**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 | Any input to code from an untrusted source needs to be validated against potential attacks. This means identifying vulnerabilities in the data type used to store the input, in the method of retrieving the input and the in the processing of the input. This is the first essential step in preventing common C security vulnerabilities like overflow and injection. |
| 1. Heed Compiler Warnings | Compiler warning exist to notify developers of potential issues with the implementation of their code. While not necessarily errors, compiler warning can be a strong indication of a vulnerability or potential bug in your code, therefor these warnings should be understood and considered. |
| 1. Architect and Design for Security Policies | Developing with security policy in mind saves a lot of time in the QA and testing step of the SDLC. Dev teams should develop a comprehensive security policy that identifies and addresses potential security concerns for their specific project, and engineers should develop code that attempts to cover those security concerns within the core functionality of their code. |
| 1. Keep It Simple | Over engineering is an excellent way to introduce unidentified security vulnerabilities to your code. The more complex your code is the higher the likelihood that exploits exist within the APIs, and the more time it will take for the dev team to identify those vulnerabilities. |
| 1. Default Deny | Default deny is the practice of denying any unknown or previously undefined request. Having any access request denied by default can help mitigate a variety of potential security vulnerabilities. |
| 1. Adhere to the Principle of Least Privilege | The principle of least privilege is the practice of ensuring that no user or section of a system has more access than is required for its functionality. This helps insulate each section of the program from potential security breaches. |
| 1. Sanitize Data Sent to Other Systems | Any data sent to other systems should be checked for privileged information or potential security issues. This is similar to the practice of input validation, except its output validation. This helps prevent accidental or erroneous use of a system’s output to transmit privileged data. |
| 1. Practice Defense in Depth | Defense in depth is the practice of developing a multi layered security protocol. This means that each major security risk should be protected by at least one redundant layer of security. The purpose of this practice is to develop a security plan that can catch the maximum number of security vulnerabilities and styme attempts to breach one layer by having additional layers to prevent further access. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance (QA) is the system by which security standards and their implementation are evaluated. Having a QA team that understands security principles and has the resources to effectively implement those techniques is paramount to ensuring securing coding. |
| 1. Adopt a Secure Coding Standard | Committing each member of the development team to a secure coding standard is an excellent way of ensuring that each layer of the developed system or application was developed with security in mind. Having that secure coding standard and enforcing it on each section of the SDLC is another effective technique for creating secure programs. |

### 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** | **Choose Appropriate Data Types** |
| --- | --- | --- |
| **Data Type** | [STD-001-C++] | Choosing appropriate data types is essential in protecting against runtime errors and exploits. |

| **Noncompliant Code** |
| --- |
| This legacy code uses a character array to store user input from the console. |
| char input[20];  cin >> input; |

| **Compliant Code** |
| --- |
| This corrected code block uses the C++ buffer safe String var instead. |
| string input;  cin >> input; |

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

| **Principles(s):** Validate Input Data: Input data should be stored in appropriate data types. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Visual Studio | Newest | Data type compile basics | Checks automatically while writing and during compile |
| CPP Check | Newest | Can check for potential variable overflows | Implemented by user |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Validate Data Values** |
| --- | --- | --- |
| **Data Value** | [STD-002-C++] | Data values should be checked before use to ensure compatibility with following code block. This includes checking for integer overflow or asserting that the correct data type is passed. |

| **Noncompliant Code** |
| --- |
| This legacy code uses a parameter integer without checking for integer overflow |
| int timesByFiveThousand(int toTimes) return toTimes \* 5000; |

| **Compliant Code** |
| --- |
| This updated code ensures that integer overflow will not occur. |
| int timesByFiveThousand(int toTimes)  {  if (toTimes > (INT\_MAX / 5000))  {  return 0;  }  return toTimes \* 5000;  } |

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

| **Principles(s):** Architect and design for security principles: Potential errors should be considered while writing function. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Visual Studio | Newest | Data type compile basics | Checks automatically while writing and during compile |
| CPP Check | Newest | Can check for potential integer overflows | Implemented by user |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Use Correct String Declarations** |
| --- | --- | --- |
| **String Correctness** | [STD-003-C++] | Use the const declaration to ensure that strings that shouldn’t be changed are not, mitigating injection into these strings. |

| **Noncompliant Code** |
| --- |
| This code declares and uses a string “introduction” but does not declare it as const, meaning it could be changed by other sections of the program. |
| string intro = “Welcome to global conglomerate inc. llc. Ent. Employee portal” |

| **Compliant Code** |
| --- |
| Simple correction of the declaration of the intro variable prevents editing. |
| const string intro = “Welcome to global conglomerate inc. llc. Ent. Employee portal” |

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

| **Principles(s):** Default Deny: If access can be restricted it should be. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Low | Low | Medium | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Visual Studio | Newest | Will recommend static variables when they aren’t changed | Warning during development |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Always Validate SQL Queries** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-C++] | SQL queries generated by users should be sanitized and checked for potential SQL injection attacks. |

| **Noncompliant Code** |
| --- |
| This code allows users to define an entire SQL string from the console to pass to the SQL engine for querying, allowing for easy SQL injection. |
| cout << “Please Input SQL query:”;  string SQLquery;  cin >> SQLquery;  QuerySQLEngine(SQLquery); |

| **Compliant Code** |
| --- |
| Compliant code asks for portions of the SQL query and builds the query separate from user input. |
| string age, species;  cout <<”Please input desired species:”;  cin >> species;  cout <<endln << “Please input desired maximum age”;  cin >> age;  // validate here that age != “age” and species != “species”  SQLquery = “SELECT \* FROM Animals WHERE species=’” + species + “’ AND age <=” + age + “’;”; |

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

| **Principles(s):** Adhere to the Principle of Least Privilege |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | High | High | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Sqlmap | Newest | Tests code for SQL injection vulnerability | Checks on the database side, errors in code will need to be manually solved |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Ensure Memory is Used Effectively** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-C++] | Unused memory should be cleared and values that can be pointers instead of copies should be. |

| **Noncompliant Code** |
| --- |
| This legacy code creates a copy of the parameter variable when it could be a pointer |
| Add(int & value1, int & value2) return value1 + value2; |

| **Compliant Code** |
| --- |
| This code converts the old function to a pointer, so additional memory allocation is limited. |
| Add(int \*value1, int \*value2) return value1 + value2; |

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

| **Principles(s):** Adhere to the Principle of Least Privilege: Manage scope and stack |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Visual Studio | Newest | Recommends between reference and value, cleans memory occasionally | Checks during development |
| CPP Check | Newest | Can identify potential memory issues and style issues with pointers | Implemented by user as static testing tool |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Use Assertions for Developing Only** |
| --- | --- | --- |
| **Assertions** | [STD-006-C++] | Assertions are disabled at runtime, so they should only be used for in-development checks, not for logic in the final build. |

| **Noncompliant Code** |
| --- |
| This code uses an assertion to control logic flow in the program |
| calculateGrade(int score){  assert(score<=100)  return score/100;  } |

| **Compliant Code** |
| --- |
| The updated code uses an if block to ensure that the score provided is within acceptable parameters |
| calculateGrade(int score){  if(score > 100){  cout << “Score too high” << endl;  return 0;  }  return score/100;  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPP Check | Newest | Checks for potential Assert errors | Implemented by user |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle All Exceptions** |
| --- | --- | --- |
| **Exceptions** | [STD-007-C++] | Unhandled exceptions call std::terminate(), which does not provide much information on the exception, therefore all potential exceptions should be handled by an exception block to provide debugging information. |

| **Noncompliant Code** |
| --- |
| This code calls a function that throws an error, but has no block to handle the error. |
| ThrowError(); |

| **Compliant Code** |
| --- |
| This compliant code implements exception handling to handle potential errors in the program. |
| try{  ThrowError()  } catch (…){  // error handled here  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | High | Low | Medium | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Visual Studio | Newest | Checks for unhandled exceptions at runtime | Implemented in development |
| CPPCheck | Newest | Can check for potential exceptions | Implemented by user as static testing tool |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Create Objects When Appropriate** |
| --- | --- | --- |
| OOP | [STD-008-C++] | Object oriented programming cuts down on redundant code, making QA and security validation easier. |

| **Noncompliant Code** |
| --- |
| This code uses the same set of variables for shelter animals multiple times, when it could be defined as a discrete object. |
| string name1, species1, description1;  int age;  GetInfo();  string name2, species2, description2;  int age2;  GetInfo();  string name3, species3, description3;  int age3;  GetInfo(); |

| **Compliant Code** |
| --- |
| This code uses a shelter animal object to trivialize variables |
| shelterAnimal animal1, animal2, animal3;  animal1.GetInfo();  animal2.GetInfo();  animal3.GetInfo(); |

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

| **Principles(s):** Keep It Simple |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Medium | Low | Low | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Visual Studio | Newest | Can warn about redundant code, has tools to create objects quickly | Implemented in development |
| CPPCheck | Newest | Can check for style errors | Implemented by user as static testing tool |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Use Uniform Naming Conventions** |
| --- | --- | --- |
| Naming Convention | [STD-009-C++] | Using uniform naming conventions helps make code much easier to understand. |

| **Noncompliant Code** |
| --- |
| This code uses a variety of naming conventions, which can be difficult to read. |
| int to\_add, toSubtract;  Add\_Numbers(to\_add);  SubtractNumbers(toSubtract); |

| **Compliant Code** |
| --- |
| This updated code uses uniform naming conventions that are much easier to follow |
| int toAdd, toSubtract;  AddNumbers(toAdd);  SubtractNumbers(toSubtract); |

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

| **Principles(s):** Keep it Simple |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Low | Low | Low | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | Newest | Checks for style errors | Implemented as static testing tool by user |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Always Check Element Access of Containers** |
| --- | --- | --- |
| Container Access | [STD-010-C++] | Ensuring that access is not out of range of the container ensures that appropriate memory is accessed |

| **Noncompliant Code** |
| --- |
| This code accesses the element of the parameter integer without ensuring that the index is within range of the container |
| GetElement(int index) return parentArray.at(index); |

| **Compliant Code** |
| --- |
| Compliant code first ensures that the index is within the array’s boundaries. |
| GetElement(int index)  {  if(index <= parentArray.size())  {  return parentArray.at(index);  }  Return 0;  } |

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

| **Principles(s):** Architect and Design for Security Policies |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | Newest | Checks for style errors | Implemented as static testing tool by user |

### 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.

In the current DevSecOps system, automated testing should be implemented during the Verify and test section of the pre-production process, and during the transition and health check section of the production process. These are the ideal times to implement automated coding checks, because they’re the times in the processes where other testing is being accomplished. Having automation will help streamline this process and make it slightly more effective.

### 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-C++ | Medium | High | Low | High | 2 |
| STD-002-C++ | High | Medium | Low | High | 2 |
| STD-003-C++ | Medium | Low | Low | Medium | 4 |
| STD-004-C++ | High | High | High | High | 1 |
| STD-005-C++ | Medium | Medium | Medium | Medium | 3 |
| STD-006-C++ | High | Low | Low | High | 3 |
| STD-007-C++ | Low | High | Low | Medium | 4 |
| STD-008-C++ | Low | Medium | Low | Low | 5 |
| STD-009-C++ | Low | Low | Low | Low | 5 |
| STD-010-C++ | Medium | Low | Medium | Medium | 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 | Sensitive data stored in any data storage should be encrypted so as to prevent unauthorized access from obtaining that sensitive data. |
| Encryption at flight | Sensitive data that is transmitted should be encrypted so that any interception of that data has a lower risk of resulting in that data leaking. |
| Encryption in use | Sensitive data used by our program should be encrypted so any runtime exploitation of our code cannot be used to access that sensitive data inappropriately. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | The practice of implementing security checks to ensure that the user is who they say they are. Whether this be a physical user or information requested through the API, validating the source of the request is essential to security. Authentication should always follow the principle of default deny. |
| Authorization | Ensuring clearly defined permissions for each type of user in your program ensures the security of non-related sections of your program. Authorization should always follow the principle of least privilege. |
| Accounting | Tracking user access helps ensure that we keep our authentication and authorization accurate and up to date. Without accurate information on user access we can’t ensure that the other two principles remain effective. |

**\***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 | 8/6/2023 | Completed Tempalte | Jared Smith |  |
| [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 |