**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 | By validating the inputs of all untrusted data, we are able to prevent the all the invulnerabilities of software. When we validate inputs we need to be aware of all types of data sources, especially external data. |
| 1. Heed Compiler Warnings | It is important not to ignore the warnings when developing code. We need to always eliminate all warnings by modifying the code. When warnings are left and not dealt with, security flaws persist, they are there for a reason. |
| 1. Architect and Design for Security Policies | When you design your software architecture, it is very important that you are following and address the security policies. |
| 1. Keep It Simple | When designing your code, it is important that you keep your code as simple as you can. When making things complex that don’t need to be, we leave room for errors which could result in invulnerabilities. |
| 1. Default Deny | Standardize the way things are done. If they are not done that specific way, then deny it always. By keeping a standard way things are done, we know something is going on if it’s not done that way. |
| 1. Adhere to the Principle of Least Privilege | Allow process to only have access for the exact amount of time they need to finish there task. When running processes that require higher access do the same thing. Reducing the amount of access reduces the chance of invulnerability. |
| 1. Sanitize Data Sent to Other Systems | Injection attacks will be taken care of by using sanitization. By sanitizing data sent it will allow the data to be put through a filter before being sent to the other system. |
| 1. Practice Defense in Depth | Always use multiple layers of security so when one layer fails there is another layer like an onion. |
| 1. Use Effective Quality Assurance Techniques | Are sure that the security is working correctly by involving a process to follow where. There is testing, reviewing, and outside point of views. |
| 1. Adopt a Secure Coding Standard | Always have a secure coding standard that you can follow that is catered to the language you are using. |

## 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] | Don’t cast to an enumeration value that is out of range. |

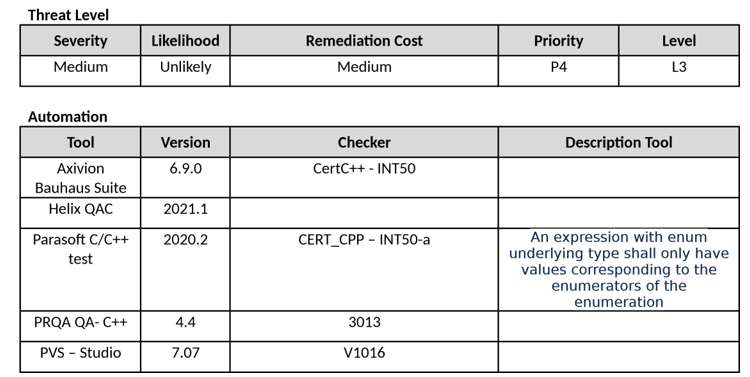
| **Noncompliant Code** |
| --- |
| Checks if the value is with in the enum value rage. |
| enum EnumType {  One,  Two,  Three  };  void func(int intVariable) {  EnumType enumVar = static\_cast<EnumType>(intVariable);  if (enumVar < One || enumVar > Three) {    } } |

| **Compliant Code** |
| --- |
| Checks value of enum before conversion to make sure the output is not an error. |
| enum EnumType {    One,  Two,  Three  };  void func(int intVariable) {  if (intVariable < One || intVariable > Three) {    }  EnumType enumVar = static\_cast<EnumType>(intVariable);  } |

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

|  |
| --- |
| **Principles(s):** Unspecified values can result in a overflow of the buffer. This can result in code that is not supposed to be ran, ran by an attacker. |

**Threat Level**

****

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Use the right type of reference or pointers. |

| **Noncompliant Code** |
| --- |
| In this example an error with pos occurs after the first iteration making the loop have an error. |
| #include <deque>  void func(const double \*objectt, std::size\_t count) {    std::deque<double> num;    auto pos = num.begin();    for (std::size\_t i = 0; i < count; ++i, ++pos) {  num.insert(pos, objectt[i] + 41.0);  }  } |

| **Compliant Code** |
| --- |
| In this example pos does not have an error since it has a valid irritator on each insertion. |
| #include <deque>  void func(const double \*objectt, std::size\_t count) {    std::deque<double> num;    auto pos = num.begin();    for (std::size\_t i = 0; i < count; ++i, ++pos) {  num.insert(pos, objectt[i] + 41.0);  }  } |

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

|  |
| --- |
| **Principles(s):** When using the wrong pointers and references, we can have a result of unidentified behavior. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree |  |  | ---- |
|  |  | ---- | ---- |
| Parasoft C/C++ test |  |  | Do not modify container while iterating over it |
|  |  |  | ---- |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | dont make an std::string from a pointer that is null |

| **Noncompliant Code** |
| --- |
| In the following code we make a std::string object and store the output of the std:detenv() value in it. The output is a null pointer so an error occurs. |
| #include <cstdlib>  #include <string>    Void func() {  const char \*tempPointer = std::getenv("TMP");  if (!temp.empty()) {    } } |

| **Compliant Code** |
| --- |
| In this solution we check to see if std::getenv() is null and then we create the std::string if it is not null. |
| #include <cstdlib>  #include <string>    Void func() {  const char \*tempPointer = std::getenv("TMP"); std::string temp(tempPointer? tempPointer: "");  if (!temp.empty()) {    } } |

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

|  |
| --- |
|  |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
|  |  | Assert\_failure | ---- |
|  |  | ---- | ---- |
| Parasoft/C++ test |  |  |  |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | The smart pointer value should not be stored in the already-owned pointer. |

| **Noncompliant Code** |
| --- |
| In this example the two smart pointers are created form the same pointer value. When the ptrTwo variable is destroyed we that is also deleted the pointer value it is managing. And then we also delete ptrOne we notice that the same pointer value is destroyed… |
| #include <memory>  void func()  {  int \*i = new int; std::shared\_ptr<int> ptrOne(i); std::shared\_ptr<int> ptrTwo(i); } |

| **Compliant Code** |
| --- |
| In this example both of the ptrOne and ptrTwo pointers are related, this means when we destroy one of them, the managed pointers count goes down by one for each related pointer and finally hits zero and is also destroyed when all other related pointers are destroyed. |
| #include <memory>  void func()  {  std::shared\_ptr<int> ptrOne = std::make\_shared<int>(); std::shared\_ptr<int> ptrTwo(ptrOne);  } |

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

|  |
| --- |
|  |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
|  |  | Dangling\_pointer\_use | ---- |
|  | 2021.1 | ---- | ---- |
| Parasoft C/C++ | 2020.2 |  | Do not store an already-owned pointer value in an unrelated smart pointer |
|  | 7.01 |  | ---- |

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Make sure all allocated resourced are deallocated in the correct way. |

**Noncompliant Code**

In this example, the local variable is space is being passed to new. Then, that pointer is passed to the ::operator delete(), this has an output of an error.

#include <iostream>

struct test {

test() { std::cout << "test::test()" << std::endl; }

~S() { std::cout << "test::~test()" << std::endl; }

;

}

void func() {

alignas(struct test) char space[sizeof(struct test)];

test \*test1 = new (&space) test;

// ...

delete test1;

}

**Compliant Code**

This example removes the ::operator delete() call and calls the destructor instead.

#include <iostream>

struct

test {

test() { std::cout <<

"test::test()"

<<

std::endl;

}

~test() { std::cout <<

"test::~test()"

<<

std::endl;

}

}

;

void

func() {

alignas(struct

test)

**char**

space[sizeof(struct

test)];

test \*tst1 =

new

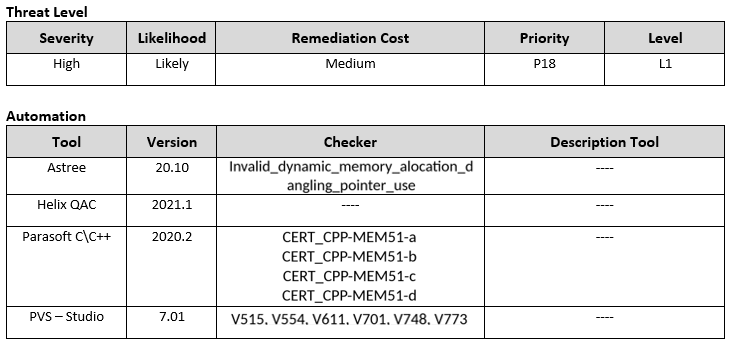
(

&space) test;

|  |
| --- |
| In this example, the local variable is space is being passed to new. Then, that pointer is passed to the ::operator delete(), this has an output of an error. |

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

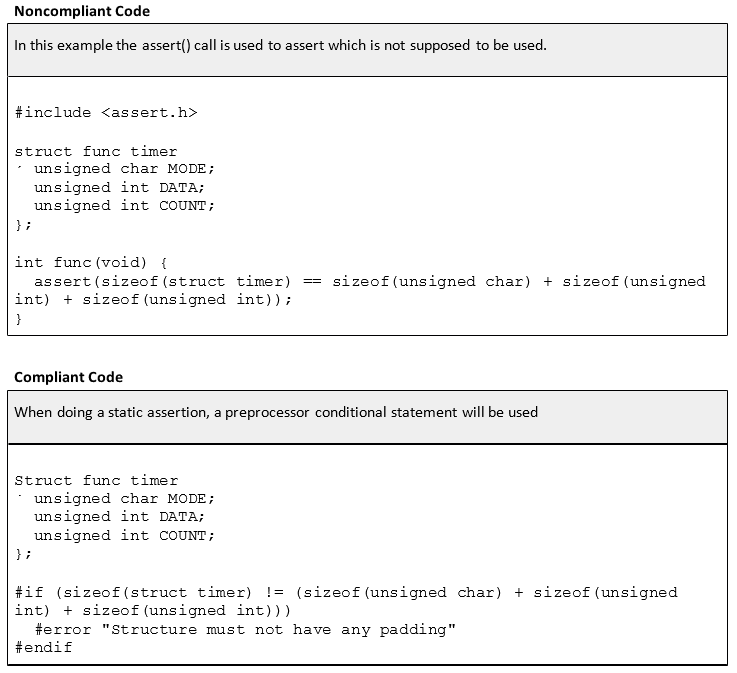
|  |
| --- |
| **Principles(s):** if a deallocation functions receives a pointer that was not obtained by an alloc function will result in an undefined behavior. |

****

### 

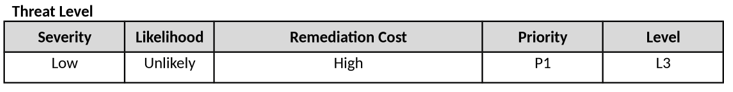
### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | constant expression, when tested needs to be tested with static insertion. |

****

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

|  |
| --- |
| **Principles(s):** Static assertion allows us to find errors and defects in our code so that we can eliminate them fast. |

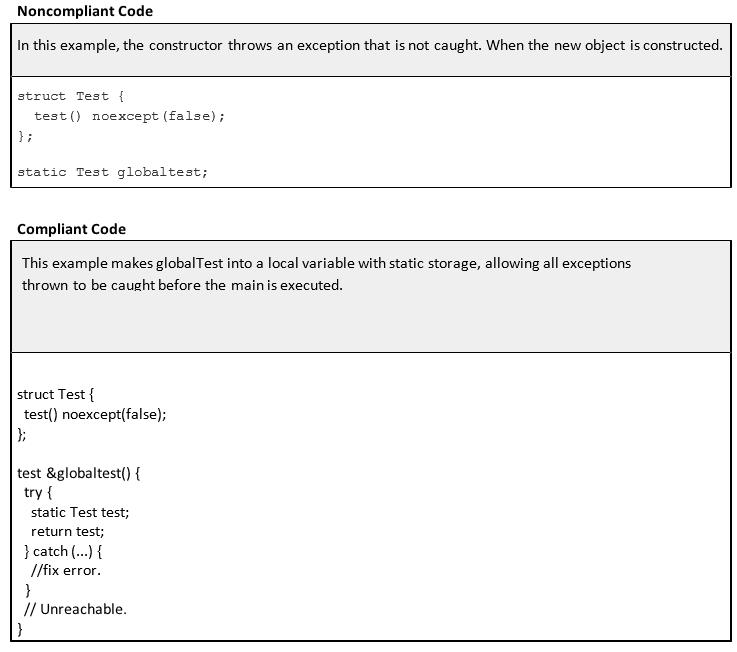
****

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
|  |  | ---- | Users can create a custom check that uses the assert() function |
|  | 1.2 | ---- | ---- |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-LLL] | Handle all exceptions before executing. |

****

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

|  |
| --- |
| **Principles(s):** When an exception is thrown and cannot be cough we have a termination of the program at a random time which lead to attacks. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Low | p9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 |  | Partially Checked |
|  | 2021.1 | ---- | ---- |
| Parasoft C/C++ | 2020.2 CERT\_CP-ERR58-a | ---- |  |

## 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.

For this company, I believe that verify and test should be the top priority of the pre-production phase. We need to make sure everything is working great with out there being any issues before production starts. If we are to start production, we cannot have any type of security flaw with this system that is exposed. When It comes to Production, we need to make sure Transition and health check is happening often. To describe the diagram. The preproduction phase basically is checking for security faults and testing to see what they all are. The Production phase is more of a way to make sure that these security flaws do not happen, and they prevent ones that were found from happening. They both work together to make sure that security is kept at a high.

## Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Medium | Unlikely | Medium | Medium | 3 |
| STD-002-CPP | High | Likely | High | Medium | 2 |
| STD-003-CPP | High | Likely | Medium | High | 1 |
| STD-004-CPP | High | Likely | Medium | High | 1 |
| STD-005-CPP | High | Likely | Medium | High | 1 |
| STD-006-CPP | Low | Unlikely | High | Low | 3 |
| STD-007-CPP | Low | Likely | Low | Medium | 1 |

## 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 | This is like storing data in a vault. It never moves. |
| Encryption at flight | This is like storing data that is moving like an email. |
| Encryption in use | This is data that is stored in no state that is always active. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Passwords and usernames to access information |
| Authorization | The information a user is authorized to have |
| Accounting | Accounts people use to log in with |

**\***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 | 8/01/2021 | Initial Template | Mason Voorhees |  |
| 2.0 | 8/14/2021 | Adding code standards | Mason Voorhees | [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 |