Using the file *original.cpp*, I tested four cases, each with reasonable input integers that produced unusual output values. For my first case, I inputted 0 as the number of registered voters surveyed to see what percentage would be produced. Entering 2 votes for Joe and 1 vote for Donald, the result was “inf%” for both. My second case was inputting a negative number for the number of registered voters surveyed; in this scenario I used -10. I entered in 6 votes for Joe and 4 votes for Donald, and the output was -60.0% for Joe and -40.0% for Donald. My third case was -20 registered voters surveyed, but -15 votes for Joe and -5 votes for Donald. The result was 75.0% for Joe and 25.0% for Donald, but it also printed, “Donald is predicted to win the election.” This was because even though Joe had a higher percentage, the compiler read that Donald’s value was less negative than Joe’s. My last case was to see how the compiler would deal with both an equal number of votes for Joe and Donald and also their individual vote counts to be higher than the number of registered voters surveyed. I inputted 50 for the number of registered voters surveyed, 100 for Joe’s votes, and 100 for Donald’s votes. The output was 200.0% for both Joe and Donald, but it also displayed, “Donald is predicted to win the election” despite them having an equal number of votes. This is a potential issue with the code, as even when integers producing reasonable output are entered, the output would incorrectly state that Donald would win the election when equal votes are given to both Joe and Donald.

The errors I introduced to the file *logic\_error.cpp* were replacing the “\*” with “/” and the “/“ with “\*” in line 21. So, the line is: “double pctDonald = 100.0 / forDonald \* numberSurveyed;”. This causes the build to be successful and no error messages to be reported, but the output is incorrect. For instance, if the number of registered voters surveyed is 100 and 50 vote for Joe while the other 50 vote for Donald, the result is 50.0% of the votes for Joe and 200.0% of the votes for Donald. Another example is if the number of registered voters surveyed is 50 and 40 vote for Joe while 10 vote for Donald, the result is 80.0% in Joe’s favor and 500.0% in Donald’s favor. These are clearly nonsensical values for Donald’s vote percentage, as it can’t be greater than 100.0%.

In the file *compile\_error.cpp*, the two compile errors I introduced were deleting the semicolon at the end of line 17 and deleting the end brace in line 33, resulting in the program failing to build. Upon execution of the code in Xcode, I was flagged with two errors: “Expected ‘;’ after expression” and “Expected ‘}’” at lines 17 and 33, respectively. When running using g31 on the SEASnet Linux server, I received 3 errors: “expected ‘;’ before ‘cin’” between lines 17 and 18; “this ‘else’ clause does not guard this statement, but the latter is misleadingly indented as if it were guarded by the ‘else’” between lines 32 and 33; and “expected ‘}’ at end of input” at line 33 “to match this ‘{’”, referring to the unmatched open brace at line 8. A semicolon is required after line 17 because it indicates the end of a statement but if it is omitted, the compiler cannot interpret the code and run it correctly. A closing brace “}” is required in line 33 because otherwise, there will be an imbalance of opening braces “{“ and closing braces. These sets of braces mark a method or loop contained inside them, and the compiler expects a closing brace to signify that the code for the method or loop is complete.