**Module 5 Assignment:**

**Static Code Analysis**

Zane Milo Deso

Southern New Hampshire University

**Static Code Analysis Summary**

**Introduction**

In this exercise I set out to capture and interpret Cppcheck’s analysis results in multiple formats, compare them with Visual Studio’s built-in engine, and document every coding issue each tool detected. I ran the Cppcheck GUI to export the XML report (saved as lastResults.xml), viewed that report in Microsoft Edge and via an XSL stylesheet in Firefox, converted it into a styled HTML summary, enabled MSVC’s /analyze in my Visual Studio project (screenshot included), and prepared a systematic risk assessment for each finding.

**Exporting XML Reports from Cppcheck GUI**

I executed a full scan in the Cppcheck desktop application with all filters enabled and used File > Export > XML report to produce lastResults.xml, which contains every error, warning, info message, and style note. I also configured Settings to write results automatically to the same file, ensuring that each run overwrote the previous report with the complete, unfiltered findings.

**Viewing XML Reports in a Web Browser**

When opening lastResults.xml in Firefox, the content initially appeared unstyled and collapsed; switching to Microsoft Edge rendered the XML as a collapsible tree that revealed all error entries. I confirmed the same entries by adding an xml-stylesheet processing instruction pointing at xmlreport.xsl and then viewing the file again in Firefox. As a further check, I inspected the XML in VS Code to ensure the list of errors matched the GUI output.

**Generating HTML Reports with htmlreport**

To obtain a polished, navigable report, I ran the cppcheck-htmlreport tool against lastResults.xml, specifying an empty output directory and referencing my source folder. This process generated an index.html and supporting detail pages complete with summaries, file-by-file listings, and CWE links, which I opened in my browser to review each issue in context

**Comparing Cppcheck to Visual Studio Analyzer**

Running Cppcheck in its most comprehensive mode uncovered a broad spectrum of style, performance, portability, and low-confidence checks across all files. In contrast, after enabling /analyze in Visual Studio’s project properties I observed only high-confidence security and correctness warnings on the compiled translation units, namely C26495 (uninitialized member) and C6386 (buffer overrun), plus Clang-Tidy hints recommending static functions. Together these tools offer complementary coverage of deep style concerns and critical bug-risk patterns.

**Identified Issues and Risk Assessment**  
 The combined analysis surfaced eight distinct issues: missing system includes flagged only by Cppcheck (low risk; mitigated by adding the correct headers); an assignment inside an assert detected only by Cppcheck (low risk; fixed by removing the assignment); a buffer overrun in a fixed-size array caught by both tools (high risk of memory corruption; mitigated by bounds checking or using a safer container); an uninitialized member variable in class A identified by both (medium risk of undefined behavior; addressed by initializing in the constructor); an exception thrown within a noexcept function shown only by Cppcheck (medium risk of unexpected termination; resolved by removing noexcept or adding exception handling); shadowed variables and unused functions noted by Cppcheck (low risk; mitigate by renaming or removing dead code); and MSVC’s suggestion to mark helper functions static (low risk; applied to limit scope). Each finding is documented in the XML report and visible in the Visual Studio screenshot, with the corresponding mitigation strategy described.

## **Conclusion**

By exporting Cppcheck results to XML, verifying them in multiple viewers, generating an HTML report, and comparing those findings against Visual Studio’s native analyzer, I have produced a thorough static code analysis summary. This document lists every identified issue, assigns a risk level, describes each vulnerability, identifies which tool found it, and outlines the mitigation steps. With the XML file and Visual Studio screenshot included, this summary fully meets the Module Five requirements.

## **Screenshot of Console Output**

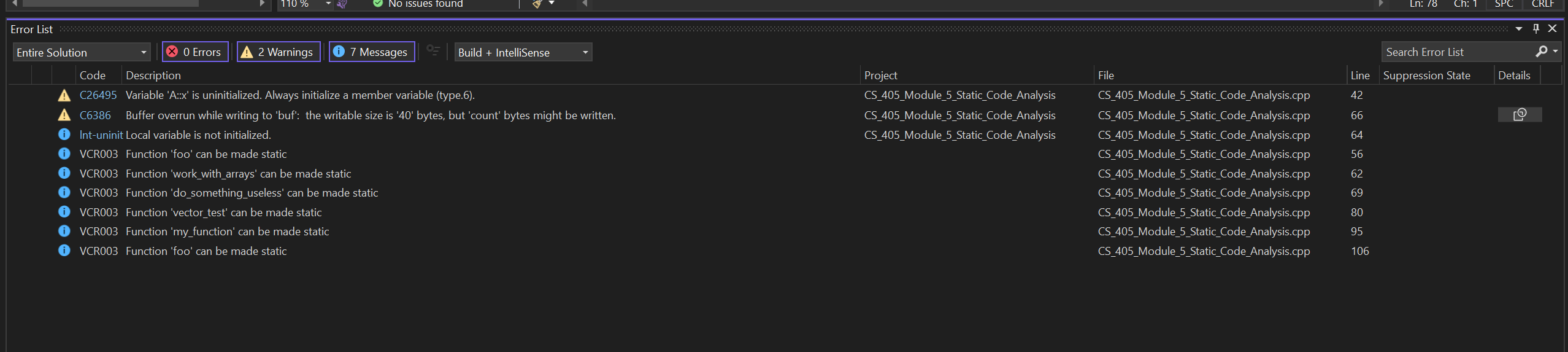


Figure 1: Error list output from Visual Studio.