重庆大学本科学生实验项目任务书

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 实验题目 | | 栈、队列及其应用 | | | | | |
| 学院 | 计算机学院 | | 专业 | | 信息安全 | 年级 | 2020 |
| 实验目的   * 1. 理解栈和队列的逻辑结构以及输入输出等基本运算。 * 2. 理解用队列实现栈的原理及实现方法。 * 3. 熟练掌握使用栈识别有效的出栈顺序的方法。 | | | | | | | |
| 实验内容：   * 1. 完成下面的栈类QStack，使用其中的双队列实现入栈、出栈等基本运算   template <typename E>  class QStack : public Stack<E>{  private:  int maxSize; //栈的容量  AQueue QA；  AQueue QB; //基于数组实现的队列  public:  QStack(int size = defaultSize): QA(size), QB(size) //初始化队列  {  maxSize = size;  }  ~QStack() { }  //完成下列函数的代码  **void clear(){ }**  **void push(const E& it) { }**  **E pop() { }**  **const E& topValue() const { }**  **virtual int length() const { }**    };   * 2. 设1,2,…,N依次入栈QStack<int>, 判断由这N个整数构成的整数序列<a1,a2,…,aN> 是否为有效的出栈顺序。同时我们限定栈中只能存储K个整数(0<K<=N)，即整数入栈必须满足stack.length() < K, 如果stack.length()==K, 只能从中弹出1个以上的整数后，下一个整数才能入栈。（注：栈的容量必须大于或等于K） * 输入格式: 第一行有三个正整数N K m：N表示入栈的最大整数，K为栈中存储的整数数量上限，m表示接下来有m行输入，每一行都有1到N的整数的一组序列（空格分开）。 * 输出格式，输出m行字符T或F, 第i行的字符T(F), 表示第i行序列为有效（无效）的出栈顺序 (1<=i<=m) * 实例：   输入  5 3 3  1 2 3 4 5  3 2 1 5 4  1 5 4 3 2  输出  T  T  F     * 3. 最后提交完整的实验报告和源程序。 | | | | | | | |
| 参考资料：   * 1. Data Structures and Algorithm Analysis (C++ Version) Clifford A. Shaffer   + 2. Data Structure and Algorithm Analysis in C++ (Third Edition)，Mark Allen Weiss， Pearson Education, 2006.   + 3. Data Structures, Algorithms, and Applications in C++，Sartaj Sahni， McGraw-Hill, 1998.   + 4.《数据结构（ C 语言版）》，严蔚敏，吴伟民编著，清华大学出版社，2007年第1版 | | | | | | | |
| 任务下达日期 2021 年 11月 1日 | | | | 完成日期 年 月 日 | | | |

说明：学院、专业、年级均填全称，如：计算机学院、计算机科学与技术、2018。

**《数据结构》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **20级信息安全专业01班** | | | **姓名** | **蔡欣彤** |
| **实验题目** | 栈、队列及其应用 | | | | | |
| **实验时间** | **2021.11.4** | | **实验地点** | **D1421** | | |
| **实验成绩** |  | | **实验性质** | **□验证性** √**设计性 □综合性** | | |
| 教师评价：  □算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  其他：  评价教师签名： | | | | | | |
| 一、实验目的   * 1. 理解栈和队列的逻辑结构以及输入输出等基本运算。 * 2. 理解用队列实现栈的原理及实现方法。 * 3. 熟练掌握使用栈识别有效的出栈顺序的方法。 | | | | | | |
| 二、实验项目内容   * 1. 完成下面的栈类QStack，使用其中的双队列实现入栈、出栈等基本运算   template <typename E>  class QStack : public Stack<E>{  private:  int maxSize; //栈的容量  AQueue QA；  AQueue QB; //基于数组实现的队列  public:  QStack(int size = defaultSize): QA(size), QB(size) //初始化队列  {  maxSize = size;  }  ~QStack() { }  //完成下列函数的代码  **void clear(){ }**  **void push(const E& it) { }**  **E pop() { }**  **const E& topValue() const { }**  **virtual int length() const { }**    };   * 2. 设1,2,…,N依次入栈QStack<int>, 判断由这N个整数构成的整数序列<a1,a2,…,aN> 是否为有效的出栈顺序。同时我们限定栈中只能存储K个整数(0<K<=N)，即整数入栈必须满足stack.length() < K, 如果stack.length()==K, 只能从中弹出1个以上的整数后，下一个整数才能入栈。（注：栈的容量必须大于或等于K） * 输入格式: 第一行有三个正整数N K m：N表示入栈的最大整数，K为栈中存储的整数数量上限，m表示接下来有m行输入，每一行都有1到N的整数的一组序列（空格分开）。 * 输出格式，输出m行字符T或F, 第i行的字符T(F), 表示第i行序列为有效（无效）的出栈顺序 (1<=i<=m) * 实例：   输入  5 3 3  1 2 3 4 5  3 2 1 5 4  1 5 4 3 2  输出  T  T  F     * 3. 最后提交完整的实验报告和源程序。 | | | | | | |
| 1. 实验过程或算法（源程序）   #include <iostream>  #include<cassert>  #define defaultSize 14  #define Assert(a,b) assert((a)&&(b))  using namespace std;  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*定义队列  template<typename E>  class Queue{  private:  void operator=(const Queue&) {}  Queue(const Queue&) {} //拷贝构造函数  public:  Queue() {}  virtual ~Queue() {}  virtual void clear() = 0;  virtual void enqueue(const E&) = 0;  virtual E dequeue() = 0;  virtual const E& frontValue() = 0;  virtual int length() = 0;  };  template<typename E>  class AQueue: public Queue<E> {  private:  int maxSize;  int front;  int rear;  E \*listArray;  public:  AQueue(int size=defaultSize) {  maxSize=size+1;  rear=0;  front=1;  listArray=new E[maxSize];  }  ~AQueue() {  delete [] listArray;  }  void clear() {  rear=0;  front=1;  }  void enqueue(const E& it) { //尾部入队列  Assert(((rear+2)%maxSize)!=front,"Queue is full");  rear=(rear+1)%maxSize;  listArray[rear]=it;  }  E dequeue() { //头部出队列  Assert(length()!=0,"Queue is empty");  E it=listArray[front];  front=(front+1)%maxSize;  return it;  }  const E& frontValue() {  Assert(length()!=0,"Queue is empty");  return listArray[front];  }  const E& rearValue() {  Assert(length()!=0,"Queue is empty");  return listArray[rear];  }  virtual int length() {  return ((rear+maxSize)-front+1)%maxSize;  }  };  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*定义栈  template<typename E>  class Stack {  private:  void operator=(const Stack&) {}  Stack(const Stack&) {}  public:  Stack() {}  virtual ~Stack() {}  virtual void clear() = 0;  virtual void push(const E& it) = 0;  virtual E pop() = 0;  virtual const E& topValue() = 0;  virtual int length() = 0;  };  template<typename E>  class QStack:public Stack<E>{  private:  int maxSize; //栈的容量  //基于数组实现的队列  AQueue<E> QA;  AQueue<E> QB;  public:  QStack(int size=defaultSize):QA(size),QB(size) {  maxSize=size;  }  ~QStack() {}  void clear() {  QA.clear();  QB.clear();  }  //两个队列互相转换除队尾的全部元素，队尾元素相当于栈顶，对栈顶进行操作  void push(const E& it) {  Assert(QA.length()<maxSize,"QStack is full");  QA.enqueue(it);  }  E pop() {  Assert(QA.length()>0,"QStack is empty");  while(QA.length()>1) {  QB.enqueue(QA.dequeue());  }  E it =QA.dequeue();  while(QB.length()>=1) {  QA.enqueue(QB.dequeue());  }  return it;    }  //哪个队列长，所有元素就在哪个队列里面  const E& topValue() {  return QA.rearValue();  }  virtual int length() {  return QA.length();  }  void validate(int N) { //验证出栈顺序是否有效  AQueue<E> Qtmp(N);  int val;  //读取待测序列  for(int i=1;i<=N;i++) {  cin>>val;  Qtmp.enqueue(val);  }  int odnr=1; //记录原顺序1，2，……，N  while(odnr<=N) {  //元素压入栈  while(Qtmp.frontValue()>=odnr) {  if(length()<maxSize) { //保证不溢出栈  push(odnr);  odnr++;  } else {  if(length()==maxSize) {  if(odnr==Qtmp.frontValue()) {  push(odnr);  odnr++;  break;  } else { //若栈压入的最后一位（栈顶）没办法和队首元素相等，该顺序必然错误·  cout<<'F'<<endl;  return;  }  }  }  }  //序列比较  for(int i=1;i<=length();i++){  if(topValue()==Qtmp.frontValue()){ //当栈首位等于队列首位，将两者都移出  pop();  Qtmp.dequeue();  }  }  }  //倘若能执行完整个循环（对1~N），则此序列正确  cout<<'T'<<endl;  return;  }  };  int main()  {  int N; //入栈的最大整数  int K; //栈存储整数的数量上限  int m; //共有m行序列需要检测  while(cin>>N>>K>>m){  QStack<int> q(K);  int i=1;  while(i<=m) {  q.validate(N);  i++;  }  }  } | | | | | | |
| 1. 实验结果及分析和（或）源程序调试过程   实验结果：    与实验要求相符。  分析：  在考虑用双队列时，考虑使用一个A队列作存储栈，pop时A队列将n-1个元素存入B队列中，将最后一个元素作为栈中的topValue弹出，再把B队列的元素全部弹进A队列。  使用函数来判断是否合理中，考虑题目特性，在一个一个push进栈之前判断一下是否与存入队列中的首个元素相同，若相同，pop，再继续判断，若不是则继续push再判断。 | | | | | | |