C++ Features to Study

Introduction

This document lists key C++ features to study, with a focus on threading and safety enhancements, organized by C++ version. Each feature is accompanied by a brief description to guide your study.

C++11 Features

- **std::thread**
  Introduces threading support in the standard library, enabling concurrent execution of code.
- **std::mutex, std::lock, std::unique_lock**
  Provides mechanisms for managing access to shared resources and preventing race conditions.
- **std::async, std::future, std::promise**
  Introduces asynchronous programming facilities, allowing tasks to run asynchronously and return results via futures.
- **std::atomic**
  Introduces atomic operations to ensure thread safety without using mutexes, providing a performance boost in some cases.
- **std::unique_ptr, std::shared_ptr**
  Introduces smart pointers to automate memory management and reduce the risk of memory leaks and dangling pointers.
- **move semantics and rvalue references**
  Enhances performance by enabling efficient transfer of resources, minimizing unnecessary copying.

C++14 Features

- **std::shared_timed_mutex**
  Extends mutex capabilities with timed locking, allowing threads to attempt locking for a specific duration.

- **Relaxed constexpr**
  Allows more complex expressions to be evaluated at compile-time, enhancing the flexibility of ‘constexpr‘ functions.
- **Generic lambdas**
  Simplifies the use of lambdas by allowing them to accept parameters of any type, promoting code reuse.

C++17 Features

- **std::shared_mutex**
  Introduces shared mutexes that allow multiple readers or one writer, improving concurrency control in read-heavy scenarios.
- **std::scoped_lock**
  Provides a convenient way to lock multiple mutexes simultaneously, preventing deadlocks.
- **std::optional**
  Represents optional values, reducing the risk of null pointer dereferencing and clarifying intent.
- **std::variant**
  Provides a type-safe union, allowing a variable to hold one of several specified types.
- **std::any**
  Allows storing values of any type with runtime type information, promoting flexibility.
- **std::byte**
  Introduces a type-safe representation of byte data, clarifying the intent when working with raw memory.

C++20 Features

- **std::jthread**
  A more convenient thread class that automatically joins upon destruction, reducing the risk of resource leaks.
- **std::atomic_ref**
  Introduces atomic operations for non-atomic data, allowing atomic operations on shared resources without altering their type.
- **Coroutines**
  Enables writing asynchronous code in a synchronous style, simplifying the management of asynchronous tasks.

- **Concepts**
  Introduces constraints for templates, improving the readability and safety of template code.
- **Ranges**
  Provides a new way to work with sequences of data, offering a safer and more expressive alternative to traditional iterators.
- **Modules**
  Facilitates better modularization of code, reducing compilation times and improving code organization.

C++23 Features

- **std::expected**
  A safer alternative to ‘std::optional‘ for error handling, providing a way to represent either a value or an error.
- **Enhanced constexpr**
  Expands ‘constexpr‘ capabilities, allowing more complex logic and data structures to be evaluated at compile time.
- **std::span**
  Provides a view over a contiguous sequence of elements, offering safer and more expressive access to array data.

Additional Resources

- cppreference.com
  Comprehensive reference for C++ features, including details on the latest standards.
- Clang Website
  Lists C++ features supported by the Clang compiler, useful for understanding what is available and implemented.