Homework 7

/\*\*

\* Shifts entries between {@code leftStack} and {@code rightStack}, keeping

\* reverse of the former concatenated with the latter fixed, and resulting

\* in length of the former equal to {@code newLeftLength}.

\*

\* **@param** <T>

\* type of {@code Stack} entries

\* **@param** leftStack

\* the left {@code Stack}

\* **@param** rightStack

\* the right {@code Stack}

\* **@param** newLeftLength

\* desired new length of {@code leftStack}

\* **@updates** leftStack, rightStack

\* **@requires** <pre>

\* 0 <= newLeftLength and

\* newLeftLength <= |leftStack| + |rightStack|

\* </pre>

\* **@ensures** <pre>

\* rev(leftStack) \* rightStack = rev(#leftStack) \* #rightStack and

\* |leftStack| = newLeftLength}

\* </pre>

\*/

private static <T> void setLengthOfLeftStack(Stack<T> leftStack,

Stack<T> rightStack, int newLeftLength) {

assert rightStack != null : "Violation of: rightStack is not null";

assert leftStack != null : "Violation of: leftStack is not null";

assert 0 <= newLeftLength : "Violation of: 0 <= newLeftLength";

assert newLeftLength <= leftStack.length() + rightStack.length() : ""

+ "Violation of: newLeftLength <= |leftStack| + |rightStack|";

while (leftStack.length() <= newLeftLength && rightStack.length() > 0) {

leftStack.push(rightStack.pop());

}

while (leftStack.length() > newLeftLength) {

rightStack.push(leftStack.pop());

}

}