线性表实现及应用

线性表提供了组织数据的一种方法，用它可以组织管理待办事项清单、礼品单、地址列表、甚至清单的清单等。这些清单为人们有条理的安排生活提供了一定的保障。所以说，线性表是一个基础性很强的数据结构。在很多的高级语言中，一般都会提供对这类数据组织方式的支持。

任务 1： 指定的 List ADT 实现

在本次实验中，主要完成的任务是：

1、为指定的 List ADT 实现三种数据结构：①使用顺序数组做为存储表示； ②使用单向链表做为存储表示； ③使用双向链表做为存储表示。不论哪种存储表示下实现的数据结构，实验中需要处理的数据类型都是 char 类型。

2、用给定的输入数据文件验证所实现的数据结构是否正确。

3、使用表格列举每种数据结构下所实现的所有方法的时间复杂度。

指定的 List 的 ADT 描述如下：

```
public interface List<T> {
        void insert(T newElement);
        /*
        * Precondition: List is not full and newElement is not null.
        * PostCondition:
        * Inserts newElement into a list after the cursor. If the list is empty, newElement is inserted as the
first(and only)element in the list.
          In either case(empty or not empty),moves the cursor to newElement.
        */
        void remove();
        /*
        * Precondition:
        * List is not empty.
        * PostCondition:
        * Removes the element marked by the cursor from a list. If the resulting list is not empty, then moves the
cursor to the element that followed the deleted element. If the deleted element was at the end of the list, then
moves the cursor to the element at the beginning of the list.
        */
        void replace(T newElement);
        /*
        * Precondition:
        * List is not empty and newElement is not null.
        * PostCondition:
        * Replaces the element marked by the cursor with newElement. The cursor remains at newElement.
        */
        void clear();
        /*
        * Precondition:
        * None
        * PostCondition:
        * Removes all the elements in a list.
        */
```

```
        boolean isEmpty();
        /*
        * Precondition:
        * None
        * PostCondition:
        * Returns true if a list is empty. Otherwise, returns false.
        */
        boolean isFull();
        /*
        * Precondition:
        * None
        * PostCondition:
        * Returns true if a list is full. Otherwise, returns false.
        */
        boolean gotoBeginning();
        /*
        * Precondition:
        * None
        * PostCondition:
        * If a list is not empty, then moves the cursor to the beginning of the list and returns true. Otherwise, returns false.
        */
        boolean gotoEnd();
        /*
        * Precondition:
        * None
        * PostCondition:
        * If a list is not empty, then moves the cursor to the end of the list and returns true. Otherwise, returns false.
        */
        boolean gotoNext();
        /*
        * Precondition:
        * List is not empty.
        * PostCondition:
        * If the cursor is not at the end of a list, then moves the cursor to the next element in the list and return true. Otherwise, returns false.
        */
        boolean gotoPrev();
        /*
        * Precondition:
        * List is not empty.
        * PostCondition:
        * If the cursor is not at the beginning of a list, then moves the cursor to the preceding element in the list and returns true. Otherwise, returns false.
        */
        T getCursor();
        /*
        * Precondition:
        * List is not empty.
        * PostCondition:
        * Returns a copy of the element marked by the cursor.
        */
        void showStructure();
        /*
        * Precondition:
        * None
        * PostCondition:
        * Outputs the elements in a list and the value of cursor. If the list is empty, outputs "Empty list". Note that this operation is intended for testing/debugging purpose only.
        */
}
```

为了方便进行测试验证，对 List 的各种操作指定了相应的命令符号，具体的符号含义如下：

+	insert
-	remove
=	replace
#	gotoBeginning
*	gotoEnd
>	gotoNext
<	gotoPrev
~	clear

假如对一个空表做如下的操作列表，并且假设该线性表中插入的元素类型为 char 类型，那么下面表格中将示例对命令符号使用的运行结果：

命令行内容	执行结果（调用 List 的 showStructure 行为的输出内容）
+a +b +c +d	a b c d 3
# > >	a b c d 2
* < <	a b c d 1
-	a c d 1
+e +f +f	a c e f f d 4
* -	a c e f f 0
-	c e f f 0
=g	g e f f 0
~	Empty list -1

实验完成之后，必须通过实验中提供的测试用例。通过对测试用例的运行检查所撰写的代码功能是否正确。测试用例中的每一行的内容都类似于上表中的每一行“命令行内容”列中所指示的内容。要求每执行一行，就调用 List 接口中的 showStructure 行为，用以验证该命令行的执行是否正确。实验包里包括了个文件，一个是“list_testcase.txt”，其内包含了测试用例；另一个是“list_result.txt”，其内包含了对应测试用例的运行结果。

任务 2：栈和递归之间的关系

递归是一种解决很多复杂问题最简单的思想方法，而任何编程语言对递归程序的支持都是通过栈实现的。现在通过字母全排列问题的实践去理解和认知递归和栈之间的关系。具体的任务如下：

1、给定一个字符串，设计一个算法完成对该字符串的一个全排列的输出。比如：字符串“abc”，输出结果应该是“abc”、“acb”、“bac”、“bca”、“cab” 和 “cba”。

要求：①使用递归思想，编写一个函数 permutationByRecursion，该函数用来生成给定的字符串的全排列结果。②使用栈数据结构，将①中编写的算法转换成非递归函数，函数名为：permutationByNoRecursion。(利用课堂上讲解的分解 Hanoi 塔的任务方式)，该函数所使用的栈数据结构只能使用自己定义的栈数据结构，不能使用任何语言提供的栈。

2、在进入递归之前如果增加一些约束条件，则可以删减一些排列的输出，从而达到特别的目

的（类似于算法设计中的“剪枝法”）。下面提出两个特别要求，请仔细思考如何修改 1）中的递归程序从而达到要求。

①变形 1：当字符串中出现相同字符时，只给出完全不一样的排列组合，比如：字符串“aac”，输出结果应该是：“aac”、“aca”、“caa”。

②变形 2：输出长度为 n 的字符串中取 k 个字符构成的所有全排列，即完成公式：$p(n,k)=\frac{n!}{(n-k)!}$ 所表示的意义。比如字符串“abcd”，当 k = 2 时，应该输出的字符串为：“ab”、“ba”、“ac”、“ca”、“ad”、“da”、“bc”、“cb”、“bd”、“db”、“cd”、“dc”。

任务 3：创建一个可自动调整空间大小的 Queue 数据结构

教材中的 Queue 实现分别采用了两种数据组织方式：一是将普通数组构成逻辑循环的方式，二是采用普通的单向链表。当选择用数组做为队列的存储标识时意味着使用前需要预先估计队列的最大容量。在 Java 中，数组一旦创建，其大小是无法改变的，因此队列使用的空间只能是这个最大容量的一部分，空间利用率将会比较低。而且一旦预先估计有误，将会造成队列溢出，对应用带来致命一击。为了解决这个问题，我们将尝试使用动态调整的方式来完成数组空间大小的有序变化，调整方案很简单，具体步骤如下：

0、使用 N 表示当前队列的最大容量（即最多能够存储的队列元素个数）；

1、 初始情况下，N=1（注：使用循环数组实现技术，则真实的数组大小应该是 N+1）；

2、 当入队时队列满，那么就重新生成一个容量为 2N+1 的数组，将原数组中的 N 个元素拷贝到新数组中，让新数组成为当前队列的存储表示；

3、 当出队时，如果当前队列中的元素个数是 N 的四分之一时，则重新生成一个容量为 N/2 的数组，将原数组中的 N 个元素拷贝到新数组中，让新数组成为当前队列的存储表示。

任务要求：

1、 按照上述方式实现一个可以动态调整的循环数组实现队列数据结构；

2、 该队列数据结构必须支持泛型，具体实现的行为如下：

public class ResizingQueue<T>		
	ResizingQueue()	构造方法，无需指定任何初始大小
void	enqueue(T element)	将元素 element 入队，如果队列满，则需要完成空间大小的调整
T	dequeue()	从队列中将队头元素删除并返回，如果队列的元素个数是当前容量的 1/4，那么完成空间大小的调整
int	size()	返回当前队列中的实际元素个数
String	toString()	将当前队列中的元素和队列中的结构信息按照格式要求生成一个字符串

3、对 toString 方法实现的字符串内容的要求如下：

```
[133 735 309 368 816 419 282 965]
elements: 8 size:16
[510 247 916 400 774  ... 422 15 282 887 503]
elements: 33 size:64
```

第 1 行为队列中的元素内容，用方括号括起来，每个元素用空格分隔。当队列中的元素个数不大于 20 个时，全部显示；否则显示前五个和后五个，中间用“...”分隔。

第 2 行按照样例格式输出当前队列中的元素个数和当前队列的最大容量值。
每一次 toString 调用就显示上面两行内容。

4、当执行入队操作时，当 element 元素为 null 时，抛出 IllegalArgumentException；当执行出队操作时，当队列为空，则抛出 java.util.NoSuchElementException。

5、为了验证所创建的 ResizingQueue 数据结构的正确性，提供了 test1000.txt 和 test5000.txt 的测试用例，同时也给出了相对应的执行结果文件 result1000.txt 和 result5000.txt，供同学们参考。在测试数据文件中当读到一个数字时，代表执行入队操作，入队元素即为该数字；当读到一个 ‘-’ 符号时，代表执行出队操作；当读到一个 ‘？’ 符号时，代表执行 toString 方法。result***.txt 文件中的内容是读到 ‘？’ 符号时的执行结果。

任务 4：基数排序

利用任务 3 实现的队列实现对某一个数据序列的排序(采用基数排序)，其中对待排序数据有如下的要求:

1、当数据序列是整数类型的数据的时候，数据序列中每个数据的位数不要求等宽，比如: 1、21、12、322、44、123、2312、765、56

2、当数据序列是字符串类型的数据的时候，数据序列中每个字符串都是等宽的，比如: "abc","bde","fad","abd","bef","fdd","abe"

注：radixsort1.txt 和 radixsort2.txt 是为上面两个数据序列提供的测试数据。