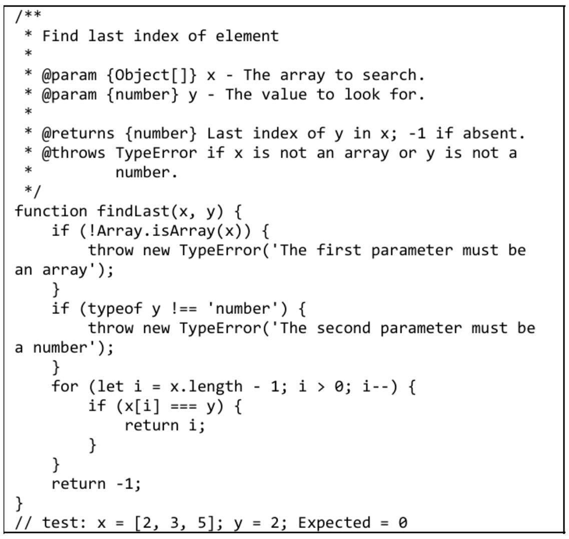
Homework 1：Faulty Programs with Faults and Failures

Student：314551102 / 曹育瑄

Part 1

Section 1：Last Index of Element



```
/**
 * Find last index of element
 *
 * @param {Object[]} x - The array to search.
 * @param {number} y - The value to look for.
 *
 * @returns {number} Last index of y in x; -1 if absent.
 * @throws TypeError if x is not an array or y is not a
 *         number.
 */
function findLast(x, y) {
    if (!Array.isArray(x)) {
        throw new TypeError('The first parameter must be
an array');
    }
    if (typeof y !== 'number') {
        throw new TypeError('The second parameter must be
a number');
    }
    for (let i = x.length - 1; i > 0; i--) {
        if (x[i] === y) {
            return i;
        }
    }
    return -1;
}
// test: x = [2, 3, 5]; y = 2; Expected = 0
```

Figure 1 The code of Last Index of Element.

(a) Explain what is wrong with the given code.
Describe the fault precisely by proposing a modification to the code.

i > 0 改為 i >= 0。

for (let i = x.length - 1; i > 0; i--)中，條件 i > 0 檢查不到 index = 0 的 element，應該改為 for (let i = x.length - 1; i >= 0; i--)，才會讀完整個 loop。

(b) If possible, give a test case that does not execute the fault.
If not, briefly explain why not. (You need to give the same number of arguments.)

test: x = [2, 3, 5]; y = "miku";

因為 y 不是 number，程式碼會在 if (typeof y !== 'number')下扔出 TypeError('The second parameter must be a number')，就不會執行到 for (let i = x.length - 1; i > 0; i–)，也就是不會執行 fault。

(c) If possible, give a test case that executes the fault, but does not result in an error state.

test: x = [2, 3, 5]; y = 3; Expected = 1, Actual = 1

會執行到 for (let i = x.length - 1; i > 0; i–)，但因為要找的元素在 index = 1，程式不需要檢查 i = 0 就會找到我們要的 index。雖然 fault 被執行（因為迴圈條件 i > 0 真的影響到檢查範圍），但因為是從最後一個 index 找，所以沒有導致 error state。

(d) If possible, give a test case that results in an error state, but not a failure.
If not, briefly explain why not. (You also need to answer expected and actual output.)

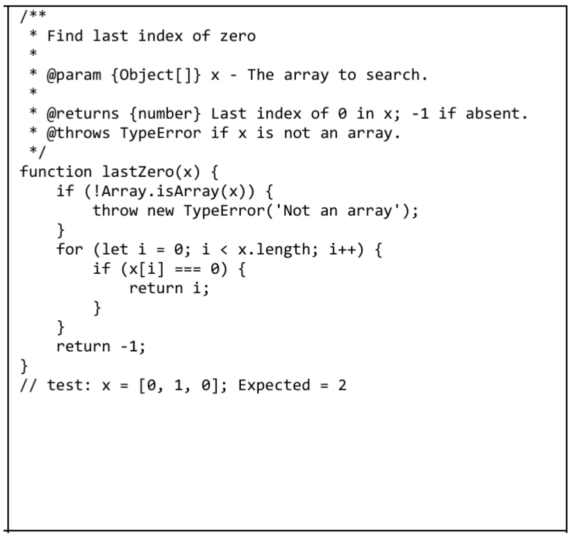
沒有一個 test case 滿足此條件，因為在這個題目的 error state 勢必要在我們要找的元素在 index = 0 的情況下才會發生，而這個 error state 發生時，就一定會導致 failure。例如，test: x = [2, 3, 5]; y = 2; Expected = 0, Actual = -1。

(e) For the given test case in (d), describe the first error state. Be sure to describe the complete state.

沒有一個 test case 滿足 results in an error state, but not a failure。

而在 d 舉例的 test case: x = [2, 3, 5]; y = 2; Expected = 0, Actual = -1。此 test case 的第一次發生 error state 是當 i 變成 0 時，因為 for-loop 沒有檢查 index = 0 的情況，會導致 index 不會變成 0，最後會因為找不到我們需要的元素 y=2 而回傳 -1，也導致 failure。

Section 2：Last Index of Zero



```
/**
 * Find last index of zero
 *
 * @param {Object[]} x - The array to search.
 *
 * @returns {number} Last index of 0 in x; -1 if absent.
 * @throws TypeError if x is not an array.
 */
function lastZero(x) {
    if (!Array.isArray(x)) {
        throw new TypeError('Not an array');
    }
    for (let i = 0; i < x.length; i++) {
        if (x[i] === 0) {
            return i;
        }
    }
    return -1;
}
// test: x = [0, 1, 0]; Expected = 2
```

Figure 2 The code of Last Index of Zero.

(a) Explain what is wrong with the given code.
Describe the fault precisely by proposing a modification to the code.

for (let i = 0; i < x.length; i++)的功能會是找到第一個 0 的 index，而不是最後的 index。應改為由後往前找，應該改為 for (let i = x.length - 1; i >= 0; i--)。

(b) If possible, give a test case that does not execute the fault.
If not, briefly explain why not. (You need to give the same number of arguments.)

test: x = "miku";

因為 x 不是 array，程式碼會在 if (!Array.isArray(x))下扔出 TypeError('Not an array')，就不會執行到 for (let i = 0; i < x.length; i++)，也就是不會執行 fault。

(c) If possible, give a test case that executes the fault, but does not result in an error state.

test: x = [3, 9, 0, 2]; Expected = 2, Actual = 2

Fault 被執行（遇到 x[2] = 0，馬上 return），但因為這個 0 是第一個 0 也是最後一個 0，所以不會導致 error state。

(d) If possible, give a test case that results in an error state, but not a failure.
If not, briefly explain why not. (You also need to answer expected and actual output.)

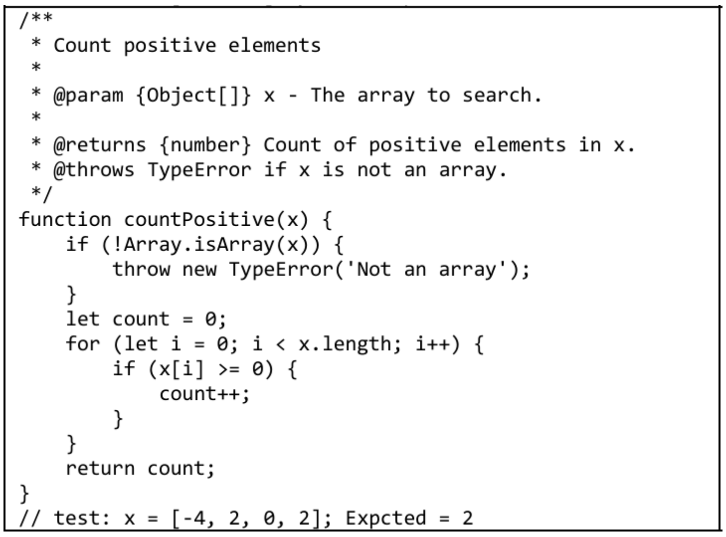
沒有一個 test case 滿足此條件，導致 error state 的情況是該元素不是陣列的最後一個 0，像是[0, 1, 0]，在 index=0 時會直接 return 0，一定會導致 failure。此題在陣列有大於等於兩個 0 時，error state 一定會發生，就會導致 failure。

(e) For the given test case in (d), describe the first error state. Be sure to describe the complete state.

沒有一個 test case 滿足 results in an error state, but not a failure。

而在 d 舉例的 test case: x = [0, 1, 0]; Expected = 2, Actual = 0。此 test case 的第一次發生 error state 是當 i 是 0 時，因為 i = 0 時，if (x[i] ===0) 成立，會直接回傳 0，導致 error state 發生，也因為他馬上回傳，導致 failure。

Section 3：Count Positive Elements



```
/**
 * Count positive elements
 *
 * @param {Object[]} x - The array to search.
 *
 * @returns {number} Count of positive elements in x.
 * @throws TypeError if x is not an array.
 */
function countPositive(x) {
    if (!Array.isArray(x)) {
        throw new TypeError('Not an array');
    }
    let count = 0;
    for (let i = 0; i < x.length; i++) {
        if (x[i] >= 0) {
            count++;
        }
    }
    return count;
}
// test: x = [-4, 2, 0, 2]; Expcted = 2
```

Figure 3 The code of Count Positive Elements.

(a) Explain what is wrong with the given code.
Describe the fault precisely by proposing a modification to the code.

for (let i = 0; i < x.length; i++)裡面的 x[i] >= 0 會包含 0，應改為 if (x[i] > 0)。

(b) If possible, give a test case that does not execute the fault.
If not, briefly explain why not. (You need to give the same number of arguments.)

test: x = "みくみくにしてあげる♪";

因為 x 不是 array，程式碼會在 if (!Array.isArray(x))下扔出 TypeError('Not an array')，就不會執行到 if (x[i] >= 0)。

(c) If possible, give a test case that executes the fault, but does not result in an error state.

test: x = [3, 9, 3, 9]; Expected = 4, Actual = 4

他會執行到 if (x[i] >= 0) { count++; }，但因為陣列裡都是正整數，所以不會導致出現 error state。

(d) If possible, give a test case that results in an error state, but not a failure.
If not, briefly explain why not. (You also need to answer expected and actual output.)

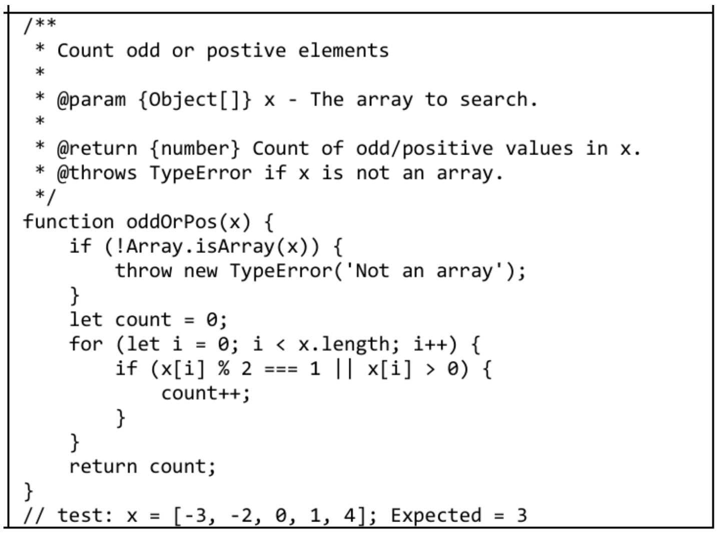
沒有一個 test case 滿足此條件，因為這個程式碼的 error state 是當陣列裡的元素是 0 時，count 會被+1，導致 count 與我們預期的不一致。此題在 error state 發生時，就一定會導致 failure。例如 test case: x = [-4, 2, 0, 2]; Expected = 2, Actual = 3，比預期的多了一。

(e) For the given test case in (d), describe the first error state. Be sure to describe the complete state.

沒有一個 test case 滿足 results in an error state, but not a failure。

而在 d 舉例的 test case: x = [-4, 2, 0, 2]; Expected = 2, Actual = 3。此 test case 的第一次發生 error state 是當 i 是 2 時，if (x[i] >= 0) 成立，count 會加 1，導致 error state 發生，因為 count 此時不該被加 1，而後續 count 會一直是錯誤的(在這個 test case 會一直多 1)，最後導致 failure。

Section 4：Count Odd or Positive Elements



```
/**
 * Count odd or postive elements
 *
 * @param {Object[]} x - The array to search.
 *
 * @return {number} Count of odd/positive values in x.
 * @throws TypeError if x is not an array.
 */
function oddOrPos(x) {
    if (!Array.isArray(x)) {
        throw new TypeError('Not an array');
    }
    let count = 0;
    for (let i = 0; i < x.length; i++) {
        if (x[i] % 2 === 1 || x[i] > 0) {
            count++;
        }
    }
    return count;
}
// test: x = [-3, -2, 0, 1, 4]; Expected = 3
```

Figure 4 The code of Count Odd or Positive Elements.

(a) Explain what is wrong with the given code.
Describe the fault precisely by proposing a modification to the code.

if (x[i] % 2 === 1 || x[i] > 0) 改為 if (x[i] % 2 !== 0 || x[i] > 0)

for (let i = 0; i < x.length; i++) 裡面的 x[i] % 2 === 1 無法處理負奇數的情況，應改為 x[i] % 2 !== 0。

(b) If possible, give a test case that does not execute the fault.
If not, briefly explain why not. (You need to give the same number of arguments.)

test: x = "みくみくにしてあげる♪";

因為 x 不是 array，程式碼會在 if (!Array.isArray(x))下扔出 TypeError('Not an array')，就不會執行到 if (x[i] % 2 === 1 || x[i] > 0)。

(c) If possible, give a test case that executes the fault, but does not result in an error state.

test: x = [3, 9, 2, 9]; Expected = 4, Actual = 4

他會執行到 if (x[i] % 2 === 1 || x[i] > 0) { count++; }，但因為陣列裡都是正整數跟奇數，所以不會導致出現 error state。

(d) If possible, give a test case that results in an error state, but not a failure.
If not, briefly explain why not. (You also need to answer expected and actual output.)

沒有一個 test case 滿足此條件，因為這個程式碼的 error state 是當陣列裡的元素有負奇數時，count 不會被+1，只要陣列有負奇數就會產生 error state，而一旦有 error state 的情況下這一題一定會導致 failure。例如，test: x = [-3, 9, 2, 9]; Expected = 4, Actual = 3。

(e) For the given test case in (d), describe the first error state. Be sure to describe the complete state.

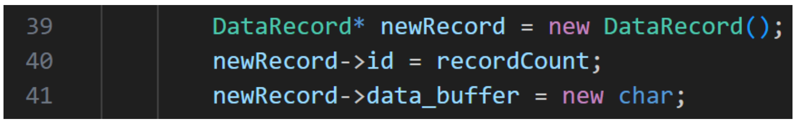
沒有一個 test case 滿足 results in an error state, but not a failure。

而在 d 舉例的 test: x = [-3, 9, 2, 9]; Expected = 4, Actual = 3。此 test case 的第一次發生 error state 是當 i 是 0 時，if (x[i] % 2 === 1 || x[i] > 0) 不成立 (-3 % 2 = -1, -3 < 0)，導致沒有算到 -3 這個負奇數，error state 發生，因為 count 此時少加 1，而後續 count 會一直是錯誤的(在這個 test case 會一直少 1)，最後導致 failure。

Part 2
Section 1：DataProcessor.cpp

Fault:

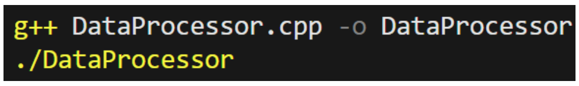
由於動態分配的 DataRecord objects 和 data_buffer 記憶體未在 processLargeFile 內釋放而造成 Memory leak。



```
39          DataRecord* newRecord = new DataRecord();
40          newRecord->id = recordCount;
41          newRecord->data_buffer = new char;
```

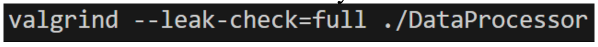
Test Case:

Run the provided main() function (which generates a data.txt file with 10,000 records and calls processLargeFile).



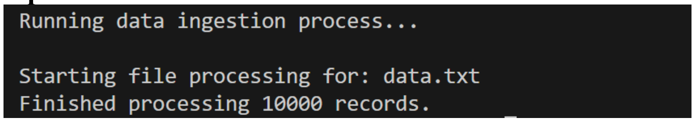
```
g++ DataProcessor.cpp -o DataProcessor
./DataProcessor
```

Then use the tool Valgrind to check if there is a memory leak.



```
valgrind --leak-check=full ./DataProcessor
```

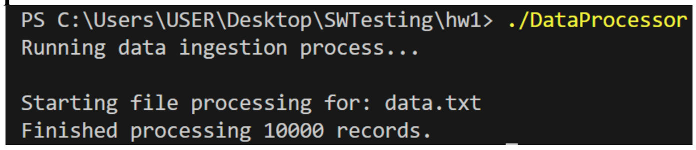
Expected Output:



```
Running data ingestion process...

Starting file processing for: data.txt
Finished processing 10000 records.
```

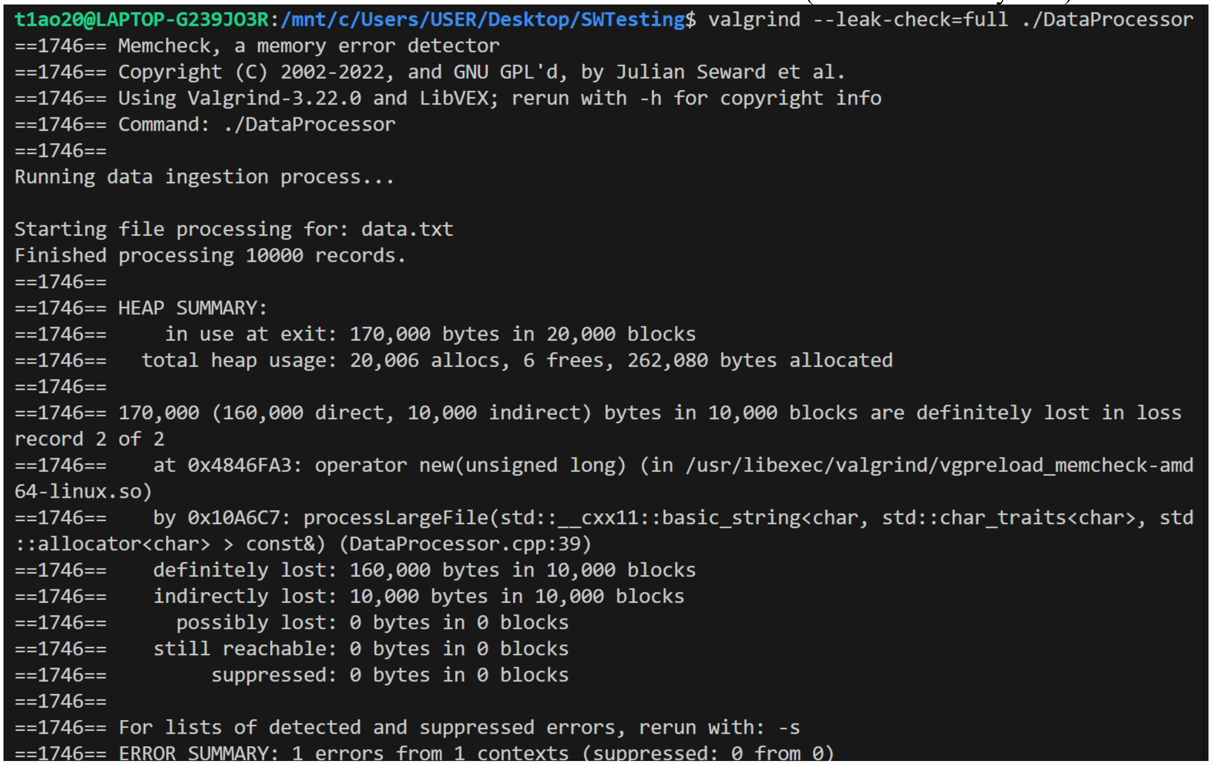
(with no memory leak)

Actual Output:



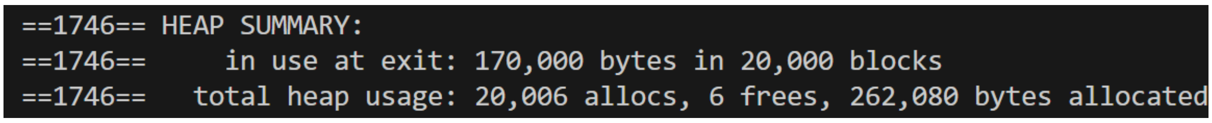
```
PS C:\Users\USER\Desktop\SWTesting\hw1> ./DataProcessor
Running data ingestion process...

Starting file processing for: data.txt
Finished processing 10000 records.
```

(but have memory leak)



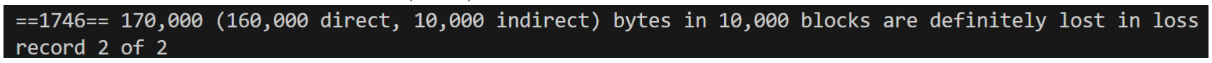
```
t1ao20@LAPTOP-G239JO3R:/mnt/c/Users/USER/Desktop/SWTesting$ valgrind --leak-check=full ./DataProcessor
==1746== Memcheck, a memory error detector
==1746== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==1746== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==1746== Command: ./DataProcessor
==1746==
Running data ingestion process...

Starting file processing for: data.txt
Finished processing 10000 records.
==1746==
==1746== HEAP SUMMARY:
==1746==     in use at exit: 170,000 bytes in 20,000 blocks
==1746==   total heap usage: 20,006 allocs, 6 frees, 262,080 bytes allocated
==1746==
==1746== 170,000 (160,000 direct, 10,000 indirect) bytes in 10,000 blocks are definitely lost in loss
record 2 of 2
==1746==    at 0x4846FA3: operator new(unsigned long) (in /usr/libexec/valgrind/vgpreload_memcheck-amd
64-linux.so)
==1746==    by 0x10A6C7: processLargeFile(std::__cxx11::basic_string<char, std::char_traits<char>, std
::allocator<char> > const&) (DataProcessor.cpp:39)
==1746==    definitely lost: 160,000 bytes in 10,000 blocks
==1746==    indirectly lost: 10,000 bytes in 10,000 blocks
==1746==      possibly lost: 0 bytes in 0 blocks
==1746==    still reachable: 0 bytes in 0 blocks
==1746==         suppressed: 0 bytes in 0 blocks
==1746==
==1746== For lists of detected and suppressed errors, rerun with: -s
==1746== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Output 說明：



```
==1746== HEAP SUMMARY:
==1746==     in use at exit: 170,000 bytes in 20,000 blocks
==1746==   total heap usage: 20,006 allocs, 6 frees, 262,080 bytes allocated
```

程式結束時還有 170,000 bytes 沒釋放。
一共做了 20,006 次記憶體配置 (alloc)，但只釋放了 6 次。



```
==1746== 170,000 (160,000 direct, 10,000 indirect) bytes in 10,000 blocks are definitely lost in loss
record 2 of 2
```

direct lost 160 KB：new DataRecord() 本身沒被 delete。
indirect lost 10 KB：每個 DataRecord 裡的 new char 沒釋放。

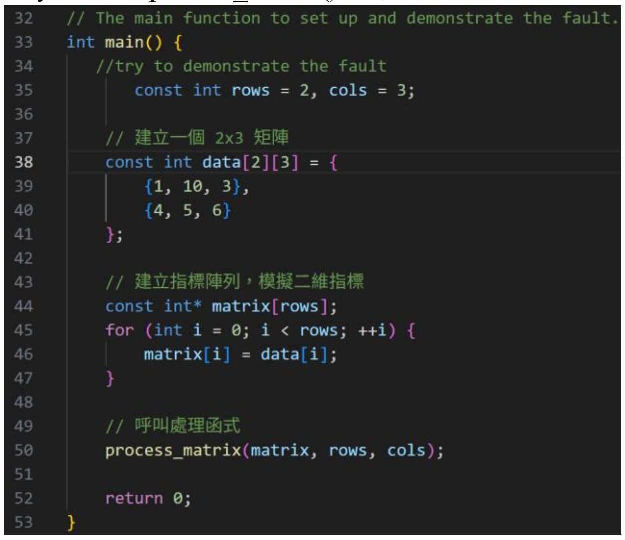
Section 2：MatrixProcessor.cpp

Fault:

在 process_matrix(const int* const* matrix, int rows, int cols); 中使用 const_cast 破壞了 matrix 變數 const 的語意，還把指標設為非法地址 999。導致在執行過程中可能導致程式 crash，輸出異常或記憶體存取錯誤。

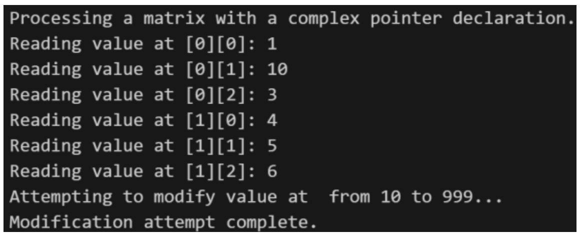
Test Case:

建立一個 const int*的 array 並呼叫 process_matrix()，即可觸發 fault。



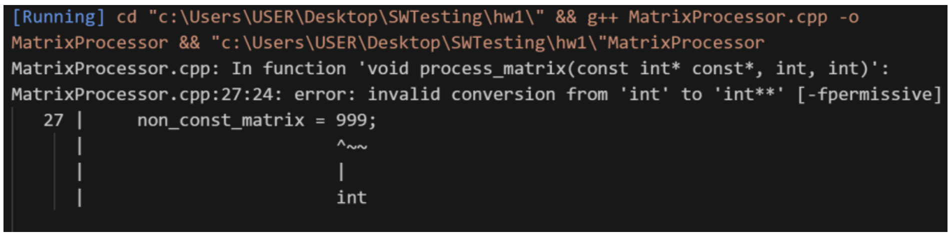
```cpp
// The main function to set up and demonstrate the fault.
int main() {
    //try to demonstrate the fault
        const int rows = 2, cols = 3;

    // 建立一個 2x3 矩陣
    const int data[2][3] = {
        {1, 10, 3},
        {4, 5, 6}
    };

    // 建立指標陣列，模擬二維指標
    const int* matrix[rows];
    for (int i = 0; i < rows; ++i) {
        matrix[i] = data[i];
    }

    // 呼叫處理函式
    process_matrix(matrix, rows, cols);

    return 0;
}
```

Expected Output:



```
Processing a matrix with a complex pointer declaration.
Reading value at [0][0]: 1
Reading value at [0][1]: 10
Reading value at [0][2]: 3
Reading value at [1][0]: 4
Reading value at [1][1]: 5
Reading value at [1][2]: 6
Attempting to modify value at  from 10 to 999...
Modification attempt complete.
```

Actual Output:



```
[Running] cd "c:\Users\USER\Desktop\SWTesting\hw1\" && g++ MatrixProcessor.cpp -o
MatrixProcessor && "c:\Users\USER\Desktop\SWTesting\hw1\"MatrixProcessor
MatrixProcessor.cpp: In function 'void process_matrix(const int* const*, int, int)':
MatrixProcessor.cpp:27:24: error: invalid conversion from 'int' to 'int**' [-fpermissive]
   27 |     non_const_matrix = 999;
      |                        ^~~
      |                        |
      |                        int
```

Section 3：ResourceScheduler.py

Fault:

Deadlock 的問題

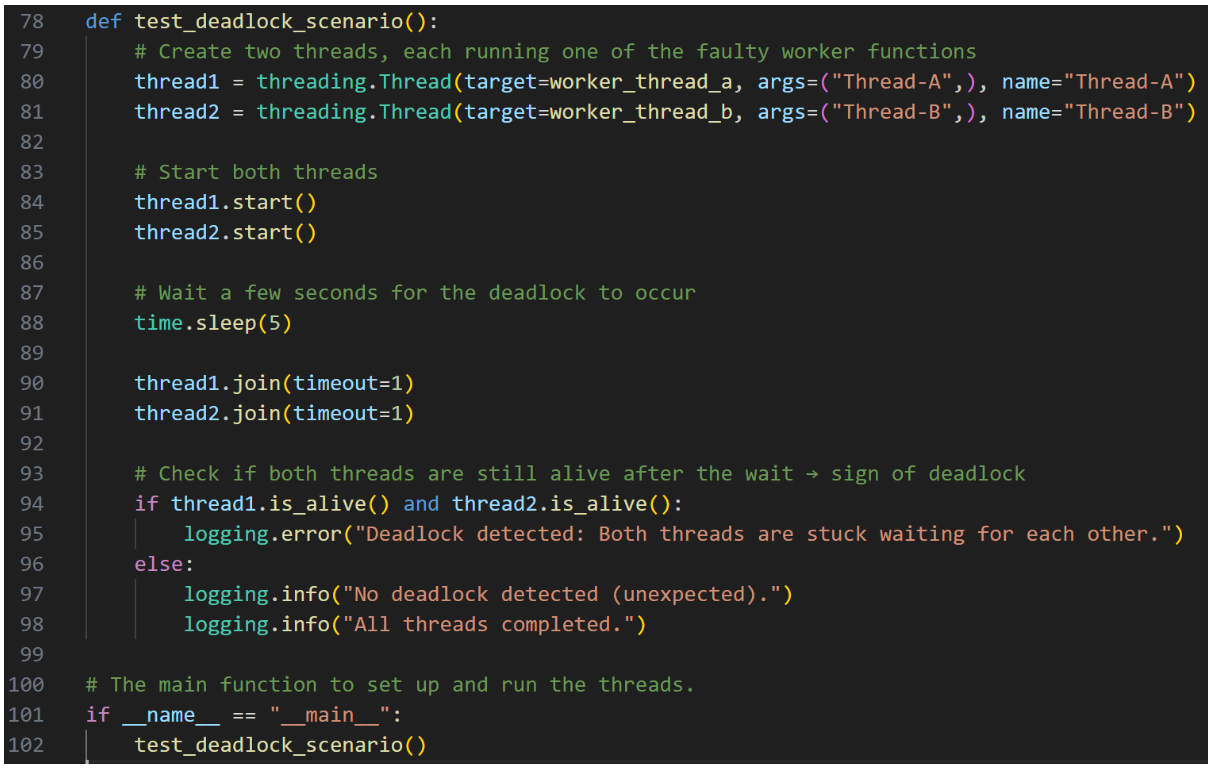
Thread A 嘗試先取得 Resource A，再等待 Resource B。

Thread B 嘗試先取得 Resource B，再等待 Resource A。

→ 導致互相等待，永遠不會結束。

Test Case:

建立兩個 threads，並執行，看到有觸發 fault。



```python
def test_deadlock_scenario():
    # Create two threads, each running one of the faulty worker functions
    thread1 = threading.Thread(target=worker_thread_a, args=("Thread-A",), name="Thread-A")
    thread2 = threading.Thread(target=worker_thread_b, args=("Thread-B",), name="Thread-B")

    # Start both threads
    thread1.start()
    thread2.start()

    # Wait a few seconds for the deadlock to occur
    time.sleep(5)

    thread1.join(timeout=1)
    thread2.join(timeout=1)

    # Check if both threads are still alive after the wait → sign of deadlock
    if thread1.is_alive() and thread2.is_alive():
        logging.error("Deadlock detected: Both threads are stuck waiting for each other.")
    else:
        logging.info("No deadlock detected (unexpected).")
        logging.info("All threads completed.")

# The main function to set up and run the threads.
if __name__ == "__main__":
    test_deadlock_scenario()
```

Expected Output:

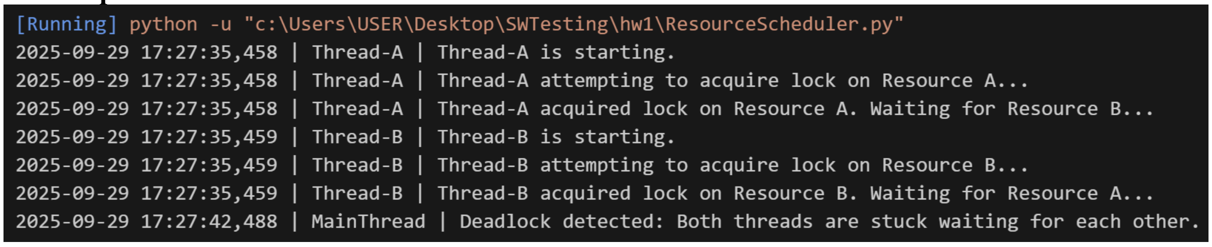
Thread-A 取得 Resource A，並嘗試取得 Resource B。

Thread-B 取得 Resource B，並嘗試取得 Resource A。

其中一個應該完成 complex_task_function，並釋放所有鎖。

程式應正常結束並印出 "All threads completed."。

Actual Output:



```
[Running] python -u "c:\Users\USER\Desktop\SWTesting\hw1\ResourceScheduler.py"
2025-09-29 17:27:35,458 | Thread-A | Thread-A is starting.
2025-09-29 17:27:35,458 | Thread-A | Thread-A attempting to acquire lock on Resource A...
2025-09-29 17:27:35,458 | Thread-A | Thread-A acquired lock on Resource A. Waiting for Resource B...
2025-09-29 17:27:35,459 | Thread-B | Thread-B is starting.
2025-09-29 17:27:35,459 | Thread-B | Thread-B attempting to acquire lock on Resource B...
2025-09-29 17:27:35,459 | Thread-B | Thread-B acquired lock on Resource B. Waiting for Resource A...
2025-09-29 17:27:42,488 | MainThread | Deadlock detected: Both threads are stuck waiting for each other.
```

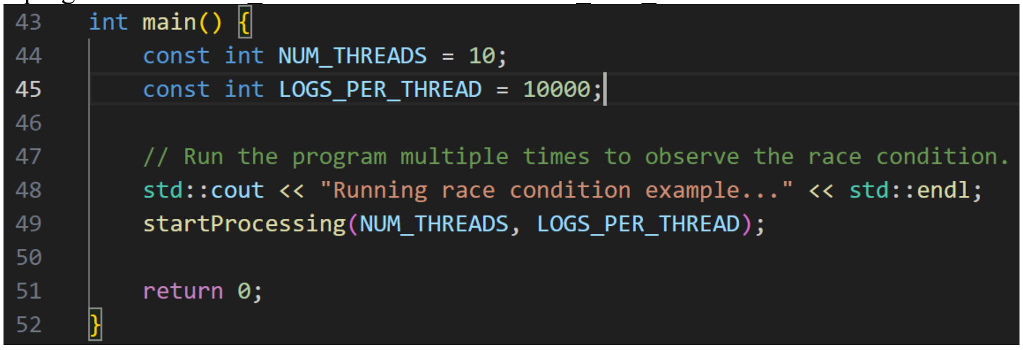
Section 4：LoggingSystem.cpp

Fault:

在遞增 total_logs_processed 時缺少 mutex，導致 race condition。

Test Case:

Run the program with NUM_THREADS = 10 and LOGS_PER_THREAD = 10000.



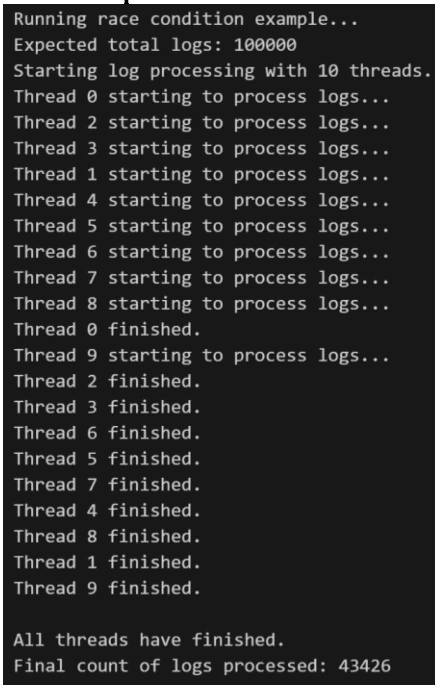
```
int main() {
    const int NUM_THREADS = 10;
    const int LOGS_PER_THREAD = 10000;

    // Run the program multiple times to observe the race condition.
    std::cout << "Running race condition example..." << std::endl;
    startProcessing(NUM_THREADS, LOGS_PER_THREAD);

    return 0;
}
```

Expected Output:

```
Running race condition example...
Expected total logs: 100000
Starting log processing with 10 threads.
Thread 0 starting to process logs...
Thread 1 starting to process logs...
...
Thread 9 finished.

All threads have finished.
Final count of logs processed: 100000
```

Final count of logs processed should be 100000.

Actual Output:

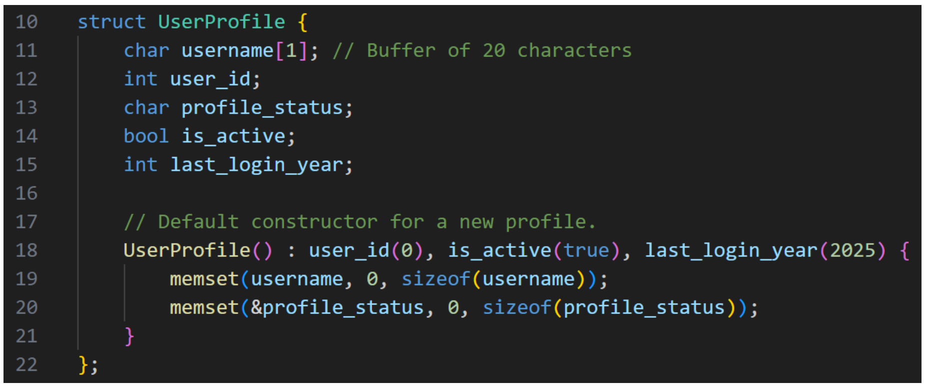


```
Running race condition example...
Expected total logs: 100000
Starting log processing with 10 threads.
Thread 0 starting to process logs...
Thread 2 starting to process logs...
Thread 3 starting to process logs...
Thread 1 starting to process logs...
Thread 4 starting to process logs...
Thread 5 starting to process logs...
Thread 6 starting to process logs...
Thread 7 starting to process logs...
Thread 8 starting to process logs...
Thread 0 finished.
Thread 9 starting to process logs...
Thread 2 finished.
Thread 3 finished.
Thread 6 finished.
Thread 5 finished.
Thread 7 finished.
Thread 4 finished.
Thread 8 finished.
Thread 1 finished.
Thread 9 finished.

All threads have finished.
Final count of logs processed: 43426
```

Final count of logs processed: 43426
-> less than expected.

Section 5：ProfileUpdater.cpp

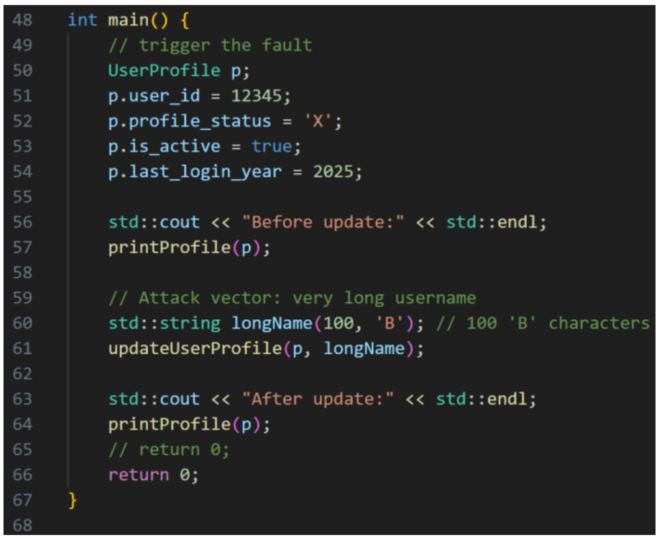
Fault:



```cpp
struct UserProfile {
    char username[1]; // Buffer of 20 characters
    int user_id;
    char profile_status;
    bool is_active;
    int last_login_year;

    // Default constructor for a new profile.
    UserProfile() : user_id(0), is_active(true), last_login_year(2025) {
        memset(username, 0, sizeof(username));
        memset(&profile_status, 0, sizeof(profile_status));
    }
};
```

char username[1]; 實際只有 buffer of 1 character。

Test Case:



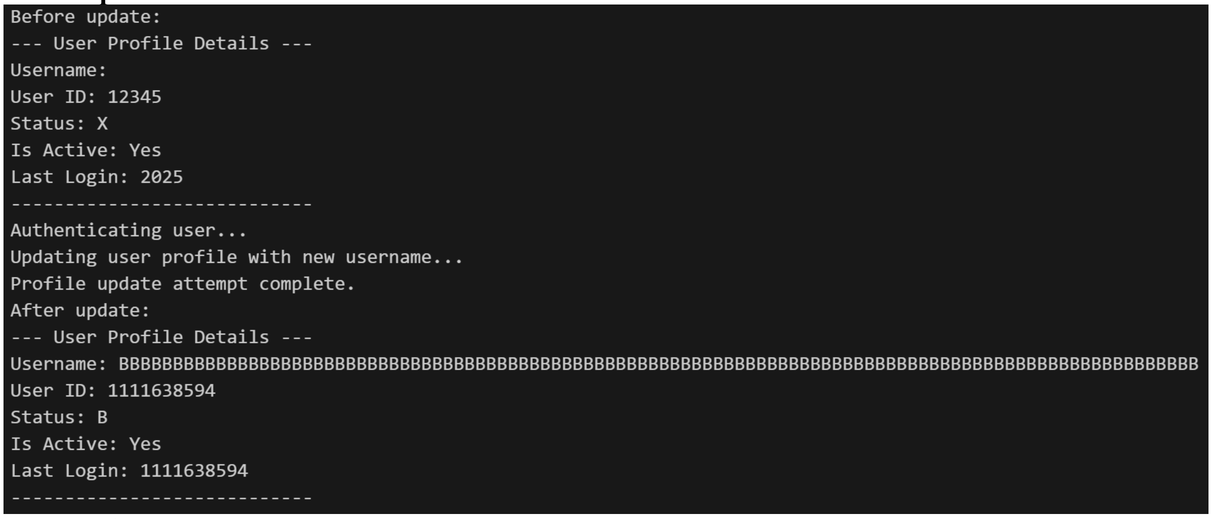
```cpp
int main() {
    // trigger the fault
    UserProfile p;
    p.user_id = 12345;
    p.profile_status = 'X';
    p.is_active = true;
    p.last_login_year = 2025;

    std::cout << "Before update:" << std::endl;
    printProfile(p);

    // Attack vector: very long username
    std::string longName(100, 'B'); // 100 'B' characters
    updateUserProfile(p, longName);

    std::cout << "After update:" << std::endl;
    printProfile(p);
    // return 0;
    return 0;
}
```

Using a very long username: longName(100, 'B'); // 100 'B' characters.

Expected Output:

```
Before update:
--- User Profile Details ---
Username:
User ID: 12345
Status: X
Is Active: Yes
Last Login: 2025
----------------------------
Updating...
After update:
--- User Profile Details ---
Username: BBBBBBBBB... B
User ID: 12345
Status: X
Is Active: Yes
Last Login: 2025
----------------------------
```

Actual Output:



```
Before update:
--- User Profile Details ---
Username:
User ID: 12345
Status: X
Is Active: Yes
Last Login: 2025
----------------------------
Authenticating user...
Updating user profile with new username...
Profile update attempt complete.
After update:
--- User Profile Details ---
Username: BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
User ID: 1111638594
Status: B
Is Active: Yes
Last Login: 1111638594
----------------------------
```

可以看到 User ID、Status 、Last Login 被溢出的 username 字元覆寫成亂數或大數值。