数据结构与算法
作业报告

第二次

姓名

班级

学号

电话

Email

日期

目录

任务 1： 为指定的 List ADT 实现各种数据结构

在本次实验中，主要完成的任务是：

1、为指定的 List ADT（该 ADT 的具体要求请参见文件 List.java）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 Character 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。

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

假如对一个初始化空间大小为 16 的空表按顺序依次执行每一行的命令列表，并且假设该线性表中插入的元素为 Character 类型，那么表格中第二列即为执行相对应的第一列中的命令列表后调用 showStructure 方法的运行结果：

命令行内容	执行结果（调用 List 的 showStructure 方法）[1]
+a +b +c +d	a b c d {capacity = 16, length = 4, cursor = 3}
# > >	a b c d {capacity = 16, length = 4, cursor = 2}
*** < <**	a b c d {capacity = 16, length = 4, cursor = 1}
-	a c d {capacity = 16, length = 3, cursor = 1}
+e +f +f	a c e f f d {capacity = 16, length = 6, cursor = 4}
*** -**	a c e f f {capacity = 16, length = 5, cursor = 0}
-	c e f f {capacity = 16, length = 4, cursor = 0}
=g	g e f f {capacity = 16, length = 4, cursor = 0}
~	Empty list {capacity = 16, length = 0, cursor = -1}

实验完成之后，必须通过实验中提供的测试用例。借助测试用例的运行结果，用来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。每行“命令行内容”都不是独立的，是针对同一个 List 类型的对象实例运行的结果。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。

[1] 如果 List 实现的存储方式为链式结构，那么 capacity 的值即为真实的链表中的结点个数。实验文件中仅提供了基于数组实现的测试用例的运行结果。

设计：

一．编写 SequenceList 类：

1.注意 SequenceList 类应接于 List 接口，并重写 List 接口的所有方法；

2.准备工作：定义一个 Character 类型的数组 list，用于存储元素；定义一个整数类型参数 defaultSize 作为 list 的大小（容量）；定义一个整数类型变量 length 用于查看 list 当前不为 null 的元素的数量；定义一个整数类型变量 curr 作为指针指向某一位置的元素（后续诸多方法都在该指针指向的元素上进行）；编写无参构造器，给 list 数组的大小赋值为 defaultSize，并初始化 length 和 curr 的值为 0；

3.重写 insert（Character newElement）方法：

（1）使用 if 语句判断插入元素不为空且 list 未满、不为空时继续执行；

（2）使用 for 循环从最后一个元素的后一位（null）遍历至 curr 指向的元素的后一位元素，并依次将上述遍历到的元素为其赋值为前一个元素的值，以达到将 curr 后的元素都向后移动一位的效果；

（3）将 list 在 curr+1 处的元素赋值为 newElement，即待插入的元素的值；

（4）顺序表长度 length+1，当前指针 curr+1；

（5）使用 else if 语句判断 length=0 的情况时：将 length 赋值为 1，令 list[0]=newElement;

4.重写 remove（）方法：

（1）用 if 语句判断表不为空时才继续执行 remove 方法的操作;

（2）调用 for 循环，遍历 curr 指向的元素至 list 的最后一个不为 null 的元素，并依次将 list 在遍历的元素的位置的值赋值为下一个元素的值，以此达到覆盖效果，将 curr 位置原有的值删除;

（3）用 if 语句判断删除的元素是 list 最后一个元素时，调用 gotoBeginning（）方法将 curr 的值赋值为 0;

（4）List 的长度 length-1;

5.重写 replace（Character newElement）方法：用 if 语句判断 list 不为空且 newElement 不为 null 时，将 list 在 curr 位置的元素赋值为 newElement;

6.重写 clear（）方法：

（1）调用 for 循环，从位置 0 开始遍历 list 的所有元素，并将每一个元素都赋值为 null;

（2）将 length 和 curr 的值都赋值为 0;

7.重写 isEmpty（）方法：用 if 语句判断 length 等于 0 时返回 true，其余情况均返回 false;

8.重写 ifFull（）方法：用 if 语句判断 list 的长度是否等于 defaultSize，若等于则返回 true，否则返回 false；

9.重写 gotoBeginning（）方法：用 if 语句判断 list 不为空时，将 curr 赋值为 0 并返回 true，否则返回 false；

10. 重写 gotoEnd（）方法：用 if 语句判断 list 不为空时，给 curr 赋值为 length-1,并返回 true，否则返回 false；

11. 重写 gotoNext（）方法：用 if 语句判断 list 不为空且 curr 的值不是 length-1 时，将 curr 的值+1 并返回 true，否则返回 false；

12. 重写 gotoPrev（）方法：用 if 语句判断 list 不为空且 curr>0 时，将 curr 的值-1 并返回 true，否则返回 false；

13. 重写 getCursor（）方法：用 if 语句判断表为空时返回 null，表不为空时返回 list 在 curr 位置处的元素的值；

14. 重写 showStructure（PrintWriter pw）方法：

（1）调用 for 循环依次遍历 list 中的每一个元素并对 pw 对象调用 print 方法将所有元素输出至指定的 txt 文本中；

（2）使用 pw.flush（）方法刷新文本，以保证输出顺利完成；

（3）调用 pw.println（）方法按照要求格式输出顺序表的容量大小、长度即当前指针指向的位置；

15. 重写 moveToNth（int n）方法：

（1）用 if 语句判断 n<length 时，定义一个 Character 类型变量 movedElement，使其等于指针 curr 指向元素的值；

（2）调用 for 循环，遍历 list 表中从 curr 位置至最后一个位置的元素并将每一个遍历到的元素的值赋值为其下一个元素的值；

（3）调用 for 循环，后序遍历 list 中最后一个元素至第 n+1 个元素，使遍历到的元素的值等于其前一个元素的值；

（4）给 list[n]赋值为 movedElement，并给 curr 赋值为 n。

16. 重写 find（Character searchElement）方法：

（1）用 if 语句判断 list 为空时直接返回 false；

（2）List 不为空时，先定义一个整数类型变量 index，其值为 curr 的值，用于后续循环遍历 list 并不改变 curr 的值；

（3）使用 while 循环，条件为 index<length，用 if 语句 equals（）方法判断 list 在 index 处的元素是否与输入的 searchElement 相同，若相同则将 index 的值赋给 curr 并返回 true，否则使 index+1 继续进行循环直至不满足循环条件；若跳出循环还未找到该元素，则使 curr 的值为 length-1（指向 list 中最后一个元素），并返回 false；

17. 编写 printlist（）方法（控制台输出）：用 if 语句判断 list 是否为空，为空则输出“Empty list ”，否则调用 for 循环遍历 list 中的每一个元素并将其输出；最后按照与

showStructure（）方法相同的格式输出 list 的容量大小、长度、当前指针指向位置；

二．编写 LinkList 类

1.LinkList 类接于 List 接口，应重写 List 接口的所有方法；

2.编写内部类 Node：

（1）定义 Character 类型变量 element，作为当前 Node 对象的值；定义 Node 对象 next，储存当前 Node 对象的下一个结点，若当前对象为链表的最后一个元素时 next 为 null；

（2）编写有参构造器 Node（Character element，Node next）：使用 this 指针分别给 element 与 next 赋值；

3.定义 Node 类对象 head 作为 list 的头结点；定义整数类型变量 N 记录链表长度；定义整数类型变量 curr，作为指针指向 list 的某一变量，方便后续各种方法对其处理；

4.编写 LinkList 类的无参构造器：初始初始化头结点 head 为 Node（null，null），初始化链表长度 N 和当前指针 curr 的值为 0；

5.重写 insert（Character newElement）方法：

（1）用 if 语句判断链表是否为空，为空时直接给头结点 head 赋值为 newElement；

（2）链表不为空时定义一个 Node 对象 currNode，使其等于 head，调用 for 循环遍历 curr 次，每次遍历都使

currNode=currNode.next，即可找到 curr 指向的结点，定义 Node 类对象 newNode=new Node（newElement, currNode.next），令 currNode.next=newNode，即可成功插入 newNode，再使 curr 的值+1;

（3）插入成功后使链表长度 N 的值+1;

6.重写 remove（）方法:

（1）用 if 语句判断链表是否为空，为空时不进行 remove 操作;

（2）链表不为空时用 if 语句判断 curr 的值是否为 0，若为 0 则令头结点 head=head.next 即可删除原有头结点;若 curr 不为 0 则定义 Node 类对象 pre=head，调用 for 循环遍历 curr-1 次使 pre 等于 curr 位置前一个结点，定义 curr 指向的结点为 currNode=pre.next，使 pre.next=currNode.next 即可删除 currNode 结点;

（3）用 if 语句判断删除的结点是否为 list 中的最后一个结点，若是则给 curr 赋值为 0;

（4）删除成功后使链表长度 N-1;

7.重写 replace（Character newElement）方法:

（1）用 if 语句判断 list 不为空且 newElement 不为 null 才进行 replace 操作;

（2）定义 Node 类对象 changeNode=head，调用 for 循环遍历 curr 次以使 changeNode 变为 curr 指向的结点;

（3）使changeNode.element=newElement即可完成替换。

8.重写 clear（）方法：将头结点 head、链表长度 N、当前指针 curr 均初始化即可；

9.重写 isEmpty（）方法：用 if 语句判断链表长度 N 是否为 0，若为 0 则返回 true，否则返回 false；

10. 重写 isFull（）方法：定义 Node 类对象 node=head，用 for 循环遍历 N 次，每次都使 node=node.next，遍历结束后 node 即为 list 最后一个结点，用 if 语句判断其是否为 null，不为 null 则返回 true，否则返回 false；

11. 重写 gotoBeginning（）方法：用 if 语句判断链表是否为空，不为空则给 curr 赋值为 0 并返回 true，否则返回 false；

12. 重写 gotoEnd（）方法：用 if 语句判断链表是否为空，不为空时使 curr=N-1 并返回 true，否则返回 false；

13. 重写 gotoNext（）方法：

（1）用 if 语句判断链表不为空时，定义 currNode=head，调用 for 循环遍历 curr 次使 currNode 等于 curr 指向的结点，用 if 语句判断 currNode.next 是否为 null，不为 null 则使 curr 的值+1 并返回 true，否则返回 false；

14. 重写 gotoPrev（）方法：用 if 语句判断链表不为空且 curr 不为 0 时，令 curr 的值-1 并返回 true，否则返回 false；

15. 重写 getCursor（）方法：定义 Node 类对象 currNode=head，调用 for 循环遍历 curr 次使其等于 curr 指向的结点，并将其值返回；

16. 重写 showStructure（PrintWriter pw）方法：

（1）定义 Node 类对象 currNode=head，使用 for 循环遍历 N 次，每次都对 pw 对象使用 print（）方法以在 txt 文本中输出链表每一个结点的值；

（2）对 pw 对象使用 println（）方法按格式输出链表容量（等于长度）、长度和当前指针的值；

（3）对 pw 对象调用 flush（）方法刷新文本。

17. 重写 moveToNth（int n）方法：

（1）用 if 语句判断 N>n 时继续执行 moveToNth 操作；

（2）定义 Node 类对象 preNode=head，调用 for 循环遍历 curr-1 次使 preNode 等于 curr 指向结点的前一节点；定义需要换位置的结点为 moveNode=preNode.next，令 preNode.next=moveNode.next 即可删除 moveNode；

（3）定义 NthNode=head，用 for 循环遍历 n 次使其指向第 n-1 个结点，并令 moveNode.next=NthNode.next，NthNode.next=moveNode，并将 curr 赋值为 n-1.

18. 重写 find（Character searchElement）方法：

（1）定义 Node 类对象 findNode=head，定义整数类型变量 index=0；

（2）调用 while 循环，条件为 index<curr，使每一次循环 index 都+1，并使 findNode=findNode.next，循环结束后 findNode 即为 curr 指向的结点；

（3）调用 while 循环，条件为 findNode.next 不为 null，调用 if 语句判断 findNode 的值是否等于 searchElement，若是则返回 true，否则令 findNode=findNode.next 并使 curr+1 以遍历下一节点；

（4）若遍历结束还未找到相同结点，则返回 false；

19. 编写 printlist（）方法（控制器输出）：输出内容与 showStructure 方法一致，仅将其输出从 txt 文本改为控制器即可。

三．编写 TwoWayLinkList 类

1.使 TwoWayLinkList 类接于 List 接口，并重写 List 接口的所有方法；

2.编写内部类 Node 类：定义 Character 类变量 element 作为 Node 类对象的值；定义 Node 类变量 pre，其为当前 Node 对象的前一 Node 对象；定义 Node 类变量 next，其为当前 Node 对象的后一 Node 对象；编写 Node 类有参构造器 Node（Character element，Node pre，Node next），分别为当前 Node 对象的 element、pre、next 赋值；

3.定义 Node 类对象 head 作为双向链表的头结点，定义整数类型变量 N 用于记录链表长度，定义整数类型变量 curr

作为指针指向链表中的某一结点，后续诸多方法的实现将在1 此结点上进行；

4.编写 TwoWayLinkList（）无参构造器：初始化 head 结点=new Node（null，null，null），初始化 N 与 curr 的值为 0；

5.重写 insert（Character newElement）方法：

（1）用 if 语句判断待插入元素不为空则执行下述：

（2）用 if 语句判断链表是否为空，为空则定义 Node 类对象 newNode=new Node（newElement，null，null），并令头结点 head=newNode；

（3）若链表不为空，定义 Node 类对象 currNode=head，调用 for 循环使 currNode 等于 curr 指向的结点，定义待插入结点为 newNode=new Node（newElement，currNode，null），用 if 语句判断 currNode 的下一节点是否为空，若不为空则使其的 pre 为 newNode，并使 newNode.next=currNode.next；

（4）使 currNode.next=newNode，并使 curr+1；

（5）使链表长度 N+1；

6.重写 remove（）方法：

（1）用 if 语句判断链表不为空时执行下述：

（2）定义 Node 类对象 preNode=head，用 for 循环使 preNode 等于 curr 指向的结点，用 if 语句判断 preNode 的下一节点是否为 null，为 null 则使用 clear（）方法删除结点；

（3）若 preNode.next 不为 null，定义其为 currNode，用 if 语句判断 currNode 是否为 null，为 null 则使 preNode.next=null 完成删除 currNode 的操作，并赋值 curr=0；若不为 null 则用 if 语句判断 curr 是否指向头结点，若非则令 preNode.next=currNode.next, currNode.next.pre=preNode，若是则令 head=head.next, head.pre=null;

（4）令链表长度 N-1;

7.重写 replace（Character newElement）方法:

（1）用 if 语句判断链表不为空且 newElement 不为 null 时执行下述:

（2）用 if 语句判断 curr 是否指向头结点，若是则将 head 的 element 赋值为 newElement，若非则定义 Node 类对象 currNode 并使其等于 curr 指向的结点，将其的 element 赋值为 newElement;

8.重写 clear（）方法：初始化 head、N、curr;

9.重写 isEmpty（）方法：用 if 语句判断链表长度 N 是否为 0，是则返回 true，否则返回 false;

10.重写 isFull（）方法：定义 Node 类对象 node=head，用 for 循环遍历整个链表，若没有结点为 null 则返回 true，否则返回 false;

11.重写 gotoBeginning（）方法：用 if 语句判断链表是否为空，不为空则令 curr=0 并返回 true，否则返回 false；

12.重写 gotoEnd（）方法：用 if 语句判断链表是否为空，不为空则令 curr=N-1 并返回 true，否则返回 false；

13.重写 gotoNext（）方法：

（1）用 if 语句判断链表不为空时执行下述：

（2）定义 Node 类对象 currNode，使其等于 curr 指向的结点，用 if 语句判断 currNode.next 是否为 null，若非则 curr+1 并返回 true，否则返回 false；

14.重写 gotoPrev () 方法：用 if 语句判断链表不为空且 curr 不指向头结点时，curr-1 并返回 true，否则返回 false；

15.重写 getCursor（）方法：定义 Node 类对象 currNode 并使其等于 curr 指向的结点，返回 currNode.element；

16.重写 showStructure（PrintWriter pw）方法：与单向链表中的 showStructure 方法一致；

17.重写 moveToNth（int n）方法：

（1）用 if 语句判断链表长度大于 n 时执行下述：

（2）定义 Node 类对象 preNode，使其等于 curr 指向结点的前一节点，定义 Node 类对象 moveNode，使其等于 preNode.next，令 preNode.next=moveNode.next 完成对 moveNode 的删除操作；

（3）定义 Node 类对象 NthNode，使其等于第 N-1 个结点，并将 curr 指向该结点，使用 insert 方法将 moveNode 插入；

18.重写 find（Character searchElement）方法：与单向链表的 find 方法如出一辙；

19.编写 printlist（）方法：令控制器输出同 showStructure 方法在 txt 文本输出的内容。

四．编写 Test 类以进行对前三个类的测试

1.编写 output（String sourceFilePath，Sting outputFilePath）方法，用于读取测试文本并输出结果：

（1）使用 try 方法：

（2）定义 BufferedReader 类对象 reader=new BufferedReader（new FileReader（sourceFilePath））；定义 PrintWriter 类对象 pw=new PrintWriter（new FileWriter（outputFilePath））；

(3)定义 SequenceList 类对象 list;定义 LinkList 类对象 list;定义 TwoWayLinkList 类对象 list，上述三行代码的其中之一用于运行，其余两行为注释（测试哪个就将另外两个注释）；

（4）定义 String 类对象 line 用于记录测试文本每一行的字符；

（5）调用 while 循环，条件为（line=reader.readLine（））不等于 null，定义 String 类对象 data=“ ”，将 line 的值赋给 data；

（6）用 for 循环遍历 data 中的每一个字符，使用 switch 方法对每一个字符进行处理：‘+’则调用 insert（data.charAt（i+1））方法，‘-’则调用 remove 方法，‘=’则调用 replace（data.charAt（i+1））方法……其中 i 为遍历到的位置；

（7）调用 list.printlist 方法和 list.showStructure（pw）方法分别在控制台和指定 txt 文本进行输出；

（8）使用 catch（IOException e），调用 e.printStakeTrace（）。

2.编写 main 函数：

（1）给 main 函数加上“throws Exception”后缀；

（2）定义两个 String 类对象 sourceFilePath 和 outputFilePath，其分别为输入文件的名称和输出文件的名称，在此测试中分别为“list_testcase.txt”和“output.txt”；

（3）调用 output（sourceFilePath，outputFilePath）方法进行输出。

运行结果展示:

下面是部分测试代码，第一张图为老师给的测试结果（部分），第二张图为本人代码运行结果（部分），可看出此部分代码完全相同，代码成功运行且未出错:

```
V J U t v O d a w c i b L {capacity = 512, length = 13, cursor = 1}
b U n w I y l B h e a t e g m C g v j {capacity = 512, length = 19, cursor = 2}
a l o b m a z N J b b j s U n w I y l B h e a t e g L w w {capacity = 512, length = 29, cursor = 0}
n g h K b z s r k i X W p s l r y h o b m a z N J b b j s U n w I y l B h e a t e g L w j g {capacity = 512, length = 46, cursor = 0}
u d h S o y e a s c b z x j R B w q {capacity = 512, length = 18, cursor = 17}
x v h S o y e a s c b z x j R B w P t {capacity = 512, length = 19, cursor = 18}
G n f e c P r s {capacity = 512, length = 8, cursor = 1}
Z F n o k B y z w D l p h b f e c P r s n y g t t h q V p {capacity = 512, length = 29, cursor = 28}
I D o S O n t w p n n y e o v b k B y z w D l p h b f e c P r s n y g t t h V C l t c a g v m {capacity = 512, length = 47, cursor = 45}
T n p g A Q m n C l z m h x {capacity = 512, length = 14, cursor = 13}
R r c J q p u V i j j t Q y e c M f u t w n p g A Q m n C l z m h x e r v q H r j {capacity = 512, length = 41, cursor = 7}
s d v y e a o u x j {capacity = 512, length = 10, cursor = 9}
```

```
1  V J U t v O d a w c i b L {capacity = 512, length = 13,cursor = 1}
2  b U n w I y l B h e a t e g m C g v j {capacity = 512, length = 19,cursor = 2}
3  a l o b m a z N J b b j s U n w I y l B h e a t e g L w w {capacity = 512, length = 29,cursor = 0}
4  n g h K b z s r k i X W p s l r y h o b m a z N J b b j s U n w I y l B h e a t e g L w j g {capacity = 512, length = 46,cursor = 0}
5  u d h S o y e a s c b z x j R B w q {capacity = 512, length = 18,cursor = 17}
6  x v h S o y e a s c b z x j R B w P t {capacity = 512, length = 19,cursor = 18}
7  G n f e c P r s {capacity = 512, length = 8,cursor = 1}
8  Z F n o k B y z w D l p h b f e c P r s n y g t t h q V p {capacity = 512, length = 29,cursor = 28}
9  I D o S O n t w p n n y e o v b k B y z w D l p h b f e c P r s n y g t t h V C l t c a g v m {capacity = 512, length = 47,cursor = 45}
10 T n p g A Q m n C l z m h x {capacity = 512, length = 14,cursor = 13}
11 R r c J q p u V i j j t Q y e c M f u t w n p g A Q m n C l z m h x e r v q H r j {capacity = 512, length = 41,cursor = 7}
12 s d v y e a o u x j {capacity = 512, length = 10,cursor = 9}
```

总结:

一.以下为每种方法的时间复杂度:

1.顺序表（n 为顺序表长度）:

简化	顺序表	时间复杂度
+	insert	O(n)
-	remove	O(n)
=	replace	O(1)
#	gotoBeginning	O(1)
*	gotoEnd	O(1)
>	gotoNext	O(1)
<	gotoPrev	O(1)
~	clear	O(1)
	isEmpty	O(1)
	isFull	O(1)
	getCursor	O(1)
	showStructure	O(n)
	moveToNth	O(n)
	find	O(n)

2. 单向链表（n 为链表长度，curr 为当前指针指向位置）：

简化	单向链表	时间复杂度
+	insert	O(curr)
-	remove	O(curr)
=	replace	O(curr)
#	gotoBeginning	O(1)
*	gotoEnd	O(1)
>	gotoNext	O(curr)
<	gotoPrev	O(1)
~	clear	O(1)
	isEmpty	O(1)
	isFull	O(1)
	getCursor	O(curr)
	showStructure	O(n)
	moveToNth	O(n)
	find	O(n)

3. 双向链表（n 为链表长度，curr 为当前指针指向位置）：

简化	双向链表	时间复杂度
+	insert	O(curr)
-	remove	O(curr)
=	replace	O(curr)
#	gotoBeginning	O(1)
*	gotoEnd	O(1)
>	gotoNext	O(curr)
<	gotoPrev	O(1)
~	clear	O(1)
	isEmpty	O(1)
	isFull	O(1)
	getCursor	O(curr)
	showStructure	O(n)
	moveToNth	O(n)
	find	O(n)

任务 2： 创建一个可自动调整空间大小的 List 数据结构

观察任务 1 中基于数组实现的线性表的测试用例的运行结果，发现大部分时候空间的使用率是不高的（length 和 capacity 的比值反映了这一事实），而且还存在有空间不够用的例外发生。当然，基于链式存储实现的线性表则不存在此类问题。为了解决空间利用率以及空间不够用的问题，任务 2 将使用动态调整的方式改善数组空间的大小，方案可以有很多种，但在本次实验中将采用如下的调整方案，具体步骤如下：

① 使用 capacity 表示当前线性表的最大容量（即最多能够存储的线性表元素个数）；

② 初始情况下，capacity=1；

③当插入元素时线性表满，那么就重新生成一个容量为 2*capacity 的数组，将原数组中的 capacity 个元素拷贝到新数组中，让新数组成为当前线性表的存储表示；

④ 当删除元素之后，如果当前线性表中的元素个数 length 是 capacity 的四分之一时，则重新生成一个容量为 capacity/2 的数组，将原数组中的 length 个元素拷贝到新数组中，让新数组成为当前线性表的存储表示。

如果基于数组存储表示的线性表按照如上的方式完成空间的动态调整，那么构造方法中就不需要再指定初始空间的大小了。假如继续使用任务 1 中的示例数据，则运行结果如下表所示：

命令行内容	执行结果（调用 L 的 showStructure 方法）
+a +b +c +d	a b c d {capacity = 4, length = 4, cursor = 3}
# > >	a b c d {capacity = 4, length = 4, cursor = 2}
* < <	a b c d {capacity = 4, length = 4, cursor = 1}
-	a c d {capacity = 4, length = 3, cursor = 1}
+e +f +f	a c e f f d {capacity = 8, length = 6, cursor = 4}
* -	a c e f f {capacity = 8, length = 5, cursor = 0}
-	c e f f {capacity = 8, length = 4, cursor = 0}
=g	g e f f {capacity = 8, length = 4, cursor = 0}
~	Empty list {capacity = 1, length = 0, cursor = -1}

该任务中需要完成的工作如下：

① 按照任务 1 中的 List 接口定义，实现一个 ResizingAList 线性表，数组空间的调整方案如该任务中所描述的；

② 继续使用“list_testcase.txt”进行测试，并将结果中的每行运行结果中当前线性表的空间使用率和任务 1 中的空间使用率用图的方式展示其变化过程。

设计：

一．编写 ResizingAList 类

1.ResizingAList 类应实现 List<Character>接口，并重写 List 接口的所有方法。

2.定义 ResizingAList 类中储存元素的数组、数组长度、当前指针指向、容量，其中前三者与 SequenceList 相同，容量命名为 capacity，其初始值为 1；

3.编写无参构造器，初始化 list 使其容量等于 capacity（当前为 1），初始化记录数组长度的 length 为 0，初始化当前指针为 0;

4.编写 resize（int newSize）方法以实现对数组容量的更新:

（1）令 capacity 等于 newSize;

（2）定义一个临时数组 temp，使其等于 list;

（3）让 list 等于新创建的数组，其大小为 capacity;

（4）调用 for 循环将 temp 中的值拷贝到 list 中。

5.重写 insert（Character newElement）方法:

（1）用 if 语句判断插入元素不为 null 且数组长度不为 0 时：用 if 语句判断数组长度是否等于容量，若是则调用 resize 方法将容量扩大至两倍，若非则如 SequenceList 中的 insert 方法实行相同操作;

（2）用 else if 判断数组长度为 0 时:调用 resize 方法令数组容量为 2，再赋值 length 为 1，令 list[0]=newElement。

6.重写 remove（）方法:

（1）将 SequenceList 中的 remove 方法拷贝到该 remove 方法中;

（2）在方法最后用 if 语句判断 length 是否小于 capacity 的四分之一，若是则调用 resize 方法将容量缩小至原来的一半;

7.重写 clear（）方法：令 list 等于 null，将 capacity、length、curr 初始化；

8.其余方法均与原 SequenceList 中的方法书写相同，仅需将 SequenceList 方法中的 defaultSize 更改为 capacity 即可；

二．编写 Test 类以完成测试

1.拷贝任务一中的 Test 类中的 main 方法，仅需对 output 方法进行修改即可：

（1）在 output 方法内定义三个 double 类型数组 X、Y1、Y2，数组的元素分别用作图形的 X 轴坐标、记录固定容量顺序表的空间使用率、记录优化容量顺序表的空间使用率；再定义一个初始值为 1 的整数类型 num，将其值赋给 X，每读取一行文本就使其值加 1，以此完成对图形 X 轴坐标对应数组的赋值；

（2）类似于任务一中的 output 方法内的 try 方法，创建 SequenceList 对象并对其实现类似任务一该方法中的操作，每处理完一行数据就将 length 除以容量的值赋给 Y1[num-1]，完成对固定容量表空间使用率 Y 轴赋值；

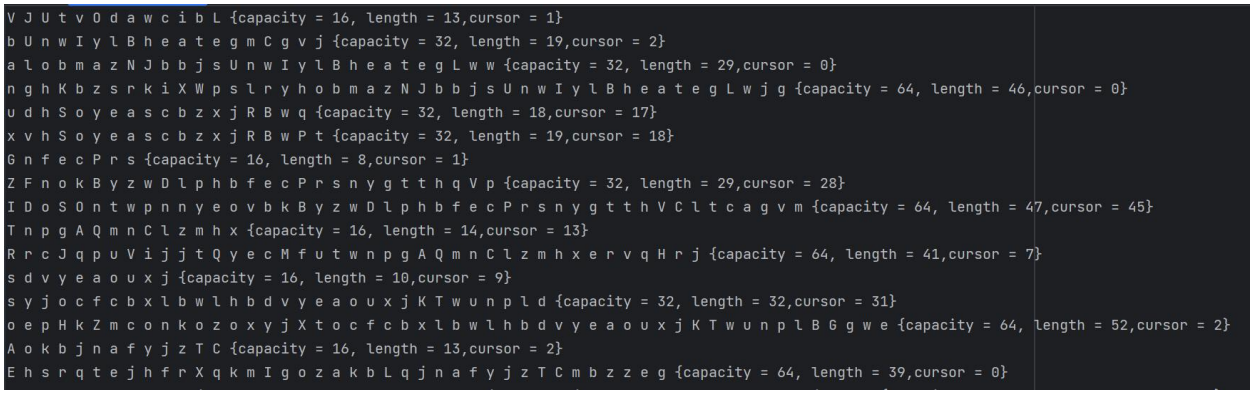
（3）完成对 SequenceList 对象的处理后将 num 的值初始化为 1；

（4）同理处理 ResizingAList 对象，对 Y2 进行赋值；

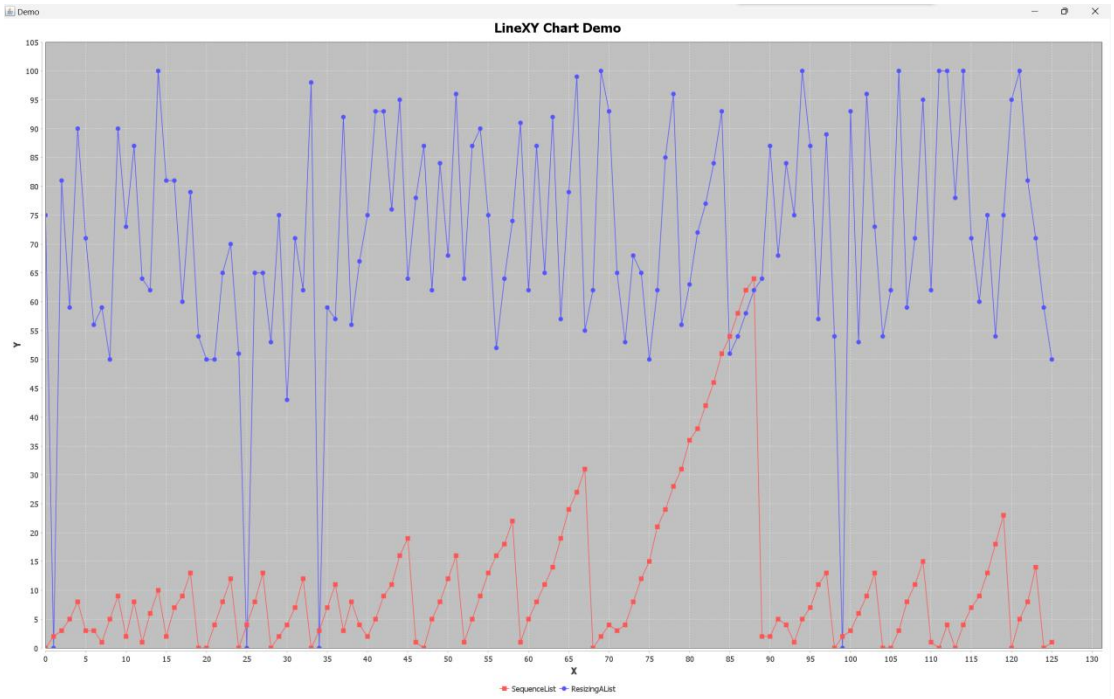
（5）方法最后创建 LineXYDemo 对象 demo，调用 demo.pack（）方法和 demo.setVisible（true）方法，完成任务。

运行结果展示:

1.下述为运行 Test 类后在 output 文本中的部分输出，可看出 length 与 curr 的值均与任务一及老师给的“result.txt”文件相同，而 capacity 经过容量更改，不会超过 length 的两倍也不会低于 length。



2. 下述为任务二增加了更改容量功能的顺序表与任务一的顺序表读取同一文本后的空间使用率比（蓝色为更改后的，红色为固定容量）：



总结与收获：

通过上述图表不难发现，通过动态调整的方式改善顺序表的容量后，其空间使用率相比固定容量的顺序表大大提高，且绝大部分都集中在 50%以上；对于固定容量的顺序表，其空间使用率取决于设定的固定容量，容量较大时其空间使用率普遍较低。

任务 3：栈

众所周知，栈虽然是一个操作受限的线性表，但是其用途却很广泛，栈的数据结构实现非常简单，因此我们只从应用层面熟悉栈。请完成下面三个子任务。

①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。请利用课堂上讲解的“Hanoi 塔”问题的非递归转化方法完成对递归快速排序的非递归转化。

②算术混合运算表达式的计算。表达式不仅能处理整数，还需要处理小数。表达式中涉及的运算符包括+、-、*、/、^(指数)。表达式可以包含括号（只包含圆括号）嵌套，因此要处理括号匹配失败的情形。

③当我们在使用很多软件时都有类似“undo”功能，比如 Web 浏览器的回退功能、文本编辑器的撤销编辑功能。这些功能都可以使用 Stack 简单实现，但是在现实中浏览器的回退功能也好，编辑器的撤销功能也好，都有一定的数量限制。因此我们需要的不是一个普通的 Stack 数据结构，而是一个空间有限制的 Stack，虽然空间有限，但这样的 Stack 在入栈时从不会溢出，因为它会采用将最久远的记录丢掉的方式让新元素入栈，也就是说总是按照规定的数量要求保持最近的历史操作。比如栈的空间是 5，当 a\b\c\d\e 入栈之后，如果继续让元素 f 入栈，那么栈中的元素将是 b\c\d\e\f。请设计一个满足上面要求的 LeakyStack 数据结构，要求该数据结构的每一个操作的时间复杂度在最坏情形下都必须满足 O(1)。

设计：

一．快速排序的非递归转化

编写 Quick 类：

1.编写 exchange（Comparable [] numbers，int i，int j）方法，用于交换 numbers 在 i 处元素和 j 处元素的位置：定义临时变量 temp，令 temp 等于 numbers[i]，令 numbers[i]等于 numbers[j]，令 numbers[j]等于 temp，完成交换。

2.编写 less（Comparable one，Comparable other）方法，用于判断 one 和 other 的大小：若 one 更小则返回 true，否则返回 false；

3.编写 printout（Interger[] nums）方法，用于输出数组：调用 for 循环遍历 nums 的每一个元素并将其输出，最后再进行一次换行操作。

4.重写 sort（Comparable[] objs）方法：

(1).定义两个整数类型变量 left 和 right，其初始值分别为 0 和 objs.length-1；

(2).使用栈模拟递归调用：创建 Stack<Interger>类型变量 stack，分别对 right 和 left 使用 stack.push 方法；

(3).调用 while 循环，条件是！stack.isEmpty（），令 left=stack.pop（）、right=stack.pop。在此 while 循环内再次调用 while 循环，循环条件为 right>left，定义整数类型变量 partition=partition(objs，left，right)，判断 partition-left 是否>right-partition，若是则对 partition-1 和 left 使用 stack.push 方法，并让 left 的值等于 partition+1；若 partition-left>right-partition 不成立，则对 right 和 partition+1 使用 stack.push 方法，并让 right 的值等于 partition-1；

5.编写 partition（Comparable[] objs, int leftobjs, int rightobjs）方法：

(1).定义 Comparable 类型变量 key=objs[leftobjs], 定义两个整数类型变量 left=leftobjs、right=rightobjs+1,作为两个指针，分别指向待切分元素的最小索引处和最大索引处的下一位置；

(2).调用 while 循环, 循环内再前后调用两次 while 循环（相互独立），第一次 while 循环条件为 less(key,objs[--right]), 即判断数组从最右侧开始依次向左的元素是否大于分界值

key，若大于则继续指向其左侧的元素，若 right==leftobjs（即指针指向了最左侧元素）或者分界值大于指针指向的元素，则跳出此 while 循环转而进入第二个 while 循环，其循环条件是 less（objs[++left],key），即从数组最左侧向右的元素小于分界值 key，再使用 if 语句，若 left==rightobjs 则使用 break 语句终止循环，否则使指针继续向右索引。结束第二个 while 循环后再调用 if 语句判断 left 是否小于 right，若 left>=right，则使用 break 语句跳出 while 循环（此时已经对数组在 leftobjs 和 rightobjs 间的元素完成排序），否则使用 exchange 方法交换左右两个指针索引的元素的值；

(3).while 循环外，再次用 exchange 方法交换分界值 key 处元素与 right 指针指向的元素，返回 right 的值。

6.编写 main 函数，测试 Quick 类是否能完成排序功能：

（1）定义 Quick 类对象 quick 和待排序数组 nums；

（2）调用 printout 方法输出未排序的数组 nums；

（3）调用 sort 方法对 nums 进行排序；

（4）调用 printout 方法输出排序后的数组 nums。

二．算术混合运算表达式的计算

1.编写 Stack 类，并编写入栈、出栈、检查栈顶元素、判断栈是否为空四个方法：

（1）在 Stack 类后加上<T>泛型，使其可根据需要的数据类型来创建对象。

（2）编写内部类 Node 结点类：定义私有 T 类型变量 item，用于储存当前节点的数据；定义 Node 类型对象 next，当做当前结点的下一结点；编写有参构造器 Node（T item，Node next），将参数的值赋给 Node 对象；

（3）定义首结点 head，其值为 null；定义整数类型变量 N，用于记录栈中的元素个数；

（4）编写 isEmpty（）方法：判断元素个数 N==0 时返回 true；

（5）编写 push（T t）方法：定义当前栈顶元素的结点为 oldFirst，定义新元素的结点为 newNode，让 newNode=head.next，并让 newNode.next=oldFirst，使 newNode 成为新的栈顶元素的结点；最后让 N 的值加一；

（6）编写 pop（）方法：定义当前栈顶元素的结点为 oldFirst，判断其是否为 null，为 null 则返回 null，否则让 head.next=oldFirst.next 并使 N 的值减一，完成对 oldFirst 的删除操作，最后返回 oldFirst.item，成功将栈顶元素弹出；

（7）编写 peek（）方法：判断栈不为空时，定义栈顶元素的结点为 node 并将 node.item 返回，否则返回 null。

2.编写 calculateOfExpression（）类，实现中缀表达式转换为后缀表达式、计算后缀表达式两种功能：

（1）编写 isOperator（char c）方法：当参数 c 为题目给定的算术运算符时返回 true，否则返回 false；

（2）编写 precedence（char c）方法：用 switch 语句判断 c 的优先级——为加号或减号时返回 1；为乘号或除号时返回 2；为次方运算符时返回 3；其他情况（左括号或右括号或非法输入）返回 0；

（3）编写 operate（double num1，double num2，char operator）方法：调用 switch 方法判断 operator 是哪个运算符，并对应的对 num1 和 num2 进行运算，最后将结果返回（除法运算中出现 0 作为除数的情况将提醒输入者并直接退出程序）；

（4）编写 calculate（String string）方法：

I）定义可储存 double 类型的栈 stack，定义 String 数组 strings 并将传入的 string 中的字符依次存入 strings 中；

II）调用 for 循环依次遍历 strings 中的元素，用 char 类型变量 c 记录当前字符，若其为数字或小数点则将其压入栈中，若为运算符则弹出栈中的两个数字并对其调用 operate 方法，定义 double 类型变量 result 记录 operate 方法计算出的结果，再将其压入栈中；

III）for 循环结束后栈中应只剩一个数字，即输入表达式的结果，使用 pop 方法将其弹出，并用 return 将其输出；

（5）编写 isMatch（String str）方法：

I）创建可存储 String 类型数据的栈 stack，用于存储 str 中的左括号；

II）调用 for 循环遍历 str，对每一个字符进行判断，若为“（”则压入栈中，若为“）”则弹出栈中一个“（”（若此时栈为空则返回 false）；

III）调用 isEmpty 方法判断 stack 是否为空，若为空则返回 true，否则返回 false。

（6）编写 turn（String string）方法：

I）定义 StringBuilder 类型变量 newString 用于存储传入参数 string 中的数字，定义可存储 Character 类型数据的栈 stack；

II）调用 for 循环遍历 string 的每一个字符，将当前遍历字符存为 char 类型变量 chars，判断其为数字或小数点时存入 newString 中；为运算符时调用 precedence 方法判断其与栈顶的运算符的优先级，若栈顶运算符优先级更高则将其弹出并加到 newString 中，否则将遍历到的运算符压入栈；为左括号时压入栈中；为右括号时调用 while 循环将栈中第一个左括号前的运算符依次弹出并加到

newString 中，并将该左括号弹出；为其他除空格外的字符时提醒用户非法输入并退出程序;

III）若 for 循环遍历完后 stack 中还有运算符，则依次弹出并加到 newString 中;

IV）在控制台输出后缀表达式（方便验证正确性），并用 return 返回 newString.toString。

（7）编写 getResult（String num）方法：调用 turn 方法将传入的 num 转换成后缀表达式，对后缀表达式调用 calculate 方法并将其输出。

（8）编写 main 函数：定义 String 类型变量 num 用于存储想要计算的表达式，用 isMatch（num）判断其左右括号数量是否相等，若不等则提醒用户检查表达式，若相等则定义 double 类型变量 result，令其等于 getResult（num），将 result 输出即可。

三．LeakyStack 数据结构的编写

1.给二中定义的 Stack 类添加两个方法：

（1）编写 size（）方法：返回元素个数 N 的值;

（2）编写 clear（）方法：调用 while 循环，判断 isEmpty 方法为 false 时调用 pop（）方法将栈中元素弹出，实现对栈的清空。

2.编写 LeakyStack 类：

（1）定义一个栈 stack 用于储存数据，定义整数类型变量 capacity，其值为栈的容量；

（2）编写 push（T item）方法：调用 stack.size（）方法并于 capacity 的值作比较，栈满则调用 removeOldest（）方法，最后调用 stack.push 将 item 压入栈中；

（3）编写 removeOldest（）方法：

I）定义一个辅助栈 tempStack，定义整数类型变量 count 用于计数；

II）调用 while 循环，每循环一次 count 的值加一，判断 stack 不为空且 count 小于 capacity-1 时，弹出一个 stack 栈的元素并压入 tempStack 中；

III）调用 clear 方法将 stack 清空；

IV）调用 while 循环将 tempStack 的元素依次压入 stack 中。

（4）编写 undo（Stack<T> stack）方法：返回 stack.pop（），实现回退功能；

（5）编写 printStack（Stack<T> stack）方法：

I）定义一个辅助栈以暂时储存所需元素；

II）调用 while 循环将 stack 的所有元素依次弹出栈，将其输出至控制台并压入到辅助栈 tempStack 中；

III）调用 while 循环将 tempStack 中元素依次压回至 stack 中进行还原功能。

运行结果展示:

一．快速排序的非递归转化:

这是待排序数组:

```
Integer[] nums = {19, 15, 4, 12, 15, 9, 2, 6, 5, 13, 25};
```

运行快速排序程序后:

```
输入的待排序数组:19 15 4 12 15 9 2 6 5 13 25
经过快排后的数组:2 4 5 6 9 12 13 15 15 19 25
```

二．算术混合运算表达式的计算:

1.正常情况的输出:

```
中缀表达式为: 3.0 * ( 10.0 - ( 2 * ( 6 - 4.0 + 5 ) ) / 2.0 ) - 2.0 ^ 2
后缀表达式为:3.0 10.0 2 6 4.0 - 5 + * 2.0 / - * 2.0 2 ^ -
结果为: 5.0
```

2.将 0 作为除数的错误输入:

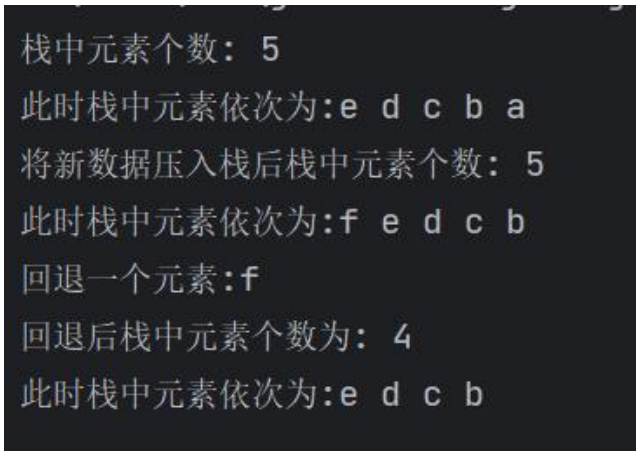
```
中缀表达式为: 5.1 + 2 / 0 +2 ^3
后缀表达式为:5.1 2 0 / + 2 3 ^ +
0不能作为除数
```

3.输入了给定范围外的字符（如#、￥等）:

```
中缀表达式为: 6 * ( 5 & 3 )
非法输入:&
```

三．LeakyStack 数据结构的测试

定义栈的容量为 5，依次压入 5 个字母 a、b、c、d、e，尝试将其输出，再压入 f，输出栈中元素个数并将此时栈中元素输出，最后调用 undo 方法测试是否能回退输出 f:



总结与收获:

1.通过重写 java 中的 Stack 类，掌握了栈的相关基础知识和部分栈的代码书写及有关栈的方法调用;

2.通过回顾排序方法相关的知识，并加上栈的知识，完成了对快速排序代码的改造，在快速排序代码中用栈的模拟递归调用完成了对快速排序的非递归转化;

3.通过对栈的相关知识学习，实践了将中缀表达式转换为后缀表达式、利用栈计算后缀表达式等相关功能;

4.学习并实践了利用栈记忆一定范围的元素、回退一定范围内的元素、将范围外的元素（栈底元素）清除等操作。

任务 4：基数排序

使用自定义的队列数据结构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。

设计：

一．编写 Queue 类，实现队列及基数排序的相关功能：

队列相关的代码：

1.编写内部类 Node 类：

（1）定义 T（任意）类型变量 item，作为每个 Node 类对象的数据，定义 Node 类对象 next，作为当前 Node 对象的指向对象；

（2）编写有参构造器为 item 和 next 赋值；

2.定义首结点 head 和尾节点 last，定义整数类型变量 N 以记录队列长度，并编写无参构造器初始化上述三个数据；

3.编写 isEmpty（）方法：判断 N 为 0 时返回 true，否则返回 false；

4.编写 enqueue（T t）方法：

（1）判断 last 为 null 时，定义 last=new Node（t，null），并让 head 指向 last；

（2）last 不为 null 时，定义 oldLast=last，令 last=new Node（t，null），再让 oldLast 指向 last；

（3）队列长度 N 加一。

5.编写 dequeue（）方法：

（1）队列为空时返回 null；

（2）定义 oldFirst=head.next，令 head 结点指向 oldFirst 指向的结点，并让记录队列长度的 N 减一，完成对 oldFirst 结点的删除操作；

（3）此时若队列为空则令 last=null；

（4）将 oldFirst.item 返回；

6.编写 clear（）方法：初始化 head、last、N；

7.重写 toString 方法：

（1）定义 StringBuilder 类对象 result，定义当前结点 currNode=head.next；

（2）用 while 循环判断 currNode 不为 null 时进行循环：将 currNode.item 加到 result 中，并让 currNode=currNode.next；

（3）返回 result.toString（）；

基数排序相关代码：

8.编写 getLength（int num）方法：num 为 0 时返回 1，否则返回 lg（num）去掉小数再加一后的结果；

9.编写 getMaxLength（）方法：

（1）定义整数类型变量 maxLength，用于记录遍历过的数字或字符串的最大位数，定义 currNode=head.next；

（2）读取 currNode.item，判断其为数字还是字符串

（3）若数据为数字，则调用 while 循环遍历整个队列，定义整数变量 length 用于记录当前遍历的数字的位数，调用 getLength 方法给 length 赋值，调用 Math 库中的 max 方法得出 maxLength 和 length 中的更大者并将其赋值给 maxLength，并使 currNode=currNode.next 以遍历下一结点；

（4）若为字符串，由于本实验测试的字符串等宽，故直接给 maxLength 赋值为任一字符串调用 length 方法后的结果

（5）返回 maxLength；

10.编写 getNum（int num，int position）方法：返回 num 除以 10 的 position 次方后余 10 的结果；

11.编写 getString（String str，int position）方法：令整数变量 index=str.length-1-position，并返回 str 在 index 位置上变为小写并-‘a’后的值；

12.编写 getValue（T item，int position）方法：

（1）如果 item 为数字，则返回 getNum（（Interger）item，position）；

（2）如果 item 为字符串，则返回 getString（（String）item，position）；

（3）若 item 为其他数据，则返回 0；

13.编写 Sort（int position）方法：

（1）定义基数 radix 为 30（数字需>=10，字母需>=26，取整后为 30）；

(2)创建队列数组 queues, 数组长度为 radix, 并给 queues 中的每一个元素初始化为一个队列;

(3) 定义 currNode=head.next，调用 while 循环遍历整个队列，定义整数变量 value 用于记录当前结点在 position 位的值, 并将当前结点的数据拷贝到 queues 的第 value 个队列中，再让 currNode=currNode.next;

（4）调用 clear 方法将队列初始化;

(5) 调用 for 循环遍历 queues 包含的每一个队列 queue，当 queue 不为空时调用 while 循环将 queue 中在 position 位置上排好序的数据通过调用 enqueue 方法传入队列中;

14.编写 radixSort（）方法:

（1）定义整数变量 maxLength，令其等于对队列调用 getMaxLength 方法后的结果;

（2）调用 for 循环，进行 maxLength 次循环，对从 0 开始到 maxLength 中的每一个位数调用 Sort 方法进行排序。

二．编写 Test 类用于测试:

1.编写 isNumber（char obj）方法：如果 obj 是数字则返回 true;

2.编写 parseLine（String line）方法:

（1）创建队列 queue;

（2）判断 line 的第一个字符是否为数字，若是则将 line 中所有数字依次存入 queue 中；

（3）若 line 为字母组成的字符串，则将所有字母的集合（以空格为界）依次存入 queue 中；

（4）返回 queue。

3.编写 output（String inputFile，String outputFile）方法：

（1）加上后缀 throws IOException，若发现异常直接抛出；

（2）定义 BufferedReader 类对象 reader，用于读取 inputFile 的数据，定义 PrintWriter 类对象 writer，用于输出数据至 outputFile；

（3）定义 String 类型变量 line，判断 line=reader.readLine（）不为 null 时，创建队列 queue，调用 parseLine（line）方法将数据添加到 queue 中；

（4）调用 radixSort 方法对 queue 进行基数排序；

（5）调用 writer.println（queue）方法将队列中的数据依次输出到指定文件中。

4.编写 main 函数进行测试：

（1）将准备的 radixSort1 以“.txt”为后缀赋值给 String 类型变量 inputFile1，再为其排序后的结果指定输出文件 output1 并以“.txt”为后缀赋值给 String 类型变量 outputFile1；

（2）调用 output（inputFile1，outputFile1）方法以完成输出，对 radixSort2 的排序输出同理。

运行结果展示：

1. 输出 radixSort1 的结果至 output1 文本中，以下是 output1 的部分内容：

（1）前几行的最左部分：

```
368 912 1284 1892 2141 2271 2451 2547 2589 2700 2864 2892 3417 4083 4178 4580 4651 4670
27 142 217 360 363 500 643 1130 1332 1514 1810 1914 1975 2090 2095 2462 2530 2657 2742 2
42 654 985 998 1020 1037 1084 1359 1361 1499 1568 1674 1716 2262 2274 2571 2698 2773 298
6 91 277 588 753 759 803 865 975 1103 1219 1339 1345 1605 1813 2153 2197 2359 2473 2640
18 85 280 354 1031 1050 1058 1212 1330 1349 1426 1978 1986 2257 2439 3119 3312 3368 3475
51 142 279 419 457 600 668 947 1367 1978 1982 2002 2051 2389 2533 2870 3059 3140 3301 33
23 69 664 887 902 1380 1381 1804 1951 1978 2169 2495 2642 2649 2748 3127 3133 3810 3942
166 263 377 386 1084 1222 1447 1470 1615 1737 1784 2036 2360 2570 2574 2657 2845 2912 29
```

（2）前几行的最右部分：

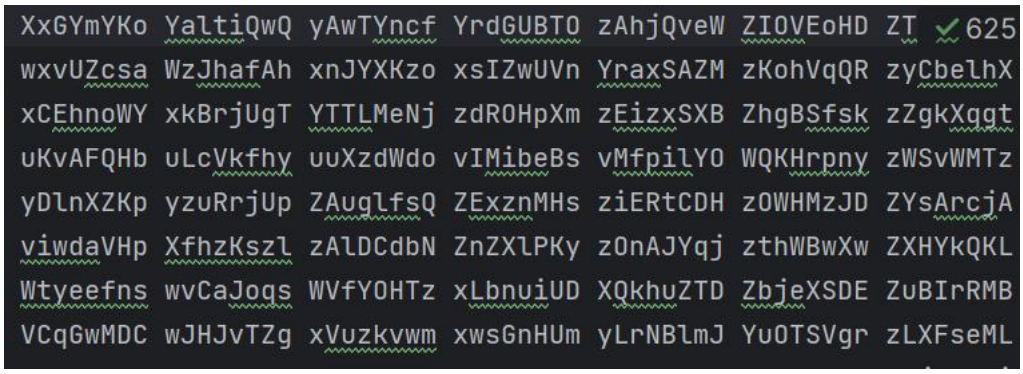
```
 986741 1005225 1074088 1089003 1153943 1161139 1169896 1174101 1220457
9487 1013477 1092189 1103453 1149127 1152640 1153878 1189582 1210693
947265 973466 976022 1046184 1068039 1118981 1146680 1188641 1203580
9546 1114106 1124560 1133015 1145847 1150142 1184391 1201068
00 998232 1016426 1021523 1024129 1058092 1090201 1118535 1180574
1004797 1005578 1020040 1086065 1152970 1171260 1191564 1213413
013883 1056249 1078142 1100687 1129869 1164718 1184917 1218801
6 978695 1041237 1052527 1065417 1108402 1153483 1183684 1189185 1207957
```

2. 输出 radixSort2 的结果至 output2 文本中，以下是 output2 的部分内容：

（1）前几行的最左部分：

```
aOwDbbJa aPXvpDSB bYEOyuMn csEgeBrx eDulsMpI eHhbJDTe ErAfCDdQ
BCKKRHgK bfUOZwul bHxYCKKz bIguGCIJ bZyWqsIW DhJkthfw eawdmqWx
aNMyTANF asyDvTbi AXQvcSbs bMxKLFOR brXfiQjL chvzhPcf cMrvPzJX
AztMmxSg BFPClXne CxwtNrcj CZGyqfyD dDKdrFPv DeQbsPHB dRvBcWPQ
AuAkCOZp BHXvhnwU BPEewCuy BQLpANqw chHoPsOS cSwsttdO DUmmGmkV
akFxAMWQ BjIpWzKO buekjsPy BwKtXaWi bZndjSKI ChYqdePX dbfJCiPi
aeauyVZZ AgYNuVcd AUNkreFd AWtDOqOs bIsTPcZt BlMgPzth BMSqeojg
AGIWmdlX BeESGied bwljZEjw CbkthTGB cHcJIpbK CtyoeBif dlIdSEkY
```

（2）前几行的最右部分：



总结与收获：

通过对该实验任务的实践，再一次巩固加深了自定义队列实现、PrintWriter 库的引用以及基数排序等相关知识的印象。通过使用自定义 Queue 类，成功实现了基数排序，对不等宽数字和等宽字母（不区分大小写）进行了排序。通过上述对 Test 类的部分运行结果的展示也可看出所写的 Queue 队列能够实现将不等宽数字或等宽字母进行基数排序的功能。