作业七 队列

## 1、什么是队列？它有哪两种存储方式？队列的运算规则是什么？

队列是一种特殊的线性表，它只允许在表的前端（front）进行删除操作，而在表的后端（rear）进行插入操作。进行插入操作的端称为队尾，进行删除操作的端称为队头。队列中没有元素时，称为空队列。

队列的两种存储方式主要是**顺序存储**和**链式存储**：

1. **顺序存储**：队列的顺序存储结构通常由一个一维数组和一个记录队列头元素位置的变量front以及一个记录队尾元素位置的变量rear组成。其中，front变量的初值为0，rear变量的初值为0。
2. **链式存储**：队列的链式存储结构实际上就是一种特殊的线性表，链式队列。

队列的运算规则有两个：**入队**和**出队**：

1. **入队**：把一个元素添加到队列的尾部。
2. **出队**：把一个元素从队列的头部删除。

当队列为空时，再对其进行出队操作会导致“下溢”，当队列已满时，再对其进行入队操作会导致“上溢”。在进行队列操作时，需要预先检查队列的状态。

## 2、队列有哪些基本操作？

队列的基本操作主要有以下几种：

1. **初始化队列**：建立一个空队列。
2. **判断队列是否为空**：检查队列中是否有元素，如果没有元素则返回真，否则返回假。
3. **入队**：在队尾插入一个新元素。
4. **出队**：删除队头元素并返回其值。
5. **获取队头元素**：返回队头元素，但不删除。
6. **销毁队列**：释放队列所占用的所有存储空间。

以上就是队列的基本操作，它们是实现队列数据结构的基础。在实际应用中，我们可以根据需要选择使用哪些操作。在使用队列时，需要注意避免队列的上溢和下溢情况。上溢是指在队列满时进行入队操作，下溢是指在队列空时进行出队操作。这两种情况都需要避免。在进行入队和出队操作时，我们通常会先检查队列的状态，以确保操作的有效性。如果队列已满，我们就不能再进行入队操作；如果队列为空，我们就不能进行出队操作。这就是队列的基本操作和使用注意事项。

## 3、请解释以下代码的含义：

public interface Queue {

public int getSize();

public boolean isEmpty();

public void enqueue(Object e);

public Object dequeue() throws QueueEmptyException;

public Object peek() throws QueueEmptyException;

}

这段代码定义了一个名为 Queue 的接口，它是一种队列数据结构，通常用于在程序中存储元素。这个接口包含以下方法：

* getSize()：这个方法返回队列中的元素数量。
* isEmpty()：这个方法检查队列是否为空。如果队列为空，它返回 true，否则返回 false。
* enqueue(Object e)：这个方法将一个对象 e 添加到队列的末尾。
* dequeue()：这个方法移除并返回队列的第一个元素。如果队列为空，它会抛出一个 QueueEmptyException 异常。
* peek()：这个方法返回队列的第一个元素，但不从队列中移除它。如果队列为空，它也会抛出一个 QueueEmptyException 异常。

这个接口定义了队列数据结构的基本操作，可以被其他类实现，以提供具体的队列功能。例如，可以有一个基于数组的队列实现，或者一个基于链表的队列实现，都实现这个接口。这样，无论底层数据结构如何，使用队列的代码都可以以相同的方式工作。这是面向对象编程中的一种常见模式，称为 “接口编程”。这种方式可以提高代码的灵活性和可重用性。

## 4、请解释以下代码的含义：

public class QueueArray implements Queue {  
 private static final int CAP=7;

private Object[] elements;

private int capacity;

private int front;

private int rear;

public QueueArray() {  
 this(CAP);

}

public QueueArray(int cap){  
 capacity =cap+1;

elements=new Object[capacity];

front = rear =0;

}  
 public int getSize() {

return (rear -front+ capacity)%capacity;

}

public boolean isEmpty() {

return front==rear;

}

public void enqueue(Object e) {

if (getSize()==capacity-1) expandSpace();

elements[rear] = e;

rear =(rear+l)%capacity;

}

private void expandSpace(){  
 Object[] a = new Object[elements.length\*2];

int i=front;

int j=0;

while (i!=rear){

a[j++] = elements[i];

i=(i+1)%capacity;}

elements= a;

capacity=elements.length;

front=0;rear=j;

}

public Object dequeue() throws QueueEmptyException {

if(isEmpty())

throw new QueueEmptyException("错误:队列为空");

Object obj = elements[front];

elements[front] = null;

front=(front+1)%capacity;

return obj;

}

public Object peek() throws QueueEmptyException

{

if(isEmpty())

throw new QueueEmptyException("错误:队列为空");

return elements[front];

}

}

这段代码实现了一个基于数组的队列数据结构。下面是每个部分的详细解释：

* QueueArray 类实现了 Queue 接口，这意味着它必须提供 Queue 接口定义的所有方法。
* CAP 是队列的默认容量，如果在创建队列时没有指定容量，那么就会使用这个值。
* elements 数组用于存储队列中的元素。
* capacity 是队列的容量，即 elements 数组的长度。
* front 和 rear 分别是队列的前端和后端的索引。
* QueueArray() 是一个构造函数，它创建一个默认容量的队列。
* QueueArray(int cap) 是一个构造函数，它创建一个指定容量的队列。
* getSize() 方法返回队列中的元素数量。
* isEmpty() 方法检查队列是否为空。
* enqueue(Object e) 方法将一个元素添加到队列的后端。
* expandSpace() 方法是一个私有方法，当队列满时，它会被 enqueue(Object e) 方法调用以扩大队列的容量。
* dequeue() 方法移除并返回队列前端的元素。如果队列为空，它会抛出一个 QueueEmptyException 异常。
* peek() 方法返回队列前端的元素但不移除它。如果队列为空，它也会抛出一个 QueueEmptyException 异常。

## 5、请解释以下代码的含义：

public bool MatchBracket(char[] charlist) {

SeqStack<char> s=new SeqStack<char>(50);

int len= charlist.Length;

for (int i=0; i < len; ++i)

{ if(s.IsEmpty()) {s.Push(charlist[i]);}

else if(((s.GetTop()==' (') && (charlist[i]==')')))

||(s.GetTop()=='['&& charlist[i]==']'))

{s.Pop();}

else {s.Push(charlist[i]);}}

if(s.IsEmpty()) {return true;}

else {return false;}

}

这段代码是一个用于检查括号是否匹配的函数。它的工作原理如下：

1. 它首先创建一个名为 s 的栈，用于存储括号字符。
2. 然后，它遍历输入的字符数组 charlist。
3. 对于 charlist 中的每个字符，它执行以下操作：
   * 如果栈 s 为空，或者当前字符不与栈顶的括号匹配，它就将当前字符压入栈 s。
   * 如果当前字符与栈顶的括号匹配（即，栈顶是 ‘(’ 并且当前字符是 ‘)’，或者栈顶是 ‘[’ 并且当前字符是 ‘]’），它就从栈 s 中弹出栈顶的括号。
4. 在遍历完 charlist 后，如果栈 s 为空，说明所有的括号都已匹配，函数返回 true；否则，说明有未匹配的括号，函数返回 false。

这个函数可以用于检查包含括号的表达式（例如，算术表达式或编程语言中的代码）是否正确地使用了括号。如果所有的括号都正确地匹配了，函数就返回 true；否则，函数返回 false。这是一个常见的问题，尤其是在编程和数学中，正确的括号使用是非常重要的。这个函数提供了一种简单有效的方法来检查括号的使用是否正确。

## 6、请解释以下代码的含义：

public static void Main(String[] args){  
 SeqStack<char> s = new SeqStack<char>(50);

CSeqQueue<char> q = new CSeqQueue<char>(50);

string str=System.in.read();  
 for(int i=0; i<str.Length; ++i)  
 { s.Push(str[i]);

q.In(str[i]);}

while(!s.IsEmpty() && !q.IsEmpty())  
 { if(s.Pop()!=q.Out()) {break;}}  
 if(!s.IsEmpty()||!q.IsEmpty())  
 { System.out.println("这不是回文!");}  
 else{System.out.println("这是回文!");}

}

这段代码是用来检测一个字符串是否为**回文**的。回文是指正读反读都一样的字符串。

首先，它创建了一个字符类型的顺序栈 s 和一个字符类型的循环队列 q，都有50个空间。

然后，它从输入中读取一个字符串 str。

接着，它遍历 str 中的每个字符，将每个字符既压入栈 s，又入队列 q。

然后，它在栈 s 和队列 q 都不为空的情况下，比较并移除栈顶和队首的字符。如果这两个字符不相等，就立即停止比较。

最后，如果栈 s 或队列 q 中还有剩余的字符，那么它就输出 “这不是回文!”。否则，它就输出 “这是回文!”。

这个算法的基本思想是利用栈的后进先出（LIFO）特性和队列的先进先出（FIFO）特性，对字符串进行正反两种方式的遍历，如果两种方式遍历的结果相同，那么这个字符串就是回文。否则，就不是回文。这是一种非常巧妙的利用数据结构性质进行问题求解的方法。