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17.3 |
| 实验题目：数组描述线性表 | | | |
| 实验学时：2 | | 实验日期： 2018.10.13 | |
| 实验目的：   1. 掌握线性表的结构、数组描述方法（顺序存储结构）、数组描述线性表的实现。 2. 掌握线性表应用。 | | | |
| 软件环境：  Windows 10  Dev-C++ 5.11 | | | |
| 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数组描述，提供操作：插入、删除、查找等。  （2）设通讯录中每一个联系人的内容有：姓名、电话号码、班级、宿舍。由键盘输入或文件录入的通讯录信息建立通讯录表，使用线性表中操作实现通讯录管理功能，包括：插入、删除、编辑、查找（按姓名查找）；键盘输入一班级，输出通讯录中该班级所有人信息。  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）  首先创建一个模板抽象类线性表作为基础，再由它派生出一个模板数组描述线性表类，并写上在虚基类中所列虚函数的具体实现，为了满足实验的特殊要求，需要把联系人设计为一个类，然后再数组线性表类中建立一个联系人数组，为此需要针对联系人的特点对数组线性表的成员函数作特殊修改。   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  如图1所示，先用键盘输入预设的数据条数，然后输入数据的内容，接下来根据菜单的提示选择要使用的功能，首先我们选择插入操作，插入了一组新数据，而插入后的联系人列表也显示正常。接下来我们执行查询操作（图2），将联系人列表中的索引为3（第4位）的联系人删除，执行完删除操作的列表正常。然后执行编辑操作（图3），先用键盘输入需要修改的联系人姓名找到该条数据，接着重新进行编辑即可。继续执行查找操作（图4），用键盘输入需要查找的联系人姓名，关于此人的所有信息会被显示，若查无此人，会返回错误信息。继续执行按班级搜索操作，输入班级信息找出属于该班级的所有人的信息。最后输入非“continue”的字符串，跳出菜单。   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. 在重载模板类arraylist的运算符<<时遇到了问题，报出“undefined reference to `operator<<(std::ostream&, Arraylist<person> const&)'”的错误。经过查找资料发现，模板类的友元函数比较特殊，如果在类内声明，类外实现，就会报连接错误。原因是编译器将友元函数实现部分的代码当成一个新定义的函数模板。所以一种解决方法是把友元函数的声明和定义全部写在类内。   （2）在编写Arraylist的构造函数是出现了这样的错误：经过分析，发现是之前为数组element开辟了新的内存空间（这里已经进行过一次初始化），又试图调用element数组元素的构造函数引起的。所以这里的赋值操作需要借助重载赋值运算符完成。   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）   #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>  class Arraylist :public linearlist<T>  {  public:  Arraylist(int initialCapacity=10)  {//不考虑智障用户输负数  element=new T [initialCapacity]; //这里已经初始化了一遍。  arrayLength=initialCapacity;  listSize=0;  string name,tel\_num,Class,dor\_room;  cout<<"请输入"<<initialCapacity<<"条数据"<<endl;  for(int i=0;i<initialCapacity;i++)  {  cin>>name>>tel\_num>>Class>>dor\_room;  T temp(name,tel\_num,Class,dor\_room);  element[i]=temp; //这里应该调用复制构造函数 ,瓜皮没法调用？！？！  }  listSize=initialCapacity;  }    Arraylist(const Arraylist<T>& A)  {  listSize=A.listSize;  arrayLength=A.arrayLength;  element=new T [arrayLength];  for(int i=0;i<listSize;i++)  {  element[i]=A.element[i];  }  }  ~Arraylist() { delete [] element; }    //ADT方法  bool empty() const  { return listSize==0; }  int size() const  { return listSize; }  T& get(int theIndex) const  {//实现随机存储，效率极高 O(1)  checkIndex(theIndex);  return element[theIndex];  }  int indexOf(T& x) const  {//返回元素x第一次出现时的索引;若不存在，则返回-1  for(int i=0;i<listSize;i++)  {  if(x==element[i]) return i;  }  return -1;  }  T& search(string str) const  {//如果找到了对应的人(str是人的名字)，则返回元组，否则返回空值  for(int i=0;i<listSize;i++)  {  if(str==element[i].name) return element[i];  }  cerr<<"未找到对应数据"<<endl;  }  void modify(string str)  {//根据姓名来找到需要修改的元组  for(int i=0;i<listSize;i++)  {  if(str==element[i].name)  {  cout<<"找到该条记录!"<<endl<<"请输入新的数据（按姓名、电话号码、班级、宿舍修改）:"<<endl;  cin>>element[i].name>>element[i].tel\_num>>element[i].Class>>element[i].dor\_room;  return;  }  }  cout<<"未找到该条记录"<<endl;  }  void erase(int theIndex)  {//从表中删除索引为theIndex的元素  for(int i=theIndex;i<listSize-1;i++)  {  element[i]=element[i+1];  }  //cout<<"right!"<<endl;  element[--listSize].~T(); //调用析构函数。？？verstehen. 单个类对象的析构，并不是动态调用  }  void insert(int theIndex,const T& theElement)  {  if(theIndex<0||theIndex>listSize)  cerr<<"请输入正确索引！"<<endl;    if(listSize==arrayLength)  {//数组空间已满  T \*p=new T[2\*arrayLength];  for(int i=0;i<listSize;i++)  {  p[i]=element[i];  }  delete [] element;  element=p;  arrayLength\*=2;  }  cout<<"listSize is "<<listSize<<endl;  for(int i=listSize-1;i>=theIndex;i--)  {  element[i+1]=element[i]; //第二次调用插入函数错误出在这里,注意数组下标是从0开始的  }  element[theIndex]=theElement;  listSize++;  }  void output() const  {//输出该线性表  for(int i=0;i<listSize;i++)  {  cout<<element[i];  }  }  void class\_output() const  {//输出该线性表  cout<<"请输入想要查找的班级信息"<<endl;  string Class;  cin>>Class;  for(int i=0;i<listSize;i++)  {  if(element[i].Class==Class)  {  cout<<"以下是复合条件的数据："<<endl;  cout<<element[i];  }  }  }  //其他方法  int capacity() const  { return arrayLength; }  friend ostream& operator<< (ostream& out,const Arraylist<T> &A) //模板类的友元函数声明与输出不能分离  {  for(int i=0;i<A.listSize;i++)  { out<<A.element[i]; }  return out;  }    protected:  void checkIndex(int theIndex) const  {//确定索引的有效性，防止越界  if(theIndex<0||theIndex>=listSize)  {  cerr<<"请检查索引的有效性！"<<endl;  }  }  //若索引无效，则抛出异常  T \*element; //存储线性表元素的一维数组  int arrayLength; //一维数组的容量  int listSize; //线性表的元素个数    };  class person  {  public:  string name,tel\_num,Class,dor\_room;  person() { }  person(string name1,string tel\_num1,string Class1,string dor\_room1)  {  name=name1; tel\_num=tel\_num1; Class=Class1; dor\_room=dor\_room1;  }  person(const person& T)  {  name=T.name; tel\_num=T.tel\_num; Class=T.Class; dor\_room=T.dor\_room;  }  ~person()  {//这里也把string变量也自动析构了么？  }    friend ostream& operator<<(ostream& out,const person &A);  bool operator ==(const person &A)  {  if(name==A.name)  {  if(tel\_num==A.tel\_num)  {  if(Class==A.Class)  {  if(dor\_room==A.dor\_room)  return true;  }  }  }  return false;  }  person& operator \*()  {  return \*this;  }  person& operator=(const person& T)  {  name=T.name; tel\_num=T.tel\_num; Class=T.Class; dor\_room=T.dor\_room;  }  };  ostream& operator<<(ostream& out,const person &A)  {  out<<A.name<<"\t"<<A.tel\_num<<"\t"<<A.Class<<"\t"<<A.dor\_room<<endl;  return out;  }  int main()  {  int n; cout<<"请输入数据的条数n"<<endl;  cin>>n;  Arraylist<person> C(n);    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<<"执行编辑操作请输入3"<<endl;  cout<<"执行查找操作请输入4"<<endl<<"执行班级搜索请输入5"<<endl;  cin>>flag;  switch(flag)  {  case 1: {  cout<<"请按顺序输入要插入的元组数据"<<endl;  string name,tel\_num,Class,dor\_room;  cin>>name>>tel\_num>>Class>>dor\_room;  person A(name,tel\_num,Class,dor\_room);  cout<<"请输入要插入的位置索引(从0开始)"<<endl;  int pos; cin>>pos;  C.insert(pos,A); cout<<C<<endl;  break;  }  case 2: {  cout<<"请输入要删除元组索引(从0开始)"<<endl;  int pos; cin>>pos;  C.erase(pos); cout<<C<<endl;  break;  }  case 3: {  cout<<"请输入要修改的联系人姓名"<<endl;  string name; cin>>name;  C.modify(name); cout<<C;  break;  }  case 4: {  cout<<"请输入要查找的联系人姓名"<<endl;  string name; cin>>name;  cout<<C.search(name)<<endl;  break;  }  case 5: {  C.class\_output();  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;  }  return 0;  } | | | | |