5-3 Activity: Static Code Analysis

**Screenshot**

**A black rectangular object with white lines

Description automatically generated**

**Summary**

In this assignment, we play the role of a senior software developer on a team responsible for a large banking application. Our manager, curious and skeptical about the value of static code analysis, has us test the Cppcheck tool to see if it can pick up errors in a file that consists of many common coding issues. We are tasked with successfully identifying the errors within the supplied code file and comparing the results of Visual Studio’s built-in analysis versus Cppcheck. Each issue found must also contain a risk analysis, brief description of the issue, proposal of mitigation, and which tool found the issue.

Upon compiling the code in Visual Studio and running the Cppcheck analysis on the project, I was genuinely surprised at how many more errors that Cppcheck found. In comparison to Visual Studio’s built-in analysis, there were at least eighteen additional errors found in the code that were not found within visual studio. Every error that Visual Studio found I was able to verify that Cppcheck also found. This highlights the importance of utilizing different static analysis tools for thorough error checking. Below I will list the errors found in the code that Cppcheck found and Visual Studio missed. Additionally, each error will contain a risk analysis, description of the error, how to mitigate the issue, and which static analysis tool was used to find the issue.

**Errors Found**

**Error:** missingIncludeSystem

Note: Not sure if this is an actual error as the include headers exist in visual studio.

**Risk:** NOT RISK

**Description:** Missing include headers for cassert, iostream, numeric, set, and vector.  
**Mitigation:** Ensure the use of proper include headers.

**Tool:** Cppcheck

**Error:** assignmentInAssert

**Risk:** RISK

**Description:** The variable ‘z’ is modified inside of an assert statement.  
**Mitigation:** Do not modify variables inside of an assert statement.

**Tool:** Cppcheck

**Error:** assertWithSideEffect

**Risk:** RISK

**Description:** ‘my\_function’ is called within an assert statement with possible desired side effects.  
**Mitigation:** Refactor the code to not call functions with side effects within an assert statement.

**Tool:** Cppcheck

**Error:** uselessAssignmentPtrArg

**Risk:** RISK

**Description:** The assignment of a function parameter has no effect outside of the function. **Mitigation:** Ensure to utilize function parameters within the function body or dereference the parameter if necessary.

**Tool:** Cppcheck

**Error:** autoVariables

**Risk:** RISK

**Description:** The address of a local auto variable is assigned to a function parameter.  
**Mitigation:** Local auto variables are reserved from the stack and become invalid when the function ends. Do not assign the address of a local auto variable to a function parameter.

**Tool:** Cppcheck

**Error:** returnNonBoolInBooleanFunction

**Risk:** RISK

**Description:** A non-boolean value is returned from a boolean function.  
**Mitigation:** If the function is supposed to return a boolean, ensure that the logic of the function is correctly returning boolean values.

**Tool:** Cppcheck

**Error:** functionStatic

**Risk:** NOT RISK

**Description:** The member functions ‘Token::next’ and MySpecialType::DontThrow can be made static for a potential performance benefit.  
**Mitigation:** Change member functions to static when they do not operate on a specific instance of the class and require the ‘this’ pointer.

**Tool:** Cppcheck

**Error:** nullPointerRedundantCheck

**Risk:** RISK

**Description:** The condition ‘tok’ is redundant or there is a possible null pointer dereference.  
**Mitigation:** Remove the condition if it is redundant or ensure that null pointer dereferences are handled correctly.

**Tool:** Cppcheck

**Error:** variableScope

**Risk:** NOT RISK

**Description:** The scope of the variable ‘buf’ can be reduced.  
**Mitigation:** Investigate if the scope of the ‘buf’ variable can be reduced. If it can be reduced, reduce the scope by 1 level.

**Tool:** Cppcheck

**Error:** shadowVariable

**Risk:** RISK

**Description:** The local variables, ‘x’, ‘y’, and ‘z’ shadow outer variables.  
**Mitigation:** Change variable names to be unique to avoid shadow variables which can cause unintended program behavior.

**Tool:** Cppcheck

**Error:** invalidContainer

**Risk:** RISK

**Description:** The iterator to the local container ‘items’ may be invalid.  
**Mitigation:** Make sure that iterators to local containers stay valid throughout their use. Avoid adding or removing elements that can invalidate an iterator on a container or use these operations outside where iterators are being used.

**Tool:** Cppcheck

**Error:** unusedStructMember

**Risk:** NOT RISK

**Description:** The class member ‘A::x’ is never used.  
**Mitigation:** If the class member has no use or purpose, remove it to eliminate any unnecessary code to enhance readability and prevent potential issues.

**Tool:** Cppcheck

**Error:** unreadVariable

**Risk:** RISK

**Description:** The variables ‘buf[count]’, ‘tok’, ‘x’, and ‘y’ are assigned values and never used.  
**Mitigation:** If the variables have no use or purpose, remove them to eliminate any unnecessary code to enhance readability and prevent potential issues.

**Tool:** Cppcheck