**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 6](#_Toc52464060)

[Coding Standard 3 8](#_Toc52464061)

[Coding Standard 4 10](#_Toc52464062)

[Coding Standard 5 13](#_Toc52464063)

[Coding Standard 6 15](#_Toc52464064)

[Coding Standard 7 17](#_Toc52464065)

[Coding Standard 8 19](#_Toc52464066)

[Coding Standard 9 21](#_Toc52464067)

[Coding Standard 10 23](#_Toc52464068)

[Defense-in-Depth Illustration 25](#_Toc52464069)

[Project One 25](#_Toc52464070)

[1. Revise the C/C++ Standards 25](#_Toc52464071)

[2. Risk Assessment 25](#_Toc52464072)

[3. Automated Detection 25](#_Toc52464073)

[4. Automation 25](#_Toc52464074)

[5. Summary of Risk Assessments 26](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 26](#_Toc52464076)

[7. Map the Principles 27](#_Toc52464077)

[Audit Controls and Management 28](#_Toc52464078)

[Enforcement 28](#_Toc52464079)

[Exceptions Process 28](#_Toc52464080)

[Distribution 29](#_Toc52464081)

[Policy Change Control 29](#_Toc52464082)

[Policy Version History 29](#_Toc52464083)

[Appendix A Lookups 29](#_Toc52464084)

[Approved C/C++ Language Acronyms 29](#_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 essential that input from the user to make sure it is valid. Invalid input is a big reason that errors occur and could cause your system to crash. More importantly, if input isn’t validated, a malicious user could get access to information that they are not supposed to have. |
| 1. Heed Compiler Warnings | It is important to make sure that if the compiler is giving you a warning that you make sure that you take your time to see why that warning is being provided. Even if the code compiles properly, that does not mean that the code is secure and there could be threats introduced by not heeding those warnings. |
| 1. Architect and Design for Security Policies | In essence this means to plan for security policies and design your code around these policies. This also means to put security on the forefront of your design process and coding practices. |
| 1. Keep It Simple | In essence, this principle is very easy; do not write more code than you need to. By writing more lines of code than necessary, you introduce the possibility of creating more errors or vulnerabilities. |
| 1. Default Deny | The importance of denying users access by default can limit the number of vulnerabilities that are written in the code. If a user does not have access to the information, there is less they can do to cause havoc. |
| 1. Adhere to the Principle of Least Privilege | The principle of lease privilege states that every user should have access to the bare minimum so that they can still do their job. If someone has access to more information than necessary, they can (either knowingly or unknowingly) get access to information that they should not have access to. |

|  |  |
| --- | --- |
| 1. Sanitize Data Sent to Other Systems | Data that is being sent to other systems should be only what the system needs. It is possible that if you just send data over without looking it over that you could send over more than you wanted. This information could be intercepted or used by other malicious individuals and cause issues with security. |
| 1. Practice Defense in Depth | Defense in depth is the process of protecting from attacks in multiple redundant layers to make sure that attacks from malicious individuals do not infiltrate your software. Protecting your system this way, gives you a better chance of catching attacks before they get too large to manage. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques allow us to catch problems before they can get out of hand. By implementing proper testing techniques, we can catch problems before they become multi-account leaks. |
| 1. Adopt a Secure Coding Standard | A secure coding standard gives everyone the same template to follow and holds people to the same standard. This also puts everyone on the same security standard so there are fewer possible chances for security leaks. |

### 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** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | If code is read into the wrong data type, the code can crash or malicious users can get data that they should not have access to. |

| **Noncompliant Code** |
| --- |
| The code block below shows that input is read from the user but we do not check if a number has been passed in. This can lead to issues later |
| std::cout << "Enter a number:" << std::endl;  int number;  std::cin >> number;  std::cout << "You entered: " << number << std::endl; |

| **Compliant Code** |
| --- |
| The code block below correctly makes sure that the value entered is actually a number. |
| std::cout << "Enter a number:" << std::endl;  int number;  std::cin >> number;  if (!(std::cin.good())) {  std::cout << "Invalid input: do something here" << std::endl;  }  else {  std::cout << "You entered: " << number << 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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | If you are making sure that a range is followed, and the user enters an invalid data value, they could cause havoc in the system. |

| **Noncompliant Code** |
| --- |
| The code below gets input from the user, but does not check if the value is in the proper range. |
| std::cout << "Enter a number between 1 and 5:" << std::endl;  int number;  std::cin >> number;  std::cout << "You entered: " << number << std::endl; |

| **Compliant Code** |
| --- |
| The code below asks for input from the user and then makes sure that the value is in the range. (Note: See STD-001-CPP for handling invalid data type; this block assumes that the data type is entered properly. |
| std::cout << "Enter a number between 1 and 5:" << std::endl;  int number;  std::cin >> number;  if (number > 5 || number < 1) {  std::cout << "Invalid number: do something here" << std::endl;  }  else {  std::cout << "You entered: " << number << 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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | It is important to make sure that a string has been entered properly and that the user’s string is correct. Invalid input can cause security vulnerabilities in the system. |

| **Noncompliant Code** |
| --- |
| The code below does not make sure that the string entered is correct and could lead to issues. |
| std::cout << "Enter a phrase:" << std::endl;  std::string phrase;  std::cin >> phrase;  std::cout << "You entered: " << phrase << std::endl; |

| **Compliant Code** |
| --- |
| This code not only check to make sure a phrase is correct, but also makes sure that the individual entered a confirmation letter properly. |
| std::cout << "Enter a phrase:" << std::endl;  std::string phrase;  std::string correct;  std::cin >> phrase;  std::cout << "You entered: " << phrase << std::endl;  std::cout << "Is this correct? (Y/N)" << std::endl;  std::cin >> correct;  if (correct.size() != 1) {  std::cout << "Invalid input: do something here" << std::endl;  }  else if (correct.compare("Y") == 0) {  std::cout <<"Thank you for verifying your entered phrase"<<std::endl;  }  else if (correct.compare("N") == 0) {  std::cout << "You did not enter that phrase" << std::endl;  }  else {  std::cout << "Invlid input: do something here" << 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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | If you do not make sure that the input to for and SQL inquiry, then a malicious user can gain access to information that they should not have access to. |

| **Noncompliant Code** |
| --- |
| This code block just lets the user enter whatever information and as such the user can enter 1=1 or a similar pattern to get more information than they are allowed to see. |
| std::string sql;  std::cin >> sql;  runQuery(sql); |

| **Compliant Code** |
| --- |
| This code block checks to see if there are any inputs where 1=1 or a similar pattern and shuts that search down before the information is shown. |
| //Create int pointers to compare values  int equalsPointer;  int wordLength;  int wordDifference;  //SQL inquiry string  std::string sql;  //Assign int values according to the values we are searching for  equalsPointer = sql.find("=");  wordLength = sql.find\_first\_of(" ", equalsPointer);  wordDifference = sql.find\_last\_of(" ", equalsPointer);  //Get input from the user for the search  std::cin >> sql;  //Do while loop that checks for any add ons to the end of an SQL statement; this is done by looking for '=' values in the SQL statement  do {  //If the values at the equals pointer and word difference are the same, then the rest of the values surrounding '=' needs to be checked  if (sql.at(equalsPointer + 1) == sql.at(wordDifference + 1)) {  //Changes each value by 1 to move to the next char in the values around the char '='  equalsPointer = equalsPointer + 1;  wordDifference = wordDifference + 1;  //If we have reached the '=' character than we know that both values are the same and we have a potential SQL injection attack  //This section will print out an error message to the console  if (sql.at(wordDifference + 1) == '=') {  std::cout << std::endl << "SQL Injection detected on:" << std::endl << sql << std::endl << "Search Stopped" << std::endl;  return false;  }  }  //If the values are not the same, we can just check if there are more '=' characters in the sql string  else {  equalsPointer = sql.find\_first\_of("=", (equalsPointer + 1));  wordLength = sql.find\_first\_of(" ", equalsPointer);  wordDifference = sql.find\_last\_of(" ", equalsPointer);  }  } while (equalsPointer > 0);  runQuery(sql); |

**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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | If memory is not protected, the user can write over data, and we can lose vital information that is stored elsewhere. |

| **Noncompliant Code** |
| --- |
| In the code block below input is read directly and it is possible that this memory location can be written over and the information could be lost forever. |
| const std::string account\_number = "CharlieBrown42";  char user\_input[20];  std::cout << "Enter a value: ";  std::cout << "You entered: " << user\_input << std::endl;  std::cout << "Account Number = " << account\_number << std::endl; |

| **Compliant Code** |
| --- |
| In this code block the input is only read to a point and will not allow the user to write over data that is stored. |
| const std::string account\_number = "CharlieBrown42";  char user\_input[20];  std::cout << "Enter a value: ";  //Added the getline command to make sure that there are not more than 20 characters added to the user\_input variable  std::cin.getline(user\_input, 20);  std::cout << "You entered: " << user\_input << std::endl;  std::cout << "Account Number = " << account\_number << 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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | If a programmer asserts that a value is something and does not check that, the code can spit out wrong data. |

| **Noncompliant Code** |
| --- |
| The code below shows an example where a value was changed, and the user did not correct it and the data calculations at the end of the block gave us the wrong output. |
| int x = 7;  int numSquared;  //do some more code here  x= 9;  //do more code here  numSquared = x \* x;  std::cout << numSquared; |

| **Compliant Code** |
| --- |
| The code block below makes sure that the value that we entered at the beginning is the same at the end to make sure that the data that we got is correct. This makes sure that a malicious user does not change any values to cause issues with the system. |
| int x = 7;  int numSquared;  //do some more code here  x= 9;  //do more code here  assert(x == 7);  numSquared = x \* x;  std::cout << numSquared; |

**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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Exceptions can be used to make sure that input or other data is valid and can keep malicious users from accessing data that they should not have access to. |

| **Noncompliant Code** |
| --- |
| This code block does not catch the error with an exception and the code will stop and crash. This can allow malicious users access to data that they really should not have. |
| func(0); //invalid function causes error |

| **Compliant Code** |
| --- |
| The code block below catches an error (in this example when x < 0) and lets the code keep running after an error is reached. |
| try{  func(0);  }  catch(exception myException){  std::cout << “Error: do something here”;  } |

**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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Number Overflow** | [STD-008-CPP] | If a user overloads a data variable, they could cause the value to overflow and run around back to the bottom. This can cause users to make data inputs invalid and cause wrong output. |

| **Noncompliant Code** |
| --- |
| This code does not check for overflow errors and will allow a malicious user to enter data that causes an overflow. |
| T result = start;  for (unsigned long int i = 0; i < steps; ++i)  {  result += increment;  }  return result; |

| **Compliant Code** |
| --- |
| This code checks to make sure that a value does not get to big and causes issues where the user can make a number negative by adding super large numbers together. |
| // Variables for checking for an overflow  T result = start;  T resultCompare;  // Loop to do the incrimental adding  for (unsigned long int i = 0; i < steps; ++i)  {  // If loop to check if there was an overflow  resultCompare = result + increment;  // Checks if the new value calculated is lower than the original (check for overflow)  if (result > resultCompare) {  result = start;  return result;  }  // Checks to see if the new value is inf  // After reading online, the isinf() function has trouble with overloads that are not double  // Casting the result to double should fix this issue (link below)  // Source: https://stackoverflow.com/questions/61646166/how-to-resolve-fpclassify-ambiguous-call-to-overloaded-function  else if (isinf(static\_cast<double>(resultCompare))){  result = start;  return result;  }  // Computes the value if no overflow is detected  else {  result += increment;  }  } |

**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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Number Underflow** | [STD-009-CPP] | When a number is subtracted to a value that is lower than possible the value might underflow back to the highest value. If someone wanted to raise their bank value to a high number, they could do that by exploiting this error. |

| **Noncompliant Code** |
| --- |
| This code does not check that the code does not underflow. In doing this a malicious user can change values to benefit themselves. |
| T result = start;  for (unsigned long int i = 0; i < steps; ++i)  {  result -= decrement;  }  return result; |

| **Compliant Code** |
| --- |
| This code block does check for underflows. It might not be an exhaustive list of checks, but it would limit the possibility for underflows. |
| // Variables for checking for an overflow  T result = start;  T resultCompare;  for (unsigned long int i = 0; i < steps; ++i){  // If loop to check if there was an overflow  resultCompare = result - decrement;  // Checks if the calculated value is greater than the prior result  if (resultCompare > result) {  result = start;  return result;  }  else if (resultCompare < 0) {  result = start;  return result;  }  // If no overflow is detected, the correct value is retuned  else {  result -= decrement;  }  }  return result; |

**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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Clearing the Buffer** | [STD-010-CPP] | If there is invalid input, it is important to clear the buffer to make sure that data doesn’t overflow. I have had the system keep running in an infinite loop because I did not clear the buffer after invalid input. |

| **Noncompliant Code** |
| --- |
| This code block does not check to make sure that the buffer is cleared. If it is not cleared, input can lead to an infinite loop or a loop that runs too far and can make data appear funky. |
| //The block below checks to see if the user wants to try more investment info  std::cout << "Do you want to try different investment information? (Y/N)" << endl;  std::cin >> tryAgainChar;  //This block changes the try again variable and the program is exited  if (tryAgainChar == 'n' || tryAgainChar == 'N') {  tryAgain = false;  }  //This block does not change the try again variable and the user can input more values  else if (tryAgainChar == 'y' || tryAgainChar == 'Y') {  } |

| **Compliant Code** |
| --- |
| This code block makes sure that the buffer is cleared after invalid input so data can not be accessed without proper protocols. |
| //The block below checks to see if the user wants to try more investment info  std::cout << "Do you want to try different investment information? (Y/N)" << endl;  std::cin >> tryAgainChar;  //This block changes the try again variable and the program is exited  if (tryAgainChar == 'n' || tryAgainChar == 'N') {  tryAgain = false;  }  //This block does not change the try again variable and the user can input more values  else if (tryAgainChar == 'y' || tryAgainChar == 'Y') {  }  //This block catches if there is an error  else {  cin.clear();  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  throw runtime\_error("Invalid Selection: You need to enter Y for Yes or N for No");  } |

**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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### 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 | High | Unlikely | Medium | High | 2 |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### 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 | [Insert text.] |
| Encryption at flight | [Insert text.] |
| Encryption in use | [Insert text.] |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | [Insert text.] |
| Authorization | [Insert text.] |
| Accounting | [Insert text.] |

**\***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 | 9/18/2022 | Updated coding standards and principles | James Porter | [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 |