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：201700140056 | 姓名：李港 | | 班级：跟18.2（17.4） |
| 实验题目：实验七 队列 | | | |
| 实验学时：2h | | 实验日期：2019.11.07 | |
| 实验目的：  1、掌握队列结构的定义与实现。  2、掌握队列结构的使用。 | | | |
| 软件开发工具：  Virtual Studio 2019 | | | |
| 1. **实验内容**    1. 创建队列类，采用链式描述；    2. 实现卡片游戏   假设桌上有一叠扑克牌，依次编号为1-n（从最上面开始）。当至少还有两张的时候，可以进行操作：把第一张牌扔掉，然后把新的第一张放到整叠牌的最后。输入n，输出每次要扔掉的牌，以及最后剩下的牌。   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）**   **总体思路：**   * 1. 采用链表作为队列底层数据结构。   2. 提供的主要功能：      1. push：向链表尾部加入新的元素      2. pop：从链表头部弹出元素      3. front：从链表头部删除元素   3. 提供常见的报错，如队列为空。   **数据结构：**  采用链表作为队列底层数据结构。  template<typename T>  typedef struct node {  T data;  node\* next;  }node;  **算法：**   * 1. 插入到队尾：      1. 链表无需担心缓冲区长度限制，直接插入即可      2. 为了提高插入的速度，链表类应该保存尾节点指针      3. 尾结点的next指向null   2. 弹出队首元素：      1. 首先判断是否为空      2. 不为空则可弹出栈顶      3. 将栈顶节点的next赋值到链表头指针后删除之前的头结点即可   3. 获得头部元素：      1. 首先判断队列是否为空      2. 为空则报错      3. 不为空则返回元素   4. 卡片游戏：      1. 只要剩余元素大于等于两个则：         1. 弹出队首元素         2. 将接下来的队首元素压入队尾并弹出         3. 重复循环      2. 最后输出剩下的队首元素即为结果   队列类设计如下所示：  queue  public:  enum queue\_err { queue\_empty }; //常见的错误  private:  struct node; //结点类型  node\* \_head; //头结点指针  node\* \_end; //尾结点指针，指向NULL  int \_length; //元素个数  public:  queue (); //构造函数  ~queue (); //析构函数  void push ( const T& in ); //入队列  T front (); //获得首元素  void pop (); //弹出首元素  bool isempty ()const; //是否为空  int size ()const; //获取元素个数   1. **测试结果（测试输入，测试输出）**    1. 验收展示：      * 1. 平台提交      1. **分析与探讨（结果分析，若存在问题，探讨解决问题的途径）**   本实验较为简单，实验过程中未遇到明显问题，但仍不能对队列这种数据结构掉以轻心，后缀表达式计算结果的过程中就可以用队列作为存贮媒介。   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）**   main.cpp  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* main.cpp \*  \* Copyright (C) 2019.10.30 TriAlley lg139@139.com \*  \* @brief 链表队列的测试 \*  \* @license GNU General Public License (GPL) \*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include<iostream>  #include"queue.h"  using namespace std;  #define DEBUG  #ifdef DEBUG  #define dprintf printf  #else  #define dprintf /\  / printf  #endif  int main () {  try {  queue<int> q;  int n;  cin >> n;  for ( int i = 1; i <= n; i++ ) {  q.push ( i );  }  while ( q.size () >= 2 ) {  dprintf ( "扔掉：");  dprintf ( "%d\n", q.front () );  q.pop ();  dprintf ("移动：%d\n", q.front ());  q.push ( q.front () );  q.pop ();  }  dprintf ( "剩余：\n" );  while ( !q.isempty () ) {  cout << q.front () << endl;  q.pop ();  }  } catch ( queue<int>::queue\_err e ) {  cout << endl << e << endl;  }  }  cal.h  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* queue.h \*  \* Copyright (C) 2019.10.30 TriAlley lg139@139.com \*  \* @brief 链表队列的实现 \*  \* @license GNU General Public License (GPL) \*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #pragma once  /\*queue  public:  enum queue\_err { queue\_empty }; //常见的错误  private:  struct node; //结点类型  node\* \_head; //头结点指针  node\* \_end; //尾结点指针，指向NULL  int \_length; //元素个数  public:  queue (); //构造函数  ~queue (); //析构函数  void push ( const T& in ); //入队列  T front (); //获得首元素  void pop (); //弹出首元素  bool isempty ()const; //是否为空  int size ()const; //获取元素个数  \*/  template<typename T>  class queue {  public:  enum queue\_err { queue\_empty };  private:  typedef struct node {  T data;  node\* next;  }node;  node\* \_head;  node\* \_end;  int \_length;  public:  queue () {  \_head = new node;  \_end = \_head;  \_length = 0;  }  ~queue () {  while ( \_head->next != NULL ) {  node\* temp = \_head;  \_head = \_head->next;  delete temp;  }  delete \_head;  }  void push ( const T& in ) {  \_length++;  node\* n\_end = new node;  n\_end->data = in;  n\_end->next = NULL;  \_end->next = n\_end;  \_end = n\_end;  }  T front () {  if ( isempty () ) {  throw queue\_empty;  }  return \_head->next->data;  }  void pop () {  if ( isempty () ) {  throw queue\_empty;  }  node\* n\_head = \_head->next;  delete \_head;  \_head = n\_head;  \_length--;  return;  }  bool isempty ()const { return \_head == \_end; }  int size ()const { return \_length; }  }; | | | |