

## **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>

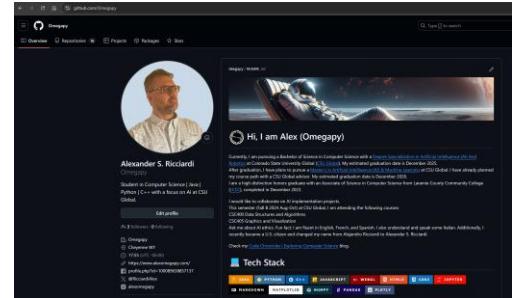
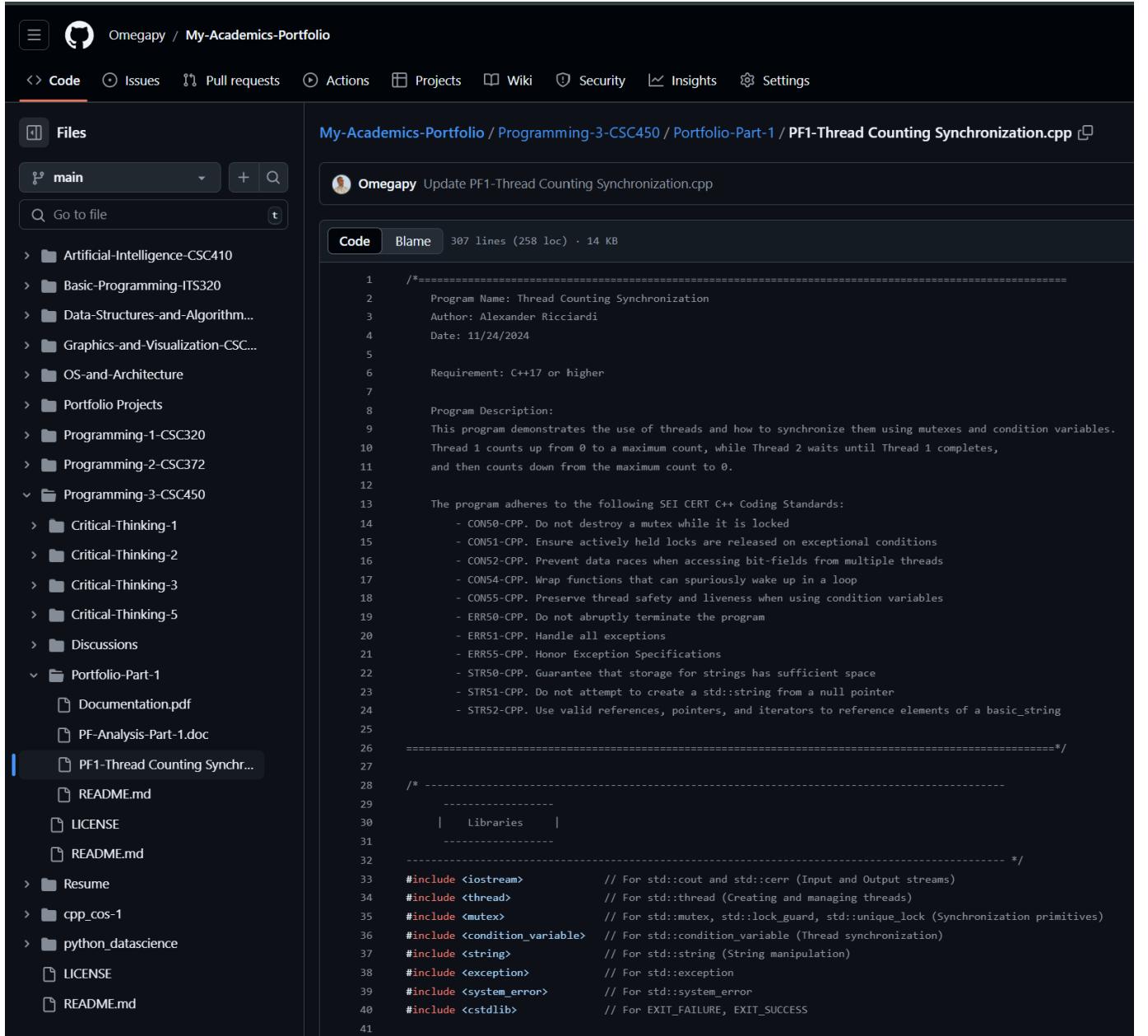


Image of the source code in the GitHub: see next page

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



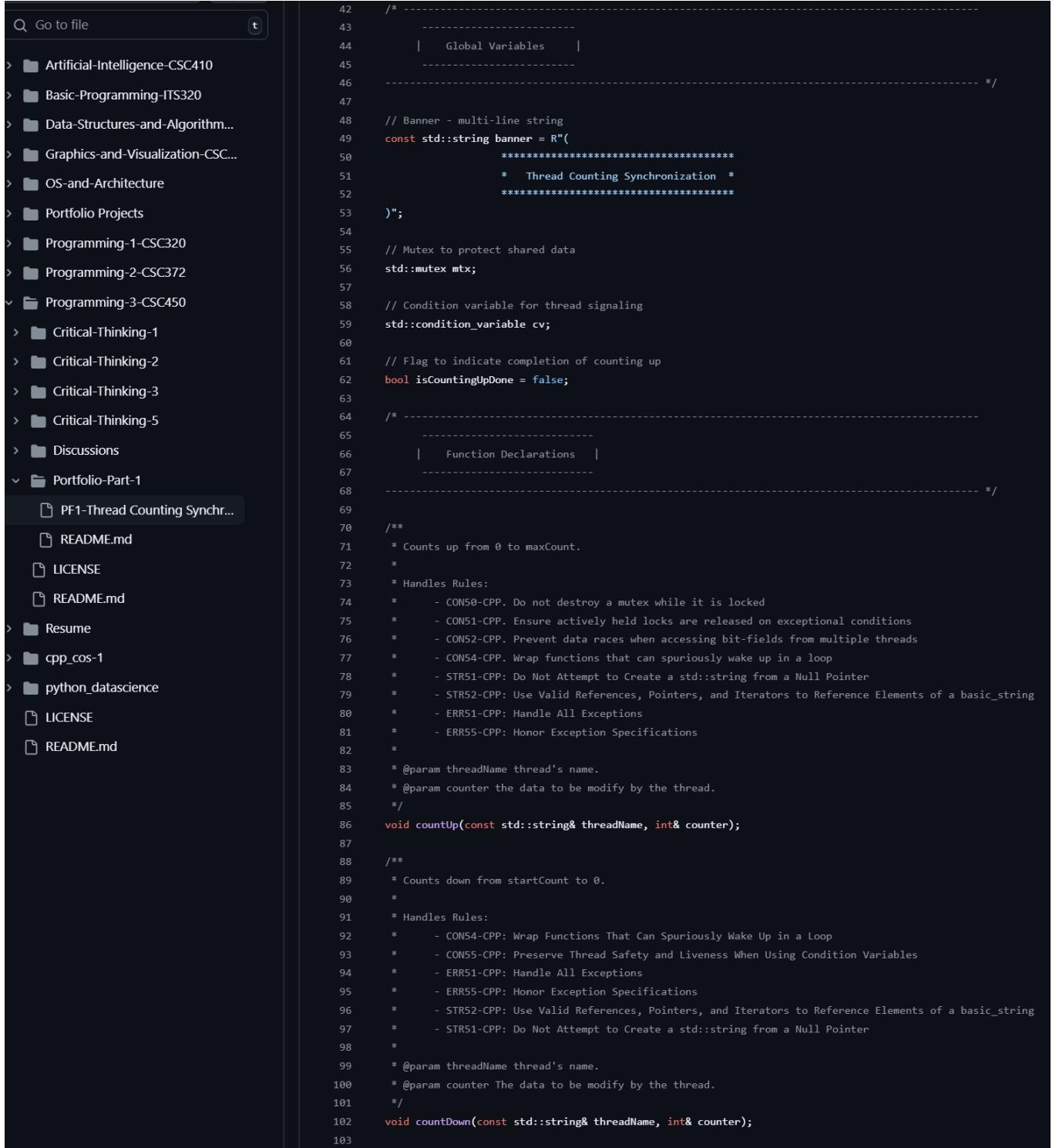
The screenshot shows a GitHub repository interface. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The main area is titled "My-Academics-Portfolio / Programming-3-CSC450 / Portfolio-Part-1 / PF1-Thread Counting Synchronization.cpp". A file list on the left shows various projects and files, with "PF1-Thread Counting Synchronization.cpp" selected. The right panel displays the code content.

```

1  //=====================================================================
2  // Program Name: Thread Counting Synchronization
3  // Author: Alexander Ricciardi
4  // Date: 11/24/2024
5
6  Requirement: C++17 or higher
7
8  Program Description:
9  This program demonstrates the use of threads and how to synchronize them using mutexes and condition variables.
10 Thread 1 counts up from 0 to a maximum count, while Thread 2 waits until Thread 1 completes,
11 and then counts down from the maximum count to 0.
12
13 The program adheres to the following SEI CERT C++ Coding Standards:
14 - CON50-CPP. Do not destroy a mutex while it is locked
15 - CON51-CPP. Ensure actively held locks are released on exceptional conditions
16 - CON52-CPP. Prevent data races when accessing bit-fields from multiple threads
17 - CON54-CPP. Wrap functions that can spuriously wake up in a loop
18 - CON55-CPP. Preserve thread safety and liveness when using condition variables
19 - ERR50-CPP. Do not abruptly terminate the program
20 - ERR51-CPP. Handle all exceptions
21 - ERR55-CPP. Honor Exception Specifications
22 - STR50-CPP. Guarantee that storage for strings has sufficient space
23 - STR51-CPP. Do not attempt to create a std::string from a null pointer
24 - STR52-CPP. Use valid references, pointers, and iterators to reference elements of a basic_string
25
26 =====*/
27
28 /*
29   -----
30   | Libraries |
31   -----
32   ----- */
33 #include <iostream>           // For std::cout and std::cerr (Input and Output streams)
34 #include <thread>             // For std::thread (Creating and managing threads)
35 #include <mutex>              // For std::mutex, std::lock_guard, std::unique_lock (Synchronization primitives)
36 #include <condition_variable> // For std::condition_variable (Thread synchronization)
37 #include <string>              // For std::string (String manipulation)
38 #include <exception>           // For std::exception
39 #include <system_error>        // For std::system_error
40 #include <cstdlib>             // For EXIT_FAILURE, EXIT_SUCCESS
41

```

*Continue next page*



The screenshot shows a file browser interface on the left and a code editor on the right. The file browser lists various projects and files, including Artificial-Intelligence-CSC410, Basic-Programming-ITS320, Data-Structures-and-Algorithm..., Graphics-and-Visualization-CSC..., OS-and-Architecture, Portfolio Projects, Programming-1-CSC320, Programming-2-CSC372, Programming-3-CSC450 (which contains Critical-Thinking-1 through 5, Discussions, and Portfolio-Part-1), and CPP and Python projects like cpp\_cos-1 and python\_datascience. The code editor displays a C++ program with comments and code related to thread synchronization and condition variables.

```

42  /* -----
43  -----|
44  |   Global Variables   |
45  -----|
46  -----*/
47
48 // Banner - multi-line string
49 const std::string banner = R"(
50 ***** * Thread Counting Synchronization * *****
51 ***** * ***** ***** ***** ***** ***** ***** *
52 )";
53
54
55 // Mutex to protect shared data
56 std::mutex mtx;
57
58 // Condition variable for thread signaling
59 std::condition_variable cv;
60
61 // Flag to indicate completion of counting up
62 bool isCountingUpDone = false;
63
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*

```

104    // =====
105    /* -----
106     -----
107     | Main Function |
108     -----
109
110     The main function creates threads for counting up and down.
111
112     Handles Rules:
113     - CON50-CPP: Do Not Destroy a Mutex While It Is Locked
114     - ERR51-CPP: Handle All Exceptions
115     - ERR50-CPP: Do Not Abruptly Terminate the Program
116     - ERR55-CPP: Honor Exception Specifications
117     - STR50-CPP: Guarantee that storage for strings has sufficient space
118
119     -----
120     // ===== */
121     int main() {
122         // Define thread names for clarity in output
123         // (STR50-CPP: Guarantee that storage for strings has sufficient space)
124         std::string thread1Name = "Thread 1";
125         std::string thread2Name = "Thread 2";
126
127         // Maximum count for Thread 1
128         int counter = -1;
129
130         try {
131             // Output banner
132             std::cout << banner << std::endl;
133
134             // Create Thread 1 (Counting Up)
135             std::thread thread1(countUp, thread1Name, std::ref(counter));
136
137             // Create Thread 2 (Counting Down)
138             std::thread thread2(countDown, thread2Name, std::ref(counter));
139
140             // Join threads to ensure they have completed execution
141             // (CON50-CPP: Do not destroy a mutex while it is locked)
142             thread1.join();
143             thread2.join();
144         }
145         catch (const std::system_error& e) {
146             // Ensure actively held locks are released on exceptional conditions (ERR51-CPP)
147             std::cerr << "Thread system error: " << e.what() << std::endl;
148             return EXIT_FAILURE; // Do not abruptly terminate the program (ERR50-CPP)
149         }
150         catch (const std::exception& e) {
151             // Handle any other standard exceptions
152             std::cerr << "Exception: " << e.what() << std::endl;
153             return EXIT_FAILURE; // Do not abruptly terminate the program (ERR50-CPP)
154         }
155         catch (...) {
156             // Catch-all handler for any other exceptions
157             std::cerr << "Unknown exception occurred." << std::endl;
158             return EXIT_FAILURE; // Do not abruptly terminate the program (ERR50-CPP)
159         }
160
161         // Final output indicating completion
162         // (STR52-CPP: Use valid references, pointers, and iterators to reference elements of a basic_string)
163         std::cout << "\nBoth threads have completed their counting without errors." << std::endl;
164
165         return EXIT_SUCCESS;
166     }

```

*Continue next page*

```

167 // =====
168 /* -----
169 |   Function Definitions   |
170 ----- */
171 // -----
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         std::cout << "\n--- Thread 1 is live ---" << std::endl;
195
196         for (int i = 0; i <= 20; i++) {
197             // Simulate some work with a sleep
198             // Note: The mutex is not locked, other threads can access the shared data (counter) during this time
199             std::this_thread::sleep_for(std::chrono::milliseconds(100));
200
201             // Using std::lock_guard ensures that the mutex is automatically released when the scope ends,
202             // even if an exception is thrown (CON51-CPP)
203             // std::lock_guard is used to lock exactly one mutex for an entire scope
204             // (Using RAII to manage mutex unlocking automatically - Related to Unlocking a mutex via RAII)
205             // Scope for lock_guard allows for automatic unlocking of the mutex when the scope ends
206             { // Scope for lock_guard
207                 std::lock_guard<std::mutex> lock(mtx); // (CON51-CPP: Automatically unlocks mutex) (at the end of the scope)
208
209                 if (i == 0) {
210                     std::cout << "\n--- Counting Up Thread 1 ---" << std::endl;
211                 }
212
213                 // Use valid references to elements of a basic_string (STR52-CPP)
214                 std::cout << threadName << " counting up: " << ++counter << std::endl;
215             } // RAII ensures mutex is unlocked automatically when the scope ends
216         }
217
218         // After counting up is done, set the flag
219         { // Lock mutex before modifying shared flag to prevent data races (CON52-CPP)
220             std::lock_guard<std::mutex> lock(mtx); // (CON51-CPP)
221             isCountingUpDone = true;
222         }
223
224         // Notify one waiting thread to start counting down
225         // (CON54-CPP: Wrap functions that can spuriously wake up in a loop)
226         cv.notify_one();
227
228         // (CON50-CPP: Do not destroy a mutex while it is locked)
229         // The mutex mtx remains valid, it is declared globally and threads are properly joined before the program is terminated.
230     }
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 // -----

```

*Continue next page*

```

242
243     /**
244      * Counts down from startCount to 0.
245      *
246      * Handles Rules:
247      *   - CON54-CPP: Wrap Functions That Can Spuriously Wake Up in a Loop
248      *   - CON55-CPP: Preserve Thread Safety and Liveness When Using Condition Variables
249      *   - ERR51-CPP: Handle All Exceptions
250      *   - ERR55-CPP: Honor Exception Specifications
251      *   - STR52-CPP: Use Valid References, Pointers, and Iterators to Reference Elements of a basic_string
252      *   - STR51-CPP: Do Not Attempt to Create a std::string from a Null Pointer
253      *
254      * @param threadName thread's name.
255      * @param counter The data to be modify by the thread.
256      */
257     void countDown(const std::string& threadName, int& counter) {
258         try {
259             // Ensure that threadName is a valid, non-null string (STR51-CPP)
260
261             std::cout << "\n--- Thread 2 is live ---" << std::endl;
262
263             // std::unique_lock<std::mutex> is used to lock the mutex and work with the condition variable (wait).
264             // Provides more flexibility compared to std::lock_guard, such as manual unlocking.
265             std::unique_lock<std::mutex> lock(mtx); // (CON51-CPP: Ensures mutex is unlocked on exceptions)
266
267             // Wait until isCountingUpDone is true
268             // (CON54-CPP: Wrap functions that can spuriously wake up in a loop)
269             cv.wait(lock, [] { return isCountingUpDone; }); // Mutex is unlocked during wait
270
271             std::cout << "\n--- Counting down Thread 2 ---" << std::endl;
272
273             // Unlock the mutex before 'Simulate' some work with a sleep in the for-loop
274             // the mutex eas locked by the unique_lock above to create a condition variable wait
275             // (Explicit Unlock - Using std::unique_lock)
276             lock.unlock();
277
278             // Start counting down
279             for (int i = counter; i >= 0; --i) {
280
281                 // Simulate some work with a sleep
282                 // Note: The mutex is not locked, other threads can access the shared data (counter) during this time
283                 std::this_thread::sleep_for(std::chrono::milliseconds(100));
284
285                 // Scope for lock_guard allows for automatic unlocking of the mutex when the scope ends
286                 std::lock_guard<std::mutex> guard(mtx); // (CON51-CPP: Automatically unlocks mutex)
287
288                 // Use valid references to elements of a basic_string (STR52-CPP)
289                 std::cout << threadName << " counting down: " << counter-- << std::endl;
290             } // RAII ensures mutex is unlocked automatically when the scope ends
291
292         }
293
294         // Ensure that threads terminate properly, maintaining liveness and thread safety
295         // (CON55-CPP: Preserve thread safety and liveness when using condition variables)
296     }
297     catch (const std::exception& e) {
298         // Handle exceptions within the thread (ERR51-CPP)
299         std::cerr << threadName << " encountered an exception: " << e.what() << std::endl;
300     }
301     catch (...) {
302         std::cerr << threadName << " encountered an unknown exception." << std::endl;
303     }
304 }
305
306 // -----
307 // End of Program

```

*Next page Source Code in IDE*

**Figure 2**  
Source Code in IDE

```

PF1-Thread C...onization.cpp ✘ X
Module-7 Portfolio Milestone (Global Scope)

1 //=====
2 | Program Name: Thread Counting Synchronization
3 | Author: Alexander Ricciardi
4 | Date: 11/24/2024
5 |
6 | Requirement: C++17 or higher
7 |
8 | Program Description:
9 | This program demonstrates the use of threads and how to synchronize them using mutexes and condition variables.
10 | Thread 1 counts up from 0 to a maximum count, while Thread 2 waits until Thread 1 completes,
11 | and then counts down from the maximum count to 0.
12 |
13 | The program adheres to the following SEI CERT C++ Coding Standards:
14 | - CON50-CPP. Do not destroy a mutex while it is locked
15 | - CON51-CPP. Ensure actively held locks are released on exceptional conditions
16 | - CON52-CPP. Prevent data races when accessing bit-fields from multiple threads
17 | - CON54-CPP. Wrap functions that can spuriously wake up in a loop
18 | - CON55-CPP. Preserve thread safety and liveness when using condition variables
19 | - ERR50-CPP. Do not abruptly terminate the program
20 | - ERR51-CPP. Handle all exceptions
21 | - ERR55-CPP. Honor Exception Specifications
22 | - STR50-CPP. Guarantee that storage for strings has sufficient space
23 | - STR51-CPP. Do not attempt to create a std::string from a null pointer
24 | - STR52-CPP. Use valid references, pointers, and iterators to reference elements of a basic_string
25 |
26 =====*/
27
28 /* -----
29 | Libraries   |
30 -----
31 */
32 // #include <iostream>           // For std::cout and std::cerr (Input and Output streams)
33 // #include <thread>            // For std::thread (Creating and managing threads)
34 // #include <mutex>             // For std::mutex, std::lock_guard, std::unique_lock (Synchronization primitives)
35 // #include <condition_variable> // For std::condition_variable (Thread synchronization)
36 // #include <string>             // For std::string (String manipulation)
37 // #include <exception>          // For std::exception
38 // #include <system_error>        // For std::system_error
39 // #include <cstdlib>            // For EXIT_FAILURE, EXIT_SUCCESS
40
41 /* -----
42 | Global Variables   |
43 -----
44 */
45
46 // Banner - multi-line string
47 const std::string banner = R"( ****
48 // ***** * Thread Counting Synchronization * ****
49 )";
50
51 // Mutex to protect shared data
52 std::mutex mtx;
53
54 // Condition variable for thread signaling
55 std::condition_variable cv;
56
57 // Flag to indicate completion of counting up
58 bool isCountingUpDone = false;
59
60
61
62
63

```

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*

```

104 // =====
105 /* -----
106 |   Main Function   |
107 -----
108
109     The main function creates threads for counting up and down.
110
111     Handles Rules:
112     - CON50-CPP: Do Not Destroy a Mutex While It Is Locked
113     - ERR51-CPP: Handle All Exceptions
114     - ERR50-CPP: Do Not Abruptly Terminate the Program
115     - ERR55-CPP: Honor Exception Specifications
116     - STR50-CPP: Guarantee that storage for strings has sufficient space
117
118 -----
119 */ // =====
120
121 int main() {
122     // Define thread names for clarity in output
123     // (STR50-CPP: Guarantee that storage for strings has sufficient space)
124     std::string thread1Name = "Thread 1";
125     std::string thread2Name = "Thread 2";
126
127     // Maximum count for Thread 1
128     int counter = -1;
129
130     try {
131         // Output banner
132         std::cout << banner << std::endl;
133
134         // Create Thread 1 (Counting Up)
135         std::thread thread1(countUp, thread1Name, std::ref(counter));
136
137         // Create Thread 2 (Counting Down)
138         std::thread thread2(countDown, thread2Name, std::ref(counter));
139
140         // Join threads to ensure they have completed execution
141         // (CON50-CPP: Do not destroy a mutex while it is locked)
142         thread1.join();
143         thread2.join();
144     }
145
146     catch (const std::system_error& e) {
147         // Ensure auxiliary held locks are released on exceptional conditions (ERR51-CPP)
148         std::cerr << "Thread system error: " << e.what() << std::endl;
149         return EXIT_FAILURE; // Do not abruptly terminate the program (ERR50-CPP)
150     }
151     catch (const std::exception& e) {
152         // Handle any other standard exceptions
153         std::cerr << "Exception: " << e.what() << std::endl;
154         return EXIT_FAILURE; // Do not abruptly terminate the program (ERR50-CPP)
155     }
156     catch (...) {
157         // Catch-all handler for any other exceptions
158         std::cerr << "Unknown exception occurred." << std::endl;
159         return EXIT_FAILURE; // Do not abruptly terminate the program (ERR50-CPP)
160     }
161
162     // Final output indicating completion
163     // (STR52-CPP: Use valid references, pointers, and iterators to reference elements of a basic_string)
164     std::cout << "\nBoth threads have completed their counting without errors." << std::endl;
165
166     return EXIT_SUCCESS;
}

```

*Continue next page*

```

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

```

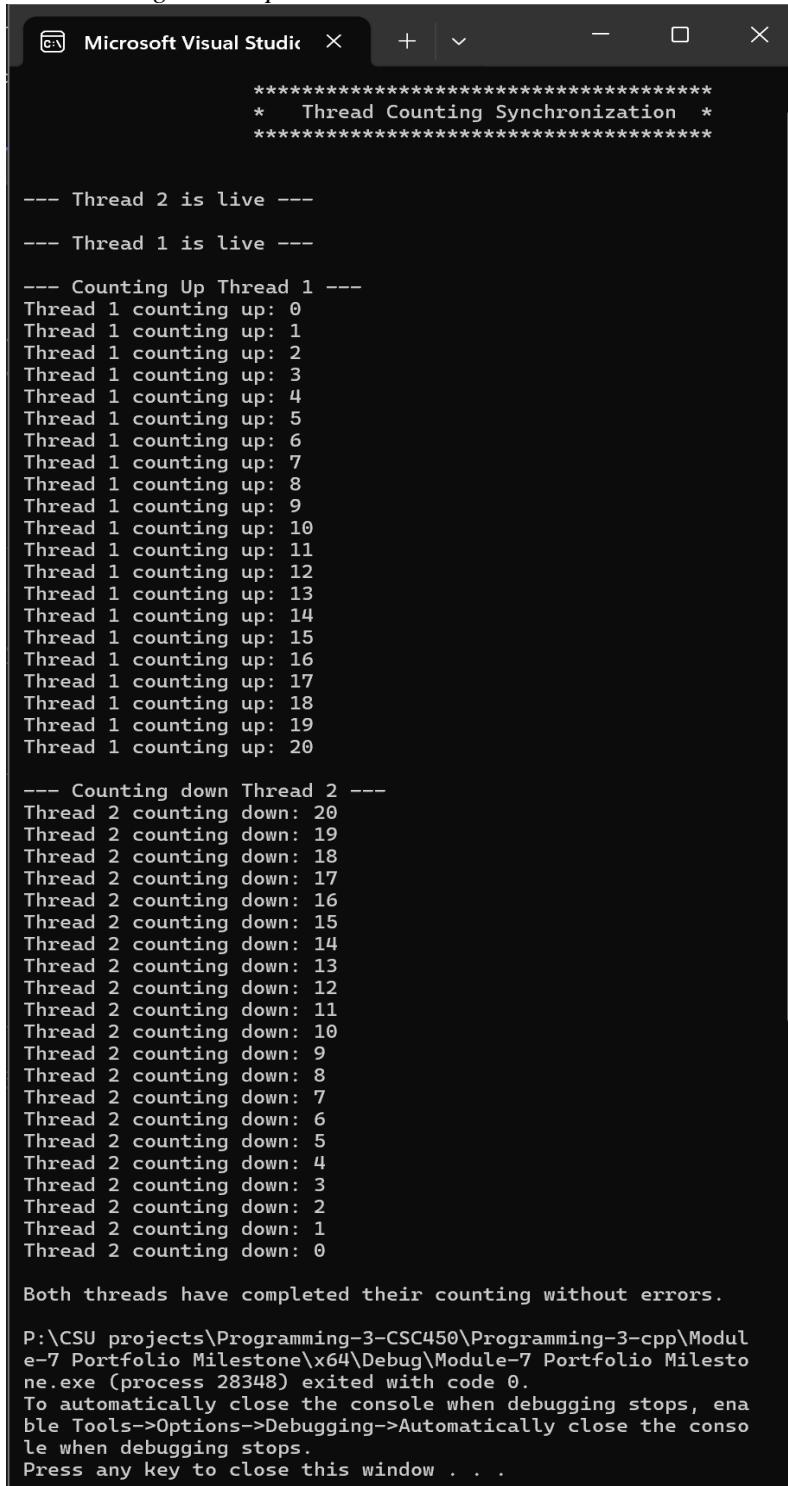
*Continue next page*

```

240 // -----
241
242 /**
243 * Counts down from startCount to 0.
244 *
245 * Handles Rules:
246 *   - CON54-CPP: Wrap Functions That Can Spuriously Wake Up in a Loop
247 *   - CON55-CPP: Preserve Thread Safety and Liveness When Using Condition Variables
248 *   - ERR51-CPP: Handle All Exceptions
249 *   - ERR55-CPP: Honor Exception Specifications
250 *   - STR52-CPP: Use Valid References, Pointers, and Iterators to Reference Elements of a basic_string
251 *   - STR51-CPP: Do Not Attempt to Create a std::string from a Null Pointer
252 *
253 * @param threadName thread's name.
254 * @param counter The data to be modify by the thread.
255 */
256 void countDown(const std::string& threadName, int& counter) {
257     try {
258         // Ensure that threadName is a valid, non-null string (STR51-CPP)
259
260         std::cout << "\n--- Thread 2 is live ---" << std::endl;
261
262         // std::unique_lock<std::mutex> is used to lock the mutex and work with the condition variable (wait).
263         // Provides more flexibility compared to std::lock_guard, such as manual unlocking.
264         std::unique_lock<std::mutex> lock(mtx); // (CON51-CPP: Ensures mutex is unlocked on exceptions)
265
266         // Wait until isCountingUpDone is true
267         // (CON54-CPP: Wrap functions that can spuriously wake up in a loop)
268         cv.wait(lock, [] { return isCountingUpDone; }); // Mutex is unlocked during wait
269
270         std::cout << "\n--- Counting down Thread 2 ---" << std::endl;
271
272         // Unlock the mutex before 'Simulate' some work with a sleep in the for-loop
273         // the mutex eas locked by the unique_lock above to create a condition variable wait
274         // (Explicit Unlock - Using std::unique_lock)
275         lock.unlock();
276
277         // Start counting down
278         for (int i = counter; i >= 0; --i) {
279
280             // Simulate some work with a sleep
281             // Note: The mutex is not locked, other threads can access the shared data (counter) during this time
282             std::this_thread::sleep_for(std::chrono::milliseconds(100));
283
284             { // Scope for lock_guard allows for automatic unlocking of the mutex when the scope ends
285                 std::lock_guard<std::mutex> guard(mtx); // (CON51-CPP: Automatically unlocks mutex)
286
287                 // Use valid references to elements of a basic_string (STR52-CPP)
288                 std::cout << threadName << " counting down: " << counter-- << std::endl;
289             } // RAI ensures mutex is unlocked automatically when the scope ends
290
291         }
292
293         // Ensure that threads terminate properly, maintaining liveness and thread safety
294         // (CON55-CPP: Preserve thread safety and liveness when using condition variables)
295     }
296
297     catch (const std::exception& e) {
298         // Handle exceptions within the thread (ERR51-CPP)
299         std::cerr << threadName << " encountered an exception: " << e.what() << std::endl;
300     }
301     catch (...) {
302         std::cerr << threadName << " encountered an unknown exception." << std::endl;
303     }
304 }
305
306 // -----
307 // End of Program

```

*Next page Console output*

**Figure 3***Console Program Output*

The screenshot shows a Microsoft Visual Studio console window titled "Console Program Output". The window displays the following text:

```
*****
*   Thread Counting Synchronization *
*****  
  
--- Thread 2 is live ---  
--- Thread 1 is live ---  
  
--- Counting Up Thread 1 ---  
Thread 1 counting up: 0  
Thread 1 counting up: 1  
Thread 1 counting up: 2  
Thread 1 counting up: 3  
Thread 1 counting up: 4  
Thread 1 counting up: 5  
Thread 1 counting up: 6  
Thread 1 counting up: 7  
Thread 1 counting up: 8  
Thread 1 counting up: 9  
Thread 1 counting up: 10  
Thread 1 counting up: 11  
Thread 1 counting up: 12  
Thread 1 counting up: 13  
Thread 1 counting up: 14  
Thread 1 counting up: 15  
Thread 1 counting up: 16  
Thread 1 counting up: 17  
Thread 1 counting up: 18  
Thread 1 counting up: 19  
Thread 1 counting up: 20  
  
--- Counting down Thread 2 ---  
Thread 2 counting down: 20  
Thread 2 counting down: 19  
Thread 2 counting down: 18  
Thread 2 counting down: 17  
Thread 2 counting down: 16  
Thread 2 counting down: 15  
Thread 2 counting down: 14  
Thread 2 counting down: 13  
Thread 2 counting down: 12  
Thread 2 counting down: 11  
Thread 2 counting down: 10  
Thread 2 counting down: 9  
Thread 2 counting down: 8  
Thread 2 counting down: 7  
Thread 2 counting down: 6  
Thread 2 counting down: 5  
Thread 2 counting down: 4  
Thread 2 counting down: 3  
Thread 2 counting down: 2  
Thread 2 counting down: 1  
Thread 2 counting down: 0  
  
Both threads have completed their counting without errors.  
  
P:\CSU projects\Programming-3-CSC450\Programming-3-cpp\Module-7 Portfolio Milestone\x64\Debug\Module-7 Portfolio Milestone.exe (process 28348) exited with code 0.  
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.  
Press any key to close this window . . .
```

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