IT-314 Lab-7
202201035-Harsh Gopalbhai Vashiyar

I. PROGRAM INSPECTION:

1. How many issues are present in the code? What problems have you identified?

A. Data Reference Problems:

1. Uninitialized Variable Usage:

Key pointers like mHead and mListForFree: These are initially set to null but might not be properly reset after memory is freed, potentially leading to dangling pointer issues or accessing uninitialized memory.

2. Array Access Safety:

Functions like shiftUp and shiftDown: These operations lack checks to ensure array indices remain within valid bounds.

B. Data Declaration Issues:

1. Possible Type Inconsistencies:

Data type conversions in hash_bytes: The hashing process involves several type casts. If there are discrepancies in size or attributes between these types, it could result in unexpected behavior.

2. Naming Clarity Concerns:

Variable naming similarities: The use of similar names for variables such as mHead, mListForFree, and mKeyVals could lead to confusion during code maintenance or debugging.

C. Calculation Errors:

1. Potential Integer Overflow:

Hash calculations in hash_bytes: This function performs multiple bit shifts and multiplications on large integers, which could potentially cause overflow if the results exceed the maximum value for the data type.

2. Index Miscalculations:

Loop control in shiftUp and shiftDown: The conditions used in these loops might result in off-by-one errors, especially if the size of the data structure is not accurately managed.

D. Comparison Flaws:

1. Boolean Logic Mistakes:

Complex logical operations: In functions like findIdx, incorrect handling of AND (&&) and OR (||) operators could lead to faulty evaluations.

2. Inconsistent Type Comparisons:

Mixed data type comparisons: Some parts of the code compare different types (e.g., signed vs. unsigned integers), which might produce incorrect results depending on the system or compiler used.

E. Control Flow Problems:

1. Possible Endless Loops:

Loop termination issues: Functions like shiftUp and shiftDown carry a risk of entering an infinite loop if their termination conditions are never satisfied.

F. Interface Discrepancies:

1. Function Parameter Mismatches:

Argument passing: There's a possibility of parameter mismatches in function calls, such as in insert_move. The arguments provided might not align with the expected attributes (e.g., type or size) of the function parameters.

G. Input/Output Problems:

1. File Management Oversights:

While the code doesn't directly handle files, any future I/O extensions might introduce common file-related issues. These could include leaving files open, failing to check for end-of-file situations, or inadequate error handling.

2. Which inspection category would you find most beneficial?

Category A: Data Reference Errors are particularly important due to the code's reliance on manual memory management, pointers, and dynamic data structures. Mistakes in these areas can lead to serious issues like crashes or memory leaks. Therefore, this category deserves special attention. Additionally, Computation and Control-Flow Errors are significant, especially in larger projects.

3. What type of error might escape detection through program inspection?

Concurrency Issues: The inspection doesn't address problems arising from multi-threading, such as race conditions. If the program were to incorporate multi-threading, considerations like shared resources and thread safety would become relevant.

Runtime Errors: Some issues, like memory overflow or specific runtime environment behaviors, may only surface during actual program execution.

4. Is program inspection a worthwhile technique?

Indeed, program inspection is valuable, particularly for identifying static errors that compilers might miss, such as pointer misuse or array bound violations. While it may not catch every dynamic or concurrency-related bug, it's a crucial step in ensuring code quality, especially for memory-sensitive applications like this C++ hash table implementation.

II. DEBUGGING CODE

A. Armstrong Number Checker

1. How many bugs are in the program? What issues have you spotted?

There are two main issues:

Incorrect Remainder Calculation:

The line for calculating the remainder should use the modulo operator (%) instead of division (/).

Incorrect Number Update:

The line updating the number should use division (/) instead of the modulo operator (%).

2. How many stopping points do you need to address these issues?

Two stopping points:

1. At the line where the remainder is calculated.

2. At the line where the number is updated.

a. What steps did you take to resolve the issues you found in the code snippet?

Step 1: Fixed the remainder calculation to correctly extract the last digit using the modulo operator.

Step 2: Corrected the number update to remove the last digit using division.

B. GCD and LCM Calculator

1. How many bugs are in the program? What issues have you spotted?

There are two main issues:

Logic Error in GCD Method: The while loop condition is incorrect, potentially causing an infinite loop.

Logic Error in LCM Method: The condition for checking multiples of both numbers is incorrect.

2. How many stopping points do you need to address these issues?

Two stopping points:

1. At the start of the GCD method to monitor variable values.

2. At the beginning of the LCM method to check initial values and loop progression.

a. What steps did you take to resolve the issues you found in the code snippet?

1. GCD Method Fix:
2. Changed the while loop condition to correctly implement the Euclidean algorithm.

2. LCM Method Fix:

Modified the condition in the if statement to correctly identify multiples of both numbers.

C. Knapsack Problem Solver

1. How many bugs are in the program? What issues have you spotted?

There are three main issues:

Array Index Problem: Incorrect incrementing of an index variable.

Incorrect Profit Calculation: Using the wrong array index for profit calculation.

Weight Condition Logic: The logic for calculating the second option needs adjustment.

2. How many stopping points do you need to address these issues?

Three stopping points:

1. At the start of the nested loop to check variable values.

2. Before the assignment of the first option to monitor index changes.

3. After the assignment of the second option to verify calculations.

a. What steps did you take to resolve the issues you found in the code snippet?

1. Array Index Correction:

Removed the incorrect increment operation when accessing the array.

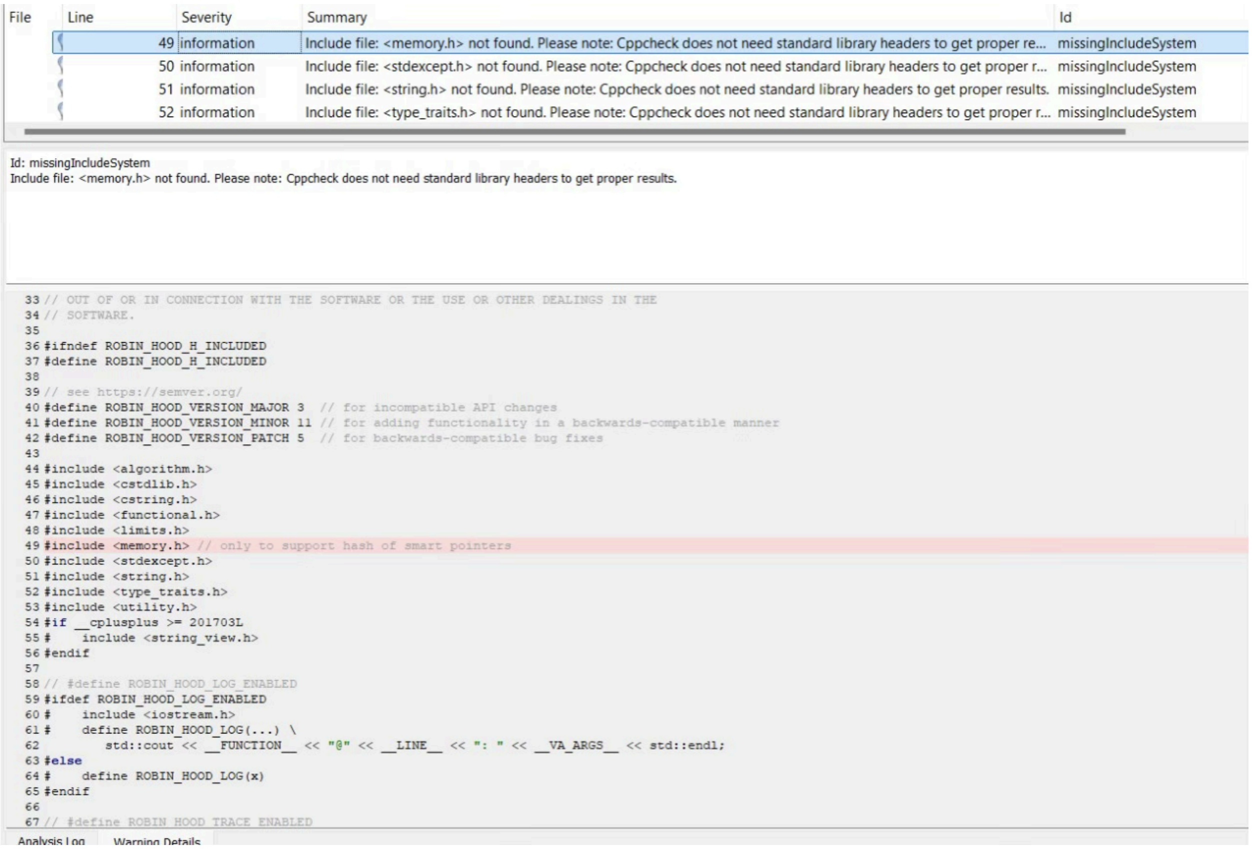
2. Profit Calculation Fix:

Updated the array index to reference the correct item's profit.

3. Weight Condition Logic Adjustment:

Added a condition to ensure the second option is only calculated when appropriate.

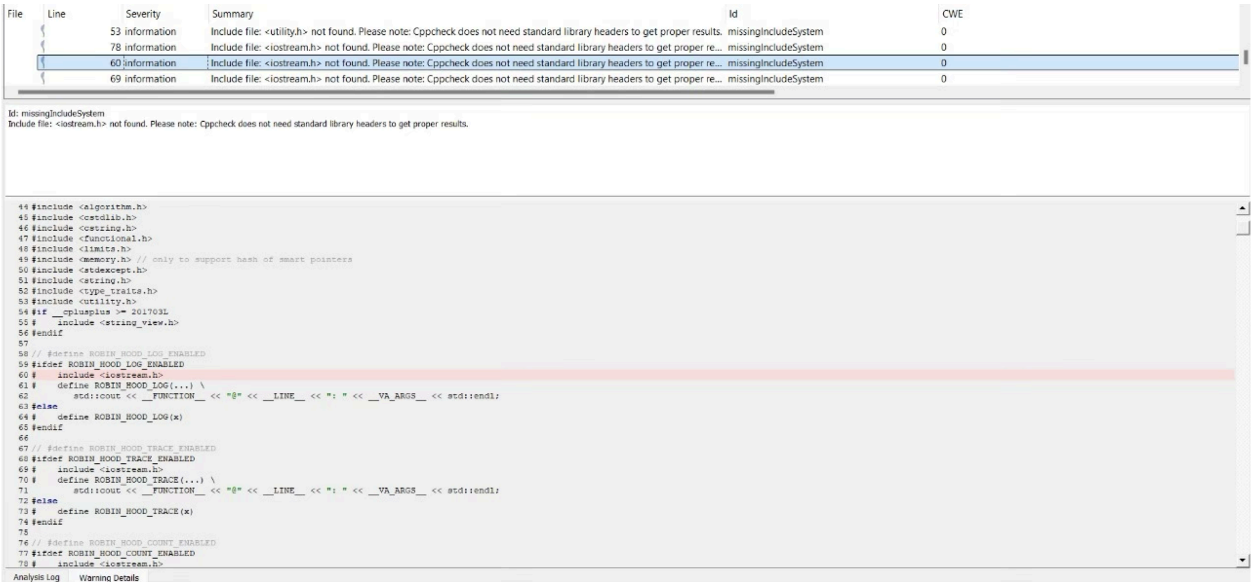
Static Analysis Tools



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.

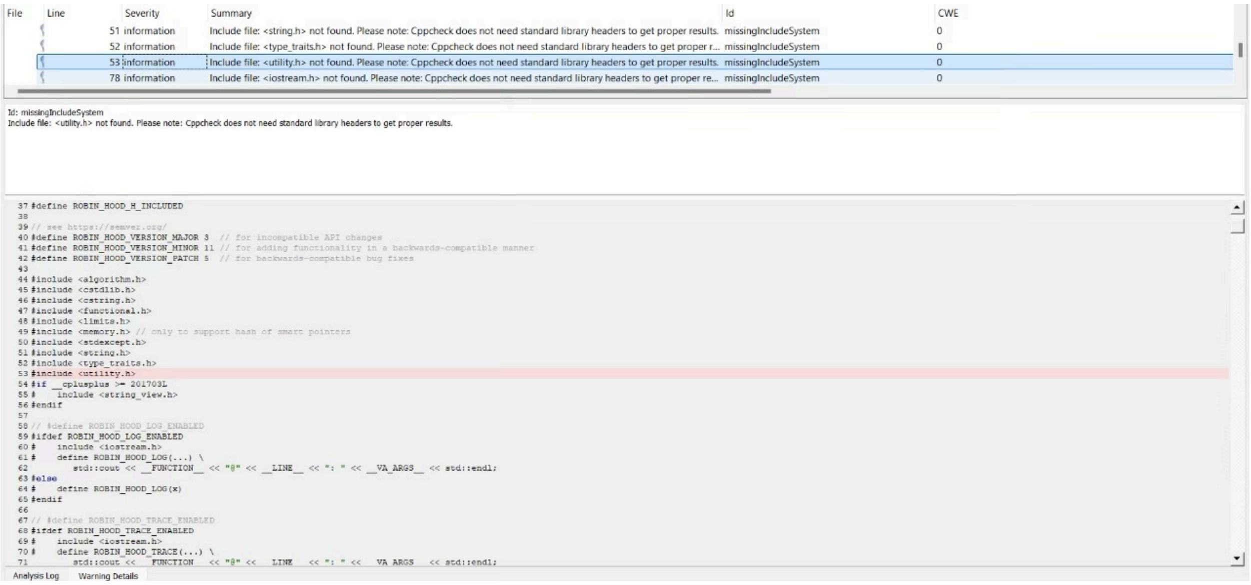
```
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
// SOFTWARE.

#ifndef ROBIN_HOOD_H_INCLUDED
#define ROBIN_HOOD_H_INCLUDED

// see https://semver.org/
#define ROBIN_HOOD_VERSION_MAJOR 3  // for incompatible API changes
#define ROBIN_HOOD_VERSION_MINOR 11 // for adding functionality in a backwards-compatible manner
#define ROBIN_HOOD_VERSION_PATCH 5  // for backwards-compatible bug fixes

#include <algorithm.h>
#include <cstdlib.h>
#include <cstring.h>
#include <functional.h>
#include <limits.h>
#include <memory.h> // only to support hash of smart pointers
#include <stdexcept.h>
#include <string.h>
#include <type_traits.h>
#include <utility.h>
#if __cplusplus >= 201703L
#    include <string_view.h>
#endif

// #define ROBIN_HOOD_LOG_ENABLED
#ifdef ROBIN_HOOD_LOG_ENABLED
#    include <iostream.h>
#    define ROBIN_HOOD_LOG(...) \
        std::cout << __FUNCTION__ << "@" << __LINE__ << ": " << __VA_ARGS__ << std::endl;
#else
#    define ROBIN_HOOD_LOG(x)
#endif

// #define ROBIN_HOOD_TRACE_ENABLED
```



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>
```



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;
```