山东大学软件学院

数据结构、算法与应用课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201800301249 | 姓名： 王帅 | | 班级： 4班 |
| 实验题目：实验三 线性表操作 | | | |
| 实验学时：4 | | 实验日期： 2019.10.10 | |
| 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：  PC机 | | | |
| 软件环境：  Microsoft Visual C++ | | | |
| 实验步骤与内容：   * 1. 输入 n 个不为零的整数作为节点元素值，遇到 0 代表输入结束（不创建元素值为 0 的节点），创建有序链表。输出整个链表。   2. 输入一个整数，将该数插入到有序表相应位置。输出整个链表。   3. 输入一个整数，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。   4. 再一次输入一个整数，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. 再一次输入 n 个不为零的整数作为节点元素值，遇到 0 代表输入结束   （不创建元素值为 0 的节点），创建有序链表。输出整个链表。   * 1. 使用链表遍历器实现上面两个有序链表的合并，输出合并后的链表。   2. 提示：注意单节点链表的测试。   实验代码如下：  #include <iostream>  using namespace std;  /\*  1、输入n个不为零的整数作为节点元素值，遇到0代表输入结束（不创建元素值为0的节点），创建有序链表。输出整个链表。  2、输入一个整数，将该数插入到有序表相应位置。输出整个链表。 insert,output  3、输入一个整数，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。 indexof  4、再一次输入一个整数，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。 indexof  5、再一次输入n个不为零的整数作为节点元素值，遇到0代表输入结束（不创建元素值为0的节点），创建有序链表。输出整个链表。  6、使用链表遍历器实现上面两个有序链表的合并，输出合并后的链表。 merge  7、提示：注意单节点链表的测试。  \*/  struct chainNode //定义节点结构  {  //数据成员  int element;  chainNode \*next;    //方法  chainNode() {}  chainNode(int e) {element = e;}  chainNode(int e,chainNode \*n)  {  element = e;  next = n;  }  };  class chain //链类  {  public:  //构造函数  chain()  {  firstNode = NULL;  }    //复制构造函数  chain(chain& list)  {  if(list.firstNode = NULL) {firstNode = NULL; return;}  chainNode\* sourceNode = list.firstNode;  firstNode = new chainNode(sourceNode->element,firstNode);  sourceNode = sourceNode->next;  chainNode\* targetNode = firstNode;  while (sourceNode!=NULL)  {  targetNode->next = new chainNode(sourceNode->element,firstNode);  targetNode = targetNode->next;  sourceNode = sourceNode->next;  }  targetNode->next = NULL;  }    //析构函数  ~chain()  {  while(firstNode != NULL)  {  chainNode\* n = firstNode;  firstNode = firstNode->next;  delete n;  }  }    //在指定位置插入一个节点的函数  void insert(int index,int& element)  {  if(index<0)  {  cout<<"索引值小于零";  }    if(index==0)  {  firstNode = new chainNode(element,firstNode);  }  else  {  chainNode\* n = firstNode;  for(int i=0;i<index-1;i++) {n = n->next;}  //节点应插入到n现在指向的节点的后边  n->next = new chainNode(element,n->next);  }  }    //在有序链表中插入一个数  void sortedInsert(int e)  {  if(firstNode==NULL||e<firstNode->element) {firstNode = new chainNode(e,firstNode);}  else  {  chainNode\* p = firstNode;  while(p->next != NULL)  {  if(p->next->element > e) {p->next = new chainNode(e,p->next); break;}  p = p->next;  }  if(p->next == NULL) {p->next = new chainNode(e,p->next);}  }  }    //输出一个整数在链表中的位置  int indexof(int targetElement)  {  chainNode\* n = firstNode;  int index = 0; //当前节点的索引值  while(n!=NULL && n->element!=targetElement)  {  n = n->next;  index++;  }  if(n==NULL) {return 0;} //n指向空节点时，说明未找到  else {return index+1;}  }    //删除指定索引值的节点  void erase(int index)  {  chainNode\* deleteNode;  if(index==0)  {  deleteNode = firstNode;  firstNode = firstNode->next;  }  else  {  chainNode\* p = firstNode;  for(int i=0;i<index-1;i++) {p = p->next;}  deleteNode = p->next;  p = p->next->next;  }  delete deleteNode;  listSize--;  }    //输出链表  void output()  {  chainNode\* n = firstNode;  while(n!=NULL&&n->next!=NULL)  {  cout<<n->element<<",";  n = n->next;  }  cout<<n->element;  }    //合并  void merge(chain& list,chain& list1)  {  chainNode\* p = list.firstNode;  while(p!=NULL)  {  this->sortedInsert(p->element);  p = p->next;  }  p = list1.firstNode;  while(p!=NULL)  {  this->sortedInsert(p->element);  p = p->next;  }  this->output();  }    protected:  chainNode \*firstNode;  int listSize;  };    int main()  {  cout<<"Input1"<<endl;  int n,m;  chain list,list1;  cin>>n;  while(n!=0)  {  list.sortedInsert(n);  cin>>n;  }  cout<<"Output1"<<endl;  list.output();  cout<<endl;  cout<<"Input2"<<endl;  cin>>n;  list.sortedInsert(n);  cout<<"Output2"<<endl;  list.output();  cout<<endl;    cout<<"Input3"<<endl;  cin>>n;  m = list.indexof(n);  cout<<"Output3"<<endl;  cout<<m<<endl;    cout<<"Input4"<<endl;  cin>>n;  m = list.indexof(n);  cout<<"Output4"<<endl;  cout<<m<<endl;    cout<<"Input5"<<endl;  cin>>n;  while(n!=0)  {  list1.sortedInsert(n);  cin>>n;  }  cout<<"Output5"<<endl;  list1.output();  cout<<endl;  chain list2;  list2.merge(list,list1);  cout<<endl;  cout<<"End";    return 0;  } | | | |
| 结论分析与体会：  线性表是后边许多数据结构的基础，必须掌握。 | | | |