**附件：**

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：202000130138 | 姓名：宋璎航 | | 班级： 20.3 |
| 实验题目：链式描述线性表 | | | |
| 实验学时：2 | | 实验日期：2021/10/21 | |
| 实验目的：   1. 掌握线性表结构、链式描述方法（链式存储结构）、链表的实现。 2. 掌握链表迭代器的实现与应用。 | | | |
| 软件开发环境：  VSC | | | |
| 1. 实验内容   **题目描述**：  要求封装链表类，链表迭代器类；  链表类需提供操作：在指定位置插入元素，删除指定元素，搜索链表中是否有指定元素，原地逆置链表，输出链表；  不得使用与链表实现相关的STL。  **输入输出格式**：  **输入**：第一行两个整数 N 和 Q。  第二行 N 个整数，作为节点的元素值，创建链表。  接下来 Q 行，执行各个操作，具体格式如下：  插入操作 : 1 idx val，在链表的idx位置插入元素val;  删除操作 : 2 val，删除链表中的 val 元素。若链表中存在多个该元素，仅删除第一个。若该元素不存在，输出 -1；  逆置操作 : 3，原地逆置链表；  查询操作 : 4 val，查询链表中的val元素，并输出其索引。若链表中存在多个该元素，仅输出第一个的索引。若不存在该元素，输出 -1；  输出操作 : 5，使用链表迭代器，输出当前链表索引与元素的异或和。；  **2、题目描述**：  要求使用题目一中实现的链表类，迭代器类完成本题；  不得使用与题目实现相关的STL；  给定两组整数序列，你需要分别创建两个有序链表，使用链表迭代器实现链表的合并，并分别输出这三个有序链表的索引与元素的异或和。  注：给定序列是无序的，你需要首先得到一个有序的链表。  **输入输出格式**：  **输入**：  第一行两个整数 N 和 M；  第二行 N 个整数，代表第一组整数序列；  第三行 M 个整数，代表第二组整数序列。  **输出**：  三行整数。分别代表第一组数、第二组数对应的有序链表与合并后有序链表的索引与元素的异或和。   1. 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）   建立链表，在基本链表功能下添加若干功能函数，如利用插入排序的interSort，再建立外部函数merge（chain a,chain b,chain c）实现链表的组合等   1. 测试结果（测试输入，测试输出）  Input 10 10  6863 35084 11427 53377 34937 14116 5000 49692 70281 73704  4 6863  1 2 44199  5  4 21466  1 6 11483  5  4 34937  5  4 6863  1 10 18635  **Output**  0  398665  -1  410141  5  410141  0 输入 3 0  3 1 2 输出 5  0  5   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   结果均正确   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）   问题1：  *#include* "bits/stdc++.h"  template <class T>  class linearList  {  public:      virtual ~linearList(){};      virtual bool empty() const = 0;      virtual int size() const = 0;      virtual T &get(int theIndex) const = 0;      virtual int indexOf(const T &theElement) const = 0;      virtual void erase(int theIndex) = 0;      virtual void insert(int theIndex, const T &theElement) = 0;  };  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;      }  };  template <class T>  class iterator  {      chainNode<T> \*node;  public:      iterator() { node = NULL; }      iterator(chainNode<T> \*thenode) { node = thenode; }      ~iterator() { node = NULL; }      T &operator\*() const { *return* node->element; }      T \*operator->() const { *return* &node->element; }      iterator &operator++()      {          node = node->next;  *return* \*this;      }      iterator operator++(int)      {          iterator old = \*this;          node = node->next;  *return* old;      }      bool operator!=(const iterator right) const { *return* this->node != right.node; }      bool operator==(const iterator right) const { *return* this->node == right.node; }  };  template <class T>  class chain : public linearList<T>  {      bool checkIndex(int theIndex) const;      int listsize;      chainNode<T> \*firstNode;  public:      chain();      chain(const chain<T> &);      ~chain();      bool empty() const { *return* listsize == 0; }      int size() const { *return* listsize; }      T &get(int theIndex) const;      int indexOf(const T &theElement) const;      void erase(int theIndex);      void insert(int theIndex, const T &theElement);      void output() const;      void reverse();      void remove(const T &theElement);      int f();      void interSort();      iterator<T> begin()      {          iterator<T> it(firstNode);  *return* it;      }      iterator<T> end()      {          iterator<T> it(NULL);  *return* it;      }  };  template <class T>  bool chain<T>::checkIndex(int theIndex) const  {  *return* true;  }  template <class T>  chain<T>::chain()  {      firstNode = NULL;      listsize = 0;  }  template <class T>  chain<T>::chain(const chain<T> &theList)  {      listsize = theList.listsize;  *if* (listsize == 0)      {          firstNode = NULL;  *return*;      }      chainNode<T> \*temp = theList.firstNode;      firstNode = new chainNode<T>(temp->element);      temp = temp->next;      chainNode<T> \*currentNode = firstNode;  *while* (temp != NULL)      {          currentNode->next = new chainNode<T>(temp->element);          currentNode = currentNode->next;          temp = temp->next;      }      currentNode->next = NULL;  }  template <class T>  chain<T>::~chain()  {  *while* (firstNode != NULL)      {          chainNode<T> \*temp = firstNode->next;          delete firstNode;          firstNode = temp;      }  }  template <class T>  T &chain<T>::get(int theIndex) const  {  *if* (!checkIndex(theIndex))          exit(-1);      chainNode<T> \*currentNode = firstNode;  *for* (int i = 0; i < theIndex; ++i)          currentNode = currentNode->next;  *return* currentNode->element;  }  template <class T>  int chain<T>::indexOf(const T &theElement) const  {      chainNode<T> \*currentNode = firstNode;      int index = 0;  *while* (currentNode != NULL and currentNode->element != theElement)      {          currentNode = currentNode->next;          index++;      }  *if* (currentNode == NULL)  *return* -1;  *else*  *return* index;  }  template <class T>  void chain<T>::erase(int theIndex)  {  *if* (!checkIndex(theIndex))  *return*;      chainNode<T> \*deleteNode = NULL;  *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 = deleteNode->next;      }      delete deleteNode;      listsize--;  }  template <class T>  void chain<T>::remove(const T &theElement)  {      chainNode<T> \*deleteNode = NULL;      chainNode<T> \*p = firstNode;  *if* (p->element == theElement)      {          deleteNode = firstNode;          firstNode = firstNode->next;      }  *else*      {  *for* (; p->next != NULL and p->next->element != theElement; p = p->next)              ;  *if* (p->next == NULL)          {              std::cout << -1 << std::endl;  *return*;          }          deleteNode = p->next;          p->next = deleteNode->next;      }      delete deleteNode;      listsize--;  }  template <class T>  void chain<T>::insert(int theIndex, const T &theElement)  {  *if* (theIndex < 0 or theIndex > listsize)  *return*;  *if* (theIndex == 0)          firstNode = new chainNode<T>(theElement, firstNode);  *else*      {          chainNode<T> \*pre = firstNode;  *for* (int i = 0; i < theIndex - 1; ++i)              pre = pre->next;          pre->next = new chainNode<T>(theElement, pre->next);      }      ++listsize;  }  template <class T>  void chain<T>::output() const  {      chainNode<T> \*currentNode = firstNode;  *for* (int i = 0; i < listsize; ++i)      {          std::cout << currentNode->element << ' ';          currentNode = currentNode->next;      }  }  template <class T>  void chain<T>::reverse()  {      chainNode<T> \*suf1, \*suf2;      suf1 = firstNode->next;      suf2 = suf1->next;      firstNode->next = NULL;  *while* (suf2 != NULL)      {          suf1->next = firstNode;          firstNode = suf1;          suf1 = suf2;          suf2 = suf2->next;      }      suf1->next = firstNode;      firstNode = suf1;      suf1 = NULL;  }  template <class T>  int chain<T>::f()  {      int ans = 0;      int cnt = 0;  *for* (iterator<T> it = this->begin(); it != this->end(); ++it)      {          ans += cnt ^ (\*it);          ++cnt;      }  *return* ans;  }  int main()  {      chain<int> thechain;      int n, q;      std::cin >> n >> q;  *for* (size\_t i = 0; i < n; ++i)      {          int temp = 0;          std::cin >> temp;          thechain.insert(i, temp);      }  *while* (q--)      {          int ctrl = 0;          std::cin >> ctrl;  *switch* (ctrl)          {  *case* 1:          {              int idx = 0, val = 0;              std::cin >> idx >> val;              thechain.insert(idx, val);              thechain.output();  *break*;          }  *case* 2:          {              int val = 0;              std::cin >> val;              thechain.remove(val);              thechain.output();  *break*;          }  *case* 3:              thechain.reverse();              thechain.output();  *break*;  *case* 4:          {              int val = 0;              std::cin >> val;              std::cout << thechain.indexOf(val) << std::endl;  *break*;          }  *case* 5:              std::cout << thechain.f() << std::endl;  *break*;  *default*:  *break*;          }      }  *return* 0;  }  问题2：  *#include* "bits/stdc++.h"  template <class T>  class linearList  {  public:      virtual ~linearList(){};      virtual bool empty() const = 0;      virtual int size() const = 0;      virtual T &get(int theIndex) const = 0;      virtual int indexOf(const T &theElement) const = 0;      virtual void erase(int theIndex) = 0;      virtual void insert(int theIndex, const T &theElement) = 0;  };  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;      }  };  template <class T>  class iterator  {      chainNode<T> \*node;  public:      iterator() { node = NULL; }      iterator(chainNode<T> \*thenode) { node = thenode; }      ~iterator() { node = NULL; }      T &operator\*() const { *return* node->element; }      T \*operator->() const { *return* &node->element; }      iterator &operator++()      {          node = node->next;  *return* \*this;      }      iterator operator++(int)      {          iterator old = \*this;          node = node->next;  *return* old;      }      bool operator!=(const iterator right) const { *return* this->node != right.node; }      bool operator==(const iterator right) const { *return* this->node == right.node; }  };  template <class T>  class chain : public linearList<T>  {      bool checkIndex(int theIndex) const;      int listsize;      chainNode<T> \*firstNode;  public:      chain();      chain(const chain<T> &);      ~chain();      bool empty() const { *return* listsize == 0; }      int size() const { *return* listsize; }      T &get(int theIndex) const;      int indexOf(const T &theElement) const;      void erase(int theIndex);      void insert(int theIndex, const T &theElement);      void output() const;      void reverse();      void interSort();      void remove(const T &theElement);      int f();      iterator<T> begin()      {          iterator<T> it(firstNode);  *return* it;      }      iterator<T> end()      {          iterator<T> it(NULL);  *return* it;      }  };  template <class T>  bool chain<T>::checkIndex(int theIndex) const  {  *return* true;  }  template <class T>  chain<T>::chain()  {      firstNode = NULL;      listsize = 0;  }  template <class T>  chain<T>::chain(const chain<T> &theList)  {      listsize = theList.listsize;  *if* (listsize == 0)      {          firstNode = NULL;  *return*;      }      chainNode<T> \*temp = theList.firstNode;      firstNode = new chainNode<T>(temp->element);      temp = temp->next;      chainNode<T> \*currentNode = firstNode;  *while* (temp != NULL)      {          currentNode->next = new chainNode<T>(temp->element);          currentNode = currentNode->next;          temp = temp->next;      }      currentNode->next = NULL;  }  template <class T>  chain<T>::~chain()  {  *while* (firstNode != NULL)      {          chainNode<T> \*temp = firstNode->next;          delete firstNode;          firstNode = temp;      }  }  template <class T>  T &chain<T>::get(int theIndex) const  {  *if* (!checkIndex(theIndex))          exit(-1);      chainNode<T> \*currentNode = firstNode;  *for* (int i = 0; i < theIndex; ++i)          currentNode = currentNode->next;  *return* currentNode->element;  }  template <class T>  int chain<T>::indexOf(const T &theElement) const  {      chainNode<T> \*currentNode = firstNode;      int index = 0;  *while* (currentNode != NULL and currentNode->element != theElement)      {          currentNode = currentNode->next;          index++;      }  *if* (currentNode == NULL)  *return* -1;  *else*  *return* index;  }  template <class T>  void chain<T>::erase(int theIndex)  {  *if* (!checkIndex(theIndex))  *return*;      chainNode<T> \*deleteNode = NULL;  *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 = deleteNode->next;      }      delete deleteNode;      listsize--;  }  template <class T>  void chain<T>::remove(const T &theElement)  {      chainNode<T> \*deleteNode = NULL;      chainNode<T> \*p = firstNode;  *for* (; p->next->element != theElement and p->next != NULL; p = p->next)          ;  *if* (p->next == NULL)  *return*;      deleteNode = p->next;      p->next = deleteNode->next;      delete deleteNode;      listsize--;  }  template <class T>  void chain<T>::insert(int theIndex, const T &theElement)  {  *if* (theIndex < 0 or theIndex > listsize)  *return*;  *if* (theIndex == 0)          firstNode = new chainNode<T>(theElement, firstNode);  *else*      {          chainNode<T> \*pre = firstNode;  *for* (int i = 0; i < theIndex - 1; ++i)              pre = pre->next;          pre->next = new chainNode<T>(theElement, pre->next);      }      ++listsize;  }  template <class T>  void chain<T>::output() const  {      chainNode<T> \*currentNode = firstNode;  *for* (int i = 0; i < listsize; ++i)      {          std::cout << currentNode->element << ' ';          currentNode = currentNode->next;      }  }  template <class T>  void chain<T>::reverse()  {      chainNode<T> \*suf1, \*suf2;      suf1 = firstNode->next;      suf2 = suf1->next;      firstNode->next = NULL;  *while* (suf2 != NULL)      {          suf1->next = firstNode;          firstNode = suf1;          suf1 = suf2;          suf2 = suf2->next;      }      suf1->next = firstNode;      firstNode = suf1;      suf1 = NULL;  }  template <class T>  int chain<T>::f()  {      int ans = 0;      int cnt = 0;  *for* (iterator<T> it = this->begin(); it != this->end(); ++it)      {          ans += cnt ^ (\*it);          ++cnt;      }  *return* ans;  }  template <class T>  void chain<T>::interSort()  {  *for* (int i = 1; i < listsize; ++i)      {          T temp = get(i);          int j;  *for* (j = i - 1; j >= 0 and temp < get(j); --j)              get(j + 1) = get(j);          get(j + 1) = temp;      }  }  template <class T>  void merge(chain<T> a, chain<T> b, chain<T> &c)  {      int cnta = 0, cntb = 0;      iterator<T> ita = a.begin();      iterator<T> itb = b.begin();  *if* (a.size() == 0)      {  *for* (; cntb < b.size(); ++cntb, ++itb)              c.insert(cntb, \*itb);  *return*;      }  *if* (b.size() == 0)      {  *for* (; cnta < a.size(); ++cnta, ++ita)              c.insert(cnta, \*ita);  *return*;      }      T tempa = \*ita, tempb = \*itb;  *while* (cnta < a.size() or cntb < b.size())      {  *if* (tempa <= tempb)          {              c.insert(cnta + cntb, tempa);              ++cnta;              ++ita;  *if* (cnta == a.size())  *break*;              tempa = \*ita;          }  *else*          {              c.insert(cnta + cntb, tempb);              ++cntb;              ++itb;  *if* (cntb == b.size())  *break*;              tempb = \*itb;          }      }  *if* (cnta == a.size())      {  *for* (; cntb < b.size(); ++cntb, ++itb)              c.insert(cnta + cntb, \*itb);  *return*;      }  *if* (cntb == b.size())      {  *for* (; cnta < a.size(); ++cnta, ++ita)              c.insert(cnta + cntb, \*ita);  *return*;      }  }  int main()  {      chain<int> a, b, c;      int n, m;      std::cin >> n >> m;  *for* (size\_t i = 0; i < n; ++i)      {          int temp = 0;          std::cin >> temp;          a.insert(0, temp);      }  *for* (size\_t i = 0; i < m; ++i)      {          int temp;          std::cin >> temp;          b.insert(0, temp);      }      a.interSort();      b.interSort();      merge(a, b, c);      std::cout << a.f() << std::endl;      std::cout << b.f() << std::endl;      std::cout << c.f() << std::endl;  } | | | |