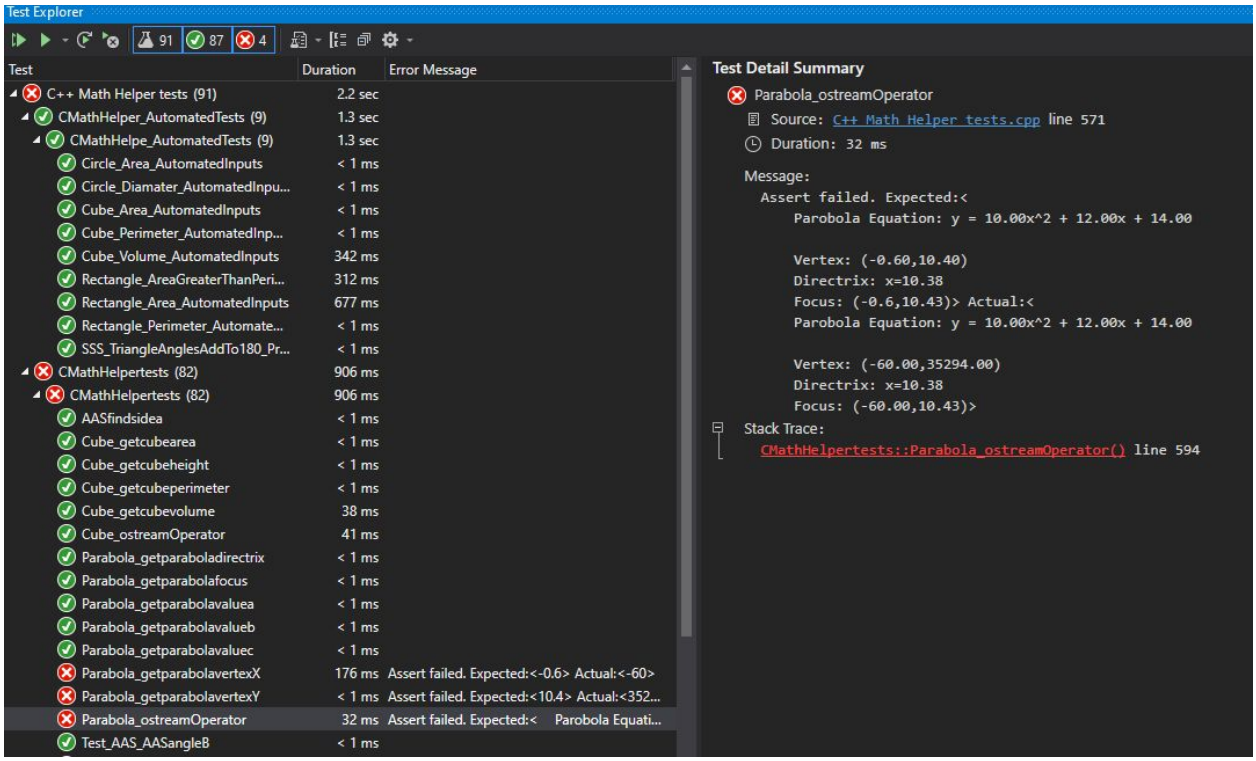
Project Report

Student Name(s): Tommy Avetisyan, Kyle Felkel
Link to SUT Source Code: https://github.com/Tomavetisyan/C--Math-Solver

Unit Testing with Sufficient Coverage

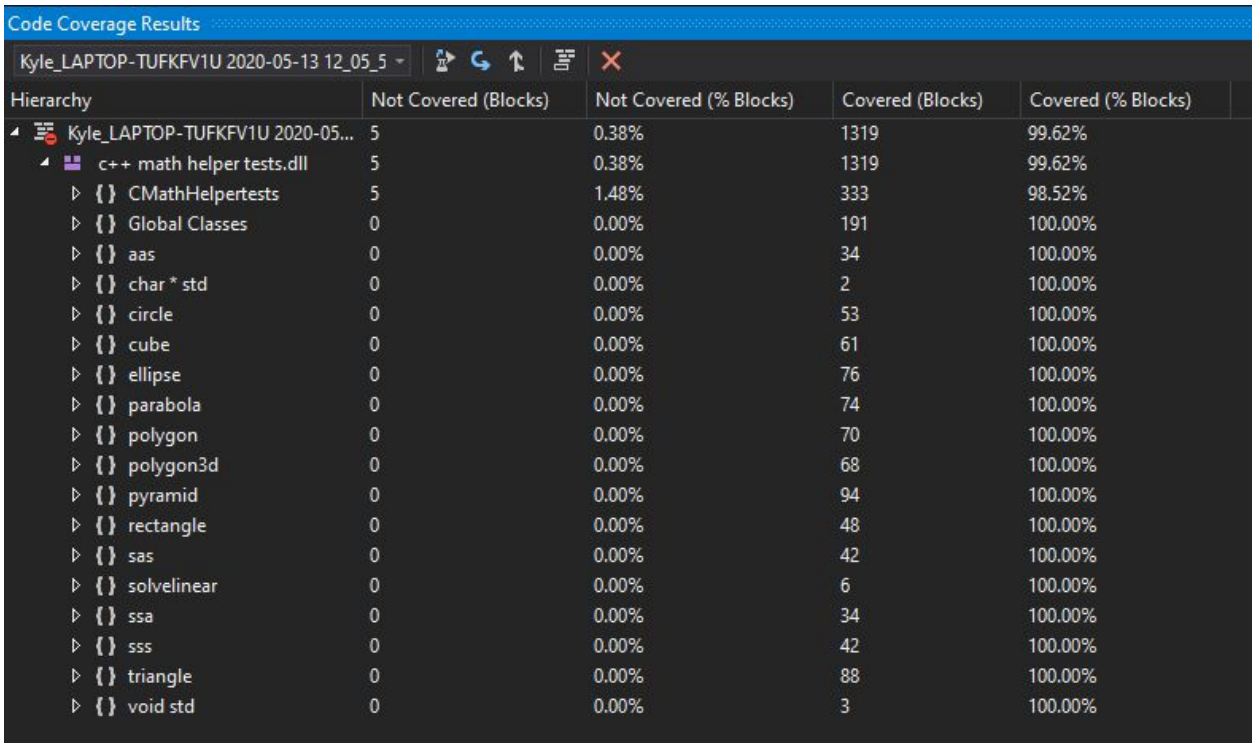
All actual tests are found in the file: C++ Math Helper tests.cpp, https://github.com/Tomavetisyan/C--Math-Solver/blob/master/C%2B%2B%20Math%20Helper%20tests/C%2B%2B%20Math%20Helper%20tests.cpp
This file contains all unit tests. Inputs are manually given and expected values calculated by hand. Expected values are compared to the actual values returned by the SUT in Assert statements. In order to avoid rounding errors of double values failing a test, an error range of 0.1 is used as a parameter of Assert statements. Actual test values are received by calling helper functions which return values rather than direct interaction with the objects under test. Helper tests functions are found in Tests.cpp, https://github.com/Tomavetisyan/C--Math-Solver/blob/master/C%2B%2B%20Math%20Helper%20tests/Tests.cpp
This Tests class is declared as a friend of the other classes being tested and is used to access private and protected variables. The functions were added here to do the work of initializing objects and values, performing the relevant function calls, and returning those values. Both of these files were added during the project for testing purposes and minimal changes were made to the implementation code.



Code Coverage

Code coverage was calculated using the built-in code coverage tool in Visual Studio 2019 Enterprise. Code coverage is found in .coverage files found under the TestResults folder. https://github.com/Tomavetisyan/C--Math-Solver/tree/master/TestResults

These files can be viewed using Visual Studio and show the percentage of code covered. By expanding and going down the list you can view coverage by the entire solution, by class, or by individual methods. Due to the small size of the project we achieved 100% code coverage. Although the tool does not show 100% coverage, this is accurate because it calculates the assertion statements from failed unit tests as not covered.



Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
Kyle_LAPTOP-TUFKFV1U 2020-05...	5	0.38%	1319	99.62%
c++ math helper tests.dll	5	0.38%	1319	99.62%
CMathHelpertests	5	1.48%	333	98.52%
Global Classes	0	0.00%	191	100.00%
aas	0	0.00%	34	100.00%
char * std	0	0.00%	2	100.00%
circle	0	0.00%	53	100.00%
cube	0	0.00%	61	100.00%
ellipse	0	0.00%	76	100.00%
parabola	0	0.00%	74	100.00%
polygon	0	0.00%	70	100.00%
polygon3d	0	0.00%	68	100.00%
pyramid	0	0.00%	94	100.00%
rectangle	0	0.00%	48	100.00%
sas	0	0.00%	42	100.00%
solvelinear	0	0.00%	6	100.00%
ssa	0	0.00%	34	100.00%
sss	0	0.00%	42	100.00%
triangle	0	0.00%	88	100.00%
void std	0	0.00%	3	100.00%

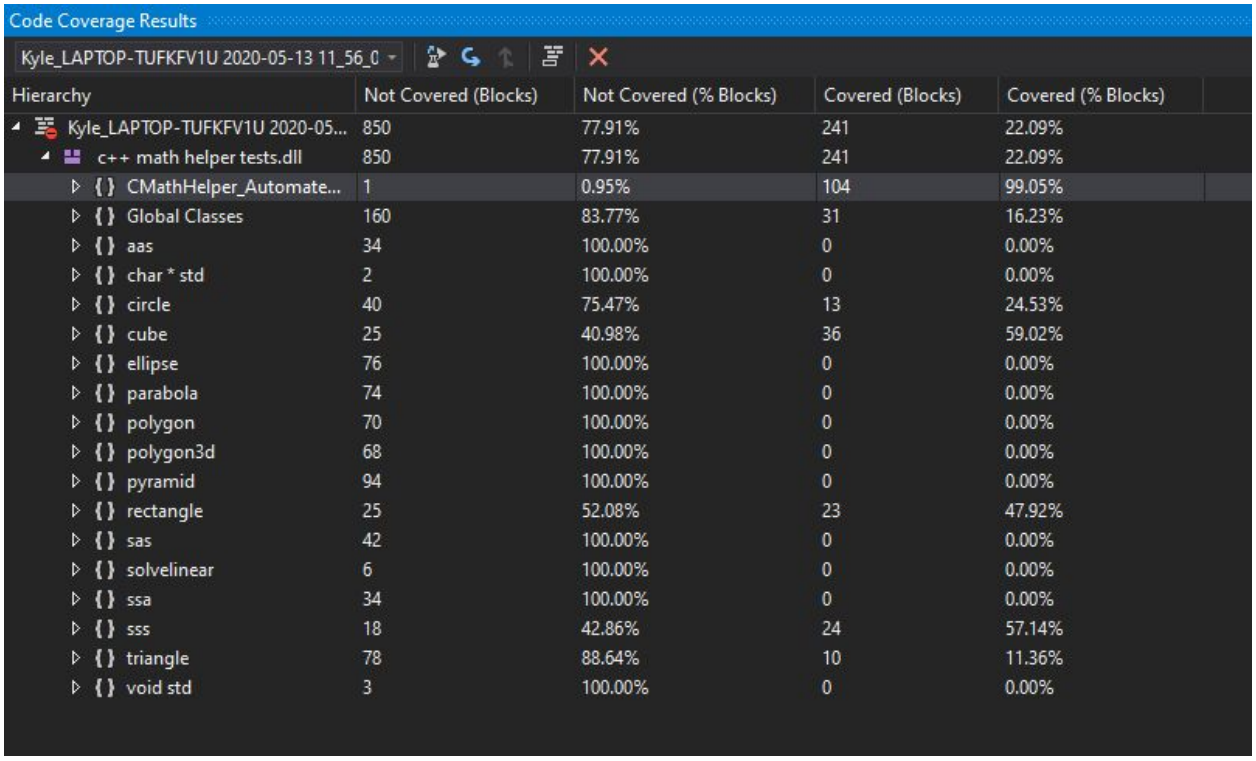
Automated Testing

Automated Tests are found in C++ Math Helper AutomtatedTests.cpp, https://github.com/Tomavetisyan/C--Math-Solver/blob/master/C%2B%2B%20Math%20Helper%20tests/C%2B%2B%20Math%20Helper%20AutomtatedTests.cpp

This file contains tests which automatically generate input values, calculate correct values based on those input values, pass the input values to the system for a return value, and compare the returned and calculated values. This is done in a loop and the input values generated are between 1 and 100. There are also Property Based Tests located at the bottom of the file which follow a similar pattern but do not calculate exact expected values.

Code Coverage

Here is the code coverage of the automated tests. The value is slightly inflated due to the tool including the Tests themselves as code that is covered, however the actual coverage value calculated by hand is still about 18%.



Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
Kyle_LAPTOP-TUFKFV1U 2020-05...	850	77.91%	241	22.09%
c++ math helper tests.dll	850	77.91%	241	22.09%
{ } CMathHelper_Automate...	1	0.95%	104	99.05%
{ } Global Classes	160	83.77%	31	16.23%
{ } aas	34	100.00%	0	0.00%
{ } char * std	2	100.00%	0	0.00%
{ } circle	40	75.47%	13	24.53%
{ } cube	25	40.98%	36	59.02%
{ } ellipse	76	100.00%	0	0.00%
{ } parabola	74	100.00%	0	0.00%
{ } polygon	70	100.00%	0	0.00%
{ } polygon3d	68	100.00%	0	0.00%
{ } pyramid	94	100.00%	0	0.00%
{ } rectangle	25	52.08%	23	47.92%
{ } sas	42	100.00%	0	0.00%
{ } solvelinear	6	100.00%	0	0.00%
{ } ssa	34	100.00%	0	0.00%
{ } sss	18	42.86%	24	57.14%
{ } triangle	78	88.64%	10	11.36%
{ } void std	3	100.00%	0	0.00%

Lessons Learned

The main thing we had trouble with through the project was learning Microsoft's testing framework on Visual Studios. Doing a C++ project, junit was not an option to do our testing with. Some of the trouble with Visual Studio was figuring out how to connect the project to the test class. We ended up deciding to create the test class in a separate file and migrate the project into the test folder. When we got the testing and the project together, we realized our test class had to friend all the other classes to access private and protected variables to run the tests and get the coverage. For the coverage reports, we did some digging and found out we had to have a paid version of Visual Studios which later we realized they offered a free trial that covered us for the semester.

I think this project we went outside of our comfort zone and used something different which ended up being another skill to add on our resumes being both upcoming graduates.I think Visual Studios documentation is a little too simplified and the forums for issues did not cover the troubles we were having. On top of that, we had to refresh ourselves on C++ again being that we have both been coding in java since we have been at CSUN.