Naming Conventions for C++ 17

2-28-2025

by Atomic Badger

“What are the naming conventions to be used as standard in C++ 17?”

ChatGPT response:

C++17 doesn’t enforce any strict naming conventions, but there are widely accepted best practices that developers follow to maintain readability, consistency, and maintainability in code. Below are the standard naming conventions used in C++17:

---

1. Variables and Data Members

- Use **camelCase** or **snake_case**.
- Prefer **snake_case** for consistency with the standard library.

```
int my_variable;  // Recommended (consistent with std library)
int myVariable;   // Alternative but less common
```

- For **class data members**, some conventions prepend m_ or _ to distinguish them:

```
class Example {
private:
    int m_value;  // Recommended
    int _value;   // Alternative (avoid leading underscore)
};
```

---

2. Constants

- Use **UPPER_CASE_SNAKE_CASE** for global or file-level constants:

```
const int MAX_BUFFER_SIZE = 1024;
constexpr double PI = 3.1415926535;
```

- For **class-level constants**, prefer static constexpr:

```
class MathConstants {
public:
    static constexpr double E = 2.71828;
};
```

---

3. Functions

- Use **camelCase** or **snake_case** (prefer snake_case for consistency with STL).

```
void processData();  // camelCase (common in legacy C++)
void process_data(); // Recommended (matches STL)
```

- Function **parameters** should follow snake_case:

```
void set_value(int new_value) { /* ... */ }
```

---

4. Classes, Structs, and Enums

- Use **PascalCase** (UpperCamelCase).

```
class DataProcessor {
    // ...
};

struct Point3D {
    float x, y, z;
};

enum class ErrorCode {
    NotFound,
    PermissionDenied,
    Unknown
};
```

---

5. Enums

- Use **PascalCase** for the enum name.
- Enum members are either PascalCase or SCREAMING_SNAKE_CASE.

```
enum class Status {
    OK,
    ERROR,
    TIMEOUT
};
```

- If using enum instead of enum class, prefix names to avoid conflicts:

- `enum ErrorCode {`

```
    ERROR_OK,
    ERROR_NOT_FOUND,
    ERROR_TIMEOUT
};
```

6. Templates and Generics

- Template type parameters should be **PascalCase**.
- Common generic names: T, U, V, Key, Value, etc.

```
template <typename T>
class Container {
    T value;
};
```

7. Macros and Preprocessor Directives

- Use **ALL_CAPS_SNAKE_CASE**.
- Prefix with a unique identifier to avoid conflicts.

```
#ifndef PROJECT_CONFIG_H
#define PROJECT_CONFIG_H

#define MAX_CONNECTIONS 100

#endif
```

- Avoid leading underscores (_NAME or __NAME) as they are reserved by the implementation.

8. Namespaces

- Use **lowercase** for namespaces (or PascalCase if project-specific).

```
namespace my_library {
    class Example {};
}

namespace MyProject {
    class Engine {};
}
```

9. Global Variables (Avoid when possible)

- Prefix with g_ to indicate a global variable.

```
int g_global_counter = 0;
```

10. Member Functions

- Use **snake_case** (or camelCase if preferred).
- Accessors and mutators often follow the getX/setX pattern.

```
class Example {
public:
    int get_value() const { return value; }
    void set_value(int new_value) { value = new_value; }

private:
    int value;
};
```

Summary Table

Category:	**Naming Convention:**	**Example:**
Variables	snake_case	my_variable
Constants	UPPER_CASE_SNAKE	MAX_SIZE
Functions	snake_case	process_data()
Classes/Structs	PascalCase	DataProcessor
Enums	PascalCase	enum class Status { OK, ERROR }
Enum Members	SCREAMING_SNAKE	ERROR_NOT_FOUND
Macros	UPPER_CASE_SNAKE	#define BUFFER_SIZE 512
Namespaces	lowercase or PascalCase	namespace my_library {}
Template Types	PascalCase	template<typename T>
Global Variables	Prefix with g_	g_config_path