IT-304 : Software Engineering

Lab Assignment: 7

Program Inspection, Debugging and Static Analysis

Name: Parth Parekh

Student ID: 202201200

I. PROGRAM INSPECTION:

GitHub Code Link: Robin Hood Hashing

1. How many errors are there in the program? Mention the errors you have identified.

A. Data Reference Errors:

1. Uninitialized Variables:

- mHead and mListForFree (Line 418): These are initialized to nullptr but are not consistently reset following memory deallocation, which could result in dangling pointers or uninitialized memory access.

```
T* allocate() {
    T* tmp = mHead;
    if (!tmp) {
        tmp = performAllocation();
    }
    mHead = *reinterpret_cast_no_cast_align_warning<T**>(tmp);
    return tmp;
}
```

2. Array Bound Violations:

- shiftUp and shiftDown operations: No checks ensure that the index is within the array bounds.

```
while (--idx != insertion_idx) {
        mKeyVals[idx] = std::move(mKeyVals[idx - 1]);
    }
```

3. Dangling Pointers:

- In BulkPoolAllocator: The reset() method frees memory but does not reset the pointer to nullptr.

  std::free(mListForFree);

B. Data-Declaration Errors:

1. Potential Data Type Mismatches:

- Casting in hash_bytes: Hashing operations require frequent castings between different data types. If there are discrepancies in the size or characteristics of these data types, it may lead to unpredictable behavior.

2. Similar Variable Names:

- Confusion between similarly named variables: Variables such as mHead, mListForFree, and mKeyVals have similar names, which may lead to misunderstandings or errors during modification and debugging.

C. Computation Errors:

1. Integer Overflow:

- Hash Computations in hash_bytes: The hash function involves several shifts and multiplications on large integers, which may result in overflow if the computed values exceed their limits.

2. Off-by-One Errors:

- Loop Indexing in shiftUp and shiftDown: The loop conditions have the potential to cause off-by-one errors, particularly if the size of the data structure is not handled correctly.

  while (--idx != insertion_idx)

D. Comparison Errors:

1. Incorrect Boolean Comparisons:

- In conditions where multiple logical operations are combined, like in findIdx, improper use of `&&` and `||` can result in incorrect evaluations and unintended behavior.

```
if (info == mInfo[idx] &&
        ROBIN_HOOD_LIKELY(WKeyEqual::operator()(key,
mKeyVals[idx].getFirst()))) {
        return idx;
    }
```

2. Mixed Comparisons:

- In some cases, different types (e.g., signed and unsigned integers) are compared, which could lead to incorrect outcomes depending on the system/compiler.

E. Control-Flow Errors:

1. Potential Infinite Loop:

- Unterminated Loops: In loops like shiftUp and shiftDown, there is a risk of the loop not terminating correctly if the termination condition is never met.

F. Interface Errors:

1. Mismatched Parameter Attributes:

- Function Calls: There is a risk of parameter mismatches in functions like insert_move. The arguments provided may not align with the expected attributes, such as data type or size, potentially leading to errors.

  insert_move(std::move(oldKeyVals[i]));

2. Global Variables:

- Global variables in different functions: When global variables are referenced across multiple functions or procedures, it is essential to ensure they are used consistently and properly initialized. While this issue may not be immediately apparent, it could become a significant source of errors as the codebase expands.

G. Input/Output Errors:

1. Missing File Handling:

- Although the code does not currently handle files, any future extensions involving I/O operations could lead to common file handling issues, such as unclosed files, failure to check for end-of-file conditions, or insufficient error handling.

2. Which category of program inspection would you find more effective?

Category A: Data Reference Errors are particularly significant due to the dependence on manual memory management, pointers, and dynamic data structures. Errors in pointer dereferencing or improper memory allocation and deallocation can lead to critical issues like crashes, segmentation faults, or memory leaks, making it essential to address these problems. Additionally, Computation Errors and Control-Flow Errors should also be taken into account, especially in large-scale projects, as they can affect the correctness and efficiency of the system.

3. Which type of error you are not able to identified using the program inspection?

Concurrency Issues: The inspection does not consider any of the problems due to multi-threading or concurrency, such as race conditions or deadlocks. If the program were to be extended to deal with multi-threading, then shared resources, locks, and thread safety would become considerations.

Dynamic Errors: Some errors, such as those related to memory overflow, underflow, or runtime environment behaviour, may not be caught until the code is executed in a real-world scenario.

4. Is the program inspection technique is worth applicable?

Yes, the program inspection technique is indeed valuable, especially for identifying static errors that compilers might not catch, such as pointer mismanagement, array bound violations, and improper control

flow. While it may not detect every dynamic issue or concurrency-related bug, it is an essential step for ensuring code quality, particularly in memory-critical applications like this C++ implementation of hash tables.

II. CODE DEBUGGING

A. Armstrong

1. How many errors are there in the program? Mention the errors you have identified.

incorrect Calculation of Remainder:

- The line remainder = num / 10; should be remainder = num % 10; because we want to extract the last digit of the number.

Updating num Incorrectly:

- The line `num = num % 10;` should be changed to `num = num / 10;`. This modification ensures that the last digit of `num` is removed after processing, rather than taking its remainder again.

2. How many breakpoints you need to fix those errors ?

Two breakpoints:

1. On the line where the remainder is calculated (`remainder = num % 10;`).

2. On the line where `num` is updated (`num = num / 10;`).

a. What are the steps you have taken to fix the error you identified in the code fragment?

- Step 1: Fix the calculation of the remainder to correctly extract the last digit (remainder = num % 10;).
- Step 2: Correctly update num to remove the last digit (num = num / 10;).

B. GCD and LCM

1. How many errors are there in the program? Mention the errors you have identified.

There are two errors in the program:

- Logical Error in the gcd Method: The condition in the while loop is incorrect. It should be `while (a % b != 0)` instead of `while (a % b == 0)`. The original condition can cause an infinite loop if `b` is not a divisor of `a`.
- Logical Error in the lcm Method: The condition for checking whether `a` is a multiple of both `x` and `y` is incorrect. It should be `if (a % x == 0 && a % y == 0)` instead of `if (a % x != 0 && a % y != 0)`.

2. How many breakpoints do you need to fix those errors?

You need two breakpoints to debug and fix the identified errors:

- A breakpoint at the beginning of the gcd method to monitor the values of a, b, and r.
- A breakpoint at the beginning of the lcm method to check the initial value of a and how it increments during the loop.

a. What are the steps you have taken to fix the errors you identified in the code fragment?

1. Fixing the gcd Method:

- Changed the condition in the while loop from while (a % b == 0) to while (a % b != 0) to correctly implement the Euclidean algorithm for calculating the GCD.

2. Fixing the lcm Method:

- Modified the condition in the if statement from if (a % x != 0 && a % y != 0) to if (a % x == 0 && a % y == 0) to ensure that the method correctly identifies when a is a multiple of both x and y.

C. Knapsack

1. How many errors are there in the program? Mention the errors you have identified.

There are three main errors in the program:

- Array Indexing Issue: The line `int option1 = opt[n++][w];` incorrectly increments `n`, which can cause out-of-bounds access in subsequent iterations. It should be changed to `int option1 = opt[n][w];`.
- Wrong Profit Calculation: In the line int option2 = profit[n-2] + opt[n-1][w-weight[n]];, the program incorrectly uses profit[n-2] instead of profit[n] to calculate the profit of the current item.
- Weight Condition Logic: The condition for taking the item is correct, but the logic for option2 should only be calculated if the item's weight does not exceed the current weight limit (w).

2. How many breakpoints do you need to fix those errors?

You would need three breakpoints to debug and fix the errors:

- Set a breakpoint at the beginning of the nested loop to check the values of n, w, opt[n][w], and other variables.
- Set a breakpoint right before the assignment of option1 to monitor how n is changing.
- Set a breakpoint after the assignment of option2 to verify the calculations for both option1 and option2.

a. What are the steps you have taken to fix the error you identified in the code fragment?

1. Correcting Array Indexing:

- Changed int option1 = opt[n++][w]; to int option1 = opt[n][w]; to prevent n from being incremented incorrectly.

2. Correcting Profit Calculation:

- Modified the line `int option2 = profit[n-2] + opt[n-1][w-weight[n]];` to `int option2 = profit[n] + opt[n-1][w-weight[n]];` to correctly reference the profit for the appropriate item.

3. Adjusting Weight Condition Logic:

- Added a condition to ensure that option2 is only calculated if the current item's weight does not exceed w. This prevents erroneous profit calculations for items that can't be added.

D. Magic Number

1. How many errors are there in the program? Mention the errors you have identified.

There are four errors in the program:

- Logical Error in the Inner Loop: The condition in the line while(sum==0) should be while(sum!=0). The current condition will not enter the loop when sum is zero, which is incorrect.
- Incorrect Calculation in the Inner Loop: The line s=s*(sum/10); should be s = s + (sum % 10); to correctly accumulate the sum of the digits.
- Missing Semicolon: The line sum=sum%10 should have a semicolon at the end: sum = sum % 10;.
- Logical Error in the While Loop: The outer loop condition while(num>9) should be while(num>9 || num == 0) to account for the scenario where the number becomes zero.

2. How many breakpoints do you need to fix those errors?

You would need three breakpoints to effectively debug and fix the errors:

1. Set a breakpoint at the beginning of the inner loop to observe the values of sum and s.

2. Set a breakpoint at the beginning of the outer loop to check the current value of num.

3. Set a breakpoint before the final if statement to verify the final value of num before making the magic number determination.

a. What are the steps you have taken to fix the error you identified in the code fragment?

1. Correcting the Inner Loop Condition:

- Changed while(sum==0) to while(sum!=0) to ensure the loop iterates while there are digits left to process.

2. Fixing the Digit Summation Logic:

- Updated the line s=s*(sum/10); to s = s + (sum % 10); to accumulate the digits correctly.

3. Adding Missing Semicolon:

- Added a semicolon at the end of sum = sum % 10;.

4. Adjusting the Outer Loop Condition:

- Changed the outer loop condition from `while(num > 9)` to `while(num > 9 || num == 0)` to properly handle the case where `num` might reduce to zero.

E. Merge Sort

1. How many errors are there in the program? Mention the errors you have identified.

There are four main errors in the program:

1. Incorrect Array Slicing: The lines `int[] left = leftHalf(array + 1);` and `int[] right = rightHalf(array - 1);` are incorrect because you cannot slice arrays by adding or subtracting integers. They should correctly split the array into halves.

2. Incorrect Parameters in Recursive Calls: When calling `merge(array, left++, right--);`, using the increment/decrement operators (++ and --) on the arrays is not valid. You should pass the arrays as they are without modifying them.

3. Incorrect Calculation of Left and Right Sizes: The size calculations in `leftHalf` and `rightHalf` should consider the entire array. The size for the left half should be `(array.length + 1) / 2` to properly handle arrays of odd lengths.

4. Missing Merging Logic: In the merge method, the original array (result) should not be passed in the manner shown. Instead, it should be the original array passed to the merge sort function which gets modified.

This logic needs to be integrated properly.

2. How many breakpoints do you need to fix those errors?

You would need three breakpoints to effectively debug and fix the errors:

1. Set a breakpoint at the beginning of the mergeSort method to inspect how the array is being split and what the left and right halves are.

2. Set a breakpoint before the merge operation to check the contents of the left and right arrays.

3. Set a breakpoint inside the merge method to see how elements are being merged back into the original array.

a. What are the steps you have taken to fix the error you identified in the code fragment?

1. Correcting Array Slicing:

- Instead of int[] left = leftHalf(array + 1); and int[] right = rightHalf(array - 1);, change it to correctly split the array using Arrays.copyOfRange.

2. Fixing Parameters in Recursive Calls:

- Update the call to merge by passing the arrays without using the increment/decrement operators: merge(array, left, right);.

3. Adjusting Size Calculations:

- Change the size calculation in leftHalf and rightHalf methods to (array.length + 1) / 2 for the left half and the rest for the right half.

4. Merging Logic:

- Ensure that the merge method correctly combines the sorted arrays back into the original array.

F. Multiply metrics

1. How many errors are there in the program? Mention the errors you have identified.

There are five main errors in the program:

1. Array Indexing Errors: In the line sum = sum + first[c-1][c-k] * second[k-1][k-d];, the indices c-1 and k-d are incorrect. They should use c and k for proper indexing since the matrix elements start from index 0.

2. Uninitialized Variables: The variable sum is being reused without resetting in the inner loop properly. This can lead to incorrect calculations in subsequent iterations. It should be reset to 0 at the start of each c and d iteration.

3. Wrong Output Input Prompt: The input prompt for the second matrix incorrectly states, “Enter the number of rows and columns of first matrix” instead of “Enter the number of rows and columns of second matrix”.

4. Multiplication Logic Issue: The multiplication logic needs to access elements of the matrices correctly. The correct formula for matrix multiplication is first[c][k] * second[k][d].

5. Potential Readability Issue: The output formatting is slightly misleading, as it shows the product matrix but doesn't include a proper header or format.

2. How many breakpoints do you need to fix those errors?

You would need three breakpoints to effectively debug and fix the errors:

1. Set a breakpoint inside the multiplication loop to inspect the indices and the values being multiplied.

2. Set a breakpoint before printing the multiplication results to check the contents of the multiplication array.

3. Set a breakpoint after reading the second matrix to verify that the inputs have been read correctly.

a. What are the steps you have taken to fix the error you identified in the code fragment?

1. Correcting Array Indexing:

- Change sum = sum + first[c-1][c-k] * second[k-1][k-d]; to sum = sum + first[c][k] * second[k][d]; to correctly access the elements of the matrices.

2. Resetting Variables:

- Move the reset of the sum variable to the beginning of the inner loop for d to ensure it starts fresh for each element calculation: sum = 0; should be at the start of the for (d = 0; d < q; d++) loop.

3. Fixing Input Prompts:

- Update the prompt for the second matrix to say “Enter the number of rows and columns of the second matrix”.

4. Adjusting Output Formatting:

- Consider adding headers to clarify that the following output is the product matrix.

G. Quadratic Probing

1. How many errors are there in the program? Mention the errors you have identified.

There are several errors in the program:

1. Syntax Error in the Insert Method: The line `i + = (i + h / h--) % maxSize;` contains a space in the `+=` operator, which results in a compilation error. It should be corrected to `i += (i + h / h--) % maxSize;`.

2. Incorrect Hashing Logic: The line `i = (i + h * h++) % maxSize;` is incorrect because it modifies `h` within the loop, which can lead to an infinite loop.

3. Key Removal Logic: In the `remove` method, `currentSize--` is decremented twice, resulting in incorrect size management.

4. Uninitialized Value Printing: When printing the hash table, the output may include null values or be improperly formatted.

5. Clear Method Logic: The `makeEmpty` method does not clear the actual objects in the arrays, potentially leading to memory issues.

2. How many breakpoints do you need to fix those errors?

To fix these errors, you would need the following breakpoints:

1. Breakpoint on the Insert Method: Before the line containing the i + = operator to check the current value of i.

2. Breakpoint on the Hash Method: To observe how the hash value is calculated for different keys.

3. Breakpoint on the Remove Method: To ensure the correct key is being removed and to check the state of the hash table after the removal.

4. Breakpoint in the Print Method: To validate the correct values are being printed from the hash table.

a. What are the steps you have taken to fix the error you identified in the code fragment?

1. Correcting the Insert Method: Remove the space in the += operator and correct the logic for incrementing h.

2. Fixing the Hash Method: Ensure that the hashing algorithm doesn't modify h directly and doesn't lead to an infinite loop.

3. Updating Removal Logic: Adjust the remove method to ensure currentSize is only decremented once after a successful removal.

4. Enhancing Print Logic: Add checks to avoid printing null values and ensure that the output format is clear.

5. Adjusting the Make Empty Logic: Modify the makeEmpty method to reset the actual contents of the keys and values arrays.

H. Sorting Array

1. How many errors are there in the program? Mention the errors you have identified.

There are several errors in the program:

1. Class Name Error: The class name Ascending _Order contains a space, which is not allowed in Java. It should be AscendingOrder.

2. Incorrect Loop Condition: The outer loop for (int i = 0; i >= n; i++); has an incorrect condition (i >= n), which will cause it to never execute. The correct condition should be i < n.

3. Unnecessary Semicolon: There is an unnecessary semicolon at the end of the outer loop declaration (for (int i = 0; i >= n; i++);), which ends the loop prematurely.

4. Sorting Logic: The comparison in the sorting condition is incorrect. It should be if (a[i] > a[j]) to ensure that the smaller number is placed before the larger number.

5. Output Formatting: The final output will have an extra comma if the elements are printed directly. It should be formatted correctly to avoid trailing commas.

2. How many breakpoints do you need to fix those errors?

To fix these errors, you would need the following breakpoints:

1. Breakpoint on Class Declaration: To check the correct naming of the class.

2. Breakpoint on Outer Loop: To observe the initial value of i and ensure that the loop condition is correct.

3. Breakpoint on Sorting Logic: To validate the values of a[i] and a[j] before and after swapping.

4. Breakpoint on Output: To check the formatting of the output and ensure it doesn't include unwanted commas.

a. What are the steps you have taken to fix the error you identified in the code fragment?

1. Renaming the Class: Change the class name from Ascending _Order to AscendingOrder.

2. Correcting the Loop Condition: Change the loop condition from i >= n to i < n.

3. Removing the Semicolon: Remove the unnecessary semicolon after the outer loop declaration.

4. Fixing the Sorting Logic: Change the condition in the sorting logic to if (a[i] > a[j]).

5. Formatting the Output: Update the output logic to avoid trailing commas.

I. Stack Implementation

1. How many errors are there in the program? Mention the errors you have identified.

There are several errors in the program:

1. Incorrect Logic in push Method: The line top--; should be top++; because we want to increment the top index to push the value onto the stack.

2. Incorrect Logic in pop Method: The line top++; should be top--; because we want to decrement the top index to remove the top element of the stack.

3. Incorrect Condition in display Method: The loop condition for (int i = 0; i > top; i++) is incorrect. It should be i <= top to ensure all elements in the stack are displayed.

4. Handling Stack Underflow: The pop method should return the popped value. This can be done by storing the value being popped before decrementing top.

5. Displaying the Stack Contents: The output format may be misleading because the elements are not displayed correctly after popping.

2. How many breakpoints do you need to fix those errors?

To fix these errors, you would need the following breakpoints:

1. Breakpoint on push Method: To check the value of top before and after the increment.

2. Breakpoint on pop Method: To observe the value being popped and the state of top.

3. Breakpoint on display Method: To verify the loop condition and ensure all elements are printed correctly.

a. What are the steps you have taken to fix the error you identified in the code fragment?

1. Corrected Logic in push Method: Change top--; to top++; so that the next element is added at the correct index.

2. Corrected Logic in pop Method: Change top++; to top--; to ensure the top element is correctly removed from the stack.

3. Updated Loop Condition in display Method: Change i > top to i <= top so that all elements in the stack are displayed.

4. Return Value in pop Method: Modify the pop method to return the value that was popped from the stack.

5. Adjust the Display Logic: Ensure the display method properly reflects the current state of the stack after popping elements.

J. Tower of Hanoi

1. How many errors are there in the program? Mention the errors you have identified.

There are several errors in the program:

1. Incorrect Increment and Decrement in Recursive Call: The line doTowers(topN ++, inter--, from+1, to+1) is incorrect. The post-increment and post-decrement operators (++ and --) are used incorrectly in this context. They should not be used this way, as they do not modify the values passed to the function.

2. Missing Recursive Call for Disk Movement: The logic for handling disk movements in the recursive calls is not accurate, leading to incorrect calculations.

3. Printing Issues: The final output does not match the expected movements of the disks correctly due to the incorrect handling of parameters.

2. How many breakpoints do you need to fix those errors?

You would need the following breakpoints to fix the errors:

1. Breakpoint on the first doTowers call: To check the values of topN, from, inter, and to before executing the recursive calls.

2. Breakpoint before the printing statement: To observe the correct flow of disk movements.

3. Breakpoint on the second doTowers call: To ensure the parameters are being correctly passed after the first recursive call.

a. What are the steps you have taken to fix the error you identified in the code fragment?

1. Corrected Recursive Call: Change `doTowers(topN++, inter--, from+1, to+1)` to `doTowers(topN - 1, inter, from, to)` in the recursive call for moving the remaining disks.

2. Removed Invalid Modifications: Ensure that the values of `from`, `inter`, and `to` are not altered using post-increment and post-decrement operators. Instead, directly pass the original variables.

3. Clarified Disk Movement Logic: Verify that the recursive logic accurately implements the Tower of Hanoi algorithm.

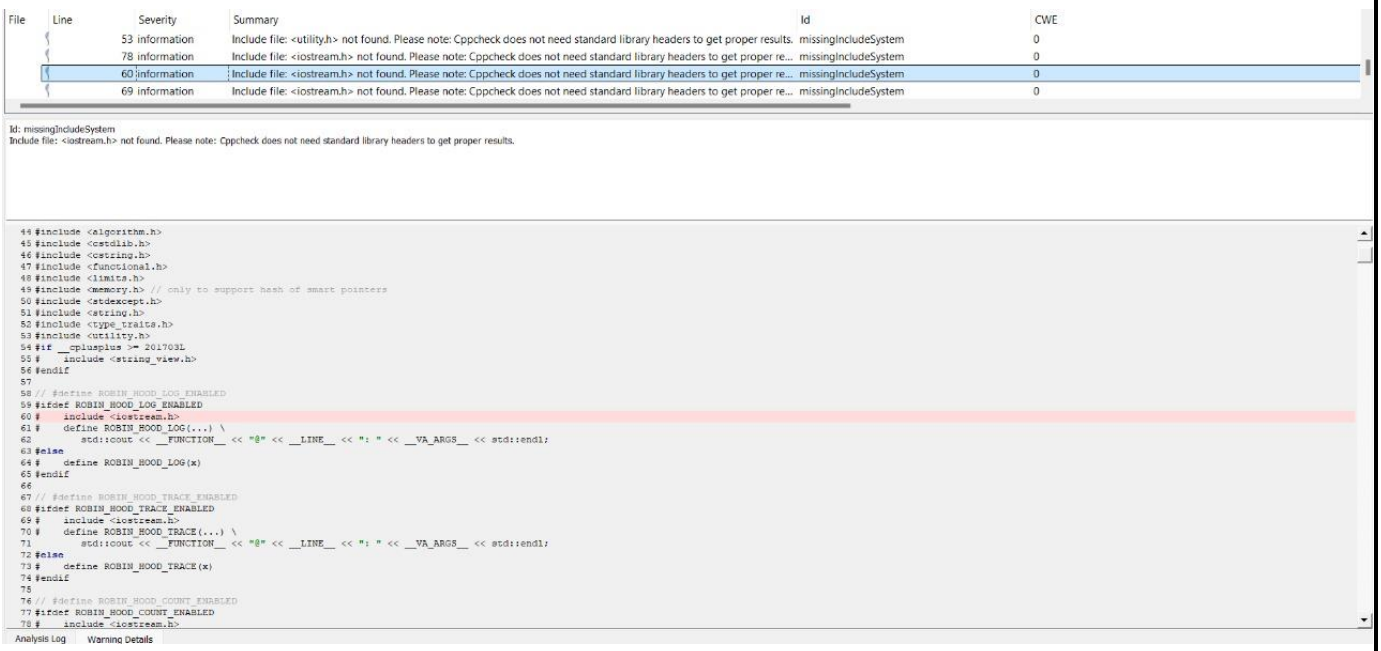
Static Analysis Tools:

File	Line	Severity	Summary	Id
	49	information	Include file: <memory.h> not found. Please note: Cppcheck does not need standard library headers to get proper re...	missingIncludeSystem
	50	information	Include file: <stdexcept.h> not found. Please note: Cppcheck does not need standard library headers to get proper r...	missingIncludeSystem
	51	information	Include file: <string.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.	missingIncludeSystem
	52	information	Include file: <type_traits.h> not found. Please note: Cppcheck does not need standard library headers to get proper r...	missingIncludeSystem

Id: missingIncludeSystem
Include file: <memory.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

```
33 // OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
34 // SOFTWARE.
35
36 #ifndef ROBIN_HOOD_H_INCLUDED
37 #define ROBIN_HOOD_H_INCLUDED
38
39 // see https://semver.org/
40 #define ROBIN_HOOD_VERSION_MAJOR 3  // for incompatible API changes
41 #define ROBIN_HOOD_VERSION_MINOR 11 // for adding functionality in a backwards-compatible manner
42 #define ROBIN_HOOD_VERSION_PATCH 5  // for backwards-compatible bug fixes
43
44 #include <algorithm.h>
45 #include <cstdlib.h>
46 #include <cstring.h>
47 #include <functional.h>
48 #include <limits.h>
49 #include <memory.h> // only to support hash of smart pointers
50 #include <stdexcept.h>
51 #include <string.h>
52 #include <type_traits.h>
53 #include <utility.h>
54 #if __cplusplus >= 201703L
55 #    include <string_view.h>
56 #endif
57
58 // #define ROBIN_HOOD_LOG_ENABLED
59 #ifdef ROBIN_HOOD_LOG_ENABLED
60 #    include <iostream.h>
61 #    define ROBIN_HOOD_LOG(...) \
62         std::cout << __FUNCTION__ << "@" << __LINE__ << ": " << __VA_ARGS__ << std::endl;
63 #else
64 #    define ROBIN_HOOD_LOG(x)
65 #endif
66
67 // #define ROBIN_HOOD_TRACE_ENABLED
```

Analysis Log Warning Details

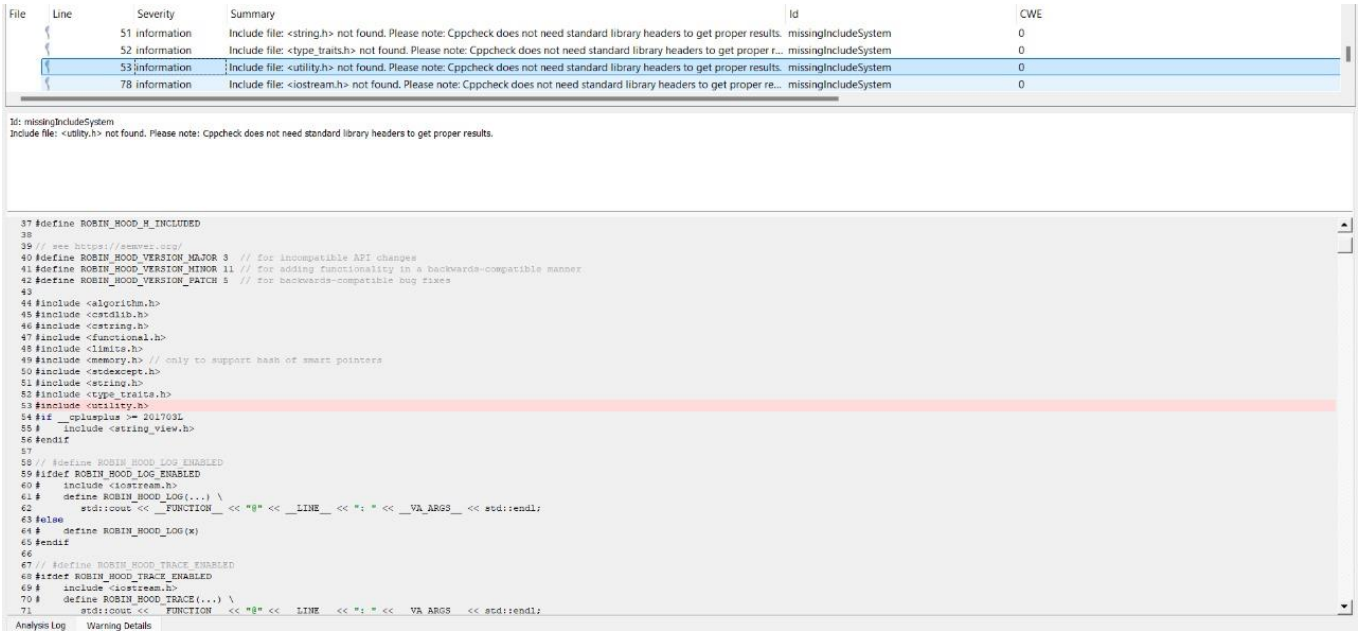


File	Line	Severity	Summary	Id	CWE
	53	information	Include file: <utility.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.	missingIncludeSystem	0
	78	information	Include file: <iostream.h> not found. Please note: Cppcheck does not need standard library headers to get proper re...	missingIncludeSystem	0
	60	information	Include file: <iostream.h> not found. Please note: Cppcheck does not need standard library headers to get proper re...	missingIncludeSystem	0
	69	information	Include file: <iostream.h> not found. Please note: Cppcheck does not need standard library headers to get proper re...	missingIncludeSystem	0

Id: missingIncludeSystem
Include file: <iostream.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

```
44 #include <algorithm.h>
45 #include <cstdlib.h>
46 #include <cstring.h>
47 #include <functional.h>
48 #include <limits.h>
49 #include <memory.h> // only to support hash of smart pointers
50 #include <stdexcept.h>
51 #include <string.h>
52 #include <type_traits.h>
53 #include <utility.h>
54 #if __cplusplus >= 201703L
55 #    include <string_view.h>
56 #endif
57
58 // #define ROBIN_HOOD_LOG_ENABLED
59 #ifdef ROBIN_HOOD_LOG_ENABLED
60 #    include <iostream.h>
61 #    define ROBIN_HOOD_LOG(...) \
62         std::cout << __FUNCTION__ << "@" << __LINE__ << ": " << __VA_ARGS__ << std::endl;
63 #else
64 #    define ROBIN_HOOD_LOG(x)
65 #endif
66
67 // #define ROBIN_HOOD_TRACE_ENABLED
68 #ifdef ROBIN_HOOD_TRACE_ENABLED
69 #    include <iostream.h>
70 #    define ROBIN_HOOD_TRACE(...) \
71         std::cout << __FUNCTION__ << "@" << __LINE__ << ": " << __VA_ARGS__ << std::endl;
72 #else
73 #    define ROBIN_HOOD_TRACE(x)
74 #endif
75
76 // #define ROBIN_HOOD_COUNT_ENABLED
77 #ifdef ROBIN_HOOD_COUNT_ENABLED
78 #    include <iostream.h>
```

Analysis Log | Warning Details



File	Line	Severity	Summary	Id	CWE
	51	information	Include file: <string.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.	missingIncludeSystem	0
	52	information	Include file: <type_traits.h> not found. Please note: Cppcheck does not need standard library headers to get proper r...	missingIncludeSystem	0
	53	information	Include file: <utility.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.	missingIncludeSystem	0
	78	information	Include file: <iostream.h> not found. Please note: Cppcheck does not need standard library headers to get proper re...	missingIncludeSystem	0

Id: missingIncludeSystem
Include file: <utility.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

```
37 #define ROBIN_HOOD_H_INCLUDED
38
39 // see https://semver.org/
40 #define ROBIN_HOOD_VERSION_MAJOR 3  // for incompatible API changes
41 #define ROBIN_HOOD_VERSION_MINOR 11 // for adding functionality in a backwards-compatible manner
42 #define ROBIN_HOOD_VERSION_PATCH 5  // for backwards-compatible bug fixes
43
44 #include <algorithm.h>
45 #include <cstdlib.h>
46 #include <cstring.h>
47 #include <functional.h>
48 #include <limits.h>
49 #include <memory.h> // only to support hash of smart pointers
50 #include <stdexcept.h>
51 #include <string.h>
52 #include <type_traits.h>
53 #include <utility.h>
54 #if __cplusplus >= 201703L
55 #    include <string_view.h>
56 #endif
57
58 // #define ROBIN_HOOD_LOG_ENABLED
59 #ifdef ROBIN_HOOD_LOG_ENABLED
60 #    include <iostream.h>
61 #    define ROBIN_HOOD_LOG(...) \
62         std::cout << __FUNCTION__ << "@" << __LINE__ << ": " << __VA_ARGS__ << std::endl;
63 #else
64 #    define ROBIN_HOOD_LOG(x)
65 #endif
66
67 // #define ROBIN_HOOD_TRACE_ENABLED
68 #ifdef ROBIN_HOOD_TRACE_ENABLED
69 #    include <iostream.h>
70 #    define ROBIN_HOOD_TRACE(...) \
71         std::cout << __FUNCTION__ << "@" << __LINE__ << ": " << __VA_ARGS__ << std::endl;
```

Analysis Log | Warning Details