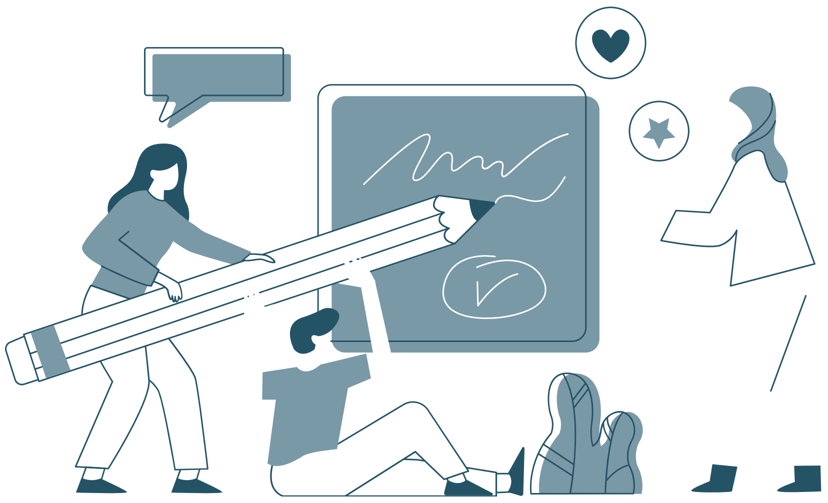
CODING STANDARDS
DIAGRAM

CONCRETE PASS

Table of Contents



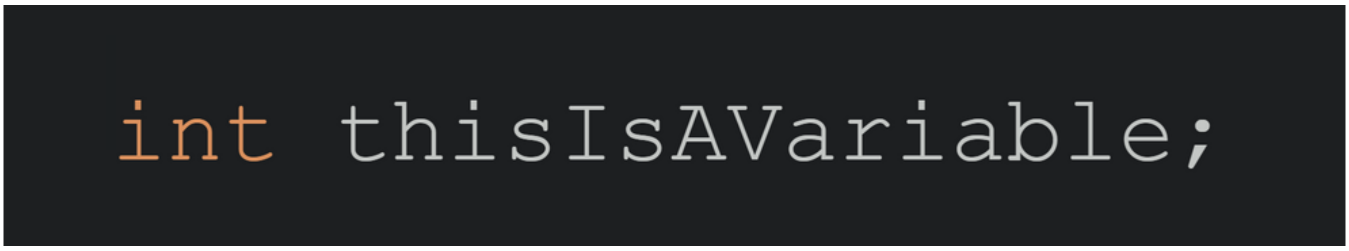
INTRODUCTION

This coding standards diagram visually outlines the essential coding standards, including naming conventions, Git practices, code structure, testing, documentation, code review, and security considerations. It serves as a quick reference to ensure consistency and quality in our code.

NAMING CONVENTIONS

VARIABLES /FUNCTIONS/METHODS

Using Camel Case

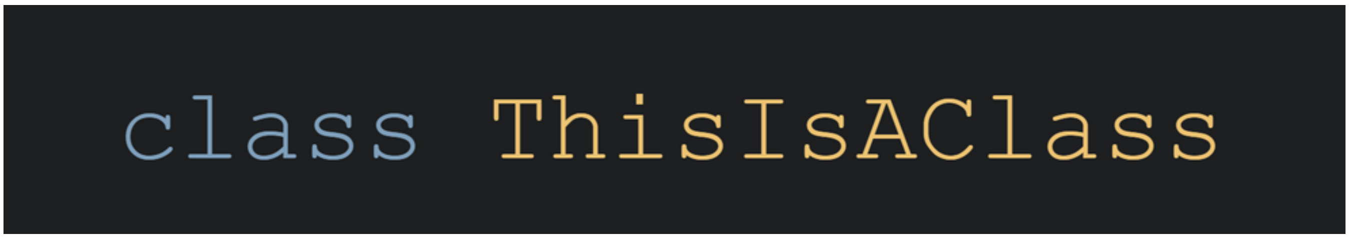


```
int thisIsAVariable;
```

Use Descriptive names

CLASSES

Use Pascal Case



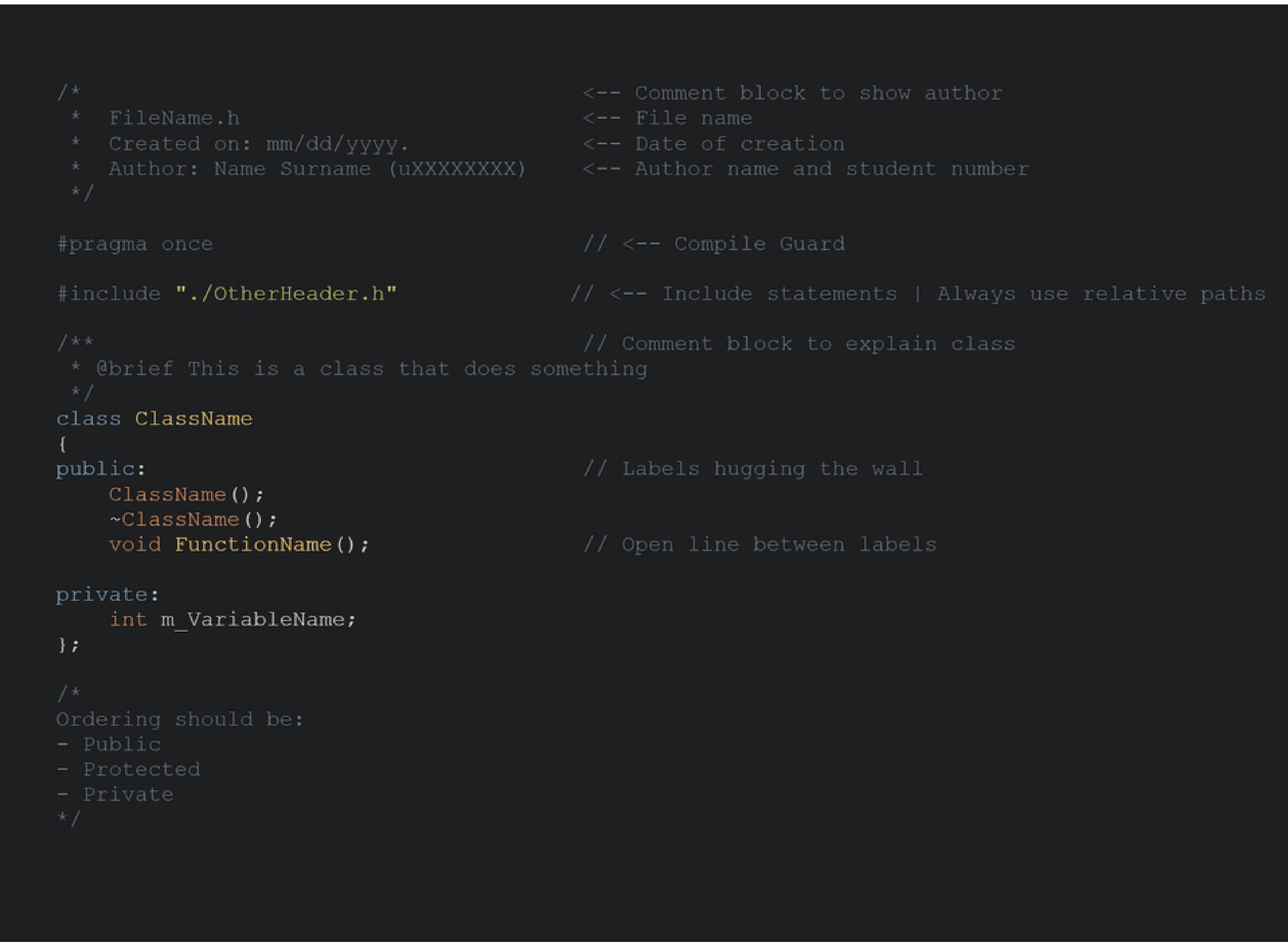
```
class ThisIsAClass
```

Use Descriptive nouns

NAMING CONVENTIONS

CLASSES

HEADER:

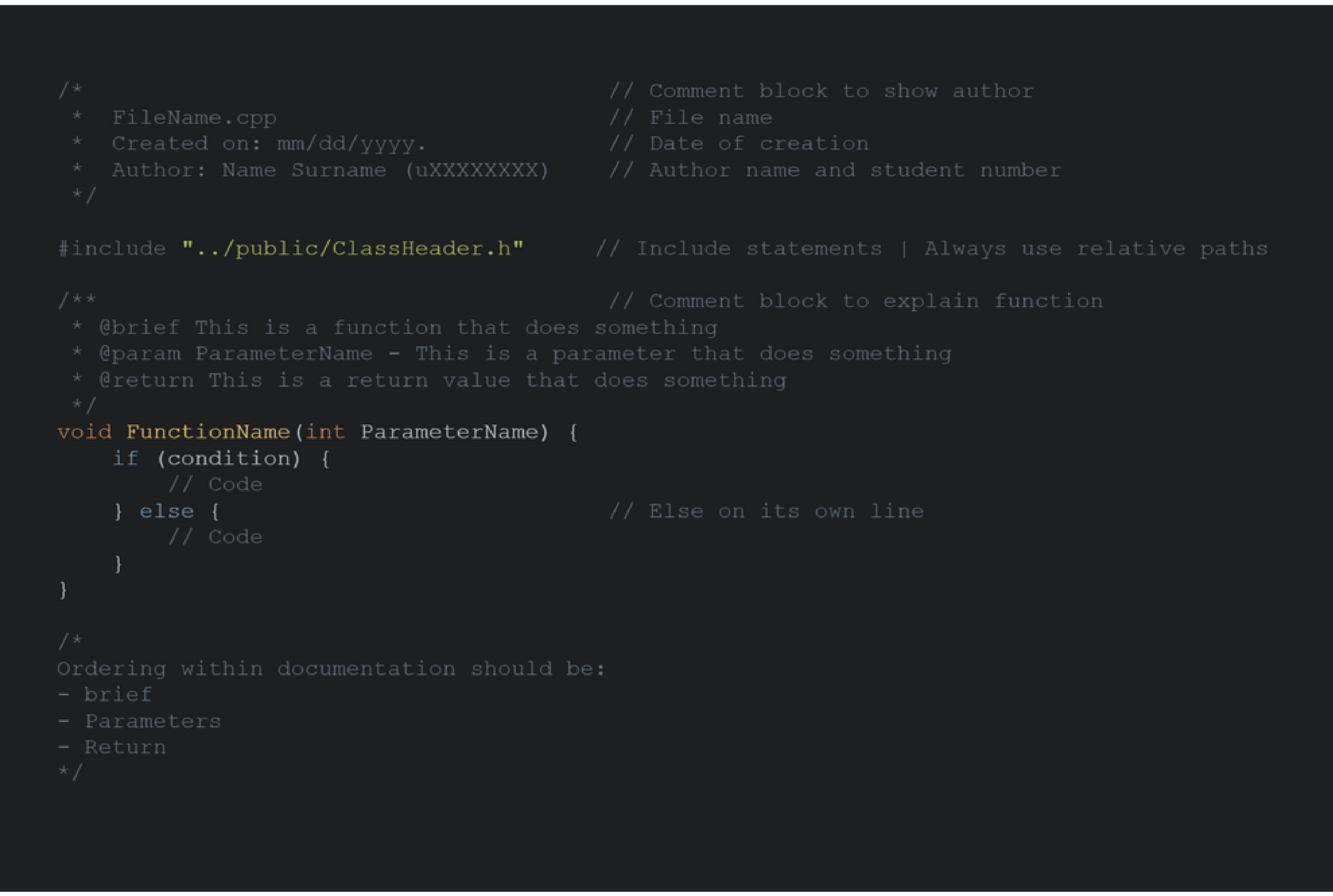


```
/*                                      <-- Comment block to show author
 *  FileName.h                          <-- File name
 *  Created on: mm/dd/yyyy.             <-- Date of creation
 *  Author: Name Surname (uXXXXXXXX)    <-- Author name and student number
 */

#pragma once                            // <-- Compile Guard

#include "./OtherHeader.h"             // <-- Include statements | Always use relative paths

/**                                     // Comment block to explain class
 * @brief This is a class that does something
 */
class ClassName
{
public:                                 // Labels hugging the wall
    ClassName();
    ~ClassName();
    void FunctionName();                // Open line between labels

private:
    int m_VariableName;
};

/*
Ordering should be:
- Public
- Protected
- Private
*/
```

NAMING CONVENTIONS

CLASSES

SOURCE FILES:



```
/*                                      // Comment block to show author
 *  FileName.cpp                        // File name
 *  Created on: mm/dd/yyyy.             // Date of creation
 *  Author: Name Surname (uXXXXXXXX)    // Author name and student number
 */

#include "../public/ClassHeader.h"     // Include statements | Always use relative paths

/**                                     // Comment block to explain function
 * @brief This is a function that does something
 * @param ParameterName - This is a parameter that does something
 * @return This is a return value that does something
 */
void FunctionName(int ParameterName) {
    if (condition) {
        // Code
    } else {                            // Else on its own line
        // Code
    }
}

/*
Ordering within documentation should be:
- brief
- Parameters
- Return
*/
```

NAMING CONVENTIONS

FILES/MODULES

They should be named using Pascal Case

GIT STANDARDS

COMMIT MESSAGES

- Commit Messages should be clear i.e., Write clear and informative commit messages that succinctly describe the purpose of each commit.
- Use the imperative mood (e.g., "Add feature" instead of "Added feature") for consistency.
- Reference issues or pull requests related to the commit by using keywords (e.g., "Closes #123" or "Fixes #456").

GIT STANDARDS

DOCUMENTATION

- Maintain a comprehensive README file with project information, setup instructions, and usage guidelines.

PULL REQUESTS

- Create a pull request for each feature, bug fix, or code change.
- Provide a meaningful title and description in the pull request.
- Assign reviewers and use labels to categorize and prioritize pull requests.
- Conduct thorough code reviews for all pull requests.

CODE STRUCTURE

```cpp
// Code Structure Guidelines for C++
// ------------------------------

// 1. Indentation: Use spaces for indentation.
void exampleFunction() {
    int variable1 = 42;
    if (variable1 > 0) {
        for (int i = 0; i < variable1; ++i) {
            // Indentation helps improve code readability.
        }
    }
}

// 2. Line Length: Limit line length to 80 characters for improved code readability.
std::string longMessage =
    "This is an example of a long message that should be wrapped to maintain a maximum line length of
80 characters.";

// 3. Imports/Includes: Organize and order import statements consistently.
#include <iostream> // Standard library includes come first.
#include "my_header.h" // Next, include your own header files.

// 4. Whitespace Usage: Use whitespace to enhance code readability.
int value = 42;
int result = value * (3 + 2);
```

DOCUMENTATION

Use Doxygen to generate detailed documentation

```cpp
// Documentation Guidelines for C++
// ------------------------------

// Inline Documentation:
// - Encourage adding comments within the code to clarify complex logic or non-obvious sections.
// - Follow a consistent commenting style (e.g., "//" for single-line comments, "/* */" for
multi-line comments).

class MyClass {
public:
    /**
     * This function performs an important task.
     * @param parameter1: A description of the first parameter.
     * @param parameter2: A description of the second parameter.
     * @return: A description of the return value.
     */
    int myFunction(int parameter1, int parameter2);
};

// API Documentation:
// - Detail how to document public APIs and interfaces for libraries and services.
// - Consider using documentation tools like Doxygen, Javadoc, or tools specific to your API.

/**
 * @class MyLibrary
 * @brief A brief description of the library.
 *
 * More detailed information about the library goes here.
 */
class MyLibrary {
public:
    /**
     * @brief Perform an action.
     *
     * This function performs a specific action and returns the result.
     *
     * @param input An input parameter.
     * @return The result of the action.
     */
    int performAction(int input);
};
```

DOCUMENTATION

```
// ReadMe Files:
// - Provide guidelines for creating comprehensive project documentation in ReadMe files.
// - Include an overview of the project, setup instructions, and usage guidelines.

# My Project

This is a description of my project.

## Getting Started

To get started, follow these steps:
1. Step 1
2. Step 2

## Usage

How to use this project:
1. Usage 1
2. Usage 2
```

SOURCES

https://google.github.io/styleguide/cppguide.html