# Documentation: Portfolio Part-1

Alexander Ricciardi

Colorado State University Global

CSC450: Programming III

Professor: Reginald Haseltine

November 24, 2024

**Documentation: Portfolio Part-1**

This documentation is part of the Module-7 Portfolio Milestone from CSC450: Programming III at Colorado State University Global. The program's name is Thread Counting Synchronization. It provides an overview of the program's functionality and testing scenarios, including console output screenshots. The program is coded in C++ 17.

**The Assignment Direction:**

Portfolio Project: Part 1
For your Portfolio Project, you will demonstrate an understanding of the various concepts discussed in each module. For the first part of your Portfolio Project, you will create a C++ application that will exhibit concurrency concepts. Your application should create two threads that will act as counters. One thread should count up to 20. Once thread one reaches 20, then a second thread should be used to count down to 0. For your created code, provide a detailed analysis of appropriate concepts that could impact your application. Specifically, address:

- Performance issues with concurrency
- Vulnerabilities exhibited with use of strings
- Security of the data types exhibited.

Compile and submit your pseudocode, source code, and screenshots of the application executing the application, the results and your GIT repository in a single document.

To receive full credit for the packaging requirements for your Critical Thinking and Portfolio assignments you must:
1) Put your C++ source code in .cpp text files. Note that I execute all your programs to check them out.
2) In a Word or PDF "documentation" file, labeled as such, put a copy of your C++ source code and execution output screen snapshots.
3) Some positive evidence that you've definitely stored your source code in a GitHub repository on GitHub.com.
4) Include a detailed analysis paper in APA Edition 7 format of the important concepts of concurrency with C++ to cover in detail performance issues, string vulnerabilities, and security of data types. Here's a link to the school's Writing Center where you can find the relevant APA Edition 7 requirements you need to follow -> https://csuglobal.libguides.com/writingcenterLinks to an external site.
5) Put all your files into a single .zip file, and submit ONLY that .zip file for grading. Do not submit any additional separate files.

⚠️ **My notes:**

- The simple C++ console application is in file **PF1-Thread Counting Synchronization.cpp**
- The program follows the following SEI CERT C/C++ Coding Standard:
    - CON50-CPP. Do not destroy a mutex while it is locked
    - CON51-CPP. Ensure actively held locks are released on exceptional conditions
    - CON52-CPP. Prevent data races when accessing bit-fields from multiple threads
    - CON54-CPP. Wrap functions that can spuriously wake up in a loop
    - CON55-CPP. Preserve thread safety and liveness when using condition variables
    - ERR50-CPP. Do not abruptly terminate the program
    - ERR51-CPP. Handle all exceptions
    - ERR55-CPP. Honor Exception Specifications
    - STR50-CPP. Guarantee that storage for strings has sufficient space
    - STR51-CPP. Do not attempt to create a std::string from a null pointer

- STR52-CPP. Use valid references, pointers, and iterators to reference elements of a basic_string

**Program Description:**

This program demonstrates the use of threads and how to synchronize them using mutexes and condition variables.
Thread 1 counts up from 0 to a maximum count, while Thread 2 waits until Thread 1 completes, and then counts down from the maximum count to 0.

**Git Repository**

I use GitHub as my Distributed Version Control System (DVCS), the following is a link to my GitHub, Omegapy.
My GitHub repository that is used to store this assignment is named My-Academics-Portfolio.
The link to this specific assignment is:
https://github.com/Omegapy/My-Academics-Portfolio/tree/main/Programming-3-CSC450/Portfolio-Part-1

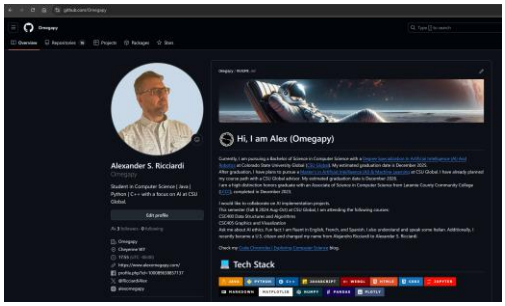
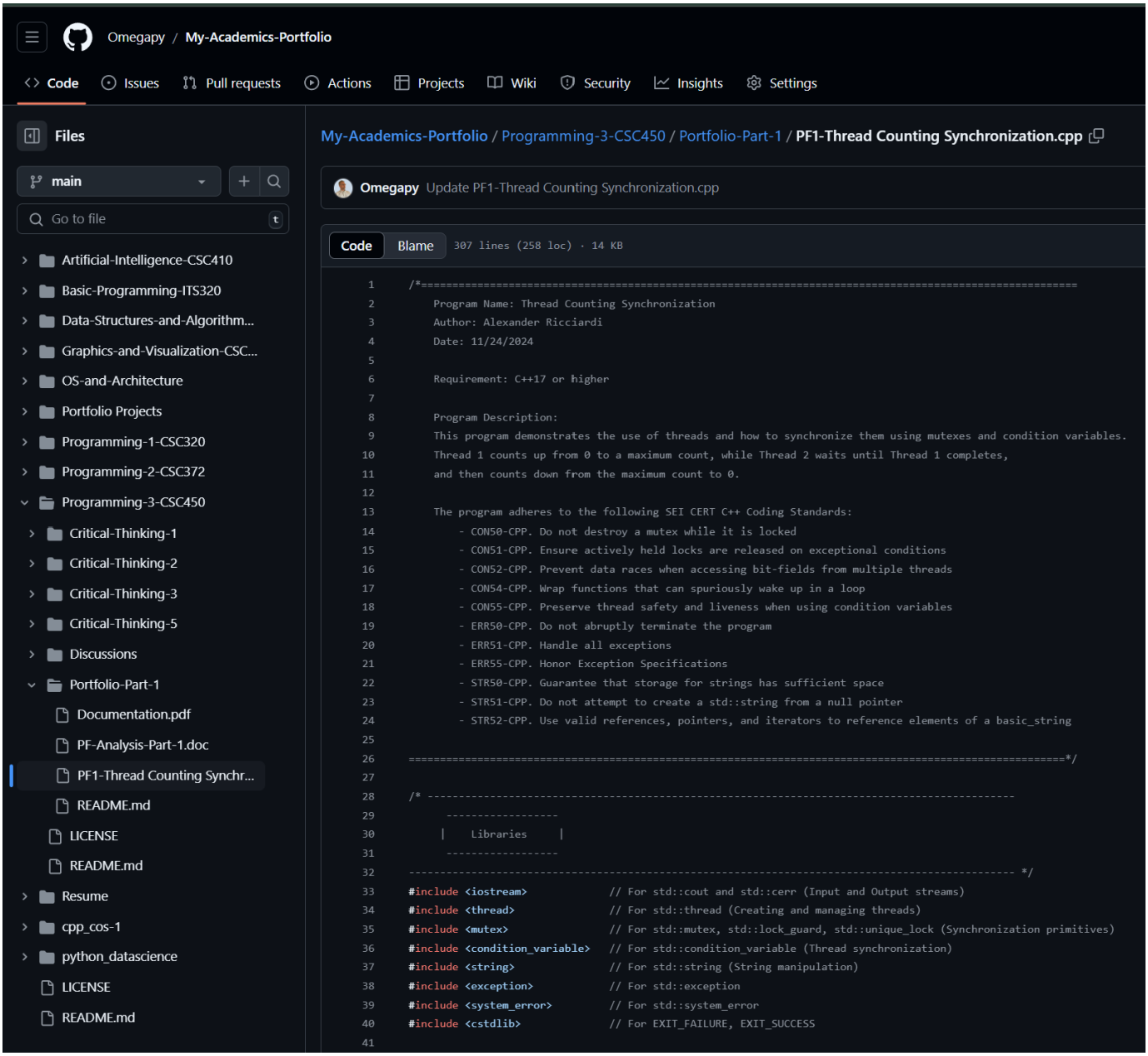
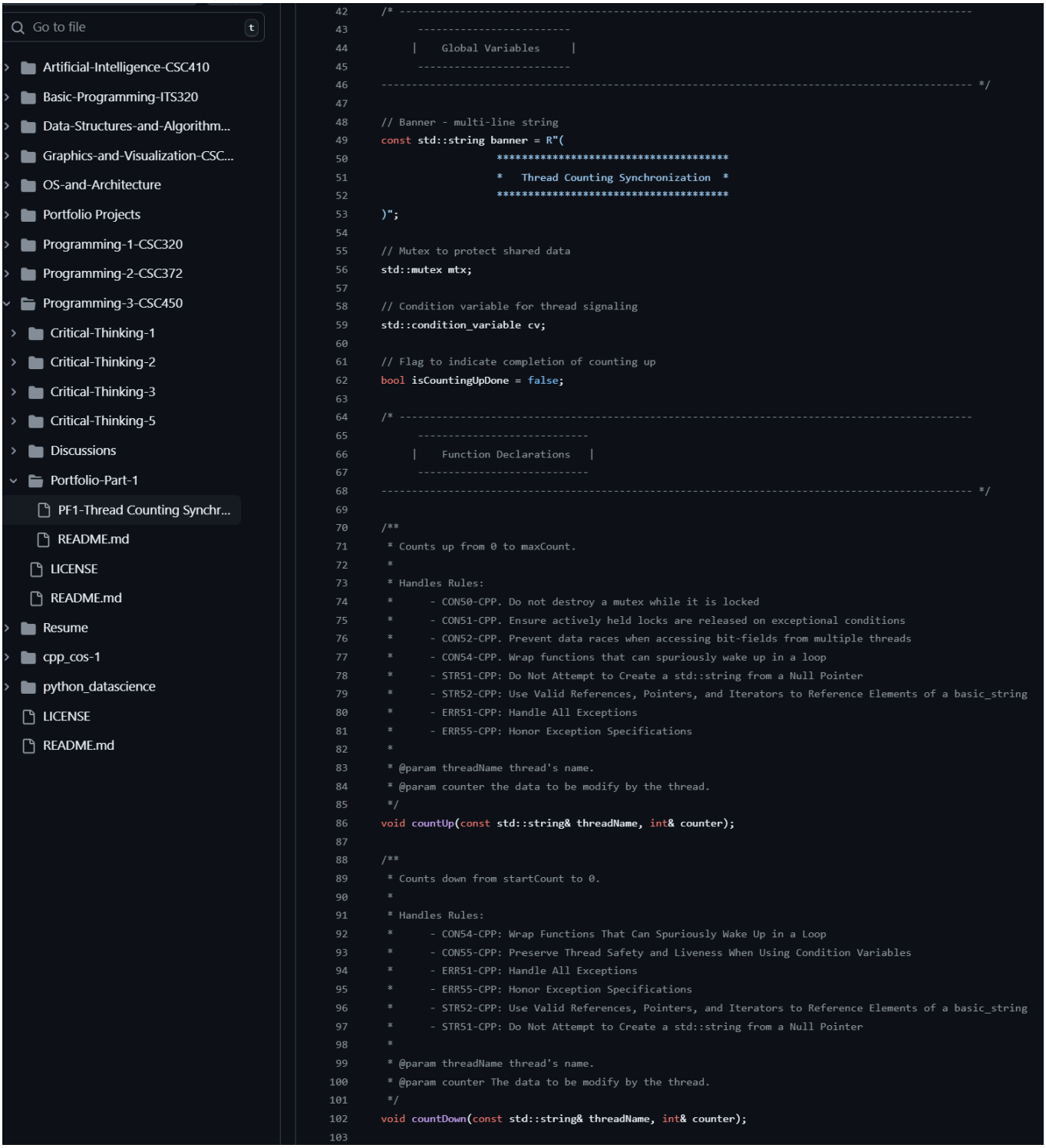



Image of the source code in the GitHub: see next page

**Figure 1**
*Source Code in GitHub*

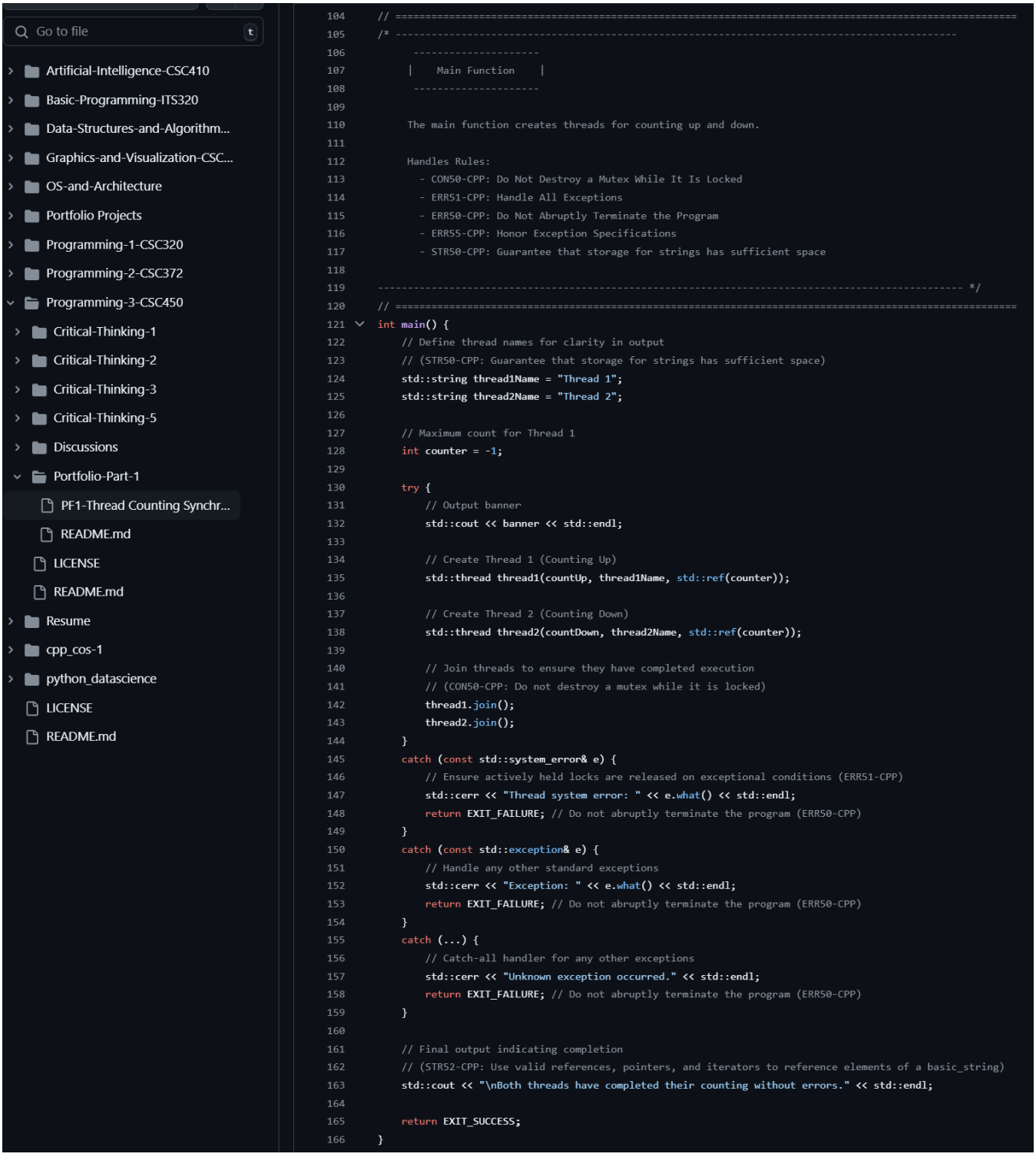



*Continue next page*



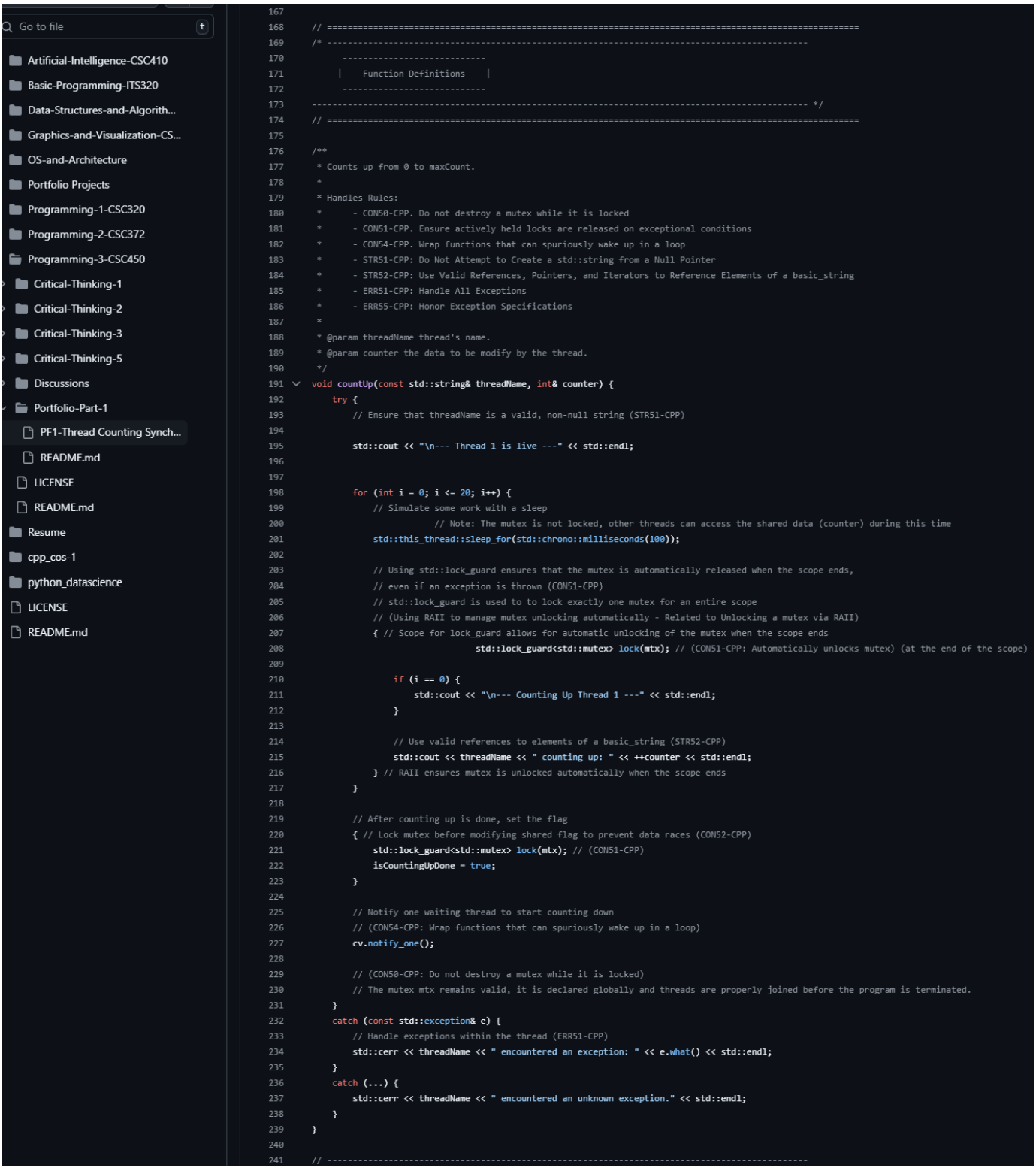

```cpp
/* ------------------------------------------------------------------------------------------
      -------------------------
     |    Global Variables     |
      -------------------------
------------------------------------------------------------------------------------------ */

// Banner - multi-line string
const std::string banner = R"(
                    ************************************
                    *   Thread Counting Synchronization  *
                    ************************************
)";

// Mutex to protect shared data
std::mutex mtx;

// Condition variable for thread signaling
std::condition_variable cv;

// Flag to indicate completion of counting up
bool isCountingUpDone = false;

/* ------------------------------------------------------------------------------------------
      ---------------------------
     |    Function Declarations   |
      ---------------------------
------------------------------------------------------------------------------------------ */

/**
 * Counts up from 0 to maxCount.
 *
 * Handles Rules:
 *      - CON50-CPP. Do not destroy a mutex while it is locked
 *      - CON51-CPP. Ensure actively held locks are released on exceptional conditions
 *      - CON52-CPP. Prevent data races when accessing bit-fields from multiple threads
 *      - CON54-CPP. Wrap functions that can spuriously wake up in a loop
 *      - STR51-CPP: Do Not Attempt to Create a std::string from a Null Pointer
 *      - STR52-CPP: Use Valid References, Pointers, and Iterators to Reference Elements of a basic_string
 *      - ERR51-CPP: Handle All Exceptions
 *      - ERR55-CPP: Honor Exception Specifications
 *
 * @param threadName thread's name.
 * @param counter the data to be modify by the thread.
 */
void countUp(const std::string& threadName, int& counter);

/**
 * Counts down from startCount to 0.
 *
 * Handles Rules:
 *      - CON54-CPP: Wrap Functions That Can Spuriously Wake Up in a Loop
 *      - CON55-CPP: Preserve Thread Safety and Liveness When Using Condition Variables
 *      - ERR51-CPP: Handle All Exceptions
 *      - ERR55-CPP: Honor Exception Specifications
 *      - STR52-CPP: Use Valid References, Pointers, and Iterators to Reference Elements of a basic_string
 *      - STR51-CPP: Do Not Attempt to Create a std::string from a Null Pointer
 *
 * @param threadName thread's name.
 * @param counter The data to be modify by the thread.
 */
void countDown(const std::string& threadName, int& counter);

```

*Continue next page*



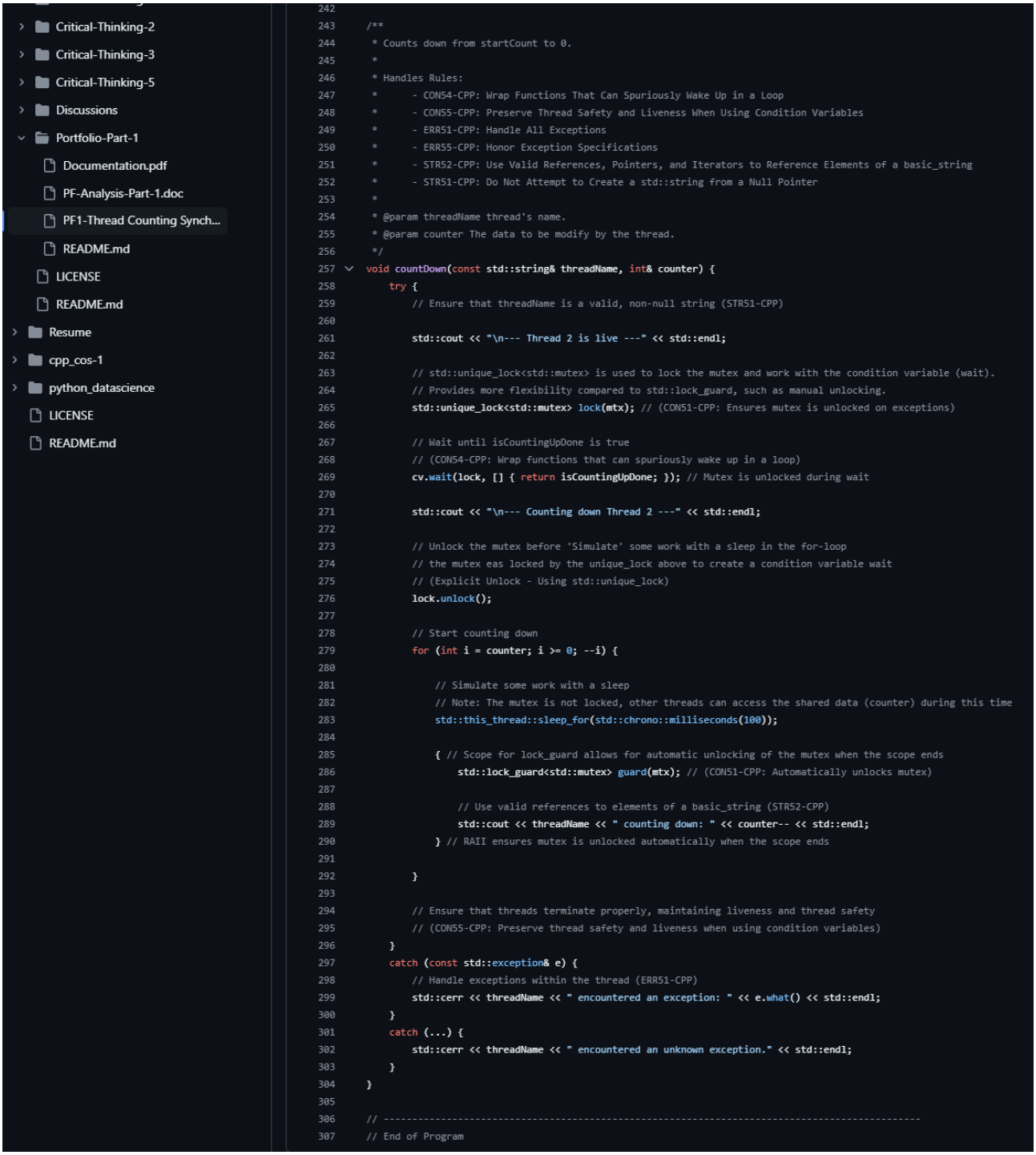

```cpp
// ============================================================================================================
/* ------------------------------------------------------------------------------------------
      --------------------
     |    Main Function   |
      --------------------

     The main function creates threads for counting up and down.

     Handles Rules:
       - CON50-CPP: Do Not Destroy a Mutex While It Is Locked
       - ERR51-CPP: Handle All Exceptions
       - ERR50-CPP: Do Not Abruptly Terminate the Program
       - ERR55-CPP: Honor Exception Specifications
       - STR50-CPP: Guarantee that storage for strings has sufficient space

------------------------------------------------------------------------------------------ */
// ============================================================================================================
int main() {
    // Define thread names for clarity in output
    // (STR50-CPP: Guarantee that storage for strings has sufficient space)
    std::string thread1Name = "Thread 1";
    std::string thread2Name = "Thread 2";

    // Maximum count for Thread 1
    int counter = -1;

    try {
        // Output banner
        std::cout << banner << std::endl;

        // Create Thread 1 (Counting Up)
        std::thread thread1(countUp, thread1Name, std::ref(counter));

        // Create Thread 2 (Counting Down)
        std::thread thread2(countDown, thread2Name, std::ref(counter));

        // Join threads to ensure they have completed execution
        // (CON50-CPP: Do not destroy a mutex while it is locked)
        thread1.join();
        thread2.join();
    }
    catch (const std::system_error& e) {
        // Ensure actively held locks are released on exceptional conditions (ERR51-CPP)
        std::cerr << "Thread system error: " << e.what() << std::endl;
        return EXIT_FAILURE; // Do not abruptly terminate the program (ERR50-CPP)
    }
    catch (const std::exception& e) {
        // Handle any other standard exceptions
        std::cerr << "Exception: " << e.what() << std::endl;
        return EXIT_FAILURE; // Do not abruptly terminate the program (ERR50-CPP)
    }
    catch (...) {
        // Catch-all handler for any other exceptions
        std::cerr << "Unknown exception occurred." << std::endl;
        return EXIT_FAILURE; // Do not abruptly terminate the program (ERR50-CPP)
    }

    // Final output indicating completion
    // (STR52-CPP: Use valid references, pointers, and iterators to reference elements of a basic_string)
    std::cout << "\nBoth threads have completed their counting without errors." << std::endl;

    return EXIT_SUCCESS;
}
```

*Continue next page*




```
167
168 // ============================================================================================================
169 /* ------------------------------------------------------------------------------------------
170         ---------------------------
171       |    Function Definitions    |
172         ---------------------------
173 ------------------------------------------------------------------------------------------ */
174 // ============================================================================================================
175
176 /**
177  * Counts up from 0 to maxCount.
178  *
179  * Handles Rules:
180  *      - CON50-CPP. Do not destroy a mutex while it is locked
181  *      - CON51-CPP. Ensure actively held locks are released on exceptional conditions
182  *      - CON54-CPP. Wrap functions that can spuriously wake up in a loop
183  *      - STR51-CPP: Do Not Attempt to Create a std::string from a Null Pointer
184  *      - STR52-CPP: Use Valid References, Pointers, and Iterators to Reference Elements of a basic_string
185  *      - ERR51-CPP: Handle All Exceptions
186  *      - ERR55-CPP: Honor Exception Specifications
187  *
188  * @param threadName thread's name.
189  * @param counter the data to be modify by the thread.
190  */
191 void countUp(const std::string& threadName, int& counter) {
192     try {
193         // Ensure that threadName is a valid, non-null string (STR51-CPP)
194
195         std::cout << "\n--- Thread 1 is live ---" << std::endl;
196
197
198         for (int i = 0; i <= 20; i++) {
199             // Simulate some work with a sleep
200                         // Note: The mutex is not locked, other threads can access the shared data (counter) during this time
201             std::this_thread::sleep_for(std::chrono::milliseconds(100));
202
203             // Using std::lock_guard ensures that the mutex is automatically released when the scope ends,
204             // even if an exception is thrown (CON51-CPP)
205             // std::lock_guard is used to to lock exactly one mutex for an entire scope
206             // (Using RAII to manage mutex unlocking automatically - Related to Unlocking a mutex via RAII)
207             { // Scope for lock_guard allows for automatic unlocking of the mutex when the scope ends
208                                 std::lock_guard<std::mutex> lock(mtx); // (CON51-CPP: Automatically unlocks mutex) (at the end of the scope)
209
210                 if (i == 0) {
211                     std::cout << "\n--- Counting Up Thread 1 ---" << std::endl;
212                 }
213
214                 // Use valid references to elements of a basic_string (STR52-CPP)
215                 std::cout << threadName << " counting up: " << ++counter << std::endl;
216             } // RAII ensures mutex is unlocked automatically when the scope ends
217         }
218
219         // After counting up is done, set the flag
220         { // Lock mutex before modifying shared flag to prevent data races (CON52-CPP)
221             std::lock_guard<std::mutex> lock(mtx); // (CON51-CPP)
222             isCountingUpDone = true;
223         }
224
225         // Notify one waiting thread to start counting down
226         // (CON54-CPP: Wrap functions that can spuriously wake up in a loop)
227         cv.notify_one();
228
229         // (CON50-CPP: Do not destroy a mutex while it is locked)
230         // The mutex mtx remains valid, it is declared globally and threads are properly joined before the program is terminated.
231     }
232     catch (const std::exception& e) {
233         // Handle exceptions within the thread (ERR51-CPP)
234         std::cerr << threadName << " encountered an exception: " << e.what() << std::endl;
235     }
236     catch (...) {
237         std::cerr << threadName << " encountered an unknown exception." << std::endl;
238     }
239 }
240
241 // ------------------------------------------------------------------------------------------
```

*Continue next page*

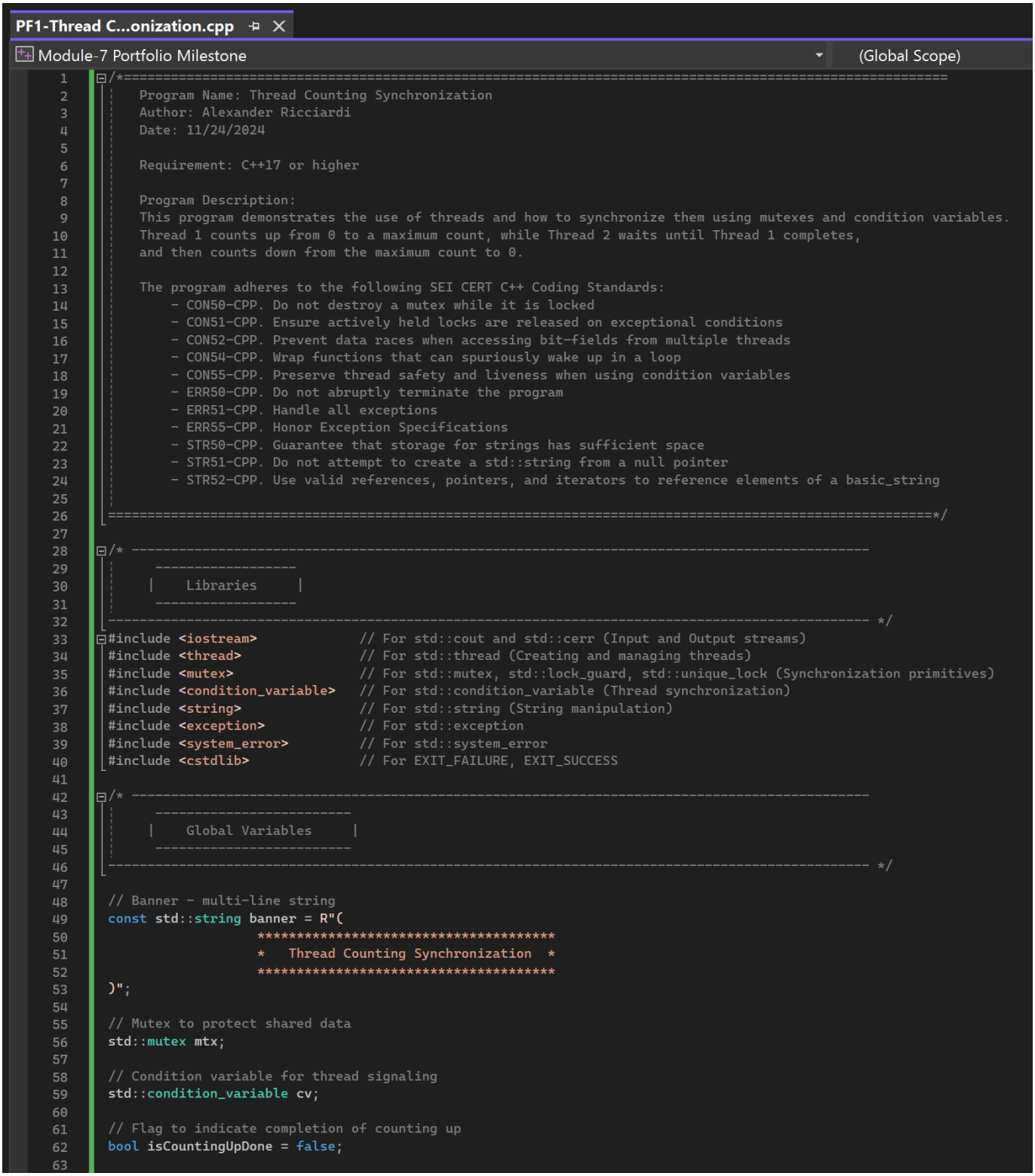



```cpp
/**
 * Counts down from startCount to 0.
 *
 * Handles Rules:
 *      - CON54-CPP: Wrap Functions That Can Spuriously Wake Up in a Loop
 *      - CON55-CPP: Preserve Thread Safety and Liveness When Using Condition Variables
 *      - ERR51-CPP: Handle All Exceptions
 *      - ERR55-CPP: Honor Exception Specifications
 *      - STR52-CPP: Use Valid References, Pointers, and Iterators to Reference Elements of a basic_string
 *      - STR51-CPP: Do Not Attempt to Create a std::string from a Null Pointer
 *
 * @param threadName thread's name.
 * @param counter The data to be modify by the thread.
 */
void countDown(const std::string& threadName, int& counter) {
    try {
        // Ensure that threadName is a valid, non-null string (STR51-CPP)

        std::cout << "\n--- Thread 2 is live ---" << std::endl;

        // std::unique_lock<std::mutex> is used to lock the mutex and work with the condition variable (wait).
        // Provides more flexibility compared to std::lock_guard, such as manual unlocking.
        std::unique_lock<std::mutex> lock(mtx); // (CON51-CPP: Ensures mutex is unlocked on exceptions)

        // Wait until isCountingUpDone is true
        // (CON54-CPP: Wrap functions that can spuriously wake up in a loop)
        cv.wait(lock, [] { return isCountingUpDone; }); // Mutex is unlocked during wait

        std::cout << "\n--- Counting down Thread 2 ---" << std::endl;

        // Unlock the mutex before 'Simulate' some work with a sleep in the for-loop
        // the mutex eas locked by the unique_lock above to create a condition variable wait
        // (Explicit Unlock - Using std::unique_lock)
        lock.unlock();

        // Start counting down
        for (int i = counter; i >= 0; --i) {

            // Simulate some work with a sleep
            // Note: The mutex is not locked, other threads can access the shared data (counter) during this time
            std::this_thread::sleep_for(std::chrono::milliseconds(100));

            { // Scope for lock_guard allows for automatic unlocking of the mutex when the scope ends
                std::lock_guard<std::mutex> guard(mtx); // (CON51-CPP: Automatically unlocks mutex)

                // Use valid references to elements of a basic_string (STR52-CPP)
                std::cout << threadName << " counting down: " << counter-- << std::endl;
            } // RAII ensures mutex is unlocked automatically when the scope ends

        }

        // Ensure that threads terminate properly, maintaining liveness and thread safety
        // (CON55-CPP: Preserve thread safety and liveness when using condition variables)
    }
    catch (const std::exception& e) {
        // Handle exceptions within the thread (ERR51-CPP)
        std::cerr << threadName << " encountered an exception: " << e.what() << std::endl;
    }
    catch (...) {
        std::cerr << threadName << " encountered an unknown exception." << std::endl;
    }
}

// ----------------------------------------------------------------------------------------
// End of Program
```

*Next page Source Code in IDE*

**Figure 2**

*Source Code in IDE*




*Continue next page*

```
64  /* ------------------------------------------------------------------------------------------------
65          ---------------------------
66      |    Function Declarations    |
67          ---------------------------
68  ------------------------------------------------------------------------------------------------ */
69
70  /**
71   * Counts up from 0 to maxCount.
72   *
73   * Handles Rules:
74   *      - CON50-CPP. Do not destroy a mutex while it is locked
75   *      - CON51-CPP. Ensure actively held locks are released on exceptional conditions
76   *      - CON52-CPP. Prevent data races when accessing bit-fields from multiple threads
77   *      - CON54-CPP. Wrap functions that can spuriously wake up in a loop
78   *      - STR51-CPP: Do Not Attempt to Create a std::string from a Null Pointer
79   *      - STR52-CPP: Use Valid References, Pointers, and Iterators to Reference Elements of a basic_string
80   *      - ERR51-CPP: Handle All Exceptions
81   *      - ERR55-CPP: Honor Exception Specifications
82   *
83   * @param threadName thread's name.
84   * @param counter the data to be modify by the thread.
85   */
86  void countUp(const std::string& threadName, int& counter);
87
88  /**
89   * Counts down from startCount to 0.
90   *
91   * Handles Rules:
92   *      - CON54-CPP: Wrap Functions That Can Spuriously Wake Up in a Loop
93   *      - CON55-CPP: Preserve Thread Safety and Liveness When Using Condition Variables
94   *      - ERR51-CPP: Handle All Exceptions
95   *      - ERR55-CPP: Honor Exception Specifications
96   *      - STR52-CPP: Use Valid References, Pointers, and Iterators to Reference Elements of a basic_string
97   *      - STR51-CPP: Do Not Attempt to Create a std::string from a Null Pointer
98   *
99   * @param threadName thread's name.
100  * @param counter The data to be modify by the thread.
101  */
102 void countDown(const std::string& threadName, int& counter);
103
```

*Continue next page*

```cpp
// ================================================================================================
/* ------------------------------------------------------------------------------------------------
     ---------------------
    |    Main Function    |
     ---------------------

    The main function creates threads for counting up and down.

    Handles Rules:
      - CON50-CPP: Do Not Destroy a Mutex While It Is Locked
      - ERR51-CPP: Handle All Exceptions
      - ERR50-CPP: Do Not Abruptly Terminate the Program
      - ERR55-CPP: Honor Exception Specifications
      - STR50-CPP: Guarantee that storage for strings has sufficient space

------------------------------------------------------------------------------------------------ */
// ================================================================================================
int main() {
    // Define thread names for clarity in output
    // (STR50-CPP: Guarantee that storage for strings has sufficient space)
    std::string thread1Name = "Thread 1";
    std::string thread2Name = "Thread 2";

    // Maximum count for Thread 1
    int counter = -1;

    try {
        // Output banner
        std::cout << banner << std::endl;

        // Create Thread 1 (Counting Up)
        std::thread thread1(countUp, thread1Name, std::ref(counter));

        // Create Thread 2 (Counting Down)
        std::thread thread2(countDown, thread2Name, std::ref(counter));

        // Join threads to ensure they have completed execution
        // (CON50-CPP: Do not destroy a mutex while it is locked)
        thread1.join();
        thread2.join();
    }
    catch (const std::system_error& e) {
        // Ensure actively held locks are released on exceptional conditions (ERR51-CPP)
        std::cerr << "Thread system error: " << e.what() << std::endl;
        return EXIT_FAILURE; // Do not abruptly terminate the program (ERR50-CPP)
    }
    catch (const std::exception& e) {
        // Handle any other standard exceptions
        std::cerr << "Exception: " << e.what() << std::endl;
        return EXIT_FAILURE; // Do not abruptly terminate the program (ERR50-CPP)
    }
    catch (...) {
        // Catch-all handler for any other exceptions
        std::cerr << "Unknown exception occurred." << std::endl;
        return EXIT_FAILURE; // Do not abruptly terminate the program (ERR50-CPP)
    }

    // Final output indicating completion
    // (STR52-CPP: Use valid references, pointers, and iterators to reference elements of a basic_string)
    std::cout << "\nBoth threads have completed their counting without errors." << std::endl;

    return EXIT_SUCCESS;
}
```

*Continue next page*

```cpp

// ================================================================================================
/* ------------------------------------------------------------------------------------------------
     ----------------------------
    |    Function Definitions    |
     ----------------------------
------------------------------------------------------------------------------------------------ */
// ================================================================================================

/**
* Counts up from 0 to maxCount.
*
* Handles Rules:
*      - CON50-CPP. Do not destroy a mutex while it is locked
*      - CON51-CPP. Ensure actively held locks are released on exceptional conditions
*      - CON54-CPP. Wrap functions that can spuriously wake up in a loop
*      - STR51-CPP: Do Not Attempt to Create a std::string from a Null Pointer
*      - STR52-CPP: Use Valid References, Pointers, and Iterators to Reference Elements of a basic_string
*      - ERR51-CPP: Handle All Exceptions
*      - ERR55-CPP: Honor Exception Specifications
*
* @param threadName thread's name.
* @param counter the data to be modify by the thread.
*/
void countUp(const std::string& threadName, int& counter) {
    try {
        // Ensure that threadName is a valid, non-null string (STR51-CPP)

        std::cout << "\n--- Thread 1 is live ---" << std::endl;


        for (int i = 0; i <= 20; i++) {
            // Simulate some work with a sleep
            // Note: The mutex is not locked, other threads can access the shared data (counter) during this time
            std::this_thread::sleep_for(std::chrono::milliseconds(100));

            // Using std::lock_guard ensures that the mutex is automatically released when the scope ends,
            // even if an exception is thrown (CON51-CPP)
            // std::lock_guard is used to to lock exactly one mutex for an entire scope
            // (Using RAII to manage mutex unlocking automatically - Related to Unlocking a mutex via RAII)
            { // Scope for lock_guard allows for automatic unlocking of the mutex when the scope ends
                std::lock_guard<std::mutex> lock(mtx); // (CON51-CPP: Automatically unlocks mutex) (at the end of the scope)

                if (i == 0) {
                    std::cout << "\n--- Counting Up Thread 1 ---" << std::endl;
                }

                // Use valid references to elements of a basic_string (STR52-CPP)
                std::cout << threadName << " counting up: " << ++counter << std::endl;
            } // RAII ensures mutex is unlocked automatically when the scope ends
        }

        // After counting up is done, set the flag
        { // Lock mutex before modifying shared flag to prevent data races (CON52-CPP)
            std::lock_guard<std::mutex> lock(mtx); // (CON51-CPP)
            isCountingUpDone = true;
        }

        // Notify one waiting thread to start counting down
        // (CON54-CPP: Wrap functions that can spuriously wake up in a loop)
        cv.notify_one();

        // (CON50-CPP: Do not destroy a mutex while it is locked)
        // The mutex mtx remains valid, it is declared globally and threads are properly joined before the program is terminated.
    }
    catch (const std::exception& e) {
        // Handle exceptions within the thread (ERR51-CPP)
        std::cerr << threadName << " encountered an exception: " << e.what() << std::endl;
    }
    catch (...) {
        std::cerr << threadName << " encountered an unknown exception." << std::endl;
    }
}
```

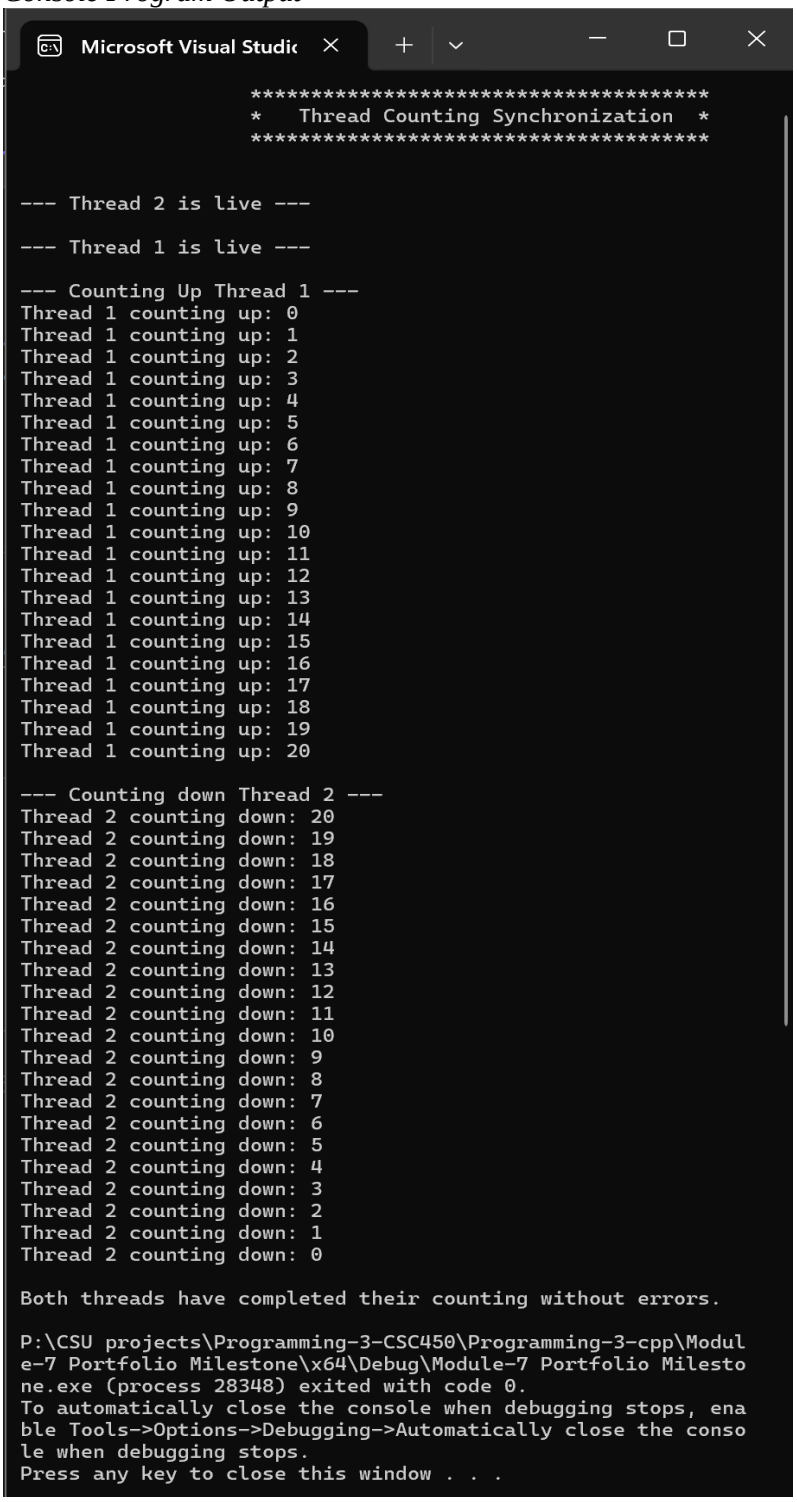
*Continue next page*

```cpp

// ------------------------------------------------------------------------------------------

/**
 * Counts down from startCount to 0.
 *
 * Handles Rules:
 *      - CON54-CPP: Wrap Functions That Can Spuriously Wake Up in a Loop
 *      - CON55-CPP: Preserve Thread Safety and Liveness When Using Condition Variables
 *      - ERR51-CPP: Handle All Exceptions
 *      - ERR55-CPP: Honor Exception Specifications
 *      - STR52-CPP: Use Valid References, Pointers, and Iterators to Reference Elements of a basic_string
 *      - STR51-CPP: Do Not Attempt to Create a std::string from a Null Pointer
 *
 * @param threadName thread's name.
 * @param counter The data to be modify by the thread.
 */
void countDown(const std::string& threadName, int& counter) {
    try {
        // Ensure that threadName is a valid, non-null string (STR51-CPP)

        std::cout << "\n--- Thread 2 is live ---" << std::endl;

        // std::unique_lock<std::mutex> is used to lock the mutex and work with the condition variable (wait).
        // Provides more flexibility compared to std::lock_guard, such as manual unlocking.
        std::unique_lock<std::mutex> lock(mtx); // (CON51-CPP: Ensures mutex is unlocked on exceptions)

        // Wait until isCountingUpDone is true
        // (CON54-CPP: Wrap functions that can spuriously wake up in a loop)
        cv.wait(lock, [] { return isCountingUpDone; }); // Mutex is unlocked during wait

        std::cout << "\n--- Counting down Thread 2 ---" << std::endl;

        // Unlock the mutex before 'Simulate' some work with a sleep in the for-loop
        // the mutex eas locked by the unique_lock above to create a condition variable wait
        // (Explicit Unlock - Using std::unique_lock)
        lock.unlock();

        // Start counting down
        for (int i = counter; i >= 0; --i) {

            // Simulate some work with a sleep
            // Note: The mutex is not locked, other threads can access the shared data (counter) during this time
            std::this_thread::sleep_for(std::chrono::milliseconds(100));

            { // Scope for lock_guard allows for automatic unlocking of the mutex when the scope ends
                std::lock_guard<std::mutex> guard(mtx); // (CON51-CPP: Automatically unlocks mutex)

                // Use valid references to elements of a basic_string (STR52-CPP)
                std::cout << threadName << " counting down: " << counter-- << std::endl;
            } // RAII ensures mutex is unlocked automatically when the scope ends

        }

        // Ensure that threads terminate properly, maintaining liveness and thread safety
        // (CON55-CPP: Preserve thread safety and liveness when using condition variables)
    }
    catch (const std::exception& e) {
        // Handle exceptions within the thread (ERR51-CPP)
        std::cerr << threadName << " encountered an exception: " << e.what() << std::endl;
    }
    catch (...) {
        std::cerr << threadName << " encountered an unknown exception." << std::endl;
    }
}

// ------------------------------------------------------------------------------------------
// End of Program
```

*Next page Console output*

**Figure 3**

*Console Program Output*




As shown in Figure 3 the program runs without any issues displaying the correct outputs as expected.