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：201900161140 | 姓名： 张文浩 | | 班级： 19级人工智能 |
| 实验题目：队列 | | | |
| 实验学时：2 | | 实验日期： 9.30 | |
| 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：  Vscode | | | |
| 1. 实验内容   **1、题目描述**：  首先创建队列类，采用数组描述；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，输出最后剩下的牌。  **输入输出格式**：  **输入**：  一个整数n，代表一开始卡片的总数。  **输出**：  最后一张卡片的值。   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）   自定义一个队列类Queue，采用循环队列的思路，对于每一个Queue，里面的成员变量包括front队首下标索引，back队尾下标索引，length当前队列容量，queue队列指针。其中，为了简化代码，特别定义front为循环队列队首前一个位置的下标。  其中封装的方法函数包括：  **Push(x)**：  从队尾插入一个元素，实现的时候第一步要判断当前队列的queue容量length是否满了，如果满了，就新开一个两倍容量的queue空间作为新的queue存储空间。原来已有的元素利用copy函数复制到新开的空间中，这样就有富余的空间了，可以进行插入操作。插入的时候只需要将back索引向后移动一位，并在将该位置的元素值设置为要插入的元素x。  **Pop()：**  弹出队首一个元素，弹出元素比较简单，只需要将front向前移动一位，并清理（析构）被弹出元素的位置的空间  **Init()：**  初始化，根据本实验的要求，将队列初始化为1-n的队列即可，直接将1-n一个一个push进队列即可。  **Output()：**  输出队列，从front+1位置开始，输出queue的元素，因为是循环队列，当所引导length-1后就要返回0位置继续索引，直到遍历的次数等于队列中元素个数的大小。  **Size()：**  获得当前队列元素的个数，利用back和front的位置即可快速的计算出当前队列中有几个元素。  **Getfront()：**  获得队首元素的值，因为front是队首的前一个位置，所以queue{front+1（对length取模）}就是队首元素。  **Solve()：**  解决本实验要求的函数，拿到一个已经根据题目要求的初始化后的队列后，只要队列的长度大于0，就按照实验题目描述，从队首pop弹出一个元素，再把队首的一个元素pop（）然后push进队尾。   1. 测试结果（测试输入，测试输出）   **n=10**    **n=50**    **n=100**     1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   很容易想到用循环队列的方式，但注意要front的设置，一开始我并没有把front定义为队首前一个元素，发现编写函数的时候总是要判断边界情况，时代码比较冗长。然后就采用了书上的改进方法，将front设置为队首前一个位置，这样不用判断边界情况，起到了简化代码的作用。  这个实验比较简单，没有什么别的问题。  题目的关键在于队列中数组的设置。在使用动态数组存储数据时要注意，数组实际容量应该为（队尾元素下标-队首元素下标+1+数组设定容量）%数组设定容量。在处理入队，出队操作之后，要记得将队首元素下标、队尾元素下标加1，再%数组设定容量。在数组内循环存储数据可以防止程序为数组开辟的空间不够用，因此每次更新队首队尾元素的下标之后都要取模。当队首下标即将到达队尾下标的位置时，要及时增加数组大小，开辟新的空间，之后要更新数组设定容量。整个操作都是基于数组设定容量实现的，因此不会造成空间浪费，无论队列结构怎么改变，都处于可控的范围之内。   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）   #include <iostream>  using namespace std;  class Queue  {  public:      Queue(int initialCapacity = 10);      ~Queue() { delete[] queue; }      void push(int x);      void pop();      void init(int l);      void output();      void solve();      int size();      int getfront();  private:      int front;  //队首前一个位置      int back;   //队尾      int length; //队列长度      int \*queue; //队列指针  };  Queue::Queue(int initialCapacity)  {      length = initialCapacity;      queue = new int[length];      front = 0;      back = 0;  }  void Queue::push(int x)  {      //如果需要，则增加数组长度      if ((back + 1) % length == front) //加倍数组长度      {          //分配新的数组空间          int \*newqueue = new int[2 \* length];          //把原数组复制到新数组          int start = (front + 1) % length;          if (start < 2)          {              //没有形成环              copy(queue + start, queue + start + length - 1, newqueue);          }          else          {              //队列形成环              copy(queue + start, queue + length, newqueue);              copy(queue, queue + back + 1, newqueue + length - start);          }          //这是新的队列的首和尾的元素位置          front = 2 \* length - 1;          back = length - 2;          length \*= 2;          delete[] queue;          queue = newqueue;      }      back = (back + 1) % length;      queue[back] = x;  }  void Queue::pop()  {      //从队首弹出一个元素      front = (front + 1) % length;      ~queue[front];  }  void Queue::init(int n)  {      for (int i = 1; i <= n; i++)          push(i);  }  void Queue::output()  {      for (int i = (front + 1) % length; i < (front + 1) % length + size(); i++)      {          cout << queue[i];      }      cout << endl;  }  int Queue::size()  {      //获得当前数组的元素个数      return (back - front + length) % length;  }  int Queue::getfront()  {      //获得队首元素      return queue[(front + 1) % length];  }  void Queue::solve()  {      while (size() >= 2)      {          pop();          int temp = getfront();          pop();          push(temp);      }      output();  }  int main()  {      int n;      cin >> n;      Queue q;      q.init(n);      q.solve();      system("pause");      return 0;  } | | | |