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：201705130113 | 姓名： 黄瑞哲 | | 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：2018.10.11 | |
| 实验目的：   1. 掌握线性表结构、数据描述方法（顺序存储结构）、数组描述线性表的实现 2. 掌握线性表的应用 | | | |
| 软件环境：  G++ 8.1.0  Visual Studio Community 2017 | | | |
| 1. 实验内容（题目内容，输入要求，输出要求） 2. 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 3. 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 4. 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） 5. 创建结构体Info表示每个联系人的信息，内有string类型的成员name、phone、cla、dorm分别记录联系人的姓名电话班级和宿舍。 6. 创建通讯录类TeleBook来实现要求操作，该类以线性表为保护成员，线性表的类型为Info 7. 程序开始从contacts.txt读入通讯录信息，程序结束时将TeleBook中的联系人信息全部写回contacts.txt 8. 测试结果（测试输入，测试输出，结果分析）   程序运行前contacts.txt中的内容    程序运行中    程序结束后contacts.txt中的内容     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. 使用类的组合时要注意完善类的构造函数，不能使用默认的构造函数、一定要完善复制构造函数、析构函数和等于号的重载，不然会出现Runtime Error。 3. 使用文件流读写要注意在结束后调用close函数来关闭文件流 4. 类似实验最好将代码分离，arrayList的声明定义放在一个文件中，TeleBook的声明和定义分别放在两个文件中，main函数的主要部分单独放在一个文件中。这样做有利于理清思路，避免代码量过大导致debug困难。 5. 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）   /\*arrayList.h\*/  #ifndef \_\_ARRAYLIST\_H\_  #define \_\_ARRAYLIST\_H\_  #include "linearList.h"  #include <stdexcept>  #include <iostream>  using namespace std;  template<typename T>  class arrayList  :virtual public linearList<T>  {  public:  arrayList(int = 10);  arrayList(const arrayList<T>&);  ~arrayList();  bool empty() const;  int size() const;  int find(const T&) const;  void erase(int);  void insert(int, const T&);  void clear();  void push\_back(const T&);  arrayList<T>& operator=(const arrayList<T>&);  T& operator[](int);  const T& operator[](int) const;  class iterator;  class const\_iterator;  const\_iterator begin() const { return const\_iterator(element); }  const\_iterator end() const { return const\_iterator(element + listSize); }  iterator begin() { return iterator(element); }  iterator end() { return iterator(element + listSize); }  class iterator  {  public:  typedef bidirectional\_iterator\_tag iterator\_category;  typedef T value\_type;  typedef ptrdiff\_t difference\_type;  typedef T\* pointer;  typedef T& reference;  iterator(T\* thePosition = nullptr) :position(thePosition) {}  T& operator\*() { return \*position; }  T\* operator->() { return &\*position; }  iterator& operator++()  {  ++position;  return \*this;  }  iterator operator++(int)  {  iterator old = \*this;  ++position;  return old;  }  iterator& operator--()  {  --position;  return \*this;  }  iterator operator--(int)  {  iterator old = \*this;  --position;  return old;  }  bool operator!=(const iterator right) const { return position != right.position; }  bool operator==(const iterator right) const { return position == right.position; }  protected:  T\* position;  };  class const\_iterator  {  public:  typedef bidirectional\_iterator\_tag iterator\_category;  typedef T value\_type;  typedef ptrdiff\_t difference\_type;  typedef T\* pointer;  typedef T& reference;  const\_iterator(T\* thePosition = nullptr) :position(thePosition) {};  const T& operator\*() { return \*position; }  const T\* operator->() { return &\*position; }  const\_iterator& operator++()  {  ++position;  return \*this;  }  const\_iterator operator++(int)  {  const\_iterator old = \*this;  ++position;  return old;  }  const\_iterator& operator--()  {  --position;  return \*this;  }  const\_iterator operator--(int)  {  const\_iterator old = \*this;  --position;  return old;  }  bool operator==(const const\_iterator right) const { return position == right.position; }  bool operator!=(const const\_iterator right) const { return position != right.position; }  protected:  T\* position;  };  protected:  int listSize;  int arrayLength;  T\* element;  void checkIndex(int theIndex) const;  };  template<typename T>  arrayList<T>::arrayList(int initialCapacity)  {  if (initialCapacity < 1)  throw out\_of\_range("the initial Capacity of arrayList mut > 0");  arrayLength = initialCapacity;  element = new T[arrayLength];  listSize = 0;  }  template<typename T>  arrayList<T>::arrayList(const arrayList<T>& a)  {  arrayLength = a.arrayLength;  listSize = a.listSize;  element = new T[arrayLength];  copy(a.element, a.element + a.listSize, element);  }  template<typename T>  arrayList<T>::~arrayList() { delete[] element; }  template<typename T>  bool arrayList<T>::empty() const { return listSize == 0; }  template<typename T>  int arrayList<T>::size() const { return listSize; }  template<typename T>  int arrayList<T>::find(const T& theElement) const  {  int i;  for (i = 0; i < listSize && element[i] != theElement; ++i);  if (i == listSize) return -1;  return i;  }  template<typename T>  void arrayList<T>::erase(int theIndex)  {  checkIndex(theIndex);  copy(element + theIndex + 1, element + listSize, element + theIndex);  element[--listSize].~T();  }  template<typename T>  void arrayList<T>::insert(int theIndex, const T& theElement)  {  if (theIndex < 0 || theIndex > listSize)  throw out\_of\_range("illegalIndex");  if (listSize == arrayLength)  {  arrayLength \*= 2;  T \*tmp = new T[arrayLength];  copy(element, element + listSize, tmp);  delete[] element;  element = tmp;  }  for (int i = theIndex + 1; i <= listSize; ++i) element[i] = element[i - 1];  element[theIndex] = theElement;  listSize++;  }  template<typename T>  void arrayList<T>::clear()  {  delete[]element;  element = new T[arrayLength];  listSize = 0;  }  template<typename T>  void arrayList<T>::push\_back(const T& theElement)  {  if (listSize == arrayLength)  {  arrayLength \*= 2;  T\* tmp = new T[arrayLength];  copy(element, element + listSize, tmp);  delete[]element;  element = tmp;  }  element[listSize++] = theElement;  }  template<typename T>  T& arrayList<T>::operator[](int theIndex)  {  checkIndex(theIndex);  return element[theIndex];  }  template<typename T>  arrayList<T>& arrayList<T>::operator=(const arrayList<T>& a)  {  if (this == &a) return \*this;  listSize = a.listSize;  arrayLength = a.arrayLength;  delete[]element;  element = new T[arrayLength];  copy(a.element, a.element + listSize, element);  return \*this;  }  template<typename T>  const T& arrayList<T>::operator[](int theIndex) const  {  checkIndex(theIndex);  return element[theIndex];  }  template<typename T>  void arrayList<T>::checkIndex(int theIndex) const  {  if (theIndex < 0 || theIndex >= listSize)  throw out\_of\_range("the index is out of range!");  }  #endif //\_\_ARRAYLIST\_H\_  /\*TeleBook.h\*/  #ifndef \_\_TELEBOOK\_H\_  #define \_\_TELEBOOK\_H\_  #include <iostream>  #include <string>  #include <stdexcept>  #include "arrayList.h"  using namespace std;  struct Info  {  string name;  string phone;  string cla;  string dorm;  bool operator==(const Info& c) const  {  if (name == c.name && phone == c.phone && cla == c.cla && dorm == c.dorm)  return true;  else return false;  }  bool operator!=(const Info& c) const  {  return !(\*this == c);  }  friend istream& operator>>(istream& in, Info& c)  {  in >> c.name >> c.phone >> c.cla >> c.dorm;  return in;  }  friend ostream& operator<<(ostream& out, const Info& c)  {  out << c.name << ' ' << c.phone << ' ' << c.cla << ' ' << c.dorm;  return out;  }  };  /\*\*  \* name: student's name  \* phone: student's phone  \* cla: student's class  \* dorm: student's dormitory  \*/  class TeleBook  {  protected:  arrayList<Info> contacts;  public:  TeleBook(int initialCapacity = 10): contacts(initialCapacity) {};  TeleBook(const TeleBook& con): contacts(con.contacts) {};  // insert information of contact to the telebook  int size() const {return contacts.size();}  void insert(const Info& contact) {contacts.push\_back(contact);};  void erase(const string&);  void edit(const string&, const Info&);  Info find(const string&) const;  arrayList<Info> find\_by\_class(const string&) const;  Info& operator[](int index) {return contacts[index];}  const Info& operator[](int index) const {return contacts[index];}  };  #endif  /\*TeleBook.cpp\*/  #include "TeleBook.h"  /\*\*  \* delete a contact's information from telebook  \* @param contact the information of contact you want to delete  \*/  void TeleBook::erase(const string& name)  {  int i;  for (i = 0; i < contacts.size(); ++i)  if (contacts[i].name == name)  break;  if (i == contacts.size())  throw domain\_error("No such contact named " + name);  contacts.erase(i);  }  /\*\*  \* edit the information  \* @param name the contact's name you want to edit  \* @param newInfo new information of contact  \*/  void TeleBook::edit(const string& name, const Info& newInfo)  {  bool find = false;  for (int i = 0; i < contacts.size(); ++i)  if (contacts[i].name == name)  {  contacts[i] = newInfo;  find = true;  break;  }  if (!find) throw domain\_error("No such contact " + name);  }  /\*\*  \* find contact by name  \* @param name contact's name you want to find  \* @return information of the person you want  \*/  Info TeleBook::find(const string& name) const  {  for (int i = 0; i < contacts.size(); ++i)  if (contacts[i].name == name)  return contacts[i];  throw domain\_error("No such contact " + name);  return Info();  }  /\*\*  \* find all people's information  \*/  arrayList<Info> TeleBook::find\_by\_class(const string& cla) const  {  arrayList<Info> res;  for (int i = 0; i < contacts.size(); ++i)  if (contacts[i].cla == cla)  res.push\_back(contacts[i]);  return res;  }  /\*main.cpp\*/  #include <iostream>  #include <fstream>  #include <string>  #include "arrayList.h"  #include "TeleBook.h"  using namespace std;  int main()  {  TeleBook t;  ifstream fin;  fin.open("contacts.txt");  if (!fin.is\_open()) exit(-1);  while (fin.good())  {  struct Info c;  fin >> c;  t.insert(c);  }  fin.close();  cout << "读取到" << t.size() << "个联系人" << endl;  int tp;  do  {  cout << "1.插入\n2.删除\n3.编辑\n4.按姓名查找\n5.按班级查找\n6.显示所有联系人\n0.退出\n";  cin >> tp;  switch (tp)  {  case 1://insert  {  struct Info c;  cout << "请依次输入联系人的姓名 号码 班级 宿舍: " << endl;  cin >> c;  try  {  t.insert(c);  cout << c.name << " 插入成功!" << endl;  }  catch (...)  {  cout << "插入失败" << endl;  }  }  break;  case 2://erase  {  string name;  cout << "请输入被删联系人的姓名: " << endl;  cin >> name;  try  {  t.erase(name);  cout << name << " 删除成功!" << endl;  }  catch (domain\_error& e)  {  cout << e.what() << endl;  }  }  break;  case 3://edit  {  string name;  cout << "请输入被编辑联系人的姓名: " << endl;  cin >> name;  struct Info c;  cout << "请依次输入联系人的姓名 号码 班级 宿舍: " << endl;  cin >> c;  try  {  t.edit(name, c);  cout << name << " 修改成功!" << endl;  }  catch (domain\_error& e)  {  cout << e.what() << endl;  }  }  break;  case 4://find by name  {  string name;  cout << "请输入姓名: " << endl;  cin >> name;  cout << t.find(name) << endl;  }  break;  case 5: // find by class  {  string cla;  cout << "请输入班级: " << endl;  cin >> cla;  arrayList<Info> all = t.find\_by\_class(cla);  for (int i = 0; i < all.size(); ++i)  cout << all[i] << endl;  }  break;  case 6:  {  for (int i = 0; i < t.size(); ++i)  cout << t[i] << endl;  }  break;  default:  break;  }  } while (tp > 0);  ofstream fout("contacts.txt", ios::trunc);  if (fout.is\_open())  {  //save while exiting  for (int i = 0; i < t.size(); ++i)  {  if (i > 0) fout << endl;  fout << t[i];  }  }  fout.close();  return 0;  } | | | |