山东大学 软件 学院

数据结构 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201400301034 | 姓名： 石兴帮 | | 班级： 软件八班 |
| 实验题目：线性表操作 | | | |
| 实验学时：4h | | 实验日期： 2015-12-13 | |
| 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：  MacBook Pro  OS X Yosemite 10.10.3 | | | |
| 软件环境：  Xcode 6.4 | | | |
| 实验内容与设计：   1. 实验内容（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链表。  2.提供操作：自表首插入元素，删除制定元素，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证书作为节点的元素值，输出链表内容。  4.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.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。  2.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）  插入：当前位置之后的元素向后一位，然后把元素插进去。  删除：那个位置之后的元素向前移一位。  搜索：遍历就行。  输出：遍历就行。  反序：创建三个指针，两个用来一前一后遍历，还有一个用来改方向。  合并：两个链表，不断取出最小的，放入新链表。  3.测试结果（测试输入，测试输出）    **Input sum of your nodelist:**  3  **现在输入元素值吧~**  2 1 3  **2  1  3**  **输入要删除的元素**  1  **2  3**  **3  2  3**  **input thevalue ~**  2  **1**    **请输入第一个有序数组的长度：**  4  **请输入第一个有序数组的元素：**  1 2 3 4  **请输入第2个有序数组的长度：**  3  **请输入第2个有序数组的元素：**  3 4 5  1  2  3  3  4  4  5    4.实现源代码（程序风格清晰易理解，有充分的注释）    #include<iostream>  using namespace std;  //链表节点的定义  template <class T>  struct chainNode  {      T element;      chainNode<T> \*next;      chainNode(){}      chainNode(const T& element ){          this->element = element;      }      chainNode(const T& element, chainNode<T>\* next)      {          this->element = element;          this->next = next;      }  };  //链表的定义  template<class T>  class chain{  public:    chain(int initial = 10);    chain(const chain<T>& x);    ~chain();      bool empty()const{return listSize == 0;}    int size() const{ return listSize; }    T& get(int theIndex) const;    int indexOf(const T& theElement);    void add(T& theElement);    void eraser(int theIndex);    void push\_back(const T& theElement);    void startinsert(const T& theElement);    void insert(int theIndex, const T& theElement);    void output(ostream& out) const;    void eraser(T &theElement);    void reverseprint();    void print();    void hebing();      void shanchu(T ele);  protected:      void checkIndex(int theIndex) const;      chainNode<T>\* firstNode;      int listSize;  };  template<class T>  void chain<T>::shanchu(T ele)  {      chainNode<T> \*p = firstNode;      chainNode<T> \*q = p;      p=p->next;      if (q->element == ele) {          firstNode = p;          return;      }      else          while (q ) {                if (p->element == ele) {                  q->next = p->next;                  //            firstNode = q;                  return;              }              p=p->next;              q=q->next;          }      return;  }    template<class T>  chain<T>::chain(int initial){      if (initial<1)      {          cout << "Error!";          // break;      }      firstNode = NULL;      listSize = 0;  }    //复制构造函数  template<class T>  chain<T>::chain(const chain<T>& theList){      listSize = theList.listSize;      if (listSize == 0){          firstNode = NULL;          return;      }      chainNode<T>\* sourceNode = theList.firstNode;      firstNode = new chainNode<T>(sourceNode->element);      sourceNode = sourceNode->next;      chainNode<T>\* targetNode = firstNode;      while (sourceNode != NULL){          targetNode->next = new chainNode<T>(sourceNode->element);          targetNode = targetNode->next;          sourceNode = sourceNode->next;      }      targetNode->next = NULL;//链表结果。  }    template <class T>//析构函数  chain<T>::~chain(){      while (firstNode != NULL){          chainNode<T>\* nextNode = firstNode->next;          delete firstNode; firstNode = nextNode;      }  }    template<class T>  void chain<T>::add(T &theElement){      if (firstNode == NULL){          firstNode = new chainNode<T>(theElement);      }      else      {          chainNode<T> \* p = firstNode;          chainNode<T> \* q = p->next;          while (q != NULL){              p = q;              q = q->next;          }          p ->next= new chainNode<T>(theElement);          listSize += 1;      }  }    template<class T>  T& chain<T>::get(int theIndex) const  {      checkIndex(theIndex);      chainNode<T>\* currentNode = firstNode;      for (int i = 0; i < theIndex; i++){          currentNode = currentNode->next;      }return currentNode->element;  }    template<class T>//搜索表中是否有指定元素，没有返回-1，有返回位置索引  int chain<T>::indexOf(const T& theElement){      chainNode<T>\* currentNode = firstNode;      int index = 1;      while (currentNode != NULL&&currentNode->element != theElement){          currentNode = currentNode->next;          index++;      }      if (currentNode == NULL)          return -1;      else return index;  }    template<class T>//删除指定索引的元素  void chain<T>::eraser(int theIndex){        chainNode<T>\* deleteNode;      if (theIndex == 0){          deleteNode = firstNode;          firstNode = firstNode->next;      }      else      {          chainNode<T>\* p = firstNode;          for (int i= 0; i < theIndex - 1; i++)              p = p->next;          deleteNode = p->next;          p->next = p->next->next;      }      listSize--;      delete deleteNode;  }    template<class T>//删除指定元素  void chain<T>::eraser(T &myElement){      int location = get(myElement);      if (location == -1) return;      eraser(location);  }    template <class T>//首位插入元素  void chain<T>::startinsert(const T &theElement){        if (firstNode==NULL)          firstNode = new chainNode<T>(theElement);      else      {     chainNode<T> current= new chainNode<T>(theElement);          current->next = firstNode;          firstNode = current;      }      listSize++;  }  template <class T>//插入元素  void chain<T>::insert(int theIndex,const T& theElement){      if (theIndex == 0)          firstNode = new chainNode<T>(theElement, firstNode);      else{          chainNode<T>\* p = firstNode;          for (int i = 0; i < theIndex - 1; i++){              p = p->next;          }          p->next = new chainNode<T>(theElement, p->next);      }      listSize += 1;  }      template<class T>//输出整个链表  void chain<T>::output(ostream& out) const{      for (chainNode < T > current = firstNode; current != NULL; current = current->next){          out << current->element << " ";      }  }  /\*template <class T>   ostream operator<<(ostream& out, const chain<T>& x){  x.output(out);  return out;   }\*/  template<class T>  void chain<T>::print(){      for (chainNode < T > \*current =this-> firstNode; current != NULL; current = current->next){          cout << current->element<< "  ";        } cout << endl;  }  template<class T>  void chain<T>::reverseprint(){      chainNode<T> \* p1= firstNode;      chainNode<T> \* p3 ;      chainNode<T> \* p2=p1->next;      while (p2 !=NULL)      {          p3 = p2->next;          p2->next = p1;          p1 = p2;          p2 = p3;      }        for (int i = 0; i < listSize;i++){          cout << p1->element << "  ";          p1= p1->next;      }  }    template<class T>  void chain<T>::hebing(){      chain<T>  A;      chain<T>  B;      int num1, num2,size;      int value1,value2;      cout << "请输入第一个有序数组的长度：" << endl;      cin >> num1;      cout << "请输入第一个有序数组的元素：" << endl;      for (int i = 0; i < num1; i++){          cin >> value1;          A.insert(i, value1);      }      cout << "请输入第2个有序数组的长度：" << endl;      cin >> num2;      cout << "请输入第2个有序数组的元素：" << endl;      for (int i = 0; i < num2; i++){          cin >> value2;          B.insert(i, value2);      }      size = num1 + num2;      chain<T> C;        for (int k = 1; k <= size; k++)      {          if (A.firstNode == NULL) {              C.insert(k - 1, B.firstNode->element);              B.firstNode = B.firstNode->next;              continue;          }          if (B.firstNode == NULL) {              C.insert(k - 1, A.firstNode->element);              A.firstNode = A.firstNode->next;              continue;          }          if (A.firstNode->element< B.firstNode->element) {              C.insert(k - 1, A.firstNode->element);              A.firstNode = A.firstNode->next;          }          else {              C.insert(k - 1, B.firstNode->element);              B.firstNode = B.firstNode->next;          }      }      C.print();  }  //#include "shiyan3.h"    //int main(){  //    chain<int>  \*mychain;  //    mychain= new chain<int>[100];  //    cout << "Input sum of your nodelist:" << endl;  //    int num = 0;  //    int value=0,forvalue,result;  //    cin >> num;  //    cout << "现在输入元素值吧~" << endl;  //    for (int i = 0; i < num; i++){  //        cin >> value;  //        mychain->insert(i, value);  //    }  //    mychain->print();  //    int yuansu;  //    cout << "输入要删除的元素" << endl;  //    cin >> yuansu;  //    mychain->shanchu(yuansu);  //    mychain->print();  //  //        mychain->reverseprint();  //        cout << endl;  //        cout << "input thevalue ~" << endl;  //  //        cin >> forvalue;  //        result=mychain->indexOf(forvalue);  //        cout << result << endl;  //        cout << endl;  //        mychain->hebing();  //    return 0;  //} | | | |
| 结论分析与体会：  加深了对线性表的理解。 | | | |