**《数据结构》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2018级物联网工程专业01班** | | | **姓名** | **邓露** |
| **实验题目** | 栈、队列及其应用 | | | | | |
| **实验时间** | **2019.11.04** | | **实验地点** | **DS1422** | | |
| **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>  #include<stdio.h>  #define defaultSize 100  using namespace std;  //Abstract queue class  template <typename E> class Queue{  private:  void operator = (const Queue&) {} //protect assignment  Queue (const Queue&) {} //protect copy constructor  public:  Queue() {} //default  virtual ~Queue(){} //base destructor  virtual void clear()=0;  virtual void enqueue(const E&) = 0;  virtual E dequeue ()=0;  virtual const E& frontValue() const=0;  virtual const E& lastValue() const=0;  virtual const E& nValue() const=0;  virtual int length() const=0;  };  //Array-based queue implementation  template <typename E> class AQueue{  private:  int maxSize;  int front;  int rear; //index of rear element  int value;  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){  if (((rear+2)%maxSize)==front)  cout<<"1 Queue is full"<<endl;  else  {  rear =(rear+1)%maxSize;  listArray[rear]=it;  }  }  E dequeue(){  if (length()==0)  cout<<"2 Queue is empty"<<endl;  else{  E it =listArray[front];  front =(front+1)%maxSize;  return it;  }  }  const E& frontValue() const {  if(length()==0)  cout<<"3 Queue is empty"<<endl;  else  return listArray[front];  }  virtual int length() const  {  return ((rear+maxSize)-front+1)%maxSize;  }  const E& lastValue() const {  if(length()==0)  cout<<"4 Queue is empty"<<endl;  else  return listArray[((rear+maxSize)-front+1)%maxSize];  }  const E& nValue(int value) const {  if(length()==0)  cout<<"5 Queue is empty"<<endl;  else  return listArray[value%maxSize];  }  };  template <typename E> class Stack  {  private:  void operator = (const Stack&) {} //protect assignment  Stack (const Stack&) {} //protect copy constructor  public:  Stack() {} //default  virtual ~Stack(){} //base destructor  virtual void clear()=0;  virtual void push(const E&it) = 0;  virtual E pop ()=0;  virtual const E& topValue() const=0;  virtual const E& nValue() const=0;  virtual int length() const=0;  };  template <typename E>  class QStack {  private:  int maxSize; //栈的容量  AQueue<E> QA;  AQueue<E> 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  //virtual void clear()=0;  void clear()  {  QA.clear();  QB.clear();  }  //virtual void push()=0;  void push(const E& it)  {  if(QA.length()==0&&QB.length()==0)  QA.enqueue(it);  else if(QA.length()!=0)  QA.enqueue(it);  else if(QB.length()!=0)  QB.enqueue(it);  }  //virtual E pop() {}=0;  E pop()  {  if(QA.length()==0&&QB.length()==0)  cout<<"6 Queue is empty"<<endl;  else if(QA.length()==0)  {  while(QB.length()>1)  {  QA.enqueue(QB.dequeue());  }  return QB.dequeue();  }  else if(QB.length()==0)  {  while(QA.length()>1)  {  QB.enqueue(QA.dequeue());  }  return QA.dequeue();  }  }  //virtual int length() const { }=0;  int length()  {  if(QB.length()==0&&QA.length()==0)  return 0;  else if(QA.length()!=0)  return QA.length();  else if(QB.length()!=0)  return QB.length();  }  //virtual const E& topValue() const {}=0;  const E& topValue()  {  if(QB.length()==0&&QA.length()==0)  cout<<"7 Queue is empty"<<endl;  else if(QA.length()!=0)  return QA.lastValue();  else if(QB.length()!=0)  return QB.lastValue();  }  const E& nValue(int value)  {  if(QB.length()==0&&QA.length()==0)  cout<<"8 Queue is empty"<<endl;  else if(QA.length()!=0)  return QA.nValue(value);  else if(QB.length()!=0)  return QB.nValue(value);  }  };  int order(int \*a,int n,int k)  {  int curr=0;  QStack <int> QS(k);  for(int v=0;v<k;v++) //先将QS里压入k个值  {  QS.push(++curr);  }  int s=0; //a[s],输入的出栈顺序  int t=1; //QS.nValue(t)  for(t=1;t<=k;t++) //确定第一次比较时t的值  {  if(QS.nValue(t)==a[s])  break;  }  int b[n]; //存入已弹出元素，避免重复弹入  int h=0; //b[h]  while(s<n)  {  if(QS.length()==0)  break;  if(QS.length()==1&&s==4)  {  if(QS.nValue(t+1)==a[s])  break;  else  {  cout<<"F"<<endl;  return 0;  }  }  else if(a[s]==QS.nValue(t))  {  int g=QS.length();  for(int v=0;v<=curr-a[s]&&v<g;v++)  {  QS.pop();  }  b[h++]=a[s];  int m=0;  for(int w=a[s]+1;w<=curr+1&&w<=n;w++)  { //如果不是B里的元素则push  for(int q=0;q<h;q++)  {  if(w==b[q])  m=1;  }  if(m==0)  {  QS.push(w);  }  }  if(curr<n)  curr++;  s++;  if(t>QS.length())  t=QS.length();  }  else if(t!=1&&a[s]==QS.nValue(t-1))  {  int g=QS.length();  for(int v=0;v<curr-a[s]&&v<g;v++)  QS.pop();  b[h++]=a[s];  int m=0;  for(int w=a[s]+1;w<=curr+1&&w<n;w++)  { //如果不是B里的元素则push  for(int q=0;q<h;q++)  {  if(w==b[q])  m=1;  }  if(m==0&&curr<n)  QS.push(w);  }  if(curr<n)  curr++;  s++;  t=t-1;  }  else if((QS.length()<=k)&&a[s]==QS.nValue(QS.length()))  {  //cout<<"33"<<endl;  b[h++]=QS.pop();  if(curr<=n)  QS.push(++curr);  s++;  t=k-1;  if(t>QS.length())  t=QS.length();  }  else  {  cout<<"F"<<endl;  return 0;  }  }  cout<<"T"<<endl;  curr=0;  QS.clear();  return 0;  }  int main()  {  int n,k,m;  scanf("%d %d %d",&n,&k,&m);  for(int i=0;i<m;i++)  {  int a[n];  for(int j=0;j<n;j++)  scanf("%d",&a[j]);  order(a,n,k);  }  return 0;  }  /\*  int main()  {  return 0;  }\*/ | | | | | | |
| 四、实验结果及分析和（或）源程序调试过程  1.OJ截图 | | | | | | |

2.实验运行结果

