山东大学 软件 学院

数据结构 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号： | 姓名： | | 班级： |
| 实验题目：线性表操作 | | | |
| 实验学时：4小时 | | 实验日期：2018.10.19 | |
| 实验目的：  1、掌握线性表的基本操作：插入、删除、查找。  2、掌握链表遍历器的使用方法。 | | | |
| 硬件环境： 笔记本电脑 | | | |
| 软件环境：Win10+Vistual Studio 2017 | | | |
| 实验步骤与内容：  实验内容：  1、创建线性表类。线性表的存储结构使用链表。  2、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、接收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，输出其在链表中的位置。如果不存在输出0。  5、使用链表遍历器实现链表的反序输出。  6、创建两个有序链表，使用链表遍历器实现链表的合并。  实验步骤：  chain.h  #ifndef chain\_H  #define chain\_H  #include "chainNode.h"  #include <iostream>  using namespace std;  class chain  {  friend class chainIterator;  public:  //析构函数  chain() { firstNode = 0; }  ~chain();  //实现抽象数据类型方法  int size();  int get(int theIndex) const;  int indexOf(const int theElement) const;  void erase(int theIndex);  void insert(int theIndex, const int theElement);  void output(ostream& out) const;  chain& add(const int& x);  protected:  chainNode\* firstNode; // 首节点的指针  int listSize; //线性表元素个数  };  #endif  chainIterator.h  #ifndef chainIterator\_H  #define chainIterator\_H  #include "chainNode.h"  class chainIterator  {  public:  int\* initialize(const chain &c);  int\* Next();  private:  chainNode \*location;  };  #endif  merge.h  #ifndef merge\_H  #define merge\_H  #include <iostream>  using namespace std;  chainNode \*merge(chainNode \*firstNode1, chainNode \*firstNode2)  {  if (firstNode1 == NULL)  return firstNode2;  if (firstNode2 == NULL)  return firstNode1;  chainNode \*firstNode = NULL;  if (firstNode1->element < firstNode2->element)  {  firstNode = firstNode1;  firstNode->next = merge(firstNode1->next, firstNode2);  }  else  {  firstNode = firstNode2;  firstNode->next = merge(firstNode1, firstNode2->next);  }  return firstNode;  }  //创建一个有序的单链表1  chainNode \*sequentialList1()  {  chainNode \*firstNode;  chainNode \*n1 = new chainNode(1);  chainNode \*n3 = new chainNode(3);  chainNode \*n5 = new chainNode(5);  chainNode \*n7 = new chainNode(7);  chainNode \*n9 = new chainNode(9);  chainNode \*n11 = new chainNode(11);  chainNode \*n13 = new chainNode(13);  firstNode = n1;  n1->next = n3;  n3->next = n5;  n5->next = n7;  n7->next = n9;  n9->next = n11;  n11->next = n13;  n13->next = NULL;  return firstNode;  }  //创建一个有序的单链表2  chainNode \*sequentialList2()  {  chainNode \*firstNode;  chainNode \*n2 = new chainNode(2);  chainNode \*n4 = new chainNode(4);  chainNode \*n6 = new chainNode(6);  chainNode \*n8 = new chainNode(8);  firstNode = n2;  n2->next = n4;  n4->next = n6;  n6->next = n8;  n8->next = NULL;  return firstNode;  }  //将链表空间释放  void freeList(chainNode \*firstNode)  {  if (firstNode == NULL)  {  return;  }  else  {  chainNode \*temp = firstNode->next;  delete firstNode;  firstNode = temp;  freeList(firstNode);  }  }  void printList(chainNode \*firstNode)  {  chainNode \*cur = firstNode;  while (cur)  {  cout << cur->element<<" ";  cur = cur->next;  }  cout << endl;  }  #endif  chainNode.h  #ifndef chainNode\_H  #define chainNode\_H  struct chainNode  {  //数据成员  friend class chain;  friend class chainIterator;  int element;  chainNode \*next;  //方法  chainNode() {}  chainNode(const int element)  {  this->element = element;  }  chainNode(const int element, chainNode \*next)  {  this->element = element;  this->next = next;  }  };  #endif  chainIterator.cpp  #include "chainIterator.h"  #include "chain.h"  using namespace std;  int\* chainIterator::initialize(const chain & c)  {  location = c.firstNode;  if (location)  return &location->element;  return 0;  }  int\* chainIterator::Next()  {  if (!location)  return 0;  location = location->next;  if (location)  return &location->element;  return 0;  }  chain.cpp  #include "chain.h"  #include "chainNode.h"  #include <iostream>  #include <sstream>  #include <string>  using namespace std;  int chain::size()  {  chainNode \*p = firstNode->next;  int n = 0;  while (p != NULL) {  n++;  p = p->next;  }  return n;  }  int chain::get(int theIndex) const  {  // 移动到所需要的节点  chainNode\* currentNode = firstNode;  for (int i = 0; i < theIndex; i++)  currentNode = currentNode->next;  return currentNode->element;  }  //返回指定元素首次出现的索引  int chain::indexOf(const int theElement) const  {  //返回该元素第一次出现时的索引  //如该元素不存在不，返回-1  //搜寻链表寻找该元素  chainNode\* currentNode = firstNode;  int index = 0; // 当前节点的索引  while (currentNode != NULL && currentNode->element != theElement)  {  // 移动到下一个节点  currentNode = currentNode->next;  index++;  }  // 判断我们是否找到了匹配的节点  if (currentNode == NULL)  return -1;  else  return index;  }  // 删除指定位置上的元素  void chain::erase(int theIndex)  {  //元素存在，寻找要删除元素的节点  chainNode \*deleteNode;  if (theIndex == 0)  {  // 如果删除的是首节点  deleteNode = firstNode;  firstNode = firstNode->next;  }  else  {  //若删除的不是首节点，则用指针p指向要删除节点的前驱节点  chainNode \*p = firstNode;  for (int i = 0; i < theIndex - 1; i++)  p = p->next;  deleteNode = p->next;  //删除deleteNode指向的节点  p->next = p->next->next;  }  listSize--;  delete deleteNode;  }  void chain::insert(int theIndex, const int theElement)  {  // 插入元素到index位置  if (theIndex < 0 || theIndex >size() + 1)  {// 无用的下标  cout << "输入位置错误！" << endl;  exit(EXIT\_FAILURE);  }  if (theIndex == 0)  // 在链表头插入  firstNode = new chainNode(theElement, firstNode);  else  {  chainNode \*p, \*t;  t = firstNode;  p = new chainNode;  p->element = theElement;  p->next = NULL;  while (theIndex-- > 1)  t = t->next;  p->next = t->next;  t->next = p;  }  }  //输出链表  void chain::output(ostream &out) const  {  // 将链表放入输出流  for (chainNode\* currentNode = firstNode;  currentNode != NULL;  currentNode = currentNode->next)  out << currentNode->element << " ";  }  //重载"<<"  template <class T>  ostream &operator<<(ostream &out, const int x)  {  x.output(out);  return out;  }  //插入方法  chain& chain::add(const int& x)  {  //在链表头进行链表的添加操作  chainNode \*p = new chainNode;  p->element = x;  p->next = firstNode;  firstNode = p;  return \*this;  }  源.cpp  #include "chainIterator.h"  #include "chainNode.h"  #include "chain.h"  #include "merge.h"  #include <stdlib.h>  #include <iostream>  #include <string>  using namespace std;  int main()  {  //接收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,并输出链表内容。  int x;  cout<<"数据结构实验三：线性表操作"<<endl;  cout << "请输入一系列整数作为链表节点的元素值(按0回车后结束输出):" << endl;  chain \*chain1 = new chain();  while (cin >> x)  {  if (x != 0)  chain1->add(x);  else  break;  }  cin.clear();  cout << "链表内容为:" << endl;  chain1->output(cout);//输出链表1\*/  cout << endl;  //自表首插入元素  int m, n;  cout << "请输入要插入的位置m以及元素n" << endl;  cin >> m >> n;  chain1->insert(m, n);  cout << "新链表为：" << endl;  chain1->output(cout);  cout << endl;    //删除指定元素  int t;  cout << "请输入要删除的元素索引" << endl;  cin >> t ;  chain1->erase(t);  cout << "剩余的链表为：" << endl;  chain1->output(cout);  cout << endl;  //搜索表中是否有指定元素  int a;  cout << "请输入要寻找的元素" << endl;  cin >> a;  int index1 = chain1->indexOf(a);  if (index1 < 0)  cout << "输入的元素没找到！" << endl;  else  cout << "输入的元素在表中" << endl;  cout << endl;  //输入一个整数（例33），在链表中进行搜索，输出其在链表中的位置，如果不存在输出0。  int b;  cout << "请输入要寻找的元素" << endl;  cin >> b;  int index2 = chain1->indexOf(b);  if (index2 < 0)  cout << "0" << endl;  else  cout << "输入的元素索引为：" << index2 << endl;  cout << endl;  //使用链表遍历器实现链表的反序输出  cout << "正序输出为: " << endl;  chain1->output(cout);  cout << endl;  cout << "反序输出为: " << endl;  chainIterator \*iterator = new chainIterator();//建立链表遍历器，遍历链表1  int \*z;  z = iterator->initialize(\*chain1);  chain \*chain2 = new chain();//建立反向链表  while (z)  {  //将链表1中的元素放入反向链表  chain2->add(\*z);  z = iterator->Next();  }  chain2->output(cout);//输出反向链表  cout << endl;  //创建两个有序链表，使用链表遍历器实现链表的合并。  cout << "链表1为: " << endl;  chainNode \*firstNode1 = sequentialList1();  printList(firstNode1);  cout << "链表2为: " << endl;  chainNode \*firstNode2 = sequentialList2();  printList(firstNode2);  chainNode \*firstNode = merge(firstNode1, firstNode2);  cout << "合并后的链表为: " << endl;  printList(firstNode);  freeList(firstNode);  system("pause");  return 0;  } | | | |
| 结论分析与体会：  运行结果： | | | |