**Coding Standards and Guidelines**

**1. Coding Standards**

**1.1 Global Variable Usage**

* Strictly limit the use of global variables due to potential side effects and reduced code predictability.
* Clearly define allowed data types for global variable declaration.
* Initialize variables on declaration whenever possible.

**1.2 Header Files**

* Employ header guards to prevent multiple inclusions of the same header file.

#ifndef MY\_HEADER\_H

#define MY\_HEADER\_H

// Header content goes here

#endif // MY\_HEADER\_H

**Naming conventions for header guards:**

* Typically use uppercase letters and underscores.
* Match the filename of the header file (in uppercase). For example, if your header file is named my\_header.h, the guard would be MY\_HEADER\_H.

**1.3 Naming Conventions**

* **Classes:** Utilize PascalCase with uppercase letters for word separators (e.g., MyClass). Class names should reflect the object's functionality, acting as nouns that describe their purpose.
* **Member Variables:** Prefix private member variables with m\_ followed by PascalCase (e.g., m\_memberVariable). This convention clearly distinguishes private members from public methods and class names.
* **Methods:** Employ camelCase with verbs at the beginning to clearly convey the action performed by the method (e.g., calculateArea).
* **Function/Method Arguments:** Begin with lowercase letters, followed by PascalCase for subsequent words (e.g., calculateArea(int radius)).
* **Variables:** Start with lowercase letters, followed by camelCase (e.g., localData). Avoid keywords and special characters (except underscore \_).
* **Pointer Variables:** Prefix with p\_ and place the asterisk (\*) directly next to the variable name for clarity (e.g., int\* p\_data).
* **Reference Variables:** Prefix with r\_ to differentiate between methods returning modifiable and non-modifiable objects (e.g., const std::string& r\_data).
* **Global Constants:** Use all uppercase letters separated by underscores (\_) (e.g., MAX\_VALUE).

**File Naming Conventions**

* Use lowercase letters, underscores (\_), and hyphens (-) for file names, adhering to a consistent naming pattern.
* File names should accurately reflect the content of the file (e.g., my\_class.cpp, data\_processing\_functions.h).
* Standard extensions are .cpp or .cc for source files and .h for header files. Avoid names that conflict with system header files.

**1.4 Indentation**

* Maintain consistent indentation (typically 4 spaces) for improved code readability.
* Indent nested code blocks to visually represent their hierarchy.

**1.5 Braces**

* Always use braces around code blocks, even for single-line statements, to improve readability and maintainability.

**1.6 Error Handling**

* Establish a standardized error handling approach. Consider using exceptions or error codes with clear and descriptive error messages.

**1.7 Pointers and References**

* Employ pointers and references judiciously, and document their usage clearly to avoid memory management issues and dangling pointers.
* Utilize smart pointers (e.g., std::unique\_ptr, std::shared\_ptr) for automatic memory management when appropriate.

**2. Coding Guidelines**

**2.1 Meaningful Comments**

* Avoid single-line comments that simply restate the code.
* Use comments to explain the "why" behind code decisions, not just the "what."

**2.2 Function Length**

* Strive for functions to be less than 50 lines of code (excluding comments) for better readability and maintainability.
* Decompose large functions into smaller, more manageable units with clear responsibilities.

**3. Additional Considerations**

**3.1 Build System and Code Formatting**

* Employ a build system (e.g., Make, CMake) to automate the compilation and linking process.
* Utilize code formatting tools (e.g., Clang-Format, AStyle) to enforce consistent coding style across the project.

**3.2 Unit Testing**

* Implement unit testing practices using frameworks like Google Test or Catch2 to ensure the functionality of individual code units.

**3.4 Reference Resources**

[Google C++ Style Guide](https://students.cs.byu.edu/~cs235ta/references/Cpp%20Style/Google%20Cpp%20Style%20Guide.pdf)