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



# Green Pace Secure Development Policy

## Contents

[Overview 3](#_Toc121040403)

[Purpose 3](#_Toc121040404)

[Scope 3](#_Toc121040405)

[Module Three Milestone 3](#_Toc121040406)

[Ten Core Security Principles 3](#_Toc121040407)

[C/C++ Ten Coding Standards 4](#_Toc121040408)

[Coding Standard 1 5](#_Toc121040409)

[Coding Standard 2 7](#_Toc121040410)

[Coding Standard 3 10](#_Toc121040411)

[Coding Standard 4 12](#_Toc121040412)

[Coding Standard 5 15](#_Toc121040413)

[Coding Standard 6 17](#_Toc121040414)

[Coding Standard 7 19](#_Toc121040415)

[Coding Standard 8 21](#_Toc121040416)

[Coding Standard 9 23](#_Toc121040417)

[Coding Standard 10 25](#_Toc121040418)

[Defense-in-Depth Illustration 28](#_Toc121040419)

[Project One 28](#_Toc121040420)

[Revise the C/C++ Standards 28](#_Toc121040421)

[Risk Assessment 28](#_Toc121040422)

[Automated Detection 28](#_Toc121040423)

[Automation 28](#_Toc121040424)

[Summary of Risk Assessments 29](#_Toc121040425)

[Create Policies for Encryption and Triple A 30](#_Toc121040426)

[Map the Principles 31](#_Toc121040427)

[Audit Controls and Management 32](#_Toc121040428)

[Enforcement 32](#_Toc121040429)

[Exceptions Process 32](#_Toc121040430)

[Distribution 33](#_Toc121040431)

[Policy Change Control 33](#_Toc121040432)

[Policy Version History 33](#_Toc121040433)

[Appendix A Lookups 33](#_Toc121040434)

[Approved C/C++ Language Acronyms 33](#_Toc121040435)

## 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 should be assumed that all input is Malicious. One cannot know what the user’s intent might be and therefore you cannot know if malicious commands are present (Meier et al., 2008). Validating input is the processes setting specific parameters for input data from untrusted sources to ensure that it is properly formatted. Parameters should be set to make sure the input is the correct data type (numeric, alphanumeric, string etc.) and length for the allotted space in the register. Validating input is essential for eliminating most software related vulnerabilities (Seacord, 2018). |
| 1. Heed Compiler Warnings | Do not ignore the compiler warnings they may signal potentially serious problems in the code. Treat warnings as if they were errors. Use the highest warning level available in the compiler being utilized and modify the code as needed to eliminate the warnings. Additionally, utilize both static and dynamic analysis to uncover and eliminate additional security flaws not caught by the compiler (Seacord, 2018). If the warnings are a false alarm, then make comments or suppress the warning if the compiler will allow it (Seacord, 2021). |
| 1. Architect and Design for Security Policies | When creating software keep security in mind while building the architecture. This includes setting role base create/read/write/delete (CRUD) privileges or dividing the components up into subsystems that restricts communication in compliance with an individual subsystem’s privileges (Seacord, 2018). Define user roles and update policies regularly. Design decisions should be considered for all potential threats and risks in the use case environment (Gupta). |
| 1. Keep It Simple | When designing and creating software, be minimalistic and remove redundant code or code that is not necessary. Minimalize the “Attack Surface Area”. Excessive code may expose functionality (Boersma). The simpler and shorter the code is, the less likely errors will exist that create security vulnerabilities. The more complex that the system becomes, the more challenging it will be to ensure the appropriate security mechanisms are implemented correctly (Seacord, 2018). |
| 1. Default Deny | A system should have varying user privilege levels for accessing resources. Give limited or no permissions to new users. When deciding on access privileges, default with the denial of access and add levels of permission if specified criteria is met. Criteria is set with a “protection scheme” based on “permission rather than exclusion (Saltzer and Schroeder, 1974 & 1975). |
| 1. Adhere to the Principle of Least Privilege | Every process should be designed so that it requires as minimal access to the system as possible to perform the task. Access should be in line with the user’s role or needs. This helps to ensure that should an attacker compromiser a user’s account, the data the gain access will be limited (Boersma). Access should likewise be limited to as minimal of a timeframe as possible to complete the task. This will limit the opportunity for arbitrary code to be executed during the access period of confidential data (Saltzer and Schroeder, 1974 & 1975). |
| 1. Sanitize Data Sent to Other Systems | All data that is be passed along to another system should be sanitized. Sanitizing data is the process of removing specific characters that are commonly used for injection attacks. The sanitation is performed by the calling process as it may have different criteria then the original input criteria (Seacord, 2018). |
| 1. Practice Defense in Depth | Practicing Defense in Depth involves considering how to prevent system access. One should assume your security will fail (Boersma). Use multiple layers of security rather than relying on just one. Multiple layers will prevent security flaws that may pass through one layer from passing through all the layers. This will limit the spread of damage or depth of breach in the system to a narrow scope. |
| 1. Use Effective Quality Assurance Techniques | Using variety of testing techniques before deployment to identify vulnerabilities. Utilizing external reviewers or auditors who are not connected to the project will bring a fresh perspective and reduce personal bias (Seacord, 2018). |
| 1. Adopt a Secure Coding Standard | Every coding language is different and therefore security practices must be adapted to fit the needs of the language. Use the appropriate security coding standards for the development language/languages being utilized (Seacord, 2018). |

### C/C++ Ten Coding Standards

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | **Understand the size of standard data types**: Data models assign sizes to standard data types. Code often has embedded assumptions about data models and may require specific sizes. Know the limits or number of bits for a given type to determine the appropriate integral range needed (Seacord and Razmyslov, 2021). |

| **Noncompliant Code** |
| --- |
| This code attempts to set the value of an unsigned long int as to unsigned int values multiplied together however, the end result is not guaranteed to fit if the allotted space for an unsigned long int. |
| unsigned int x, y;  unsigned int long z;  z = (unsigned long)x \* y; //May not fit |

| **Compliant Code** |
| --- |
| Using the largest unsigned integer type available if it can be guaranteed to be large enough to hold the results makes the code block compliant. |
| #if UINT\_MAX > UINTMAX\_MAX/UINT\_MAX  #error No safe type available.  #endif  unsigned int x, y;  unsigned int long z;  z = (uintmax\_t)x \* y; //guaranteed to fit |

| **Principles(s):**  1. Validate data ranges, length, and type to prevent buffer overflow/underflow. Input may be received internally or externally but it should never be assumed that it is safe. Set parameters to make sure the entered data fits the receiving data type.  2. If the compiler warns of a type mismatch determine if a type change is needed to ensure memory management safety. Make sure the receiving end has space to hold the incoming data.  4. Use standard data types and minimalize transforming them to do operations with other data types.  5. If the input does not meet specified criteria deny it.  8. Be defensive, mismanaged memory allocation including both overflow and underflow will make the system vulnerable to attack.  10. Know the assumptions and space allotted to the language’s standard types. Chose data types that use the minimal space needed for the maximum possible size of the data the variable must be able to hold. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | 3 | Unlikely | High | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-Int00 | A static analysis, architecture analysis and code smell detector tool that helps identify technical debt to keep software maintainable and expandable (Axivion, 2022).  [Axivion Suite - Code Quality for C, C++ and C# | Axivion](https://www.axivion.com/en/products/axivion-suite/) |
| PC-lint Plus | 1.4 | 559, 705, 706, 2430 | Reports data type inconsistencies in the format strings.  [PC-lint Plus | Static Code Analysis for C and C++ (pclintplus.com)](https://pclintplus.com/pc-lint-plus/) |
| Polyspace Bug Finder | R2022b | CERT C: Rec. INT00-C | Checks for the use of basic numerical types instead of typedef-s, Integer overflow or integer constant overflow, format string specifiers, and arguments mismatch.  [Polyspace - MATLAB & Simulink (mathworks.com)](https://www.mathworks.com/products/polyspace.html) |
| PVS-Studio | 7.21 | V629, V5004 | Static code analyzer for code quality, security, and safety. Detects typos, dead code, and potential vulnerabilities. Matches warnings to SEI CERT coding standards (Wikipedia, 2022).  [PVS‑Studio is a solution to enhance code quality, security (SAST), and safety (pvs-studio.com)](https://pvs-studio.com/en/) |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | **Make sure that performing operations with signed integers does not result in overflow:** Performing operations on signed integers can lead to integer overflow which is classified as an undefined behavior. This means an operation may return unexpected results or cause the system to enter into an infinite loop(Seacord and Britton, 2022). |

| **Noncompliant Code** |
| --- |
| If a signed int is subtracted from a second int of a lesser value then the code can result in integer overflow. |
| std::cout << "Underflow Test of Type = " << typeid(T).name() << std::endl;  // END DO NOT CHANGE  std::cout << "\tSubtracting Numbers Without Overflow (" << +start << ", " << +decrement << ", " << steps << ") = ";  auto result = subtract\_numbers<T>(start, decrement, steps);  std::cout << +result << std::endl;  std::cout << "\tSubtracting Numbers With Overflow (" << +start << ", " << +decrement << ", " << (steps + 1) << ") = ";  result = subtract\_numbers<T>(start, decrement, steps + 1);  std::cout << +result << std::endl;  } |

| **Compliant Code** |
| --- |
| Checking to make sure the subtraction will result in positive value will make the code compliant. |
| void func(signed int si\_a, signed in si\_b) {  signed int diff;    if((si\_b > 0 && si\_a < INT\_MIN + si\_b) ||  (si\_b < 0 && si\_a > INT\_MAX +si\_b)) {  //Handle error  }  else{  Diff = si\_a – si\_b;  }  } |

| **Principles(s):**  1. Validate data ranges, length and type. Inputting data of the wrong type into an operation can result on out of bounds ranges.  8. Be defensive and do not allow operations to accept incorrect data types or values that are too large for the allotted memory.  9. Test the operations in ways that will push the bounds to determine if accidental overflow/underflow might occur.  10. Avoid calculation errors by understanding how the used language interacts with numeric calculations. Pay attention to byte size discrepancies, precision, signed/unsigned types, truncation, conversion, and casting between types. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | 5 | Likely | High | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 | Integer-overflow | Static code analyzer primarily for embedded systems that reports many operational errors including but not limited to division by zero and arithmetic overflow.  [Astrée Static Analyzer for C and C++ (absint.com)](https://www.absint.com/astree/index.htm) |
| CodeSonar | 7.1p0 | **ALLOC.SIZE.ADDOFLOW ALLOC.SIZE.IOFLOW ALLOC.SIZE.MULOFLOW ALLOC.SIZE.SUBUFLOW MISC.MEM.SIZE.ADDOFLOW MISC.MEM.SIZE.BAD MISC.MEM.SIZE.MULOFLOW MISC.MEM.SIZE.SUBUFLOW** | Static code analyzer that finds and fixes bugs and security vulnerabilities in source and binary code.  [CodeSonar SAST for C/C++: Static Code Analysis Tool | GrammaTech](https://www.grammatech.com/codesonar-cc) |
| Parasoft C/C++ test | 2022.1 | CERT\_C-INT32-a  CER\_C\_INT32-b  CERT\_C-INT32-c | Helps avoid integer overflows/underflow in constant expressions by checking arithmetic operators in output expressions.  [Automated Testing to Deliver Superior Quality Software | Parasoft](https://www.parasoft.com/) |
| Polyspace Bug Finder | R2022b | CERT C: Rule INT32-C | Checks for integer overflow, tainted division operands and modulo operands.  [Polyspace - MATLAB & Simulink (mathworks.com)](https://www.mathworks.com/products/polyspace.html) |
| TrustInSoft Analyzer | 1.38 | Signed\_overflow | Focuses on the code semantics via exhaustive testing.  [Home - TrustInSoft, exhaustive static analysis tools for software security and safety (trust-in-soft.com)](https://trust-in-soft.com/) |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | **Guarantee that storage for strings must have sufficient space:** Data that is copied to a buffer that is not large enough to contain it will result in a buffer overflow. Limit copies though truncation if necessarily but preferably ensure that the destination is large enough to hold both the copied data and the null terminator (Svoboda, 2022). |

| **Noncompliant Code** |
| --- |
| Off-by-one error: A loop copies the data from the source to a destination. The loop fails to accommodate space for the null terminator causing data to be written one byte past the allotted space for the destination. |
| Void copy(size\_t n, char src[n], char dest[n]) {  Size\_t i;  For (i = 0; src[i] && (i < n); ++i) {  Dest[i] = src[i];  }  Dest[i] = ‘\0’;  } |

| **Compliant Code** |
| --- |
| Fixing the loop to account for the null terminator makes the code compliant. |
| Void copy (size\_t n, char src[n], char dest[n] {  Size\_t i;    For (i = 0; src[i] && (i < n – 1); ++i) {  Dest[i] = src[i];  }  Dest[i] = ‘\0’;  } |

| **Principles(s):**  1. Validate data ranges, length, and type. Set up specific parameters for all input data and ensure it is properly formatted.  5. Should the string value not meet the required parameters deny acceptance of the string into the holding variable.  8. Memory mismanagement will leave the system vulnerable. Allocate appropriate memory to minimalize an attack.  9. Use in depth testing to make sure overflow does not occur.  10. Check that the buffer is as large as specified, if using a function that accepts a number of bytes to copy be aware of the destination buffer size, truncate input strings to a reasonable length before passing to any copying or concatenation function. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | 4 | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Reports buffer overflows resulting from copying data to a buffer that doesn’t have sufficient space to hold the data.  [Astrée Static Analyzer for C and C++ (absint.com)](https://www.absint.com/astree/index.htm) |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR31 | Detects calls to unsafe string functions that may cause buffer overflow and potential buffer overruns.  [Axivion Suite - Code Quality for C, C++ and C# | Axivion](https://www.axivion.com/en/products/axivion-suite/) |
| CodeSonar | 7.1p0 | **LANG.MEM.BO LANG.MEM.TO MISC.MEM.NTERM BADFUNC.BO.\*** | Detects buffer overrun and for space for null terminators.  [CodeSonar SAST for C/C++: Static Code Analysis Tool | GrammaTech](https://www.grammatech.com/codesonar-cc) |
| Parasoft C/C++ test | 2022.1 | **CERT\_C-STR31-a CERT\_C-STR31-b CERT\_C-STR31-c CERT\_C-STR31-d CERT\_C-STR31-e** | Helps avoid unsafe string functions that may cause buffer overflow.  [Automated Testing to Deliver Superior Quality Software | Parasoft](https://www.parasoft.com/) |
| Polyspace Bug Finder | R2022b | CERT C: Rule STR31-C | Checks for incorrect string format specifiers and destination buffer overflow due to string manipulation or insufficient receiving buffer size.  [Polyspace - MATLAB & Simulink (mathworks.com)](https://www.mathworks.com/products/polyspace.html) |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | **Sanitize all input data** : Data passed into the subsystem can contain special characters that are used in SQL Injection to trigger commands. Sanitizing the data by replacing or removing those characters will prevent Injection vulnerabilities (Seacord and Britton, 2022). |

| **Noncompliant Code** |
| --- |
| This code block intakes user input and fails to sanitize the data allowing for the user to inject “or” statements with the append function. |
| bool run\_query(sqlite3\* db, const std::string& sql, std::vector  <user\_record>& records) {  //clear any prior results  Records.clear();  char\* error\_message;  if(sqlite3\_exec(db, sql.c\_str(), callback, &records,  &error\_message) != SQLITE\_OK)  {  std::cout << "Data failed to be queried from USERS table. ERROR  = " << error\_message << std::endl;  sqlite3\_free(error\_message);  return false;  }  return true;  }  } |

| **Compliant Code** |
| --- |
| The above example has been modified to make a copy of the input data, transform it, and check the input for operators that are used for injection. |
| bool run\_query(sqlite3\* db, const std::string& sql, std::vector  <user\_record>& records) {  //clear any prior results  Records.clear();  // Create a temporary copy of string  std::string sqlCopy(sql);  // Transform sqlCopy to all lower case  std::transform(sqlCopy.begin(), sqlCopy.end(), sqlCopy.begin(),  ::tolower);  //Search for common injections  // or  std::size\_t pos = sqlCopy.find(" or ");  if (pos != std::string::npos) {  //Error message if found  std::cout << "---Possible SQL Injection Detected---\n Query  terminated: " << sql.c\_str() << std::endl;  return false;  }  } |

| **Principles(s):**  1. check input against allowed characters and check for hazardous/alteration characters. Assume that all input is malicious.  3. Set role-based CRUD privileges and divide the components into subsystems to restrict communication to limit the reach of user threats.  5. If SQL injection occurs, default denying privileges will minimize the damage.  6. Allow each process minimal access to the system to limit a database breach.  7. Sanitize all output of un-trusted data to queries.  10. Restrict users from generating new code or altering existing code. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | 4 | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Stubbing/taint analysis  [Astrée Static Analyzer for C and C++ (absint.com)](https://www.absint.com/astree/index.htm) |
| CodeSonar | 7.1p0 | **IO.INJ.COMMAND IO.INJ.FMT IO.INJ.LDAP IO.INJ.LIB IO.INJ.SQL IO.UT.LIB IO.UT.PROC** | Checks for command injection, format string injection, LDAP injection, Library injection, SQL injection untrusted Library load, and untrusted process creation.  [CodeSonar SAST for C/C++: Static Code Analysis Tool | GrammaTech](https://www.grammatech.com/codesonar-cc) |
| Parasoft C/C++ test | 2022.1 | **CERT\_C-STR02-a CERT\_C-STR02-b CERT\_C-STR02-c CERT\_C-STR31-d CERT\_C-STR31-e** | Protects against command injection, name injection, and SQL injection.  [Automated Testing to Deliver Superior Quality Software | Parasoft](https://www.parasoft.com/) |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | **Do not access dangling pointers:** Accessing a dangling pointer (a pointer to deallocated memory) will result in undefined behavior and exploitable vulnerabilities. Freed memory makes all pointers to said memory invalid (Pincar and Britton, 2022). |

| **Noncompliant Code** |
| --- |
| The block of code attempts to allocate zero bytes of memory which would lead to a null pointer being returned. |
| void function() noexcept(false){  Unsigned char \*ptr = static\_cast<unsigned char \*>(::operator  new(0));  \*ptr = 0;  //…  ::operator delete(ptr);  } |

| **Compliant Code** |
| --- |
| There are multiple ways to resolve the above code block but let us assume the programmer purposefully allocated zero bytes of memory. A Unique pointer that cannot be reused such as a void \* cannot be dereferenced. |
| void function() noexcept(false) {  void \*ptr = :: operator new();  //…  ::operator delete(ptr); |

| **Principles(s):**  4. Do not try to overcomplicate the use of pointers. Make sure you always know what memory address your pointers are allocated.  8. avoid denial of service attacks which can occur when reading memory that has been deallocated. Writing to deallocated memory can allow for the execution or arbitrary code.  9. Run extensive testing to make sure deallocated memory is not accessible. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | 4 | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |

|  |  |  |  |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | **ALLOC.UAF** | Checks for used pointers after they are freed.  [CodeSonar SAST for C/C++: Static Code Analysis Tool | GrammaTech](https://www.grammatech.com/codesonar-cc) |

|  |  |  |  |
| --- | --- | --- | --- |
| Coverity | v7.5.0 | USE\_AFTER\_FREE | Detects instances where memory is deallocated multiple times or read/written to a freed pointer.  [Coverity Scan - Static Analysis](https://scan.coverity.com/) |
| Parasoft C/C++ test | 2022.1 | **CERT\_CPP-MEM50-a** | Checks that resources are not used after being freed.  [Automated Testing to Deliver Superior Quality Software | Parasoft](https://www.parasoft.com/) |
| Polyspace Bug Finder | R2022b | CERT C++: MEM50-CPP | Checks for pointers access out of bounds, deallocation of previously deallocated pointers, and use of previously freed pointers.  [Polyspace - MATLAB & Simulink (mathworks.com)](https://www.mathworks.com/products/polyspace.html) |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | **Use a static assertion to test the value of a constant expression**: Assertions diagnostic tools help find and eliminate code defects that may lead to vulnerabilities. It is important to use static assertions rather than just runtime assertions to ensure that they are checked (Seacord and Britton, 2018). |

| **Noncompliant Code** |
| --- |
| This code block uses runtime assertions that are not executed inside of a function meaning it occurs only inf the code takes the path containing the assertion. |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };  int function(void){  assert(sizeof(struct timer) == sizeof(unsigned char) +  sizeof(unsigned int) + sizeof(unsigned int));  } |

| **Compliant Code** |
| --- |
| Use a static assertion to allow incorrect assumptions to be diagnosed when compiling rather then creating a runtime error. |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };  static assert(sizeof(struct timer) == sizeof(unsigned char) +  sizeof(unsigned int) + sizeof(unsigned int), “Structure must  not have any padding”); |

| **Principles(s):**  2. Using assertions will generate compiler warnings. Address all of the warnings and document if any are false positives that will not be corrected.  8. Assume that if an assertion fails then there is a hole in the security of the system.  9. Use different testing techniques to identify vulnerabilities. Assertions in unit testing can identify vulnerabilities. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | 2 | Unlikely | High | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | (Customization) | Users can implement a custom check to report assert() uses.  [CodeSonar SAST for C/C++: Static Code Analysis Tool | GrammaTech](https://www.grammatech.com/codesonar-cc) |
| Compass/Rose | \_\_\_\_ | \_\_\_\_ | Detects violations by looking for assert() calls.  [Rose Compiler – Program Analysis and Transformation](http://rosecompiler.org/?page_id%3D16) |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | **Handle all exceptions** : Any time an exception is thrown it must be caught by an exception handler to ensure that the stack is unwound correctly so that the application can terminate in a controlled manner (Ballman and O’Donnell, 2022). |

| **Noncompliant Code** |
| --- |
| This code block does not provide a handler for the exceptions making it impossible to catch the exceptions that are thrown. |
| void throwing\_function() noexcept(false);  void f(){  throwing\_function();  }  int main(){  f();  } |

| **Compliant Code** |
| --- |
| Adding handles for all of the exceptions will make the above code block compliant. |
| void throwing\_function() noexcept(false);  void f(){  throwing\_function();  }  int main(){  try{  f();  } catch (…){  // handle errors  }  } |

| **Principles(s):**  2. Exceptions need to be addressed, if exceptions are thrown then the errors need to be handled appropriately.  8. If an exception is thrown determine if a process or the entire system access should be aborted.  9. Using various techniques to catch errors and vulnerabilities. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | 2 | Probable | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | LANG.STRUCT.UCTCH | Detects unreachable catch.  [CodeSonar SAST for C/C++: Static Code Analysis Tool | GrammaTech](https://www.grammatech.com/codesonar-cc) |
| Parasoft C/C++ test | 2022.1 | **CERT\_CPP-ERR51-a**  CERT\_CPP-ERR51-b | Checks that all exceptions are caught and that each exception thrown has a handler of a compatible type.  [Automated Testing to Deliver Superior Quality Software | Parasoft](https://www.parasoft.com/) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | [STD-008-CPP] | **Guarantee that container indices and iterators are within the valid range**: Making sure that array or vector references are within the arrays or vectors lower and upper bounds. Not staying within the bounds can result in overflow (Pincar and Britton, 2022). |

| **Noncompliant Code** |
| --- |
| This code block checks the upper bound but fails to check the lower bound of the table which can result in overwriting data outside of the allocated memory. |
| Void insert\_in\_table(int \*table, std::size\_t tableSize, int pos,  int value) {  if(pos >= tableSize) {  //Handle error  return;  }  Table[pos] = value;  } |

| **Compliant Code** |
| --- |
| Declaring pos as a size\_t parameter will prevent data from being written outside of the bounds. |
| Void insert\_in\_table(int \*table, std::size\_t tableSize, std::size\_t pos,  int value) {  if(pos >= tableSize) {  //Handle error  return;  }  Table[pos] = value;  } |

| **Principles(s):**  1. Verify that the data being input fits within the variable type or structure, if an iterator verify that it does not exceed the structures capacity.  4. The simpler the code the less likely errors will occur. Be careful with loops and structures.  5. Understand the chosen languages data types and structures so ensure proper variables and sizes are used. Initiate all variables, including iterators to avoid unexpected behavior. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | 5 | Likely | High | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | **LANG.MEM.BO LANG.MEM.BU LANG.MEM.TO LANG.MEM.TU LANG.MEM.TBA LANG.STRUCT.PBB LANG.STRUCT.PPE LANG.STRUCT.PARITH** | Detects buffer overrun, buffer underrun, type overrun, type underrun, tainted buffer access, pointer before an object begins, pointer after and object ends, and pointer arithmetic.  [CodeSonar SAST for C/C++: Static Code Analysis Tool | GrammaTech](https://www.grammatech.com/codesonar-cc) |
| Parasoft C/C++ test | 2022.1 | CERT\_CPP-CTR50-a | Guarantees that container indices are within the valid range.  [Automated Testing to Deliver Superior Quality Software | Parasoft](https://www.parasoft.com/) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output | [STD-009-CPP] | **Close files when they are no longer needed**: Any call to open a file must be matched but a call to close the file before any pointer storing the return value of the has call ended or the program is terminated (Pincar and Britton, 2022). |

| **Noncompliant Code** |
| --- |
| This code block fails to close the file before terminating. |
| void function(const std::sting &filename) {  std::fstream file(filename);  if(!file.is\_open()) {  //Handle error  return;  }  //…  std::terminate();  } |

| **Compliant Code** |
| --- |
| Adding a call to close the file stream will make the above code compliant. |
| void function(const std::sting &filename) {  std::fstream file(filename);  if(!file.is\_open()) {  //Handle error  return;  }  //…  file.close();  if(file.fail()) {  //Handle error  }  std::terminate();  } |

| **Principles(s):**  3. Limit CRUD functionality by use roll and access needed.  5. Give users limited access to resources.  6. A process should require minimal access to perform tasks to minimalize the reach of data breaches.  8. Prevent data access. Closes files adds an additional layer of security. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | 2 | Unlikely | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | **ALLOC.LEAK** | Checks for leaks  [CodeSonar SAST for C/C++: Static Code Analysis Tool | GrammaTech](https://www.grammatech.com/codesonar-cc) |
| Parasoft C/C++ test | 2022.1 | CERT\_CPP-FIO51-a | Ensures resources are freed.  [Automated Testing to Deliver Superior Quality Software | Parasoft](https://www.parasoft.com/) |
| Polyspace Bug Finder | R2022b | CERT C++: FIO51-CPP | Checks for resource leaks.  [Polyspace - MATLAB & Simulink (mathworks.com)](https://www.mathworks.com/products/polyspace.html) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | [STD-010-CPP] | **Copy operations must not mutate the source object**: Constructors copy the primary properties of a source object to a destination object. When copying the source object, it is important not to mutate it or any globally accessible data. Mutating the source object can invalidate assumptions made by standard library algorithms (Ballman and O’Donnell, 2022). |

| **Noncompliant Code** |
| --- |
| This code block utilizes a copy constructor that mutates the source operand causing all future copies to contain the mutation. |
| class A {  mutable int m;  public:  A() : m(0){}  explicit A(int m) : m(m){}  A(const A &other) : m(other.m) {  other.m = 0;  }  A& operator = (const A &other) {  if(&other != this) {  m = other.m;  other.m = 0;  }  return \*this;  };  void f() {  std::vector<A> v{10};  A obj(12);  std::fill(v.begin(), b.end().obj);  } |

| **Compliant Code** |
| --- |
| Modifying the code so that the copy no longer mutates the source object but rather mutates the destination object at a later time will make the code block compliant. |
| class A {  int m;  public:  A() : m(0){}  explicit A(int m) : m(m){}  A(const A &other) : m(other.m) {}  A(A &&other) : m(other.m) {other.m = 0;}  A& operator = (const A &other) {  if(&other != this) {  m = other.m;  }  return \*this;  }  A& operator = (const A &other) {  m = other.m;  other.m = 0;  return \*this;  }  Int get\_m() const {return m; }  };  void f() {  std::vector<A> v{10};  A obj(12);  std::fill(v.begin(), b.end().obj);  } |

| **Principles(s):**  4. Use classes and functions appropriately to create copies of objects rather than mutating source properties.  7. Sanitize input to make sure it does not mutate the source object.  9. Test both newly created objects and the source object to make sure the source object has not ben mutated. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | 2 | Likely | Low | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | **LANG.FUNCS.COPING** | Detects if copy operation parameters are not constant.  [CodeSonar SAST for C/C++: Static Code Analysis Tool | GrammaTech](https://www.grammatech.com/codesonar-cc) |
| Parasoft C/C++ test | 2022.1 | CERT\_CPP-OOP58-a | Copy operations must not mutate the source object.  [Automated Testing to Deliver Superior Quality Software | Parasoft](https://www.parasoft.com/) |
| Polyspace Bug Finder | R2022b | CERT C++: OOP58-CPP | Checks for copy operations modifying source operands.  [Polyspace - MATLAB & Simulink (mathworks.com)](https://www.mathworks.com/products/polyspace.html) |

### 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. The following defines guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy.

-DevSecOps-

Transitioning from DevOps to DevSecOps will help Green Pace’s security to continually evolve. Security tasks will be integrated into each phase of the pipeline via automated static testing. The diagram above depicts how security fits into each of the phases making the entire pipeline emersed in a security centered development process. OWASP best coding practices will be implemented from day one.

How to ingrate secure coding practices adapt to regulatory changes, and how to respond to threats will be planned out. Security tests will be designed as well as making sure all outside sources be it libraries or open-source tools will be selected based on their own security assessment. The security tests and best coding practices will be built into the initial build rather then being implemented later by a separate security team. Security testing will be performed in the Verify and test phase along side functionally unit testing.

All of the security features will be monitored, continually tested, and upgraded on a regular basis. As the new phases of the system roll out, they will continue to be designed with security in mind. Using automated testing tools has the advantage of continually evolving as new risks are discovered and in turn, they will continue to protect Green Pace from those new risks.

### Summary of Risk Assessments

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High 3 | Unlikely | High | P3 | L3 |
| STD-002-CPP | High 5 | Likely | High | P9 | L2 |
| STD-003-CPP | High 4 | Likely | Medium | P18 | L1 |
| STD-004-CPP | High 4 | Likely | Medium | P18 | L1 |
| STD-005-CPP | High 4 | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low 2 | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low 2 | Probable | Medium | P4 | L3 |
| STD-008-CPP | High 5 | Likely | High | P9 | L2 |
| STD-009-CPP | Medium 2 | Unlikely | Medium | P4 | L3 |
| STD-010-CPP | Low 2 | Likely | Low | P9 | L2 |

### 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 refers to additional layers of security implemented to protect stored data that is not in use. The first step is to identify the data, how it is stored, and who has/needs access to it. The data is then encrypted in a way that unauthorized users cannot decrypt. User role-based access controls should be carefully implemented to minimalize the chance of a data breach and multi factor authentication should be required for those that do have access to the data (Brightline, 2017). Green Pace develops and maintains a database of their clients most sensitive data. Protecting that data while not in use is important for the integrity of Green Pace and for maintaining the trust of their clients. |
| Encryption at flight | Encryption at flight refers to protecting data as it transfers from its resting location to the user. Ensure that data is always transferred via an HTTPS protocol to ensure SSL or TLS security  is implemented to prevent “eavesdropping” on the data being transferred. This encrypts the data with symmetric and asymmetric key encryption that cannot be intercepted and decrypted until it reaches the end user (Lo Faso, 2015). Due to Green Pace’s need to maintain a secure database, it is essential that when accessing resting data, it is transferred to the user in a secure way. Encryption in flight will allow an authorized user anywhere in the world to access the database securely on a need only basis. |
| Encryption in use | Encryption in use projected data that has been accessed by a user from an attacker who has gained unauthorized access to the database. It ensures that sensitive data is always secure. AES-256 encryption is used in all data fields in all applications and data requests are analyzed in real time to block suspicious requests. Green pace will use this encryption model to protect for user end vulnerably such as phishing attacks, misconfigurations, and fake application data requests. This will ensure that Green Pace is in compliance with international data regulations (Das). |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the identification of a user and verifying that they are who they claim they are through a login process. Typically, this is done with a unique set of credentials that may include a user name and password, single sign-on, biometrics, digital certificate or public key. Authentication tells they system what level of authorization the user has (Mylonas, 2018). Green Pace with use Authentication to ensure that various users are granted appropriate access level to the database based off of their role with the corporation. |
| Authorization | Authorization determines what access level a user is granted to the system and what functions they can perform within said system. Authorization determines if a user can Create, Read, Update, and/or Delete entries in the database. Some users may be granted read only access while others, such as administrators will be able to make changes to the database. Authorization may be based off of factors beyond user authentication as well including geographical location, date/time restrictions, number of logins, IP address, bandwidth traffic or other customized, pre-determined restrictions (Mylonas, 2018). System administrators will be responsible for adding, monitoring, and deleting user’s and/or their level of access. |
| Accounting | Accounting documents the resources that are used by a user when they access the system. A statistical evaluation is made to determine data that is sent and received as well as how long the user is accessing the system. The results obtained from accounting can be used for a variety of reasons including monitoring authorization. A secondary server is used to hold these records and is only accessible by administrators to perform audits. Using an accounting system will allow Green Pace to monitor what files are being accessed by each user or if failed authentication or authorization attempts have been made (Mylonas, 2018). |

**\***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 logsThe 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 |  |
| 1.5 | 11/13/2022 | Coding Standards Added | Jessica Ayer |  |
| 2.0 | 12/4/2022 | Security Policy Finalized | Jessica Aer |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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

**References:**

*Axivion Suite* (2022) <https://www.axivion.com/en/products/axivion-suite/>

Ballman, A., O’Donnel, J. (May 19, 2022).  *OOP58-CPP. Copy operations must not mutate the source object*. Confluence. <https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP58-CPP.+Copy+operations+must+not+mutate+the+source+object>

Ballman, A., O’Donnel, J. (May 19, 2022).  *ERR51-CPP. Handle all exceptions*. Confluence. <https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions>

Boersma, Eric. *Security by Design: 7 Application Security Principles You Need to Know.* Cprime. https://www.cprime.com/resources/blog/security-by-design-7-principles-you-need-to-know/

Brightline IT (May 17, 2017). *What is Encryption at Rest, and Why Is It Important for Your Business?*. https://brightlineit.com/encryption-at-rest-important-business/

Das, Purandar. *In-Use Encryption – What It Is and How companies Benefit*. Sotero. https://www.soterosoft.com/blog/data-in-use-encryption-data-in-motion-encryption/

Gupta, Pooja. *Security Architecture*. EDUCBA. https://www.educba.com/security-architecture/

Lo Faso, Vince (April 28, 2015). *How To Protect Data in Flight*. Skillsoft. https://www.globalknowledge.com/us-en/resources/resource-library/articles/how-to-protect-data-in-flight/

Meier, J.D., Mackman, A., Dunner, M., Srinath, V., Escamilla, R., Murukan, A. (April 7, 2008). *Security Design Principles -Input/Data Validation*. <https://guidanceshare.com/wiki/Security_Design_Principles_-_Input/Data_Validation>

Mylonas, Leo (November 27, 2018). *What is AAA security? An introduction to authentication, authorization, and accounting*. Codebots. https://codebots.com/application-security/aaa-security-an-introduction-to-authentication-authorisation-accounting

Pincar, J. & Britton, J. (April 27, 2022). *FIO51-CPP. Close files when they are no longer needed*. Confluence.

<https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51-CPP.+Close+files+when+they+are+no+longer+needed>

Pincar, J. & Britton, J. (April 27, 2022). *MEM50-CPP. Do Not access freed memory*. Confluence. https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory

Pincar, J. & Britton, J. (June 30, 2022).  *CTR50-CPP. Guarantee that container indices and iterators are within the valid range.*  Confluence. <https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR50-CPP.+Guarantee+that+container+indices+and+iterators+are+within+the+valid+range>

*PVS-Studio* (November 23, 2022). Wikipedia. https://en.wikipedia.org/wiki/PVS-Studio

Saltzer, J.H (July 1974). *Protection and the Control of Information Sharing in Multics: Communication of the ACM 17,7*. Pages 388-404.

Saltzer, J.H & Schroeder, M.D. (September 1975). *The protection of Information Sharing in Computer Systems: Proceedings of the IEEE 63, 9*. Pages 1278-1308.

Seacord, Robert (May 2, 2018) *Top 10 Secure Coding Practices.* Confluence. <https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices>

Seacord, Robert (June 24, 2021).  *MSC00-C. Compile cleanly at high warning levels*. Confluence. [MSC00-C. Compile cleanly at high warning levels - SEI CERT C Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/c/MSC00-C.+Compile+cleanly+at+high+warning+levels)

Seacord, R. & Britton, J. (December 11, 2018).  *DCL03-C. Use a static assertion to test the value of a constant expression*. Confluence. https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression

Seacord, R. & Britton, J. (April 27, 2022). *STR02-C. Sanitize data passed to complex subsystems*. Confluence. https://wiki.sei.cmu.edu/confluence/display/c/STR02-C.+Sanitize+data+passed+to+complex+subsystems

Seacord, R. & Britton, J. (October 3, 2022). *Ensure that operations on signed integers do not result in overflow*. Confluence. https://wiki.sei.cmu.edu/confluence/display/c/INT32-C.+Ensure+that+operations+on+signed+integers+do+not+result+in+overflow

Seacord, R., Razmyslov, S. (June 25, 2021). *INT00-C. Understand the data model used by your implementation(s)*. Confluence. <https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152417>

Svoboda, David (October 27, 2022).  *STR31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator*. Confluence. https://wiki.sei.cmu.edu/confluence/display/c/STR31-C.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator