1. 1) Firstly, I ran the Reverse program with array type of stack on both the "bot.dat" and the "secret.dat" profiles, moved the outcome ".data" files to the same folder with the files of "sox", and then converted ".data" files into “.wav” profiles by sox. I listened to both “.wav” profiles and found they were both reversed successfully. The implementation using linked list type was tested in the same way.

2) Moreover, since the “.dat” profiles given are neither empty, I testified if the “isEmpty()” method is working by adding

“System.out.println(s.isEmpty());”

below the

“System.out.println(count+" samples in file");”

in another copy of class “Reverse.java” which was named “Testify”. When I delete all the words in “bot.dat” except the first two lines, and run the “Testify” on the new “bot.dat” file, messages including “0 samples in file” and “true” were printed.

3) Additionally, the “peek()” method hasn’t been used by “Reverse.java”, Thus, to test the method “peek()” I added

“System.out.println (s.peek());”

below the

“while (!s.isEmpty()) {”

in class “Testify”.

When I deleted all the words in “bot.dat” except the first few lines, and run the “Testify” on the new “bot.dat” file, the numbers were printed out in a reverse order.

1. The scent of bitter almonds always reminded him of the fate of unrequited love.
2. As the initial size is 10, say there are n lines in the “.data” file, and the resizing occurs m times. Then if 10\*2^(m-1)<n, m increases from m-1to m. Thus, we need find minimum m to let 10\*2^m>=n, which is 2^m>=(n/10).

When n=1 million, 2^16<n/10, 2^17>n/10, so m=17, which means the resizing occurs 17times;

In the same way, when n=1 billion, m=27; when n=1 trillion, m=40;

However, if we start from the size of 1, the resizing occurs 20, 30, 40 times separately for million, billion, trillion lines of data, since we already know, 2^20=1MB, 2^30=1GB,2^40=1TB.

1. 1) Push()

Queue q;

Set value to be pushed in;

If (q.isEmpty()) {

q.enqueue(value);

}

Else {

Quque qNew;

qNew.enqueue(value);

while q is not empty{

qNew.enque(q.dequeue());

}

Q=qNew;

}

2) pop

throw new EmptyStackException() {

return q.deque()

}

1. In a Queue implementation, a reverse process is needed in the push method, and the pop method is quite simple; while in array-based implementation, a loop process is need to find the end of the array where the new value should be added and where the existed value need be popped out.

I’d rather choose the array-based one, for it’s more reasonable in logistical sense and the process is much simpler.

1. (1 point) In the pop method, when the numbers of data in the array is less than 1/4 of the distributed length(maxNumber) of the array, a new array with length of half the maxNumber is defined and data in the initial array is transferred to the new array. Then the new array is set to the original array name.
2. I enjoyed the process of finding the logistics and algorithm to have the work done, but I suffered the coding process since I have never write in Java and am not familiar with the Java syntax. I could done better with the concision of my program if I knew Java better. Fortunately, I have used C++ a lot before, which helped me a lot. I think I will do better in Java in the following months.
3. Nothing.