TSTL Stack Testing

Neil Parmar

Introduction

Stack is an important and a well-verified concept which has been tested using TSTL in this particular project. The documentation explains various tests, coverage concerns and how it's affects upon the output, accomplishments, and TSTL testing functionalities in relation towards the stacks.

Code Directions

In this code there are certain directions for understanding, a brief overview of the same has been provided below

<count> = increments and decrements the count based upon push and pop operations performed upon the stack.

<stack> = refers to the stack when referring to the tstl code

popOk = performs poping operations if the stack is not empty, however stackv1848.tstl has been tested differently for this case.

pushOk = pushes an element without checking any conditions while incrementing the counter.

testOk = check whether the size of the stack is equal to the number of counts and cleanses the stack accordingly. Has been only implemented in stackv1848, and will be further explained in this documentation.

cleanOk = is referred to by the testOk whenever the stack structure is inconsistent. This has only been implemented in stacktstlv1848, where it reinitiates the stack and counter.

Test Cases

In the particular folder, one will find *two* versions of stacktstl. These versions particularly functionality represent different testing operation versions.

The stacktstlv1637: This particular version of the code is basically concerned with the push and pop conditions. Firstly, pushing the elements onto the stacks and popping them which is logically the normal stack operation. Here, in the particular you might find a lot of commented statements. Please note these statements had been executed to understand the order of execution and randomness. Random testing is a powerful tool however the test cases which generated the output could get easily inconsistent. This was encountered while firstly excluding any raw python functionalities. This has been explained in later sections .

The stacktstl1848: This particular stack version internally runs a clean-up in order to maintain the count. However, the disadvantage of this method used with random testing is that it completely eliminates many stack operations whenever it encounters "testOk" condition. Here it cleanses the stack if the size of the stack is not equal to the number of elements in the stack, as the stack size would increment with the stack counter and should have the same values. If the values aren't similar, it runs a while under "cleanOk", where all the values are cleaned and the stack with the counter is flushed to initialization. The basic intention of involving the clean-up function was to maintain the stack. As the count value easily made the pops and the push completely random and inconsistent. This particular case made me test from scratch whenever the stack hits inconsistency.

Coverage Concerns

The stack when implemented without the intervention of python raw code, made a coverage of 15-16%(approx). However, the accuracy considerably increased when the stacktstl (various versions) were implemented involving python raw code. The test coverage ranged between 25-35%. (Please find the related out.txt file which contains the variety of test conditions and their outputs/results).

Testing Accomplishments

The stack is a well-implemented library, however the problem faced was when the testing asserts statements. The asserts have now been excluded from the implementation, but the code still maintains certain conditional implementation capabilities. For example, the stacktstlv1848 on line number 61 tests for a condition of the counter. The particular basically avoids running into pop conditions whenever the counter has decremented to a 0. However, the results eliminate the test conditions every time the element pushes something onto the stack and it senses inconsistency. However, the coverage is 27.69% which is a considerably higher execution to the normal output while calling the normal stack pop operations ranging from 25-27 percentage. Therefore, complete random testing for stacks does not provide a justified results there are inconsistencies and anomalies which aren’t tested.

TSTL

TSTL is a strong testing language providing capabilities which are better towards the future of python program testing. However, TSTL when referring to the randomtester for related errors, becomes difficult to understand in certain scenarios.