Deliverable Two

The samplePlug module delivered contains 11 white box tests, each of which tests the five Halstead checks, the operator check, the operand check, the loop check, the expression check, the comment check, and the comment line check.

HalsteadDifficultyWhiteTest.java
HalsteadEffortWhiteTest.java
HalsteadLengthWhiteTest.java
HalsteadVocabularyWhiteTest.java
HalsteadVolumeWhiteTest.java
NumberOfCommentsWhiteTest.java
NumberOfExpressionsWhiteTest.java
NumberOfLinesOfCommentsWhiteTest.java
NumberOfLoopingStatementsWhiteTest.java
NumberOfOperandsWhiteTest.java

Since each check is structured largely the same way, all of the tests follow the same basic structure, with the overall goal being complete or near-complete code coverage.

Each white box test has a method setup() which is called at the beginning of each test, to instantiate mock variables for the DetailAST's used to mock getting tokens, and a spy of the check style checker we want to check behavior of.

Each white box test class checks the getAcceptableTokens, getExpectedTokens, and getRequiredTokens to make sure they run when called, and to make sure that they return the correct list of tokens to look for. These are always done in three separate checks.

Each white box test does a separate check for the GetResult() method, which checks and makes sure that GetResult() runs when called upon. All of these checks are nearly identical.

Each white box class includes a beginTree white test, which starts the beginTree method and makes sure it was run on a AST.

Each white box class includes a visitTree white test, which starts visitTree on a token or several tokens of the required type and makes sure that the method was run with the correct input variables. Sometimes, such as in the Halstead and comment white tests, when different token types warrant different responses by the checker, multiple tokens are tested to make sure we get full coverage.

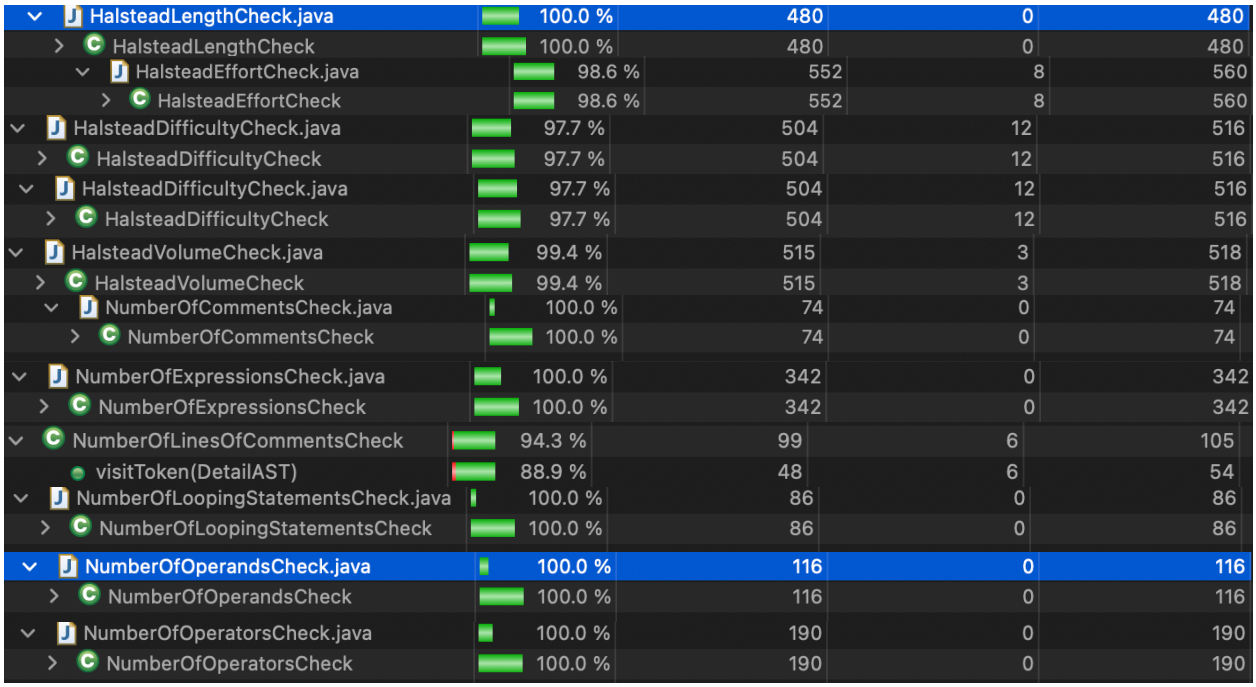
Finally, each white box class contains a test for finish tree. This makes sure that the logging function is being run by spying that function in the check style check we are testing, and verifies that the finish tree method was run.

Coverage:

The coverage for each set of cases is as follows:

Explanation for Halstead Checks lower than 100 coverage: Several Halstead checks had less than 100 percent coverage. This was because code was added to the Halstead checks to make sure that there was not a below 0 error, assuring that if there was a possible divide by zero error with a variable, it would return 0 instead of doing the operation and returning an exception. The half of the branch that returns 0 instead is what is not covered in certain Halstead checks.

Explanation for CommentLines Check lower than 100 coverage: In this case, the contents are checked for a multi line comment to see if it has the phrase “\n”. While we test for something with the contents of “\n”, we failed to test for something that does not have the \n in it.



If I could do this again, I would probably cover the divide by zero branch and the empty comment content branch with a blank AST, which would have covered those issues.

A note about checks involving operands or operators:

Operators and operands are recognized within code even if they are not being used in combination with operands or operators. Additionally, because this is for java code and not just math operations alone, it may accept some things as operands/operators that traditionally wouldn’t be considered operators/operands outside of a programming scope. This means that Halstead