<\*bi) { \*ti=\*(ai++); \*(ti++); }  else { \*ti=\*(bi++); \*(ti++); }  }  if(ai==NULL)  {  while(bi!=NULL){ \*ti=\*(bi++); \*(ti++); }  }  else  {  while(ai!=NULL){ \*ti=\*(ai++); \*(ti++); }  }  }  void clear()  {//清表操作  ChainNode<T> \*deletenode,\*p=firstnode;  for(int i=0;i<listSize;i++)  {  deletenode=p;  p=p->next;  delete deletenode;  }  firstnode=NULL;  listSize=0;  }  friend ostream& operator<<(ostream& out,const Chain<T> A)  {//这里要调用复制构造函数  if(A.listSize==0){ out<<"null"<<endl; return out; }  else{  ChainNode<T> \*p=A.firstnode;  for(int i=0;i<A.listSize;i++)  {  out<<p->element<<" ";  p=p->next;  }  return out;  }  }  class iterator  {//构造函数、\*&、前后自加、！=、==  protected:  ChainNode<T> \*node=NULL;  public:  iterator(ChainNode<T> \*theNode)  { node=theNode; }  iterator() { }    T& operator\*() { return node->element; } //取内容运算符，作为引用返回  T\* operator&() { return &(node->element); } //取地址运算符，作为指针返回  iterator& operator++()  {//前加  node =node->next;  return \*this;  }  iterator operator++(int)  {//后加  iterator temp=\*this;  node =node->next;  return temp;  }  bool operator!=(const iterator I)const  {  if(node==I.node) return false; //这里我没有定义ChainNode的==，为什么没有报错？  else return true;  }  bool operator==(const iterator I)const  {  if(node==I.node) return true;  else return false;  }    };    protected:  void checkIndex(int theIndex) const  {//确定索引的有效性，防止越界  if(theIndex<0||theIndex>listSize-1)  {  cerr<<"请检查索引的有效性！"<<endl;  }  }  ChainNode<T> \*firstnode; //指向链表第一个元素的指针  int listSize; //链表的元素个数  };  int main()  {  cout<<"please the number of datas:";  int n; cin>>n;  int \*a=new int[n];  cout<<"please the value of datas:";  for(int i=0;i<n;i++) { cin>>a[i]; }  cout<<endl<<"The initialization operation is performed below："<<endl;  Chain<int> A(n,a); cout<<"Print A:"<<A<<endl;  //以上是初始化部分  bool k=true;  while(k)  {  int flag;  cout<<"执行插入操作请输入1"<<endl<<"执行删除操作请输入2"<<endl;  cout<<"执行查找操作请输入3"<<endl<<"输出线性表(迭代器)请输入4"<<endl<<"执行合并操作请输入5"<<endl;  cin>>flag;  switch(flag)  {  case 1: {  cout<<"请输入要插入的数据"<<endl;  int ele; cin>>ele;  cout<<"请输入要插入的位置索引(从0开始)"<<endl;  int pos; cin>>pos;  A.insert(pos,ele);  cout<<"成功插入！"<<endl<<A<<endl;  break;  }  case 2: {  cout<<"请输入要删除元素的索引(从0开始)"<<endl;  int pos; cin>>pos;  A.erase(pos);  cout<<"成功删除！"<<endl<<A<<endl;  break;  }  case 3: {  cout<<"请输入要查找的元素："<<endl;  int ele; cin>>ele;  cout<<"所查找元素的索引是："<<A.indexOf(ele)<<endl;  break;  }  case 4: {  A.i\_print();  break;  }  case 5: {  cout<<"请再输入一个有序数组B："<<endl;  cout<<"please the number of datas:";  int n; cin>>n;  int \*b=new int[n];  cout<<"please the value of datas:";  for(int i=0;i<n;i++) { cin>>b[i]; }  cout<<endl<<"The initialization operation is performed below："<<endl;  Chain<int> B(n,b); cout<<"Print B:"<<B<<endl;  Chain<int> C(0);  C.merge(A,B); cout<<"合并以后的内容："<<C<<endl;  break;  }  }  cout<<"如要继续操作请输入continue,否则输入任意字符"<<endl;  string str; cin>>str;  if(str=="continue") k=true;  else k=false;  }  delete []a;  return 0;  } | | | |