**Green Pace Developer: Security Policy Guide Template**



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

## Overview

Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.

## Purpose

This policy defines the core security principles; C/C++ coding standards; authorization, authentication, and auditing standards; and data encryption standards. This article explains the differences between policy, standards, principles, and practices (guidelines and procedure): [Understanding the Hierarchy of Principles, Policies, Standards, Procedures, and Guidelines](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

## Scope

This document applies to all staff that create, deploy, or support custom software at Green Pace.

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | When validating inputs the default standpoint is to take all inputs as malicious and to grant inputs based on the scope of what is expected by the code, such as limiting only numbers when creating a simple calculator. This can be done through systems like regex where we validate the class of the input compared to the expected context. |
| 1. Heed Compiler Warnings | When warnings are created when compiling code its best to address them correctly but finding root cause and correcting issues such as class structure, type comparisons, these warnings will still allow code to compile but will add unnecessary risk into the code and could lead to avoidable issues like unwanted access into a system. |
| 1. Architect and Design for Security Policies | When creating a design or adding to an existing structure always enforce security and rule policies, always contain user permissions and privileges, even when transversing from one isolated system to another. This can prevent the unintended escalation of privileges to a system that could cause issues. |
| 1. Keep It Simple | Software and systems should be kept simple to avoid unnecessary complexity. As such any unneeded bloat in a software will cause issues when trying to add security . |
| 1. Default Deny | Default behaviors of a system should always be to deny access and to reduce impact. |
| 1. Adhere to the Principle of Least Privilege | At all access points users should always be at the lowest access permissions, and when elevated permissions are needed then they should be done in a timely manner to reduce the chances of attacks. |
| 1. Sanitize Data Sent to Other Systems | When passing data from one system to another limit the information being passed to what’s absolutely needed, a system just asking for what day it is doesn’t need to know that there is some event going on the next week. |
| 1. Practice Defense in Depth | Add layers to security, make the process as wide covering as possible, such as a first layer should only allow smith to access a system, the second should only allow john smith, and the third layer will deny all others. |
| 1. Use Effective Quality Assurance Techniques | Always place ways to identify and eliminate security risks and vulnerabilities, testing should be done continuously to fins holes in a system and measure should be in place to isolate such attacks and what caused them. |
| 1. Adopt a Secure Coding Standard | When creating a system identity and work on fixing security requirements as soon as possible, always start and end every cycle of development with security measures. |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **EXP58-CPP. Pass an object of the correct type to va\_start** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | While rule [DCL50-CPP. Do not define a C-style variadic function](https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL50-CPP.+Do+not+define+a+C-style+variadic+function) forbids creation of such functions, they may still be defined when that function has external, C language linkage. Under these circumstances, care must be taken when invoking the va\_start() macro. The C-standard library macro va\_start() imposes several semantic restrictions on the type of the value of its second parameter. The C Standard, subclause 7.16.1.4, paragraph 4 |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the object passed to va\_start() will undergo a default argument promotion, which results in undefined behavior. |
| #include <cstdarg>    extern "C" void f(float a, ...) {    va\_list list;    va\_start(list, a);    // ...    va\_end(list);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, f() accepts a double instead of a float. |
| #include <cstdarg>    extern "C" void f(double a, ...) {    va\_list list;    va\_start(list, a);    // ...    va\_end(list);  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wvarargs | Does not catch the violation in the third noncompliant code example (it is conditionally supported by [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-Clang)) |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | BADMACRO.STDARG\_H | Use of <stdarg.h> Feature |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | C++3852, C++3853 |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.2 | CERT.VA\_START.TYPE |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **DCL59-CPP. Do not define an unnamed namespace in a header file** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Production-quality C++ code frequently uses *header files* as a means to share code between translation units. A header file is any file that is inserted into a translation unit through an #include directive. Do not define an unnamed namespace in a header file. When an unnamed namespace is defined in a header file, it can lead to surprising results. Due to default internal linkage, each translation unit will define its own unique instance of members of the unnamed namespace that are [ODR-used](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-odr-use) within that translation unit. This can cause unexpected results, bloat the resulting executable, or inadvertently trigger [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior) due to one-definition rule (ODR) violations |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the variable v is defined in an unnamed namespace within a header file and is accessed from two separate translation units. Each translation unit prints the current value of v and then assigns a new value into it. However, because v is defined within an unnamed namespace, each translation unit operates on its own instance of v, resulting in unexpected output. |
| // a.h  #ifndef A\_HEADER\_FILE  #define A\_HEADER\_FILE    namespace {  int v;  }    #endif // A\_HEADER\_FILE    // a.cpp  #include "a.h"  #include <iostream>    void f() {    std::cout << "f(): " << v << std::endl;    v = 42;    // ...  }    // b.cpp  #include "a.h"  #include <iostream>    void g() {    std::cout << "g(): " << v << std::endl;    v = 100;  }    int main() {    extern void f();    f(); // Prints v, sets it to 42    g(); // Prints v, sets it to 100    f();    g();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, v is defined in only one translation unit but is externally visible to all translation units, resulting in the expected behavior. |
| // a.h  #ifndef A\_HEADER\_FILE  #define A\_HEADER\_FILE    extern int v;    #endif // A\_HEADER\_FILE    // a.cpp  #include "a.h"  #include <iostream>    int v; // Definition of global variable v    void f() {    std::cout << "f(): " << v << std::endl;    v = 42;    // ...  }    // b.cpp  #include "a.h"  #include <iostream>    void g() {    std::cout << "g(): " << v << std::endl;    v = 100;  }  int main() {    extern void f();    f(); // Prints v, sets it to 42    g(); // Prints v, sets it to 100    f(); // Prints v, sets it back to 42    g(); // Prints v, sets it back to 100  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | unnamed-namespace-header | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-DCL59 |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | cert-dcl59-cpp | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | LANG.STRUCT.DECL.ANH | Anonymous Namespace in Header File |

#### Coding Standard 3

| **Coding Standard** | **Label** | **STR53-CPP. Range check element access** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | The std::string index operators const\_reference operator[](size\_type) const and reference operator[](size\_type) return the character stored at the specified position, pos. When pos >= size(), a reference to an object of type charT with value charT() is returned. The index operators are unchecked (no exceptions are thrown for range errors), and attempting to modify the resulting out-of-range object results in [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the value returned by the call to get\_index() may be greater than the number of elements stored in the string, resulting in [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #include <string>    extern std::size\_t get\_index();    void f() {    std::string s("01234567");    s[get\_index()] = '1';  } |

| **Compliant Code** |
| --- |
| This compliant solution uses the std::basic\_string::at() function, which behaves in a similar fashion to the index operator[] but throws a std::out\_of\_range exception if pos >= size(). |
| #include <stdexcept>  #include <string>  extern std::size\_t get\_index();    void f() {    std::string s("01234567");    try {      s.at(get\_index()) = '1';    } catch (std::out\_of\_range &) {      // Handle error    }  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | Medium | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | assert\_failure |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | LANG.MEM.BO LANG.MEM.BU LANG.MEM.TBA LANG.MEM.TO LANG.MEM.TU | Buffer overrun Buffer underrun Tainted buffer access Type overrun Type underrun |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | C++3162, C++3163, C++3164, C++3165 |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-STR53-a | Guarantee that container indices are within the valid range |

#### Coding Standard 4

| **Coding Standard** | **Label** | **MEM10-C. Define and use a pointer validation function** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CCC] | Many functions accept pointers as arguments. If the function dereferences an [invalid pointer](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-invalidpointer) (as in [EXP34-C. Do not dereference null pointers](https://wiki.sei.cmu.edu/confluence/display/c/EXP34-C.+Do+not+dereference+null+pointers)) or reads or writes to a pointer that does not refer to an object, the results are [undefined](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-undefined). Typically, the program will [terminate abnormally](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-abnormaltermination) when an invalid pointer is dereferenced, but it is possible for an invalid pointer to be dereferenced and its memory changed without abnormal termination [[Jack 2007](https://wiki.sei.cmu.edu/confluence/display/c/AA.+Bibliography#AA.Bibliography-Jack07)]. Such programs can be difficult to debug because of the difficulty in determining if a pointer is [valid](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-validpointer). |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the incr() function increments the value referenced by its argument. It also ensures that its argument is not a null pointer. But the pointer could still be invalid, causing the function to corrupt memory or terminate abnormally |
| void incr(int \*intptr) {    if (intptr == NULL) {      /\* Handle error \*/    }    (\*intptr)++;  } |

| **Compliant Code** |
| --- |
| This incr() function can be improved by using the valid() function. The resulting implementation is less likely to dereference an invalid pointer or write to memory that is outside the bounds of a valid object. |
| void incr(int \*intptr) {    if (!valid(intptr)) {      /\* Handle error \*/    }    (\*intptr)++;  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | High | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 159 S | Enhanced enforcement |

#### Coding Standard 5

| **Coding Standard** | **Label** | **MEM51-CPP. Properly deallocate dynamically allocated resources** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | The C programming language provides several ways to allocate memory, such as std::malloc(), std::calloc(), and std::realloc(), which can be used by a C++ program. However, the C programming language defines only a single way to free the allocated memory: std::free(). See [MEM31-C. Free dynamically allocated memory when no longer needed](https://wiki.sei.cmu.edu/confluence/display/c/MEM31-C.+Free+dynamically+allocated+memory+when+no+longer+needed) and [MEM34-C. Only free memory allocated dynamically](https://wiki.sei.cmu.edu/confluence/display/c/MEM34-C.+Only+free+memory+allocated+dynamically) for rules specifically regarding C allocation and deallocation requirements. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the local variable space is passed as the expression to the placement new operator. The resulting pointer of that call is then passed to ::operator delete(), resulting in [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior) due to ::operator delete() attempting to free memory that was not returned by ::operator new(). |
| #include <iostream>    struct S {    S() { std::cout << "S::S()" << std::endl; }    ~S() { std::cout << "S::~S()" << std::endl; }  };    void f() {    alignas(struct S) char space[sizeof(struct S)];    S \*s1 = new (&space) S;      // ...      delete s1;  } |

| **Compliant Code** |
| --- |
| This compliant solution removes the call to ::operator delete(), instead explicitly calling s1's destructor. This is one of the few times when explicitly invoking a destructor is warranted. |
| #include <iostream>    struct S {    S() { std::cout << "S::S()" << std::endl; }    ~S() { std::cout << "S::~S()" << std::endl; }  };    void f() {    alignas(struct S) char space[sizeof(struct S)];    S \*s1 = new (&space) S;      // ...      s1->~S(); |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | invalid\_dynamic\_memory\_allocation dangling\_pointer\_use |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-MEM51 |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-cplusplus.NewDeleteLeaks  -Wmismatched-new-delete clang-analyzer-unix.MismatchedDeallocator | Checked by clang-tidy, but does not catch all violations of this rule |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | ALLOC.FNH ALLOC.DF ALLOC.TM ALLOC.LEAK | Free non-heap variable Double free Type mismatch Leak |

#### Coding Standard 6

| **Coding Standard** | **Label** | **MSC60-J. Do not use assertions to verify the absence of runtime errors** |
| --- | --- | --- |
| **Assertions** | [STD-006-JJJ] | Diagnostic tests can be incorporated into programs by using the assert statement. Assertions are primarily intended for use during debugging and are often turned off before code is deployed by using the -disableassertions (or -da) Java runtime switch. Consequently, assertions should be used to protect against incorrect programmer assumptions and not for runtime error checking. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example uses the assert statement to verify that input was available: |
| BufferedReader br;    // Set up the BufferedReader br    String line;    // ...    line = br.readLine();    assert line != null; |

| **Compliant Code** |
| --- |
| This compliant solution demonstrates the recommended way to detect and handle unavailability of input: |
| BufferedReader br;    // Set up the BufferedReader br    String line;    // ...    line = br.readLine();    if (line == null) {    // Handle error  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| N/A | N/A | N/A | N/A | N/A |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft Jtest](https://wiki.sei.cmu.edu/confluence/display/java/Parasoft) | 2024.1 | CERT.MSC60.ASSERT | Do not use assertions in production code |

#### Coding Standard 7

| **Coding Standard** | **Label** | **ERR58-CPP. Handle all exceptions thrown before main() begins executing** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | When declaring an object with static or thread storage duration, and that object is not declared within a function block scope, the type's constructor must be declared noexcept and must comply with [ERR55-CPP. Honor exception specifications](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR55-CPP.+Honor+exception+specifications). Additionally, the initializer for such a declaration, if any, must not throw an uncaught exception (including from any implicitly constructed objects that are created as a part of the initialization). If an uncaught exception is thrown before main() is executed, or if an uncaught exception is thrown after main() has finished executing, there are no further opportunities to handle the exception and it results in implementation-defined behavior. (See [ERR50-CPP. Do not abruptly terminate the program](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR50-CPP.+Do+not+abruptly+terminate+the+program) for further details.) |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the constructor for S may throw an exception that is not caught when globalS is constructed during program startup. |
| struct S {    S() noexcept(false);  };    static S globalS; |

| **Compliant Code** |
| --- |
| This compliant solution makes globalS into a local variable with static storage duration, allowing any exceptions thrown during object construction to be caught because the constructor for S will be executed the first time the function globalS() is called rather than at program startup. This solution does require the programmer to modify source code so that previous uses of globalS are replaced by a function call to globalS(). |
| struct S {    S() noexcept(false);  };    S &globalS() {    try {      static S s;      return s;    } catch (...) {      // Handle error, perhaps by logging it and gracefully terminating the application.    }    // Unreachable.  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Low | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **potentially-throwing-static-initialization** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-ERR58** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | cert-err58-cpp | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **LANG.STRUCT.EXCP.THROW** | Use of throw |

#### Coding Standard 8

| **Coding Standard** | **Label** | **CTR57-CPP. Provide a valid ordering predicate** |
| --- | --- | --- |
| [Student Choice] | [STD-008-CPP] | Providing an invalid ordering predicate for an associative container (e.g., sets, maps, multisets, and multimaps), or as a comparison criterion with the sorting algorithms, can result in erratic behavior or infinite loops [[Meyers 01](https://wiki.sei.cmu.edu/confluence/display/cplusplus/AA.+Bibliography#AA.Bibliography-Meyers01)]. When an ordering predicate is required for an associative container or a generic standard template library algorithm, the predicate must meet the requirements for inducing a strict weak ordering. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the std::set object is created with a comparator that does not adhere to the strict weak ordering requirement. Specifically, it fails to return false for equivalent values. As a result, the behavior of iterating over the results from std::set::equal\_range results in [unspecified behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-unspecifiedbehavior). |
| #include <functional>  #include <iostream>  #include <set>    void f() {    std::set<int, std::less\_equal<int>> s{5, 10, 20};    for (auto r = s.equal\_range(10); r.first != r.second; ++r.first) {      std::cout << \*r.first << std::endl;    }  } |

| **Compliant Code** |
| --- |
| This compliant solution uses the default comparator with std::set instead of providing an invalid one. |
| #include <iostream>  #include <set>    void f() {    std::set<int> s{5, 10, 20};    for (auto r = s.equal\_range(10); r.first != r.second; ++r.first) {      std::cout << \*r.first << std::endl;    }  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | High | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | C++3293 |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-CTR57-a | For associative containers never use comparison function returning true for equal values |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: CTR57-CPP](https://www.mathworks.com/help/bugfinder/ref/certcctr57cpp.html) | Checks for predicate lacking strict weak ordering (rule partially covered). |

#### Coding Standard 9

| **Coding Standard** | **Label** | **FIO51-CPP. Close files when they are no longer needed** |
| --- | --- | --- |
| [Student Choice] | [STD-009-CPP] | A call to the std::basic\_filebuf<T>::open() function must be matched with a call to std::basic\_filebuf<T>::close() before the lifetime of the last pointer that stores the return value of the call has ended or before normal program termination, whichever occurs first. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a std::fstream object file is constructed. The constructor for std::fstream calls std::basic\_filebuf<T>::open(), and the default std::terminate\_handler called by std::terminate() is std::abort(), which does not call destructors. Consequently, the underlying std::basic\_filebuf<T> object maintained by the object is not properly closed. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }    // ...    std::terminate();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, std::fstream::close() is called before std::terminate() is called, ensuring that the file resources are properly closed. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }    // ...    file.close();    if (file.fail()) {      // Handle error    }    std::terminate();  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | ALLOC.LEAK | Leak |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | DF4786, DF4787, DF4788 |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.2 | RH.LEAK |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-FIO51-a | Ensure resources are freed |

#### Coding Standard 10

| **Coding Standard** | **Label** | **ERR57-CPP. Do not leak resources when handling exceptions** |
| --- | --- | --- |
| [Student Choice] | [STD-010-CPP] | Reclaiming resources when exceptions are thrown is important. An exception being thrown may result in cleanup code being bypassed or an object being left in a partially initialized state. Such a partially initialized object would violate basic exception safety, as described in [ERR56-CPP. Guarantee exception safety](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR56-CPP.+Guarantee+exception+safety). It is preferable that resources be reclaimed automatically, using the [RAII](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-RAII) design pattern [[Stroustrup 2001](https://wiki.sei.cmu.edu/confluence/display/cplusplus/AA.+Bibliography#AA.Bibliography-Stroustrup01)], when objects go out of scope. This technique avoids the need to write complex cleanup code when allocating resources. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, pst is not properly released when process\_item throws an exception, causing a resource leak. |
| #include <new>    struct SomeType {    SomeType() noexcept; // Performs nontrivial initialization.    ~SomeType(); // Performs nontrivial finalization.    void process\_item() noexcept(false);  };    void f() {    SomeType \*pst = new (std::nothrow) SomeType();    if (!pst) {      // Handle error      return;    }      try {      pst->process\_item();    } catch (...) {      // Process error, but do not recover from it; rethrow.      throw;    }    delete pst;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the exception handler frees pst by calling delete. |
| #include <new>    struct SomeType {    SomeType() noexcept; // Performs nontrivial initialization.    ~SomeType(); // Performs nontrivial finalization.      void process\_item() noexcept(false);  };    void f() {    SomeType \*pst = new (std::nothrow) SomeType();    if (!pst) {      // Handle error      return;    }    try {      pst->process\_item();    } catch (...) {      // Process error, but do not recover from it; rethrow.      delete pst;      throw;    }    delete pst;  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | High | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | ALLOC.LEAK | Leak |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | DF4756, DF4757, DF4758 |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.2 | CL.MLK MLK.MIGHT MLK.MUST MLK.RET.MIGHT MLK.RET.MUST RH.LEAK |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 50 D | Partially implemented |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

[Insert your written explanations here.]

### Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-002-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-003-CPP | High | Unlikely | Medium | P6 | L2 |
| STD-004-CCC | High | Unlikely | High | P3 | L3 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-JJJ | N/A | N/A | N/A | N/A | N/A |
| STD-007-CPP | Low | Likely | Low | P9 | L2 |
| STD-008-CPP | Low | Probable | High | P2 | L3 |
| STD-009-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-010-CPP | Low | Probable | High | P2 | L3 |

### Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Protecting data that is stored in devices by encryption, this prevents the collection and use of data by an attack. |
| Encryption in flight | Encrypting data as it passes from location to location, allowing the use of keys prevents unauthorized use of data. |
| Encryption in use | Restricting access to data and limiting who can interact with information allows data to not get into the wrong hands. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Verification of the user, confirming the user is who they say they are by using a method such as something they know, something they have. |
| Authorization | Limiting the users access to systems by using least privilege. |
| Accounting | Keeping correct logs for access to systems and checking what is happening on a system or device. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

### Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

The only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

## Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

## Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

## Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

## Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 10/20/2024 | Project 1 | Eric Galtieri | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

| Language | Acronym |
| --- | --- |
| C++ | CPP |
| C | CLG |
| Java | JAV |