**Introduction**

To begin, we set up the environment by creating a new Console Project in Visual Studio and adding the QuestionableCode.cpp file. Additionally, CppCheck was installed from the official CppCheck website.

**Analyzing Code with Visual Studio**

We built the project using Visual Studio to identify any warnings and errors. Two main warnings were identified. The first, Warning C4297, indicated that the function MySpecialType::DontThrow was marked with noexcept but contained a throw statement, presenting a high risk of contradiction and leading to potential runtime errors. The second, Warning C4806, pointed out an unsafe operation involving a comparison between bool and int, specifically in the assertion assert(z = 2), where the assignment should be a comparison using ==.

**Analyzing Code with CppCheck**

The next step involved running CppCheck on the QuestionableCode.cpp file to generate an XML report. This was accomplished using the command line interface with the command cppcheck --enable=all --inconclusive --xml --xml-version=2 QuestionableCode.cpp 2> cppcheck\_report.xml. CppCheck identified several issues: endless recursion in C::is\_type, iterator invalidation in vector\_test, and a potential null dereference in foo.

**Comparing Results**

We compared the results from Visual Studio and CppCheck. Visual Studio identified issues like endless recursion in C::is\_type, buffer overflow in work\_with\_arrays, and an assertion error with assignment in main. On the other hand, CppCheck identified the same recursion issue in C::is\_type but also found additional issues such as iterator invalidation in vector\_test and potential null dereference in foo. These findings highlighted the unique strengths of each tool.

**Conclusion**

Both Visual Studio and CppCheck were effective in identifying critical issues in the code. Visual Studio provided immediate build-time feedback and identified issues related to function specifications and unsafe operations, while CppCheck offered additional insights into recursion, iterator invalidation, and potential null dereferences. Using both tools together enhances the robustness and reliability of the code by ensuring comprehensive identification and mitigation of potential issues.

By following this process, we were able to gain a thorough understanding of the types of errors that can be found through static code analysis and the respective benefits and limitations of Visual Studio and CppCheck.

A screenshot of a computer program

Description automatically generated