**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 | It is important to ensure that data when provided is correct and validated to prevent any unwanted or preventable vulnerabilities like SQL injection buffer overflow. Verifying the data inputted is correct can ensure any risks can be managed and minimized. |
| 1. Heed Compiler Warnings | When warnings occur by the compiler, we must pay attention, since they are occurring for a reason and trying to alert us of a potential security issue, ignoring these can become detrimental and lead to vulnerabilities to be exploited. |
| 1. Architect and Design for Security Policies | During designing a system, we need to always keep in mind the overhead policies that we must abide by to avoid unwanted outcomes, financially or ethically. When designing or building with this in mind, we reduce any chances of vulnerabilities occurring later. |
| 1. Keep It Simple | Keep code and the system simple so that longevity and reusability can be possible, reducing the complexity of a system will create positive effects and make it easier to identify an issue when or if they occur. |
| 1. Default Deny | When keeping access control in mind, denial should be the start position, only allow resources to be accessible after identity and intention is proven to avoid any unauthorized access to sensitive information. |
| 1. Adhere to the Principle of Least Privilege | Allowing only the most minimal access or permission to resources to an individual to the point they can do their duty but also not allowing them more access than they will need to operate their daily duties. |
| 1. Sanitize Data Sent to Other Systems | Data that is shared and communicated between systems should always be cleaned or reviewed to remove any sort of malicious attachments and preventing sensitive data to be compromised. |
| 1. Practice Defense in Depth | Having more than one break through point any having multiple layers of security to bypass will prevent against multiple type of threats and using this approach will help when sensitive data is at stake and if one security wall is breached, at least there are multiple others that will have to be hurdled before this data can be risked. |
| 1. Use Effective Quality Assurance Techniques | Implementing routine testing, reviews and practicing quality assurance will help detect security flaws early on but should also be practiced after launch for maintenance and upkeep. |
| 1. Adopt a Secure Coding Standard | Setting the standard and having a guideline to develop and abide by will set a bar for others to follow and reduce the possibility of vulnerabilities being developed. |

### 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** | **Obey the one-definition rule** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | The DCL60-CPP guideline from the CERT C++ Coding Standard ensures that there is exactly one definition of non-inline entities in a C++ program, preventing potential issues related to multiple definitions, which could lead to undefined behavior. Top of Form |

| **Noncompliant Code** |
| --- |
| [Noncompliant description]  In this bad code, two parts of the program define a class named "S" differently, even though they both do the same thing. This breaks the rule that says there should be only one clear definition, leading to undefined behavior. |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  Resource: https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL60-CPP.+Obey+the+one-definition+rule  // a.cpp  **struct** S {  **int** a;  };    // b.cpp  **class** S {  **public**:  **int** a;  }; |

| **Compliant Code** |
| --- |
| [Compliant description]  It would be best to want the same class definition to be available in different parts of the program because it's commonly used, they should use a header file to make it accessible in both places. |
| [Compliant code block; code should be indented using 12-point Courier New font.]  Resource: https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL60-CPP.+Obey+the+one-definition+rule  // S.h  **struct** S {  **int** a;  };    // a.cpp  #include "S.h"    // b.cpp  #include "S.h" |

**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.]  Principle DCL60-CPP "Obey the one-definition rule" helps developers to follow this crucial rule in C++ programming. This principle ensures that there is only one exact definition in a C++ program, preventing any potential errors. When applied to the data type standard it helps maintain consistency and reduces the risk of undefined behavior or security vulnerabilities. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Likely | Low | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2022.1 | DCL60-CPP Checker | Auto scan and enforce one definition rule. Integrate into CI/CD pipeline to ensure developers receive feedback during build process |
| Axivion Bauhaus Suite | 7.0 | MISRA C++ | Focuses on detecting multiple declarations in the same scope |
| PVS-Studio | 7.17 | V201 Multiple Definitions | Detects issues with multiple definitions and provides an additional layer against violations |
| Checkmarx | 10.0 | C++ MultDef | SAST tool focuses on identifying multiple definitions of the symbols in the code |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Do not access an object outside of its lifetime** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | The EXP54-CPP standard shows the importance of ensuring that objects in a C++ program are accessed only within their valid scope or lifetime. This rule is crucial for preventing undefined behavior, memory corruption, and security vulnerabilities caused by accessing objects that have been destroyed or have gone out of scope. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description]  In this example code, a pointer to an object is used to call a non-static member function of the object prior to the beginning of the pointer's lifetime, resulting in an undefined behavior. |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  Resource: https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP54CPP.+Do+not+access+an+object+outside+of+its+lifetime  **struct** S {  **void** mem\_fn();  };    **void** f() {    S \*s;    s->mem\_fn();  } |

| **Compliant Code** |
| --- |
| [Compliant description]  It is best for storage to be obtained for the pointer prior being called upon and would not dynamically allocate memory directly but use an automatic variable. |
| [Compliant code block; code should be indented using 12-point Courier New font.]  Resource: https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP54CPP.+Do+not+access+an+object+outside+of+its+lifetime  **struct** S {  **void** mem\_fn();  };    **void** f() {    S \*s = **new** S;    s->mem\_fn();  **delete** 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.]  The EXP54-CPP standard “Do not access an object outside of its lifetime" falls within the Resource Acquisition Is Initialization principle in C++. This prioritizes that resource management should be tied to object lifetimes. This ensures that when an object is accessed, it is in a valid state, and when it goes out of scope, resources are released. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2022.1 | EXP54-CPP Checker | Auto scan for violations integrating this tool will ensure continuous monitoring |
| Cppcheck | 2.6 | Object Lifetime Checking | Open-source tool that can detect issues related to object lifetime and provide extra layer of identifying issues |
| Clang Static Analyzer | 14.0 | Object Use After Free | Clang Static Analyzer includes checker for detecting object use after free which aligns with EXP54 standard this will enhance code quality and security |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Guarantee that storage for strings has sufficient space for character data and the null terminator** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | This standard ensures that memory allocated for strings includes enough space for the actual text data and the special null character that marks the end of the string. This helps prevent issues like data overflow and keeps your program running smoothly and securely. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description]  This example code leads to a buffer overflow due to the input being unbounded |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  Resource:  https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR50CPP.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator  #include <iostream>    **void** f() {  **char** buf[12];    std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| [Compliant description]  It is best to ensure the data is not shorted and to use std::string to guard against buffer overflows |
| [Compliant code block; code should be indented using 12-point Courier New font.]  Resource:  https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR50CPP.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator  #include <iostream>  #include <string>    **void** f() {    std::string input;    std::string stringOne, stringTwo;    std::cin >> stringOne >> stringTwo;  } |

**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.]  This standard aligns with the principle of 'Validate Input Data,' ensuring that strings entering the program are properly null terminated. Without null termination functions relying on it may fail, leading to undefined behavior. Additionally, it supports the principle of 'Develop a Coding Standard,' emphasizing the general aim to avoid incorrect string data within our codebase and internal application structures. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2022.1 | STR31-CPP | Auto scan for violations related to insufficient storage for strings to ensure continuous monitoring |
| PVS-Studio | 17.7 | V518 Checker | Designed to detect various types of errors and vulnerabilities and checks for issue and security concerns. |
| TrustInSoft Analyzer | 1.38 | Mem\_Access Check | Helps developers to identify and eliminate security vulnerabilities, compliance issues and other critical flaws. |
| Parasoft | 2022.2 | STR31-e Check | Security testing tool aimed to reduce defects, improve efficiency during the software development process and avoid potential breaches. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Prevent SQL injection** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | IDS00-J helps protect software from security threats related to SQL injection which could happen when untrusted data is used in SQL queries, leading to unauthorized data access or changes. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description]  This code example shows it attempting to authenticate the user to the system but this could allow a SQL injection attack because using an unsanitized input argument allows an attacker to inject |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  Resource : https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection  **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.ResultSet;  **import** java.sql.SQLException;  **import** java.sql.Statement;    **class** Login {  **public** Connection getConnection() **throws** SQLException {      DriverManager.registerDriver(**new**              com.microsoft.sqlserver.jdbc.SQLServerDriver());      String dbConnection =        PropertyManager.getProperty("db.connection");      // Can hold some value like      // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"  **return** DriverManager.getConnection(dbConnection);    }      String hashPassword(**char**[] password) {      // Create hash of password    }    **public** **void** doPrivilegedAction(String username, **char**[] password)  **throws** SQLException {      Connection connection = getConnection();  **if** (connection == **null**) {        // Handle error      }  **try** {        String pwd = hashPassword(password);          String sqlString = "SELECT \* FROM db\_user WHERE username = '"                           + username +                           "' AND password = '" + pwd + "'";        Statement stmt = connection.createStatement();        ResultSet rs = stmt.executeQuery(sqlString);    **if** (!rs.next()) {  **throw** **new** SecurityException(            "User name or password incorrect"          );        }          // Authenticated; proceed      } **finally** {  **try** {          connection.close();        } **catch** (SQLException x) {          // Forward to handler        }      }    }  } |

| **Compliant Code** |
| --- |
| [Compliant description]  This right way of doing it involves using a special placeholder in the query for the input. It also checks the length of the username to stop attackers from using very long usernames. |
| [Compliant code block; code should be indented using 12-point Courier New font.]  Resource : https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection  **public** **void** doPrivilegedAction(    String username, **char**[] password  ) **throws** SQLException {    Connection connection = getConnection();  **if** (connection == **null**) {      // Handle error    }  **try** {      String pwd = hashPassword(password);        // Validate username length  **if** (username.length() > 8) {        // Handle error      }        String sqlString =        "select \* from db\_user where username=? and password=?";      PreparedStatement stmt = connection.prepareStatement(sqlString);      stmt.setString(1, username);      stmt.setString(2, pwd);      ResultSet rs = stmt.executeQuery();  **if** (!rs.next()) {  **throw** **new** SecurityException("User name or password incorrect");      }        // Authenticated; proceed    } **finally** {  **try** {        connection.close();      } **catch** (SQLException x) {        // Forward to handler      }    }  } |

**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.]  The standard ‘Prevent SQL injection’ is relevant as it is made to mitigate a specific type of attack vector like SQL Injection attacks. This showcases the practice of developing policies to counteract well-defined security threats. The principle of 'Sanitize Data sent to other systems' is applicable ensuring that data in the form of a query is reviewed before transmission to prevent potential compromise of data/system integrity and unexpected responses. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Occasional | Medium | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft | 2022.2 | Security SQL Check | Tool can assist with SQL check and ensure coding standards are upkept and violations are addressed |
| Coverity | 2022.2 | SQLI Check | Static code analysis tool that can focus on identifying security vulnerabilities and check for early on vulnerabilities |
| FindBugs | 3.0.1 | SQL String Passed | Static analysis tool identifying common bugs that helps developers maintain code quality and detect issues. |
| SQLMap | 1.4.12 | Sqlmap-H Check | Automatic scanning tool designed to identify and exploit SQL injection vulnerabilities and automates the process of detection |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Do not access freed memory** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | MEM50-CPP is a rule that says you should not use or refer to memory that has been released or "freed" by a program. This is because when memory is freed, it can be recycled or erased by the computer's memory management system. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description]  In this bad code example, the program tries to use something after it has been thrown away. This could be a risky because it can lead to a security problem called "write-after-free," allowing hackers to run their code using the program's permissions. |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  Resource: https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory  #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...  **delete** s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| [Compliant description]  This is a good code example because the allocated memory is not deallocated until is no longer needed. |
| [Compliant code block; code should be indented using 12-point Courier New font.]  Resource: https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory  #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...    s->f();  **delete** 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.]  The MEM50-CPP principle emphasizes the critical practice of not using or referring to memory that has been freed. This principle prevents future issues from coming up when accessing memory that has been recycled/deleted by the computer’s system. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Rarely | Medium | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2022.1 | MEM50-CPP Checker | Auto scan violations related to accessing freed memory and integrating this tool into the pipeline to ensure ongoing monitoring and feedback to be relayed. |
| Cppcheck | 2.12 | Return Value Check | Open-source tool that includes checks instances where accessing memory has been freed could occur or where invalidated memory is called upon |
| TrustInSoft Analyzer | 1.38 | Location Value Check | Tool that uses formal methods to mathematically verify code and aids in rigorous analysis related to matters of accessing freed memory |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Do not write syntactically ambiguous declarations** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | The DCL53-CPP standard reminds us not to create code that is confusing or unclear due to its structure. We should make sure it's easy to understand and doesn't have statements or declarations that can be interpreted in multiple ways. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description]  In this bad code example, they try to attempt to automatically lock and unlock a mutex using a special object called `std::unique\_lock` but since its written in an unclear way, it makes it confusing. |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  Resource: https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL53CPP.+Do+not+write+syntactically+ambiguous+declarations  #include <mutex>    **static** std::mutex m;  **static** **int** shared\_resource;    **void** increment\_by\_42() {    std::unique\_lock<std::mutex>(m);    shared\_resource += 42;  } |

| **Compliant Code** |
| --- |
| [Compliant description]  In this good code example, the proper converting constructor is called and the lock object is given an ID |
| [Compliant code block; code should be indented using 12-point Courier New font.]  Resource: https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL53CPP.+Do+not+write+syntactically+ambiguous+declarations  #include <mutex>    **static** std::mutex m;  **static** **int** shared\_resource;    **void** increment\_by\_42() {    std::unique\_lock<std::mutex> lock(m);    shared\_resource += 42;  } |

**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.]  This principle emphasizes the importance of avoiding code that can be interpreted in more than just one way. This principle ensures that the code is clear, easy to understand and doesn’t create uncertainty within the structure. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Low | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2022.1 | DCL53 CPP Check | Tool will assist by automatically detecting and preventing confusing or unclear code and maintain clarity. |
| PVS-Studio | 17.7 | Assertion Checker | Static code analysis tool that checks for an array of issues and can identify assertion usage ensuring that they are implemented correctly. |
| FindBugs | 3.0.1 | ClrCdCheck | This will help identify problems with assertion statements and provide suggestions for improvement. |
| Clang Static Analyzer | 14.0 | DCL53-e Check | This tool is part of the clang compiler and can assist with detecting problems with assertion statements during static analysis. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | With the ERR51-CPP standard, when something goes wrong in the program and an exception is raised, the code should have a plan to manage it. This standard says that when an exception happens, the program should look for a specific solution for that problem. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description]  In this example, when a problem happens in the `throwing\_func()` function, neither the `f()` function nor the main part of the program knows how to handle it. Since there's no plan in place to deal with the issue, the program is forced to stop and terminate. |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  Resources: https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions  **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {    f();  } |

| **Compliant Code** |
| --- |
| [Compliant description]  The main part of the program is ready to deal with any problems that might occur. This way, if something goes wrong, it can clean up everything and manage outside things in a controlled manner. |
| [Compliant code block; code should be indented using 12-point Courier New font.]  Resources: https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions  **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {  **try** {      f();    } **catch** (...) {      // 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.]  ERR51-CPP emphasizes the importance of having a clear plan for managing exceptions in C++ programs. When an exception is raised, the code should be designed to handle it gracefully by providing a specific solution for the encountered problem. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Occasional | Medium | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft | 2022.2 | ERR51-a | Comprehensive tool meant for quality assurance and can provide a static code analysis and aims to enhance quality and identify defects and improve overall reliability. |
| LDRA Tool Suite | 9.5 | ERR51-CPP Cert. | LDRA is a comprehensive tool designed for critical software development and can specialize in following compliance with safety and security standards. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Gracefully handle self-copy assignment** |
| --- | --- | --- |
| Object Oriented Programming | STD-008-CPP | This OOP54-CPP standard is about making sure that when a part of a program copies itself, it doesn't mess things up. The rules say that if something copies itself, it shouldn't end up in a messed-up state and it should handle the situation smartly. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description]  In this bad code example, there's a problem when something tries to make a copy of itself. It doesn't check to make sure it's not copying itself, which can lead to trouble and issues to occur. |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  Resource: https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP54-CPP.+Gracefully+handle+self-copy+assignment  #include <new>    **struct** S { S(**const** S &) noexcept; /\* ... \*/ };    **class** T {  **int** n;    S \*s1;    **public**:    T(**const** T &rhs) : n(rhs.n), s1(rhs.s1 ? **new** S(\*rhs.s1) : nullptr) {}    ~T() { **delete** s1; }      // ...      T& operator=(**const** T &rhs) {      n = rhs.n;  **delete** s1;      s1 = **new** S(\*rhs.s1);  **return** \***this**;    }  }; |

| **Compliant Code** |
| --- |
| [Compliant description]  This good code example checks if it's trying to make a copy of itself. If it is, it doesn't do anything and just stops. But if it's copying something else, it works as usual. |
| [Compliant code block; code should be indented using 12-point Courier New font.]  Resource: https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP54-CPP.+Gracefully+handle+self-copy+assignment  #include <new>    **struct** S { S(**const** S &) noexcept; /\* ... \*/ };    **class** T {  **int** n;    S \*s1;    **public**:    T(**const** T &rhs) : n(rhs.n), s1(rhs.s1 ? **new** S(\*rhs.s1) : nullptr) {}    ~T() { **delete** s1; }      // ...      T& operator=(**const** T &rhs) {  **if** (**this** != &rhs) {        n = rhs.n;  **delete** s1;  **try** {          s1 = **new** S(\*rhs.s1);        } **catch** (std::bad\_alloc &) {          s1 = nullptr; // For basic exception guarantees  **throw**;        }      }  **return** \***this**;    }  }; |

**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.]  The OOP54-CPP principle, emphasizing the necessity of handling self-copy assignments intelligently in object-oriented programming. This ensures that the program maintains a consistent and functional state even when performing operations involving self-copying. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Likely | Medium | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft | 2022.2 | Ck Copy | Parasoft is a comprehensive software testing and quality assurance tool that supports static code analysis, unit testing, functional testing, and API testing. It focuses on enhancing code quality, identifying defects, and improving overall software reliability. |
| LDRA Tool Suite | 9.5 | OOP54 Check | The LDRA Tool Suite is a specialized software analysis and testing solution, particularly designed for safety and security-critical software development. It offers static and dynamic analysis, unit testing, and code coverage analysis. |
| Axivion Bauhaus Suite | 7.0 | Cert++ OOP54 | Axivion Bauhaus Suite is a software analytics and quality management tool that focuses on maintaining and improving code quality. It supports static code analysis, architecture verification, and compliance monitoring. |
| Coverity | 2022.1 | OOP54-a Check | Coverity is a static code analysis tool that identifies security vulnerabilities, defects, and coding standards violations. It offers a wide range of checks to catch potential issues early in the development process. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Avoid deadlock by locking in a predefined order** |
| --- | --- | --- |
| Concurrency | STD-009-CPP | This standard CON53-CPP is about preventing problems that can happen when different parts of a computer program are trying to use the same thing at the same time. This can lead to the program getting stuck or "deadlocking." |

| **Noncompliant Code** |
| --- |
| [Noncompliant description]  In this bad code example, if one part of the program tries to lock something while another part is trying to lock something else at the same time, the program can get stuck and not work correctly. |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  Resource: https://wiki.sei.cmu.edu/confluence/display/cplusplus/CON53CPP.+Avoid+deadlock+by+locking+in+a+predefined+order  #include <mutex>  #include <thread>    **class** BankAccount {  **int** balance;  **public**:    std::mutex balanceMutex;    BankAccount() = **delete**;  **explicit** BankAccount(**int** initialAmount) : balance(initialAmount) {}  **int** get\_balance() **const** { **return** balance; }  **void** set\_balance(**int** amount) { balance = amount; }  };    **int** deposit(BankAccount \*from, BankAccount \*to, **int** amount) {    std::lock\_guard<std::mutex> from\_lock(from->balanceMutex);      // Not enough balance to transfer.  **if** (from->get\_balance() < amount) {  **return** -1; // Indicate error    }    std::lock\_guard<std::mutex> to\_lock(to->balanceMutex);      from->set\_balance(from->get\_balance() - amount);    to->set\_balance(to->get\_balance() + amount);    **return** 0;  }    **void** f(BankAccount \*ba1, BankAccount \*ba2) {    // Perform the deposits.    std::**thread** thr1(deposit, ba1, ba2, 100);    std::**thread** thr2(deposit, ba2, ba1, 100);    thr1.join();    thr2.join();  } |

| **Compliant Code** |
| --- |
| [Compliant description]  This good code example, the program avoids the problem of getting stuck by making sure that threads take turns in a specific order when using the `deposit()` function. This order helps prevent issues and keeps the program running smoothly. |
| [Compliant code block; code should be indented using 12-point Courier New font.]  Resource: https://wiki.sei.cmu.edu/confluence/display/cplusplus/CON53CPP.+Avoid+deadlock+by+locking+in+a+predefined+order  #include <atomic>  #include <mutex>  #include <thread>    **class** BankAccount {  **static** std::atomic<unsigned **int**> globalId;  **const** unsigned **int** id;  **int** balance;  **public**:    std::mutex balanceMutex;    BankAccount() = **delete**;  **explicit** BankAccount(**int** initialAmount) : id(globalId++), balance(initialAmount) {}    unsigned **int** get\_id() **const** { **return** id; }  **int** get\_balance() **const** { **return** balance; }  **void** set\_balance(**int** amount) { balance = amount; }  };    std::atomic<unsigned **int**> BankAccount::globalId(1);    **int** deposit(BankAccount \*from, BankAccount \*to, **int** amount) {    std::mutex \*first;    std::mutex \*second;    **if** (from->get\_id() == to->get\_id()) {  **return** -1; // Indicate error    }      // Ensure proper ordering for locking.  **if** (from->get\_id() < to->get\_id()) {      first = &from->balanceMutex;      second = &to->balanceMutex;    } **else** {      first = &to->balanceMutex;      second = &from->balanceMutex;    }    std::lock\_guard<std::mutex> firstLock(\*first);    std::lock\_guard<std::mutex> secondLock(\*second);      // Check for enough balance to transfer.  **if** (from->get\_balance() >= amount) {      from->set\_balance(from->get\_balance() - amount);      to->set\_balance(to->get\_balance() + amount);  **return** 0;    }  **return** -1;  }    **void** f(BankAccount \*ba1, BankAccount \*ba2) {    // Perform the deposits.    std::**thread** thr1(deposit, ba1, ba2, 100);    std::**thread** thr2(deposit, ba2, ba1, 100);    thr1.join();    thr2.join();  } |

**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.]  The CON53-CPP principle emphasizes the importance of preventing deadlock situations in concurrent programming. These principal guides developers to establish a predefined order for locking shared resources reducing the chances of deadlocks and enhancing program reliability. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| ThreadSanitizer | 18.0.0 | CERT CON53-CPP | ThreadSanitizer specializes in detecting data races and deadlocks in multithreaded applications. |
| Helgrind | 3.22.0 | CON53 Check | Helgrind is designed to detect synchronization errors, including deadlocks, in multithreaded C, C++, and Fortran programs. |
| Microsoft Concurrency Visualizer | 2022.2 |  | The Concurrency Visualizer is part of Microsoft's Visual Studio suite and provides visual insights into the behavior of concurrent applications, aiding in the identification of deadlocks and other concurrency-related issues. |
| Intel Inspector | 2023.3 | THCON53 Checker | Intel Inspector helps identify memory and threading errors, including deadlocks, in parallel and multithreaded applications. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Close files when they are no longer needed** |
| --- | --- | --- |
| Input Output | STD-010-CPP | FIO42-C is about making sure to close files in a computer program when they're not needed anymore. This is important to prevent problems and to free up resources. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description]  This bad code example shows that the file was left opened by the call fopen() and is not being closed before the function returns. |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  Resource: https://wiki.sei.cmu.edu/confluence/display/c/FIO42-C.+Close+files+when+they+are+no+longer+needed  #include <stdio.h>    **int** func(**const** **char** \*filename) {  **FILE** \*f = **fopen**(filename, "r");  **if** (NULL == f) {  **return** -1;    }    /\* ... \*/  **return** 0;  } |

| **Compliant Code** |
| --- |
| [Compliant description]  In this good code example, you can now see fclose() is now being incorporated which is best practice to ensure the program terminates properly. |
| [Compliant code block; code should be indented using 12-point Courier New font.]  Resource: https://wiki.sei.cmu.edu/confluence/display/c/FIO42-C.+Close+files+when+they+are+no+longer+needed  #include <stdio.h>    **int** func(**const** **char** \*filename) {  **FILE** \*f = **fopen**(filename, "r");  **if** (NULL == f) {  **return** -1;    }    /\* ... \*/  **if** (**fclose**(f) == EOF) {  **return** -1;    }  **return** 0;  } |

**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.]  FIO42-CPP emphasizes the importance of closing files in a C++ computer program when they are no longer needed. This standard is crucial to prevent issues and free up resources associated with open files. This standard aligns with the principle of resource management. Efficiently closing files when they are no longer needed adheres to the principle of responsible resource utilization, ensuring that system resources are released in a timely manner. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Low | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2022.1 | File handling rules | Coverity provides detailed reports on resource management issues, including file handling, and suggests remediation strategies. |
| Clang Static Analyzer | 14.0 | Resource management check | The Clang Static Analyzer provides detailed diagnostic messages for resource management issues, aiding developers in identifying and fixing problems. |
| PVS Studio | 17.7 | Resource leak check | PVS-Studio provides a range of checks related to resource management, offering insights into potential issues, and suggesting improvements. |
| Cppcheck | 2.12 | Resource leak check | Cppcheck generates reports highlighting potential resource management issues, aiding developers in maintaining code quality |

### 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.]

DevOps policies serve as a foundational framework for implementing a DevSecOps platform; however, the traditional DevOps approach, while proficient in expeditiously releasing software, falls short in adequately addressing modern security concerns. DevSecOps enhances this model by integrating security considerations from the outset, prompting a cultural shift in engineering and positioning security, software development, and operations as equal partners. The overarching objective of DevSecOps is to apply security principles throughout the entire Software Development Lifecycle. To align with the standards outlined in our policy, I propose several modifications across each phase of the conventional DevOps lifecycle to elevate it to DevSecOps standards.

In the planning phase, I advocate brainstorming traditional security attack patterns, such as SQL Injection or Man-in-the-Middle attacks, and devising countermeasures to be integrated into the product's inception. Moving to the development and build steps, incorporating mitigation approaches from our coding standards and practicing secure coding practices becomes paramount to minimize instability and mitigate external manipulations. The test phase should embrace automated unit testing for individual units of work and implement middle-tier integration testing to assess the entire application stack, covering potential security vulnerabilities like SQL Injection and memory management attacks.

For the release, deploy, operate, and monitor stages, adopting a secure container system is essential to prevent unauthorized access to the underlying OS. Utilizing log collection and automated log sniffing enhances our ability to detect and halt instructions before they disrupt the system. Monitoring network traffic can also help identify abnormal patterns, indicating a potential Distributed Denial of Service (DDoS) attack aimed at compromising a specific node. This holistic approach ensures a security-focused integration into every facet of the software development and deployment lifecycle.

### 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 | Likely | Low | Medium | 3 |
| STD-002-CPP | High | Likely | Medium | High | 2 |
| STD-003-CPP | High | Likely | Medium | High | 2 |
| STD-004-CPP | High | Occasional | Medium | Medium | 3 |
| STD-005-CPP | Medium | Unlikely | Medium | Medium | 2 |
| STD-006-CPP | Medium | Unlikely | Low | Low | 3 |
| STD-007-CPP | Low | Occasional | Medium | Low | 3 |
| STD-008-CPP | Medium | Likely | Medium | Low | 3 |
| STD-009-CPP | High | Likely | High | High | 2 |
| STD-010-CPP | Low | Likely | Low | Low | 3 |

### 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 in rest | Encryption in rest involves encrypting data that is not actively in use, such as data stored on a hard disk or in a database. The primary objective of this policy is to safeguard data in the event of a breach and subsequent theft of these files. In the case of an attacker gaining unauthorized access to the encrypted files, they would be faced with the challenge of either employing a time-consuming brute force attack on the encryption or acquiring the encryption keys to decrypt the data and render it usable. This intricate process extends over several years, significantly diminishing the effectiveness of the compromised data. |
| Encryption at flight | Encryption at flight pertains to the practice of safeguarding data as it traverses the network. An instance of this scenario is a web application retrieving data from a database. This policy is of utmost significance as it endeavors to secure data during its arguably most vulnerable phase, exposed to potential threats. The protection of data in transit is realized through the implementation of SSL/TLS connections between the web server and the database. An additional layer of defense is introduced through the utilization of a VPN, particularly in situations where network segments require connection. This precaution ensures that common packet sniffing tools like WireShark and TCPDump are incapable of intercepting and reconstructing the transmitted packets. |
| Encryption in use | Encryption in use involves safeguarding data while it is actively being employed. For instance, when a web server retrieves data from a database and performs computations or delivers data to a consumer. The protection of data-in-use is achieved through programming techniques such as employing secure memory practices such as the .NET ProtectedMemory class and leveraging Homomorphic encryption, enabling the manipulation of encrypted strings as if they were plaintext. Implementing these protective measures restricts the viability of attacking an application server, as breaching these safeguards would necessitate a formidable brute force effort. This constraint on decryption significantly diminishes the utility of the accessed data, as the decryption process is time-intensive and could span years. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the verification process through which a server or application establishes the identity of an individual. Its significance lies in determining rightful access to the application, delineating who should and should not have entry. This authentication procedure is typically governed by a login process, where users possess a login associated with their email address or a specific username, and a password is required for access verification. Enhancing security measures, additional authentication methods, such as two-factor OTP or the implementation of OAUTH technology for Single Sign-On (SSO) should be used. |
| Authorization | Authorization follows the authentication process and dictates the extent of access granted to a system, its files, and resources. Its purpose is to regulate a user's access level, often achieved through role-based permissions that outline permissible actions. Aligned with the principle of Default Deny, new users should not automatically possess access to any resources upon authentication. Access to specific resources should only be permitted through the assignment of a role. This policy is crucial as it prevents authenticated users lacking proper authorization from accessing data that should remain off-limits to them. Additionally, it facilitates a streamlined process for elevating an individual to a higher tier with enhanced resource access. |
| Accounting | Accounting is the process of monitoring alterations to a specific system or resource. Accounting includes tracking user access to files and changes made to database files. While systems can be engineered to offer this functionality, numerous software-based applications exist to aggregate such data and generate reports. For instance, if a particular user account is accessing a resource before a breach, having this information enables the security team to scrutinize the access and potentially prevent the breach. Following a breach, this data is instrumental for conducting a root cause analysis and swiftly sealing off the compromised pathway. This policy is pertinent to our scenario as it aligns with our objective of safeguarding protected data from unauthorized intrusion. |

**\***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.

|  |  |  |
| --- | --- | --- |
| **Principle** | **Standard** | **Justify** |
| Validate Input Data | STD-001-CPP | The validation of input data aligns with ensuring that the data types in the code are accurately and securely handled, preventing potential vulnerabilities. |
| Validate Input Data | STD-002-CPP | Ensuring that data values are validated corresponds to the principle of validating input data, preventing issues related to invalid or malicious inputs. |
| Validate Input Data | STD-003-CPP | String correctness is maintained by validating input data, ensuring that strings are correctly null-terminated and preventing issues related to malformed input. |
| Sanitize Data Sent to Other Systems | STD-004-CPP | Sanitizing data sent to other systems, as emphasized by the principle, directly addresses SQL injection concerns by ensuring that data is free from malicious content |
| Practice Defense in Depth | STD-005-CPP | Memory protection aligns with the principle of defense in depth, ensuring that memory-related vulnerabilities are addressed comprehensively. |
| Use Effective Quality Assurance Techniques | STD-006-CPP | Utilizing assertions corresponds to the effective quality assurance technique, ensuring that code behavior aligns with expectations. |
| Practice Defense in Depth | STD-007-CPP | Exception handling as per this standard aligns with the principle of defense in depth, providing additional layers of protection against unexpected events. |
| Adopt a Secure Coding Standard | STD-008-CPP | The standard related to object-oriented programming is in line with adopting a secure coding standard, ensuring that object-oriented practices adhere to security guidelines. |
| Adhere to the Principle of Least Privilege | STD-009-CPP | Preventing deadlocks in concurrency aligns with the principle of least privilege, minimizing access points and potential conflicts in the system |
| Default Deny | STD-010-CPP | The input-output standard corresponds to the principle of default deny, emphasizing that new users should not inherently have access to any resources without proper authorization. |

**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 |  |
| 1.0 | 11/10/2023 | Milestone Three | Paloma Rodriguez | Paloma Rodriguez |
| 1.0 | 11/25/2023 | Project One | Paloma Rodriguez | Paloma Rodriguez |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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