Executing task: clang++ -g /home/francis/Documents/cs405m5StaticTesting/\*.cpp -o /home/francis/Documents/cs405m5StaticTesting/output -I /usr/local/include -L /usr/l

ocal/lib -lgtest -lgtest\_main -lpthread -lssl -lcrypto

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:52:5: warning: 'DontThrow' has a non-throwing exception specification but can still throw [-Wexception**

s]

52 | throw "Ha! I threw anyway!";

| **^**

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:50:8: note:** function declared non-throwing here

50 | void DontThrow() noexcept

| **^ ~~~~~~~~**

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:129:24: warning: result of comparison of constant 3 with expression of type 'bool' is always false [-W**

tautological-constant-out-of-range-compare]

129 | assert(my\_function() == 3);

| **~~~~~~~~~~~~~ ^ ~**

**/usr/include/assert.h:93:27: note:** expanded from macro 'assert'

93 | (static\_cast <bool> (expr) \

| **^~~~**

2 warnings generated.

\* Terminal will be reused by tasks, press any key to close it.

\* Executing task: clang++ -g /home/francis/Documents/cs405m5StaticTesting/\*.cpp -o /home/francis/Documents/cs405m5StaticTesting/output -I /usr/local/include -L /usr/l

ocal/lib -lgtest -lgtest\_main -lpthread -lssl -lcrypto

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:52:5: warning: 'DontThrow' has a non-throwing exception specification but can still throw [-Wexception**

s]

52 | throw "Ha! I threw anyway!";

| **^**

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:50:8: note:** function declared non-throwing here

50 | void DontThrow() noexcept

| **^ ~~~~~~~~**

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:129:24: warning: result of comparison of constant 3 with expression of type 'bool' is always false [-W**

tautological-constant-out-of-range-compare]

129 | assert(my\_function() == 3);

| **~~~~~~~~~~~~~ ^ ~**

**/usr/include/assert.h:93:27: note:** expanded from macro 'assert'

93 | (static\_cast <bool> (expr) \

| **^~~~**

2 warnings generated.

\* Terminal will be reused by tasks, press any key to close it.

\* Executing task: cppcheck --enable=all --inconclusive --std=c++17 --verbose --language=c++ /home/francis/Documents/cs405m5StaticTesting/\*.cpp

Checking /home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp ...

Defines:

Undefines:

Includes:

Platform:Native

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:127:12: warning:** **Variable 'z' is modified inside assert statement. Assert statements are removed from**

release builds so the code inside assert statement is not executed. If the code is needed also in release builds, this is a bug. [assignmentInAssert]

assert(z = 2);

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:129:10: warning:** **Non-pure function: 'my\_function' is called inside assert statement. Assert statements**

are removed from release builds so the code inside assert statement is not executed. If the code is needed also in release builds, this is a bug. [assertWithSideEffect

]

assert(my\_function() == 3);

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:109:3: warning:** **Assignment of function parameter has no effect outside the function. Did you forget de**

referencing it? [uselessAssignmentPtrArg]

tok = tok->next();

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:59:3: error:** **Dangerous assignment - the function parameter is assigned the address of a local auto-var**

iable. Local auto-variables are reserved from the stack which is freed when the function ends. So the pointer to a local variable is invalid after the function ends. [a

utoVariables]

\*a = &b;

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:129:24: warning:** **Comparison of a boolean expression with an integer other than 0 or 1. [compareBoolExp**

ressionWithInt]

assert(my\_function() == 3);

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:98:3: style:** **Non-boolean value returned from function returning bool [returnNonBoolInBooleanFunction]**

return a;

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:66:8: warning:** **Either the condition 'count==1000' is redundant or the array 'buf[10]' is accessed at i**

ndex 1000, which is out of bounds. [arrayIndexOutOfBoundsCond]

buf[count] = 0; // <- ERROR

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:65:13: note:** Assuming that condition 'count==1000' is not redundant

if (count == 1000)

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:66:8: note:** Array index out of bounds

buf[count] = 0; // <- ERROR

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:42:3: warning:** **Member variable 'A::x' is not initialized in the constructor. [uninitMemberVarPrivate]**

A(const A& other) {}

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:50:8: performance: inconclusive:** **The member function 'MySpecialType::DontThrow' can be made a static f**

unction. Making a function static can bring a performance benefit since no 'this' instance is passed to the function. This change should not cause compiler errors but i

t does not necessarily make sense conceptually. Think about your design and the task of the function first - is it a function that must not access members of class inst

ances? And maybe it is more appropriate to move this function to a unnamed namespace. [functionStatic]

void DontThrow() noexcept

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:52:5: error:** **Exception thrown in function declared not to throw exceptions. [throwInNoexceptFunction]**

throw "Ha! I threw anyway!";

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:109:9: warning:** **Either the condition 'tok' is redundant or there is possible null pointer dereference:**

tok. [nullPointerRedundantCheck]

tok = tok->next();

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:108:10: note:** Assuming that condition 'tok' is not redundant

while (tok);

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:109:9: note:** Null pointer dereference

tok = tok->next();

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:64:7: style:** **The scope of the variable 'buf' can be reduced. Warning: Be careful when fixing this mess**

age, especially when there are inner loops. Here is an example where cppcheck will write that the scope for 'i' can be reduced:

**void f(int x)**

**{**

**int i = 0;**

**if (x) {**

**// it's safe to move 'int i = 0;' here**

**for (int n = 0; n < 10; ++n) {**

**// it is possible but not safe to move 'int i = 0;' here**

**do\_something(&i);**

**}**

**}**

**}**

**When you see this message it is always safe to reduce the variable scope 1 level. [variableScope]**

int buf[10];

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:133:9: style:** **Local variable 'x' shadows outer variable [shadowVariable]**

int x = 5;

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:117:7: note:** Shadowed declaration

int x = 0;

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:133:9: note:** Shadow variable

int x = 5;

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:134:9: style:** **Local variable 'y' shadows outer variable [shadowVariable]**

int y = 5;

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:118:7: note:** Shadowed declaration

int y = 0;

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:134:9: note:** Shadow variable

int y = 5;

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:135:9: style:** **Local variable 'z' shadows outer variable [shadowVariable]**

int z = 5;

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:119:7: note:** Shadowed declaration

int z = 0;

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:135:9: note:** Shadow variable

int z = 5;

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:87:30: error:** **Using iterator to local container 'items' that may be invalid. [invalidContainer]**

for (iter = items.begin(); iter != items.end(); ++iter) {

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:87:26: note:** Iterator to container is created here.

for (iter = items.begin(); iter != items.end(); ++iter) {

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:88:15: note:** Assuming condition is true.

if (\*iter == 2) {

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:88:15: note:** Assuming condition is true.

if (\*iter == 2) {

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:87:35: note:** Assuming condition is true.

for (iter = items.begin(); iter != items.end(); ++iter) {

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:89:13: note:** After calling 'erase', iterators or references to the container's data may be invalid .

items.erase(iter);

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:82:20: note:** Variable created here.

std::vector<int> items;

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:87:30: note:** Using iterator to local container 'items' that may be invalid.

for (iter = items.begin(); iter != items.end(); ++iter) {

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:66:16: style:** **Variable 'buf[count]' is assigned a value that is never used. [unreadVariable]**

buf[count] = 0; // <- ERROR

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:109:7: style:** **Variable 'tok' is assigned a value that is never used. [unreadVariable]**

tok = tok->next();

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:50:0: style:** **The function 'DontThrow' is never used. [unusedFunction]**

^

**/home/francis/Documents/cs405m5StaticTesting/QuestionableCode.cpp:69:0: style:** **The function 'do\_something\_useless' is never used. [unusedFunction]**

^

**nofile:0:0: information:** **Cppcheck cannot find all the include files. Cppcheck can check the code without the include files found. But the results will probably be more**

accurate if all the include files are found. Please check your project's include directories and add all of them as include directories for Cppcheck. To see what files

Cppcheck cannot find use --check-config. [missingIncludeSystem]

\* Terminal will be reused by tasks, press any key to close it.