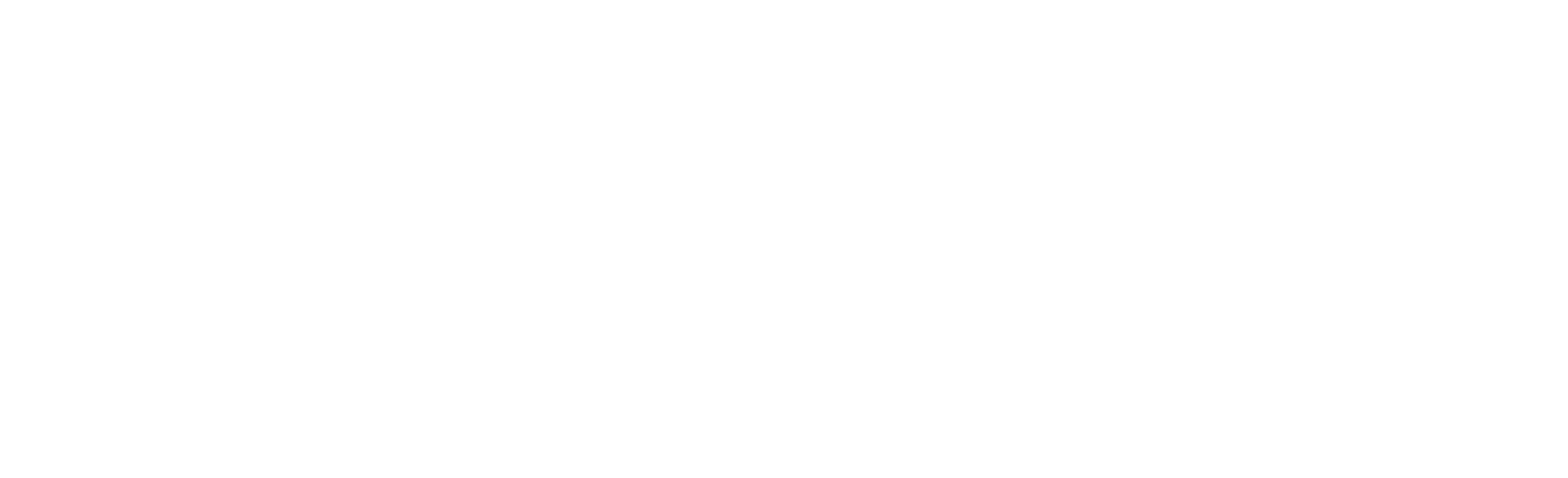


AIRGEAD

BANKING

|

2019



A

IRGEAD

B

ANKING

S

TANDARDS

D

OCUMENT

# TABLE OF CONTENTS

[TABLE OF CONTENTS 1](#_Toc5034)

[Coding Best Practices to be Followed 2](#_Toc5035)

[Safety 2](#_Toc5036)

[Style 2](#_Toc5037)

[Maintainability 3](#_Toc5038)

[Portability 3](#_Toc5039)

[References 4](#_Toc5040)

# Coding Best Practices to be Followed

## Safety

* Returning values from functions by reference can have significant performance savings when the normal use of the returned value is for observation.
* Avoid raw memory access.

|  |  |  |
| --- | --- | --- |
|  | // Bad Idea  MyClass \*myobj = new MyClass;    // ...  delete myobj;      // Good Idea auto myobj = std::make\_unique<MyClass>(constructor\_param1, constructor\_param2); // C++14 auto myobj = std::unique\_ptr<MyClass>(new MyClass(constructor\_param1, constructor\_param2)); // C++11  auto mybuffer = std::make\_unique<char[]>(length); // C++14  auto mybuffer = std::unique\_ptr<char[]>(new char[length]); // C++11    // or for reference counted objects  auto myobj = std::make\_shared<MyClass>();    // ...  // myobj is automatically freed for you whenever it is no longer used. |  |

* Use exceptions.
* Use C++-style cast instead of C-style cast.

## Style

* Common C++ naming conventions:
  + Types start with upper case: MyClass.
  + Functions and variables start with lower case: myMethod.
  + Constants are all upper case: const double PI=3.14159265358979323;.
* The C++ Standard Library and other well-known C++ libraries like Boost use these guidelines:
  + Macro names use upper case with underscores: INT\_MAX. o Template parameter names use camel case: InputIterator.
  + All other names use snake case: unordered\_map.
* Variable names follow [camel case notation.](https://en.wikipedia.org/wiki/Camel_case)
* Distinguish private object data.
  + Name private data with an m\_ prefix to distinguish it from public data. The m\_ stands for "member" data.
* Distinguish function parameters.
  + Name function parameters with a t\_ prefix. The t\_ can be thought of as "the,” but the meaning is arbitrary. The point is to distinguish function parameters from other variables in scope while using a consistent naming strategy.

|  |
| --- |
| Example  struct Size  {  int width; int height;  Size(int t\_width, int t\_height) : width(t\_width), height(t\_height) {}  };    // This version might make sense for thread safety or something,  // but more to the point, sometimes we need to hide data, sometimes we don't.  class PrivateSize  { public:  int width() const { return m\_width; } int height() const { return m\_height; }  PrivateSize(int t\_width, int t\_height) : m\_width(t\_width), m\_height(t\_height) {}    private: int m\_width;  int m\_height;  }; |

## Maintainability

* Avoid using assert() in your code. This causes the program to fail when something unexpected happens. Consider using try/catch.
* Readable code is easy to understand. Write your code and comment on it as if a non-programmer were reading it.
* When needed, refactor code to improve its use.
* Properly utilize try/catch statements.

o These keywords prevent your code from running at the client side (due to run-time errors) and can enable the program to “exit gracefully” with proper error messages as well as continue the program execution.

## Portability

* Have minimal code in main(). Most of your code should be in classes and main is used simply as a driver to instantiate objects of these classes and call proper functions.
* Use header files. Your class prototype should exist in an H file while the body of that class should exist in a CPP file.

|  |
| --- |
| // tree.h  class tree  { public:  double height;  void SetHight(double h); |
| double GetHeight();  };      // tree.cpp #include “tree.h”  void Tree::SetHight(double h)  {  height = h;  }  double Tree::GetHeight()  {  return height;  } |

● The #define guard:

* All header files should have #define guards to prevent multiple inclusion. The format of the symbol name should be <PROJECT>\_<PATH>\_<FILE>\_H\_.
* To guarantee uniqueness, they should be based on the full path in a project's source tree. For example, the file foo/src/bar/baz.h in project foo should have the following guard:

|  |
| --- |
| #ifndef FOO\_BAR\_BAZ\_H\_  #define FOO\_BAR\_BAZ\_H\_  ...    #endif // FOO\_BAR\_BAZ\_H\_ |

# References

Google C++ Style Guide. (n.d.). Retrieved from <https://google.github.io/styleguide/cppguide.html>

Turner, J. (2019, June 06). GitHub Collaborative Collection of C++ Best Practices. Retrieved from <https://github.com/lefticus/cppbestpractices>