**《数据结构》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **计算机科学与技术2020级01班** | | | **姓名** | **陈鹏宇** |
| **实验题目** | 栈、队列及其应用 | | | | | |
| **实验时间** | **2021.11.2** | | **实验地点** | **Ds1408** | | |
| **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。 | | | | | | |
| 三、实验过程或算法（源程序）  #include <iostream>  #define defaultSize 100  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() const = 0;  virtual int length() const = 0;  };  template <typename E>  class Queue  {  private:  void operator = (const Queue&) {}  Queue(const Queue&) {}  public:  Queue() {}  virtual ~Queue() {}  virtual void clear() = 0;  virtual void enqueue(const E&) = 0;  virtual E dequeue() = 0;  virtual const E& frontValue() const = 0;  virtual int length() const =0;  };  void Assert(bool b , std::string s)  {  if (!b) {  std::cout << "Assertion Failed: " << s << std::endl;  exit(-1);  }  }  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){  Assert((rear+2) % maxSize != front , "Queue is full");  rear = (rear+1) % maxSize;  listArray[rear] = it;  }  E dequeue(){  Assert(length() != 0 , "Queue is empty");  E it = listArray[front];  front = (front+1) % maxSize;  return it;  }  const E& frontValue() const{  Assert(length() != 0 , "Queue is empty");  return listArray[front];  }  virtual int length() const{  return ((rear + maxSize) - front + 1) % maxSize;  }  };  template <typename E>  class QStack: public Stack<E>{  private:  int maxSize;  AQueue<E> QA; //formal  AQueue<E> QB; //temp  public:  QStack(int size = defaultSize): QA(size), QB(size) {  maxSize = size;  }  ~QStack(){}  void clear(){  QA.clear();  QB.clear();  }  E top;  void push(const E& it){ //A，B中要保持其中一个队列为空  if(QA.length()){ //如果A不为空，则入队A  QA.enqueue(it);  }else{ //其他情况入队B  QB.enqueue(it);  }  top = it;  }  E pop(){  if(QA.length()){//如果A不为空  while(QA.length() > 2){  QB.enqueue(QA.dequeue());  }  if(QA.length() == 2){  QB.enqueue(top = QA.dequeue());  return QA.dequeue();  }  else  return top = QA.dequeue();  }else{  while(QB.length() > 2){  QA.enqueue(QB.dequeue());  }  if(QB.length() == 2){  QA.enqueue(top = QB.dequeue());  return QB.dequeue();  }  else  return top = QB.dequeue();  }  }  const E& topValue() const {  return top;  }  virtual int length() const{  return (QA.length())? QA.length() : QB.length();  }  };  using namespace std;  //int main()  //{  // QStack<int> a(6);  // for(int i = 0 ; i < 6 ; i++){  // int temp ; cin >> temp ; a.push(temp);  // }  // for(int i = 0 ; i < 6 ; i++){  // cout<<"topvalue: "<<a.topValue()<<endl;  // cout<<"length: "<<a.length()<<endl;  // cout<<"popvalue: "<<a.pop()<<endl;  // }  // cout<<a.length();  // return 0;  //}  int main(){  int n , k , m ; cin >> n >> k >> m;  QStack<int> temp(n); //临时栈  int box[n]; //储存出栈顺序  while(m--){  int ele = 1 , ind = 0; //输入顺序和出栈数组下标  int flag = 1;  temp.clear();  for(int i = 0 ; i < n ; i++)  cin >> box[i];  while(true){  if(ele == box[ind]){  //cout<<ele<<" with "<<box[ind]<<endl;  ele++;  ind++;  }  else if(temp.length() && temp.topValue() == box[ind]){  //cout<<temp.topValue()<<" with "<<box[ind]<<endl;  temp.pop();  ind++;  }  else{  if(ele > n) break;  //cout<<ele<<" with "<<box[ind]<<endl;  temp.push(ele);  ele++;  if(temp.length() + 1 > k){  flag = 0;  break;  }  }  }  if(!flag || temp.length())  cout<<"F"<<endl;  else  cout<<"T"<<endl;  }  } | | | | | | |
| 四、实验结果及分析和（或）源程序调试过程  实验结果：  基本功能实现：    实验：        实验分析：  本程序算法将其分为两种情况，入栈立刻出栈，入栈后无立刻出栈。  将出栈顺序放入数组中，按输入顺序ele与数组元素box[ind]依次比较，  如果ele = box[ind]，则入栈立刻出栈，ele++，ind++；  如果ele != box[ind]，则入栈没有立刻出栈，ele存入临时栈，ele++；  如果临时栈里面有元素且临时栈栈顶元素等于box[ind]，栈顶出栈，ind++；  当ele > n或stack.length() + 1 > k时，结束循环；  此时，如果栈内还有元素或者由于stack.length() + 1 > k而退出循环时，说明不是出栈的有效顺序，输出F。  为什么是stack.length() + 1 > k：  这个判断语句位于入栈没有立刻出栈时，但是还没有入栈，由于只能存k个数据，所以要预留一个位置给即将入栈的元素。 | | | | | | |