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 | | 计算机科学与技术 | 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 2021 年 11 月 4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、2020。

**《数据结构》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2020级计算机科学与技术卓越一班** | | | **姓名** | **孙莹莹** |
| **实验题目** | 栈、队列及其应用 | | | | | |
| **实验时间** | **2021.11.4** | | **实验地点** | **DS1404** | | |
| **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） 2. 用基于数组实现的队列AQueue实现栈QStack 3. AQueue实现   Queue为AQueue的抽象父类，AQueue有成员listArray为数组指针存放元素，*front*为头，*rear*为尾，*maxSize*为可存放的最多元素个数+1.  有成员函数：*enqueue()*将元素加入队列末尾，*dequeue()*将第一个元素出队，*frontValue()*返回第一个元素值，*length()*返回队列中元素的个数，*full()*和*empty()*判断队列是否为满或空。  const int defaultSize = 50;  //队列实现  template<typename E>  class Queue  {  private:      void operator=(const Queue & ) {}      Queue(const Queue&) {}  public:      Queue() {}      virtual ~Queue() {}      virtual void enqueue(const E &) = 0;      virtual E dequeue() = 0;      virtual void clear() = 0;      virtual const E & frontValue() const = 0;      virtual int length() const = 0;      virtual bool full() const = 0;      virtual bool empty() const = 0;  };  //顺序队列  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)      {          if(((rear+2)%maxSize) ==front)          {              cout << "Queue is full" << endl;              return;          }          rear = (rear + 1) % maxSize;          listArray[rear] = it;      }      E dequeue()      {          if(length()==0)          {              cout << "Queue is empty" << endl;              return -10000;          }          E it = listArray[front];          front = (front+1) % maxSize;          return it;      }      const E & frontValue() const{          if(length()==0)          {              cout << "Queue is empty" << endl;              return defaultSize;          }          return listArray[front];      }      int length() const{          return ((rear+maxSize)-front+1) % maxSize;      }      bool full() const      {          return ((rear+2)%maxSize==front);      }      bool empty() const      {          return (front==(rear+1)%maxSize);      }  };   1. 栈QStack的实现   Stack为QStack的父类，QStack有成员*maxSize*为栈的最大容量，队列*QA,QB*。其中QA的作用是存储栈内元素，且满足栈的先入后出要求，既最先入栈的元素处于队列QA的末尾，最后入栈的元素处于QA的头部。而QB则起到辅助作用，具体如下：  *push()*函数：先将QA中的所有元素一个个出队并加入QB中，然后在QA中将待入栈元素入队，再将QB中的所有元素出队重新加入QA中。  *pop()*函数：弹出栈顶元素，既让QA进行dequeue（）操作。  *clear()*函数：对QA进行的queue（）操作直到QA为空。  *topValue()*函数：返回栈顶元素值，即对QA进行frontValue()操作。  *length()*函数：返回栈内元素个数，即对QA进行length()操作。  *full()、empty()*函数:判断栈是否为满或为空，对QA进行full(),empty()操作。  check()函数：判断是否为合理出队顺序，具体见③  //栈实现  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;      virtual bool full() const = 0;      virtual bool empty() const = 0;    };  template <typename E>  class QStack : public Stack<E>{  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       void clear(){           while(!QA.empty())           {               QA.dequeue();           }           maxSize = 0;      }       void push(const E& it) {           if(!QA.full())           {               while(!QA.empty())              {                  E temp = QA.dequeue();                  QB.enqueue(temp);              }              QA.enqueue(it);              while(!QB.empty())              {                  E temp = QB.dequeue();                  QA.enqueue(temp);              }          }        }       E pop() {           int s = QA.frontValue();           QA.dequeue();           return s;        }       const E &topValue() const {           return QA.frontValue();        }       virtual int length() const {           return QA.length();        }        bool full() const      {          return QA.full();      }      bool empty() const      {          return QA.empty();      }      bool check(int num) {   //验证出栈顺序是否有效          AQueue <E> output;          for(int i=0; i<num; i++)          {              E s;              cin >> s;              output.enqueue(s);          }          for(int i=1; i<=num; i++)          {              if(!full())              {                  push(i);              }              else              {                  E k = topValue();                  if(k !=output.frontValue())                  {                      return false;                  }                  else                  {                      pop();                      output.dequeue();                      push(i);                  }              }              while(!empty() && topValue() == output.frontValue())              {                  pop();                  output.dequeue();              }          }          if(!empty())          {              return false;          }          return true;      }  };   1. 实现判断是否为正确出队序列（代码见上述check函数）   用一个队列output存储输入的出队序列，i从1开始一直到N，如果栈不是满的，就将i压入栈中，否则记录栈顶元素并与output队首元素比较，如果相同就将栈顶元素弹出，继续执行下面的代码，否则返回false,即栈无法满足这个出队序列。然后当栈顶元素等于队首元素且栈不为空时，就弹出栈顶元素和队首元素(栈可以满足这一部分出队序列）。然后i++直到i等于N。最后如果栈不是空的，就说明该出队序列不合理，否则合理。 | | | | | | |
| 1. 实验结果及分析和（或）源程序调试过程     由图，上述代码可以实现QStack并判断是否为合理出栈顺序。  QStack的实现中出来push()函数会消耗O(N）的时间之外，其他的依赖AQueue的实现并不会耗费过多时间和空间。 | | | | | | |