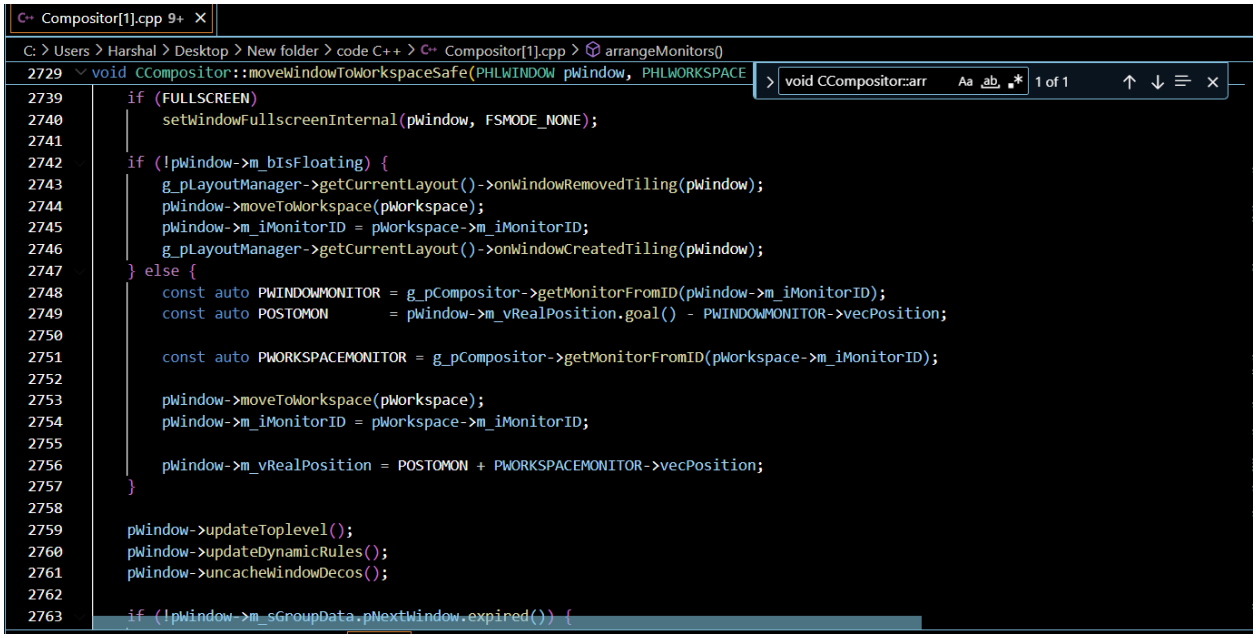
Lab Assignment

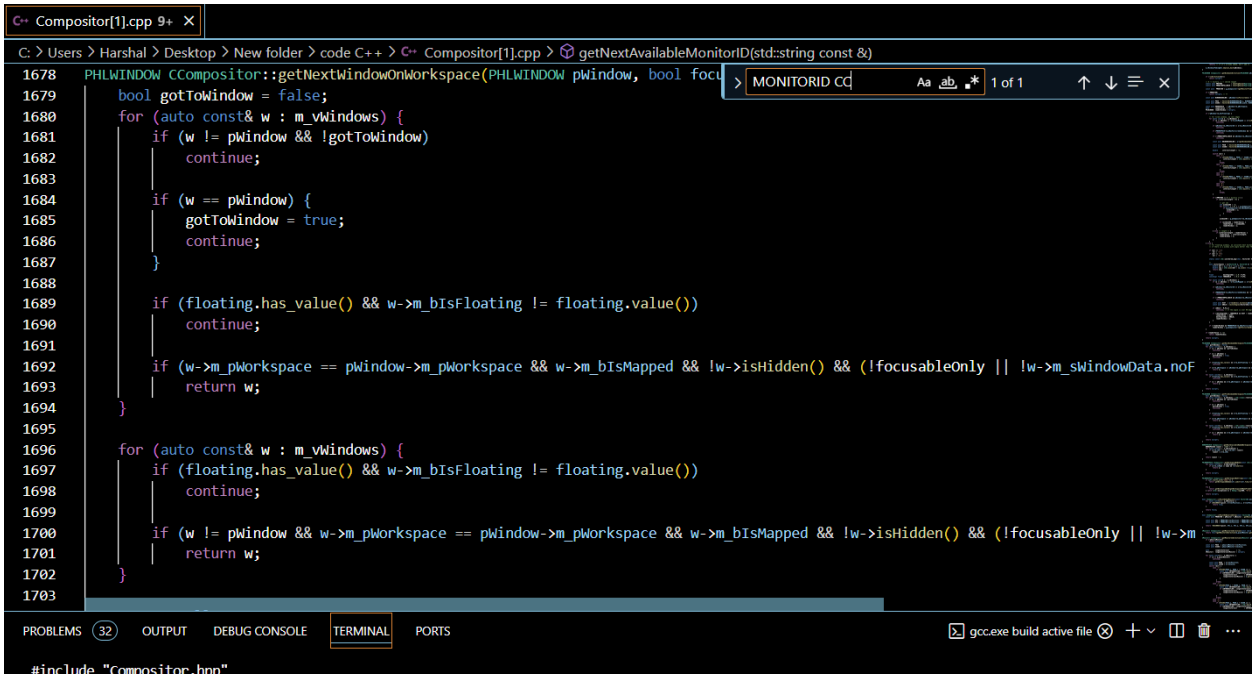
Harshal Patel - 202201070

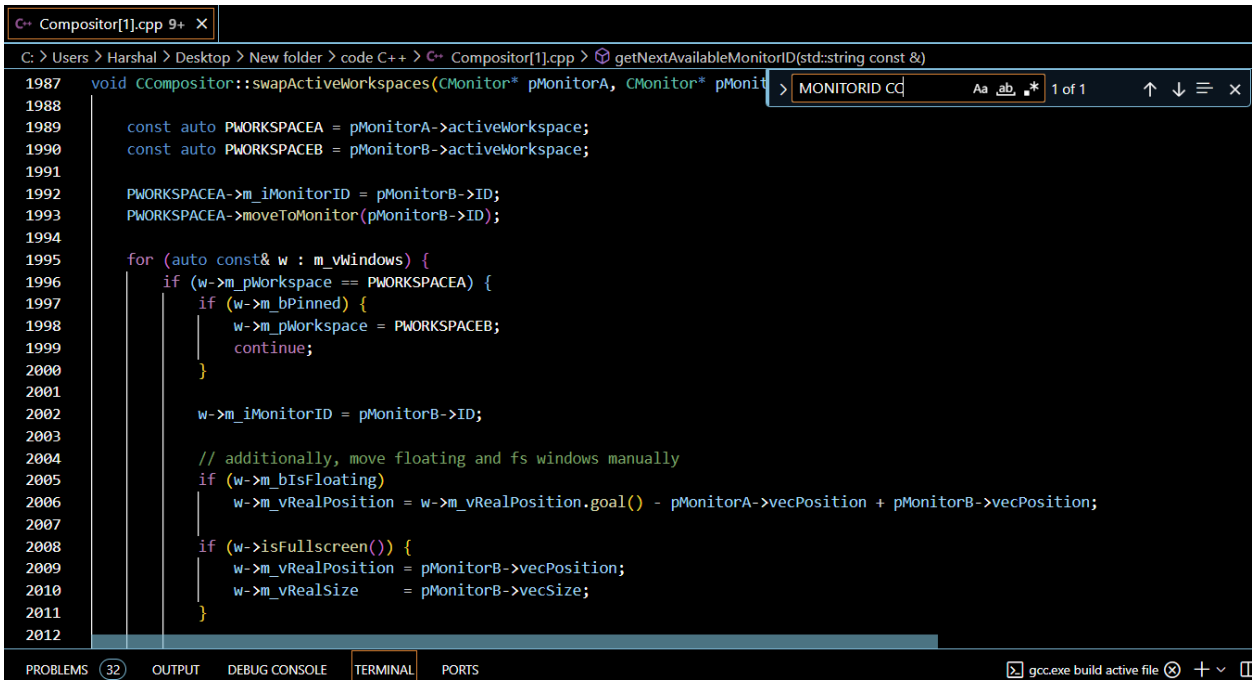
I. PROGRAM INSPECTION:

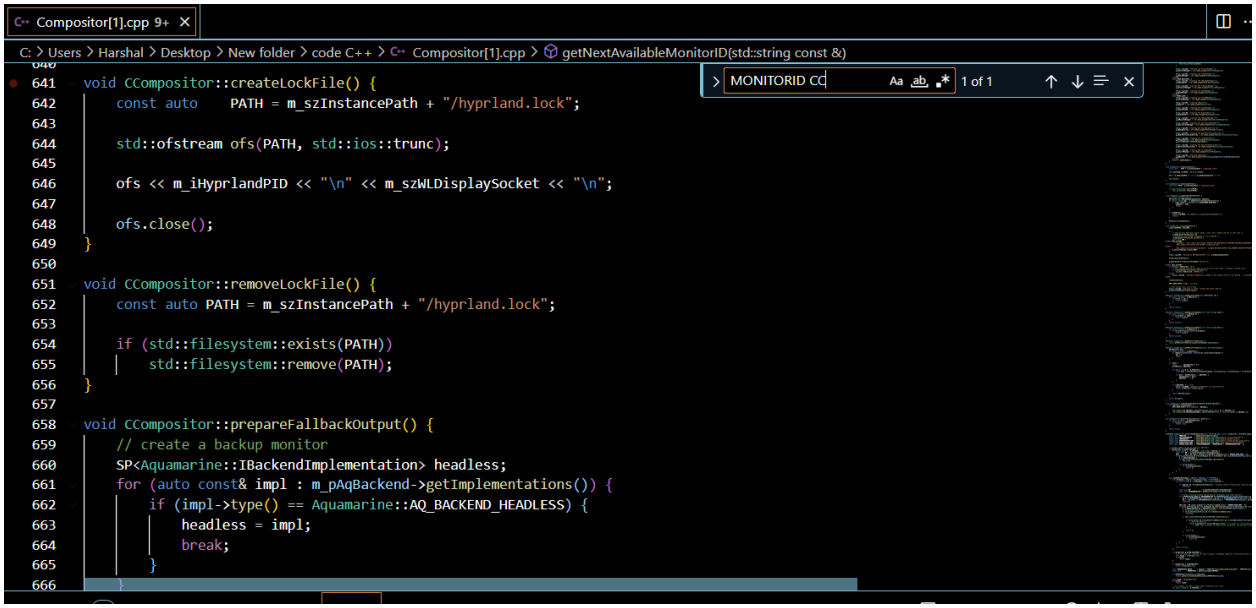
Debigging 2000 lines of
code:https://github.com/hyprwm/Hyprland/blob/main/src/Compositor.cpp

Category A:


Compositor[1].cpp 9+
C: > Users > Harshal > Desktop > New folder > code C++ > Compositor[1].cpp > arrangeMonitors()
void CCompositor::moveWindowToWorkspaceSafe(PHLWINDOW pWindow, PHLWORKSPACE
void CCompositor::arr
1 of 1
2729 void CCompositor::moveWindowToWorkspaceSafe(PHLWINDOW pWindow, PHLWORKSPACE
2739 if (FULLSCREEN)
2740 setWindowFullscreenInternal(pWindow, FSMODE_NONE);
2741
2742 if (!pWindow->m_bIsFloating) {
2743 g_pLayoutManager->getCurrentLayout()->onWindowRemovedTiling(pWindow);
2744 pWindow->moveToWorkspace(pWorkspace);
2745 pWindow->m_iMonitorID = pWorkspace->m_iMonitorID;
2746 g_pLayoutManager->getCurrentLayout()->onWindowCreatedTiling(pWindow);
2747 } else {
2748 const auto PWINDOWMONITOR = g_pCompositor->getMonitorFromID(pWindow->m_iMonitorID);
2749 const auto POSTOMON = pWindow->m_vRealPosition.goal() - PWINDOWMONITOR->vecPosition;
2750
2751 const auto PWORKSPACEMONITOR = g_pCompositor->getMonitorFromID(pWorkspace->m_iMonitorID);
2752
2753 pWindow->moveToWorkspace(pWorkspace);
2754 pWindow->m_iMonitorID = pWorkspace->m_iMonitorID;
2755
2756 pWindow->m_vRealPosition = POSTOMON + PWORKSPACEMONITOR->vecPosition;
2757 }
2758
2759 pWindow->updateToplevel();
2760 pWindow->updateDynamicRules();
2761 pWindow->uncacheWindowDecos();
2762
2763 if (!pWindow->m_sGroupData.pNextWindow.expired()) {


Compositor[1].cpp 9+
C: > Users > Harshal > Desktop > New folder > code C++ > Compositor[1].cpp > getNextAvailableMonitorID(std::string const &)
MONITORID CC
Aa ab .* 1 of 1
1678 PHLWINDOW CCompositor::getNextWindowOnWorkspace(PHLWINDOW pWindow, bool focu
1679 bool gotToWindow = false;
1680 for (auto const& w : m_vWindows) {
1681 if (w != pWindow && !gotToWindow)
1682 continue;
1683
1684 if (w == pWindow) {
1685 gotToWindow = true;
1686 continue;
1687 }
1688
1689 if (floating.has_value() && w->m_bIsFloating != floating.value())
1690 continue;
1691
1692 if (w->m_pWorkspace == pWindow->m_pWorkspace && w->m_bIsMapped && !w->isHidden() && (!focusableOnly || !w->m_sWindowData.noF
1693 return w;
1694 }
1695
1696 for (auto const& w : m_vWindows) {
1697 if (floating.has_value() && w->m_bIsFloating != floating.value())
1698 continue;
1699
1700 if (w != pWindow && w->m_pWorkspace == pWindow->m_pWorkspace && w->m_bIsMapped && !w->isHidden() && (!focusableOnly || !w->m
1701 return w;
1702 }
1703
PROBLEMS 32 OUTPUT DEBUG CONSOLE TERMINAL PORTS
gcc.exe build active file
#include "Compositor.hpp"


Compositor[1].cpp 9+
C: > Users > Harshal > Desktop > New folder > code C++ > Compositor[1].cpp > getNextAvailableMonitorID(std::string const &)
MONITORID CC
Aa ab .* 1 of 1
1987 void CCompositor::swapActiveWorkspaces(CMonitor* pMonitorA, CMonitor* pMonit
1988
1989 const auto PWORKSPACEA = pMonitorA->activeWorkspace;
1990 const auto PWORKSPACEB = pMonitorB->activeWorkspace;
1991
1992 PWORKSPACEA->m_iMonitorID = pMonitorB->ID;
1993 PWORKSPACEA->moveToMonitor(pMonitorB->ID);
1994
1995 for (auto const& w : m_vWindows) {
1996 if (w->m_pWorkspace == PWORKSPACEA) {
1997 if (w->m_bPinned) {
1998 w->m_pWorkspace = PWORKSPACEB;
1999 continue;
2000 }
2001
2002 w->m_iMonitorID = pMonitorB->ID;
2003
2004 // additionally, move floating and fs windows manually
2005 if (w->m_bIsFloating)
2006 w->m_vRealPosition = w->m_vRealPosition.goal() - pMonitorA->vecPosition + pMonitorB->vecPosition;
2007
2008 if (w->isFullscreen()) {
2009 w->m_vRealPosition = pMonitorB->vecPosition;
2010 w->m_vRealSize = pMonitorB->vecSize;
2011 }
2012
PROBLEMS 32 OUTPUT DEBUG CONSOLE TERMINAL PORTS
gcc.exe build active file



Category errors in all:
Potential Issues Identified:
Category A:
Line 2794

1. Pointer Initialization is wrong

Category B
Line : 1347

Out of scope

Category C
Line: 2739

3. Implicit Type Conversions that are wrong

4. The function performs conversions from strings to numbers

   Category E
   Moniterid Ccompositer line:

5. Boolean wrong Logic use
6. Infinite Loop.
7. Unreachable Code Segments

   Category F
   Line 1987:

8. `CCompositor::swapActiveWorkspaces()`, wrongly addressed

   Category G
   Line 641:

9. File handling error.

Effectiveness of Inspection Categories:

Commonality in C++

→uninitialized variables and null pointer dereferencing.

Subtle Bugs

Widespread Impact : A single error can impact many execution parts of the code related to it.

Errors Hard to Identify Through Inspection

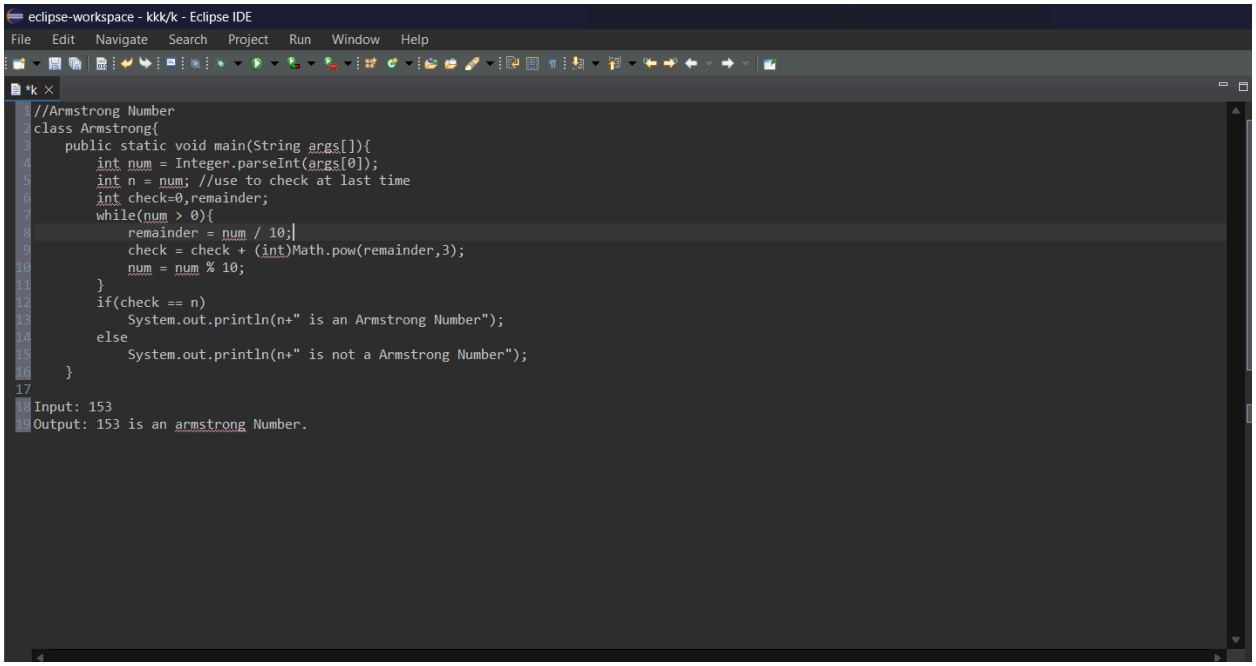
Some runtime errors are challenging to detect during inspections, including:

- Concurrency Issues: Race conditions and deadlocks.
- Performance Bottlenecks: Memory leaks affecting performance.
- Dynamic Memory Allocation Failures.
- File Handling Errors: Issues with external dependencies.
- Logic Errors: Resulting from unexpected user input.

Value of Program Inspection Techniques

Inspection techniques are crucial for identifying common problems early, such as data reference errors and logical inconsistencies, ultimately enhancing code reliability and maintainability.

Question 2: **Liines represent the break points**
Code 1 : Line 5



Errors in the Program:

- **Logical Error in Remainder Calculation:** The statement `remainder = num / 10;` is incorrect. It should be `remainder = num % 10;`
- **Logical Error in Updating num:** The statement `num = num % 10;` should be `num = num / 10;`

Effective Category of Program Inspection:

- The computation errors (Category C) would be the most effective here. Specifically, ensuring correct calculations in the loop.

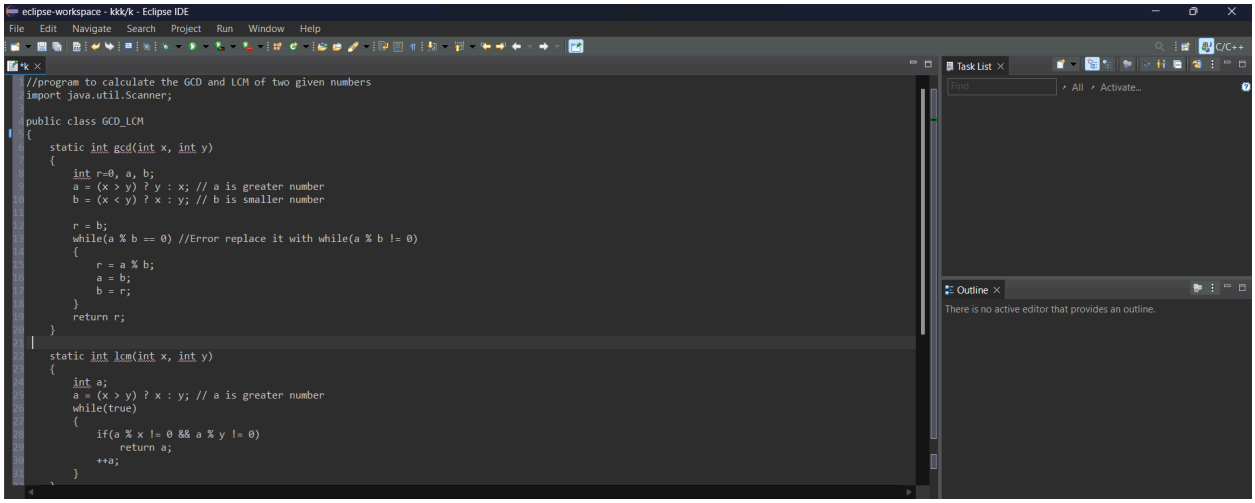
Type of Error Not Identified by Program Inspection:

- **Runtime Errors (e.g., Input Validation):** The program does not handle invalid input. For example, if no input or a non-numeric input is provided, the program will throw an exception. Such issues are difficult to identify solely through program inspection unless explicit checks for input are considered.

Applicability of the Program Inspection Technique:

- **Yes**, program inspection is worth applying in this case. It helps in catching logical mistakes, such as incorrect use of modulus and division operations.

- Code 2:**Error in line 5 and 21**

- 

Errors in the Program:

- **Error in the GCD Function:**
  - The condition inside the `while` loop should be `while(a % b != 0)` rather than `while(a % b == 0)` as already noted in the comment. This is because the loop should continue until a becomes divisible by b.
- **Error in the LCM Function:**
  - The condition `if(a % x != 0 && a % y != 0)` inside the `lcm()` method is incorrect. It should be `if(a % x == 0 && a % y == 0)` because the least common multiple is the smallest number divisible by both `x` and `y`..

Effective Category of Program Inspection:

- **Computation Errors (Category C)** are most effective for this code. This category will help identify the incorrect conditions in both the `gcd()` and `lcm()` functions, which lead to incorrect results.

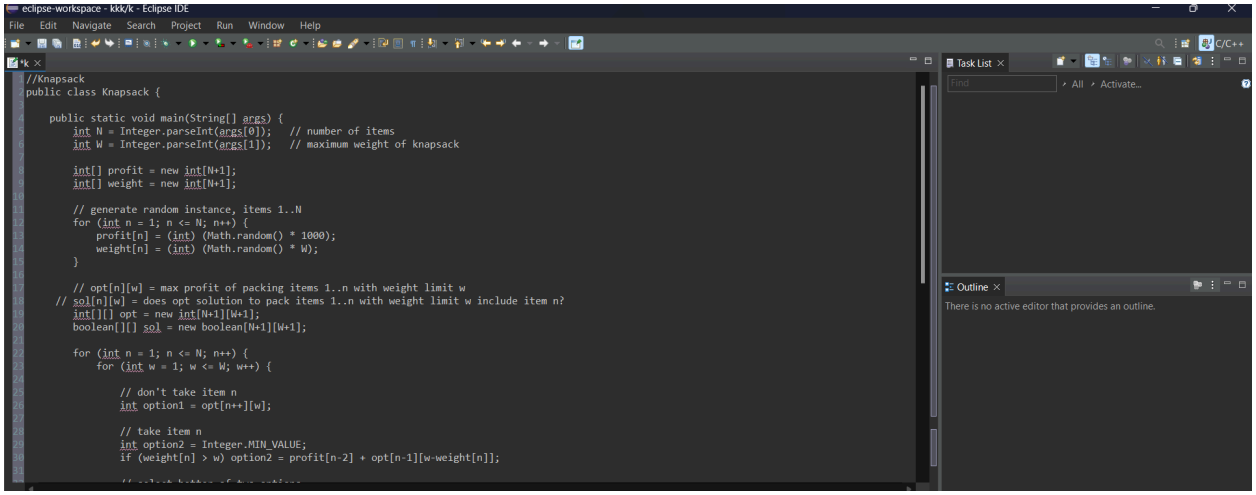
Type of Error Not Identified by Program Inspection:

- **Runtime Performance:** The `lcm()` function could potentially take a long time to compute for large inputs, and a more efficient method would be beneficial.

Applicability of the Program Inspection Technique:

- **Yes**

Code 3:Line 6 break point



1. Errors in the Program:

- **Error in Line 25:** The statement `int option1 = opt[n++][w];` should be `int option1 = opt[n-1][w].`
- **Error in Line 28:** The condition `if (weight[n] > w)` should be `if (weight[n] <= w)`. The logic is supposed to check if the weight of the item can fit within the current weight limit `w`, and only then the item can be considered.
- **Error in Line 29:** The expression `profit[n-2] + opt[n-1][w-weight[n]]` is incorrect because it should be `profit[n] + opt[n-1][w-weight[n]]`.

2. Effective Category of Program Inspection:

- **Control-flow errors (Category E)** and **computation errors (Category C)** are the most effective here.

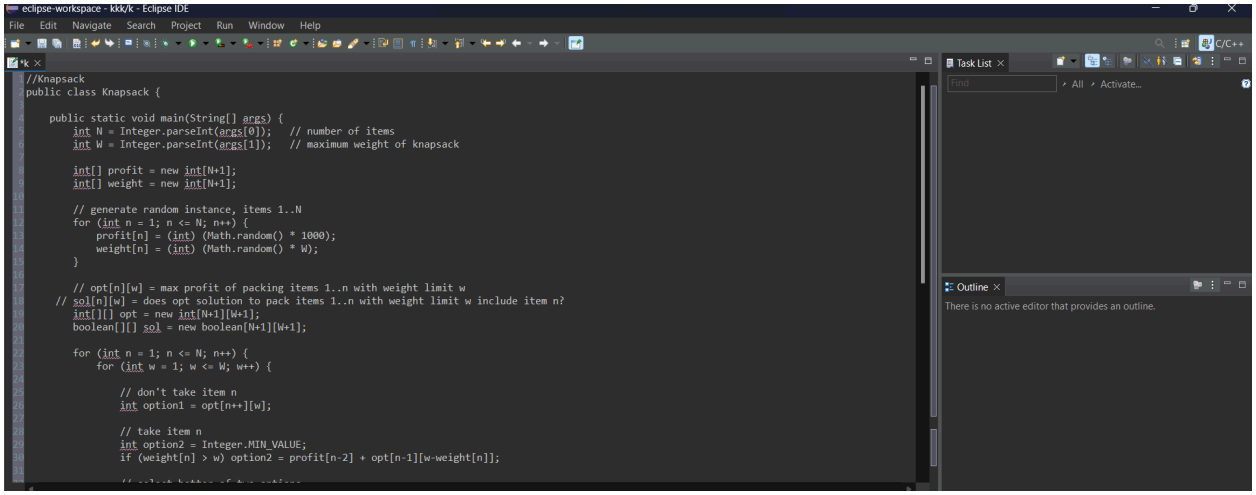
3. Type of Error Not Identified by Program Inspection:

- **Memory Constraints and Input Validation:** Program inspection alone does not check for edge cases like when `N = 0` or `W = 0`. This would cause the array sizes to become incorrect.

4. Applicability of Program Inspection:

- **Yes**

Code 4 Line 19 has error and 7 break point



1. Errors in the Program:

- **Error in Line 25:** The statement `int option1 = opt[n++][w];` should be `int option1 = opt[n-1][w];`
- **Error in Line 28:** The condition `if (weight[n] > w)` should be `if (weight[n] <= w)`. The logic is supposed to check if the weight of the item can fit within the current weight limit `w`, and only then the item can be considered.
- **Error in Line 29:** The expression `profit[n-2] + opt[n-1][w-weight[n]]` is incorrect because it should be `profit[n] + opt[n-1][w-weight[n]]`. `n-2` does not make sense in this context, and it will lead to wrong calculations.

2. Effective Category of Program Inspection:

- **Control-flow errors (Category E)** and **computation errors (Category C)** are the most effective here.

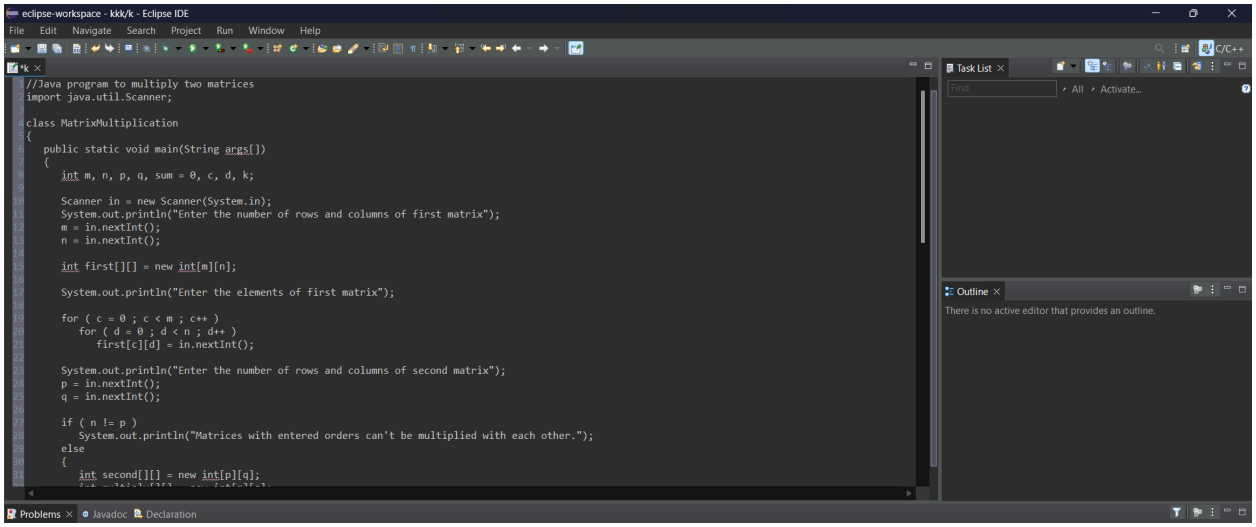
3. Type of Error Not Identified by Program Inspection:

- **Memory Constraints and Input Validation:** Program inspection alone does not check for edge cases like when `N = 0` or `W = 0`.

4. Applicability of Program Inspection:

- **Yes**

Code 5:



1. Errors in the Program:

- **Error in Inner While Loop (Line 12):** The condition `while(sum == 0)` is incorrect. It should be `while(sum > 0)` to continue extracting digits from the number.
- **Error in Line 13 (Digit Extraction):** The statement `s = s * (sum / 10)` is wrong The correct statement is `s = s + (sum % 10)`. To sum the individual digits of the number.
- **Syntax Error (Line 14):** The statement `sum = sum % 10` is missing a semicolon.
- **Initial Value of s in Inner Loop:** s should be initialized to 0 at the beginning of the loop, but it is initialized inside the loop and not resetting properly in subsequent iterations. The initialization should be `s = 0;` at the start of each iteration.

2. Effective Category of Program Inspection:

- **Computation Errors (Category C)** and **Control-flow errors (Category E)** are relevant here. The logic for calculating the sum of digits and control flow within the loops was incorrect, which leads to an infinite loop or incorrect results.

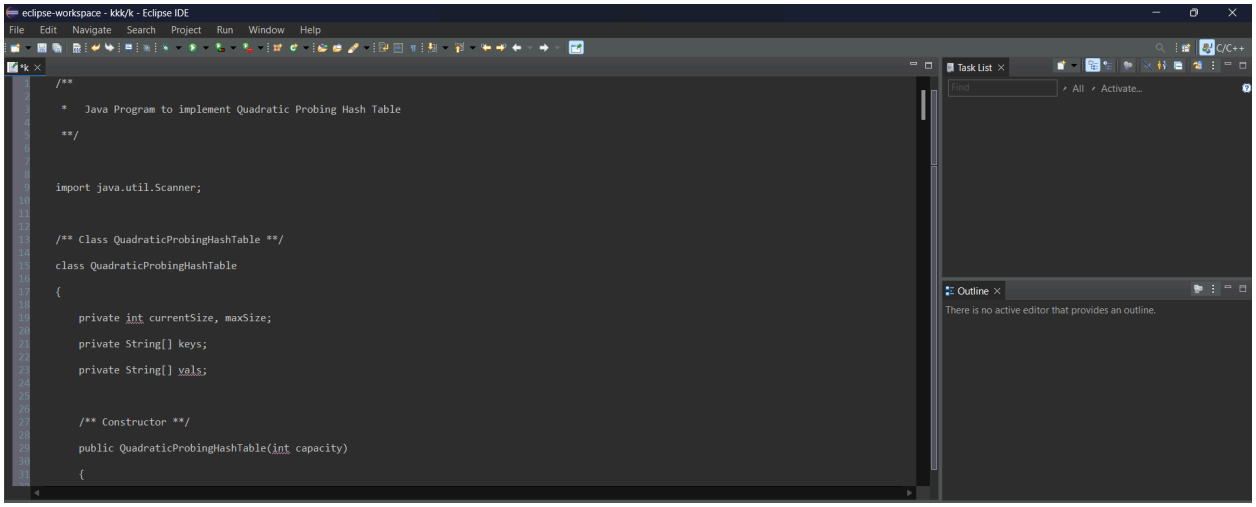
3. Type of Error Not Identified by Program Inspection:

- **Input Validation:** There's no input validation for negative numbers or non-integer inputs, which could cause unexpected behavior.

4. Applicability of the Program Inspection Technique:

- **Yes**

Code 6 :



1. Errors in the Program:

- **Error in Line 14:** The expression `int[] left = leftHalf(array+1);` is incorrect You should pass the array directly: `int[] left = leftHalf(array);`.
- **Error in Line 15:** Similarly, `int[] right = rightHalf(array-1);` should be `int[] right = rightHalf(array);`
- **Error in Line 22 (Merge Call):** The call `merge(array, left++, right--);` is incorrect. Correct `merge(array, left, right);`

2. Effective Category of Program Inspection:

- **Control-flow errors (Category E)** and **Data-reference errors (Category A)** are the most effective categories here.

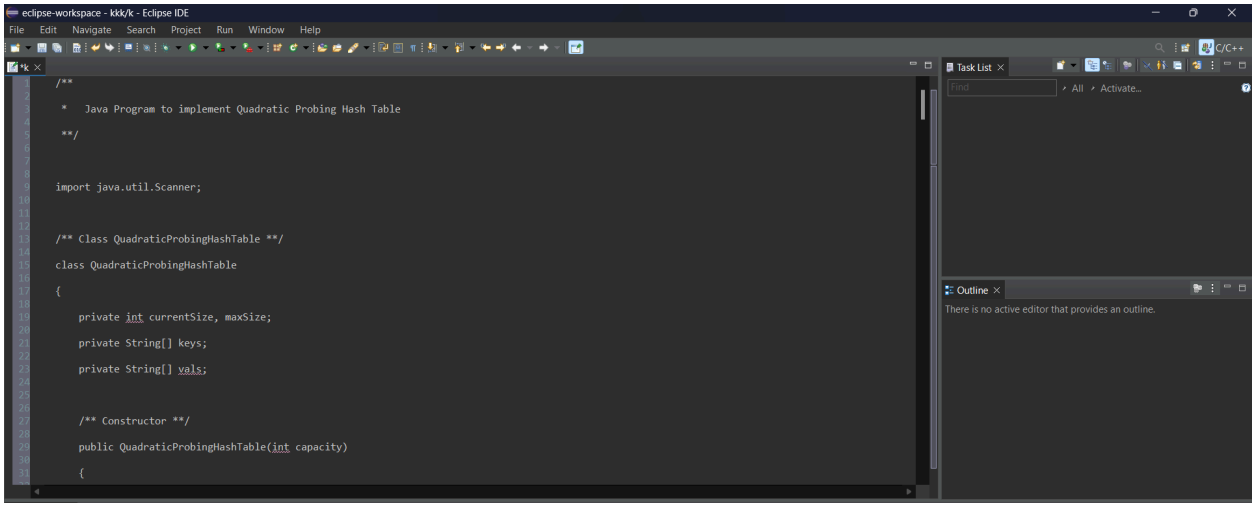
3. Type of Error Not Identified by Program Inspection:

- **Edge Case Handling:** Program inspection does not identify potential issues such as handling empty arrays or arrays with a single element.

4. Applicability of Program Inspection:

- **Yes**

Code 7:



1. Errors in the Program:

- **Error in Line 46 (Increment Operator):** `i + = (i + h / h--) % maxSize;` contains a syntax error. The += operator is incorrectly written as `+ =`. The correct line should be `i = (i + h * h++) % maxSize;` to correctly handle quadratic probing.
- **Rehash Issue in Line 96:** The rehashing logic after removing an element seems incorrect, the current implementation mixes up the index computation, leading to incorrect rehashing behavior.
- **Improper Decrementing of `currentSize` Twice (Line 102):** The `currentSize--` is called twice, both after rehashing and inside the `remove` function. This would reduce the count incorrectly.

2. Effective Category of Program Inspection:

- **Control-flow errors (Category E)** and **computation errors (Category C)** are the most relevant categories here.

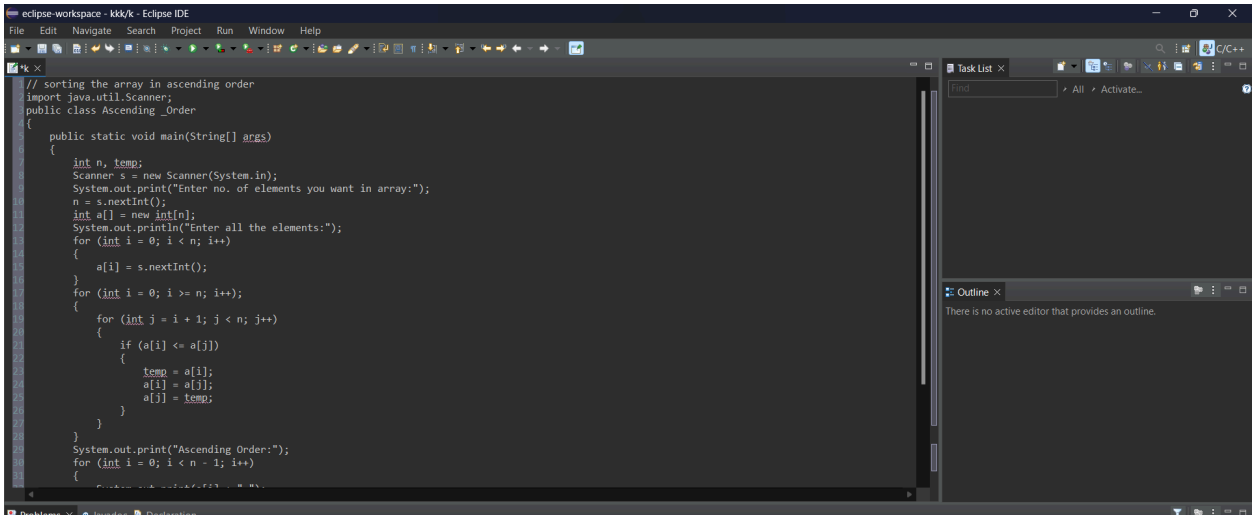
3. Type of Error Not Identified by Program Inspection:

- **Load Factor Consideration:** Program inspection doesn't highlight the need for handling load factors (how full the table is). At high load factors, rehashing the entire table with increased size would be optimal to avoid performance degradation.

4. Applicability of Program Inspection:

- **Yes**

Code 8:



1. Errors in the Program:

- **Error in Class Name (Line 3):** The class name contains a space (`Ascending _Order`). Spaces are not allowed in class names. It should be `AscendingOrder`.
- **Error in the Outer Loop Condition (Line 16):** The condition `for (int i = 0; i >= n; i++);` is incorrect. It should be `for (int i = 0; i < n; i++)` to iterate over the array. Also, there is an extra semicolon `;`

  Error in the Sorting Logic (Line 18): The condition `if (a[i] <= a[j])` is incorrect because this will perform a descending sort. It should be `if (a[i] > a[j])` to swap elements in ascending order.

2. Effective Category of Program Inspection:

- **Control-flow errors (Category E)** and **computation errors (Category C)** are relevant here.

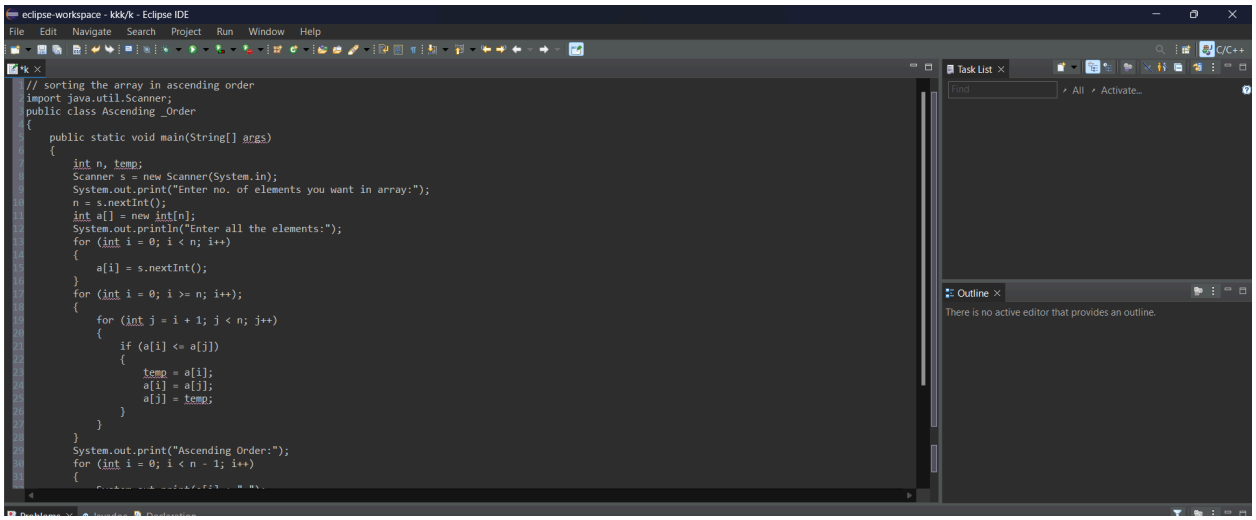
3. Type of Error Not Identified by Program Inspection:

- **Edge Case Handling:** The program does not handle edge cases like an array with one element or an empty array.

4. Applicability of Program Inspection:

- **Yes**

Code 9 :



1. Errors in the Program:

- **Error in `push()` Method (Line 15):** The line `top--;` should be `top++;`
- **Error in `display()` Method (Line 32):** The condition in the `for` loop `for(int i=0;i>top;i++)` is incorrect. The condition `i>top` will prevent the loop from running because the initial value of `i=0`. It should be `i <= top`.
- **Error in `pop()` Method (Line 24):** When popping from the stack, you should decrement `top` (i.e., `top--`), not increment it.

2. Effective Category of Program Inspection:

- **Control-flow errors (Category E)** and **computation errors (Category C)** are relevant here. These errors involve incorrectly updating the `top` index, causing the stack operations to misbehave.

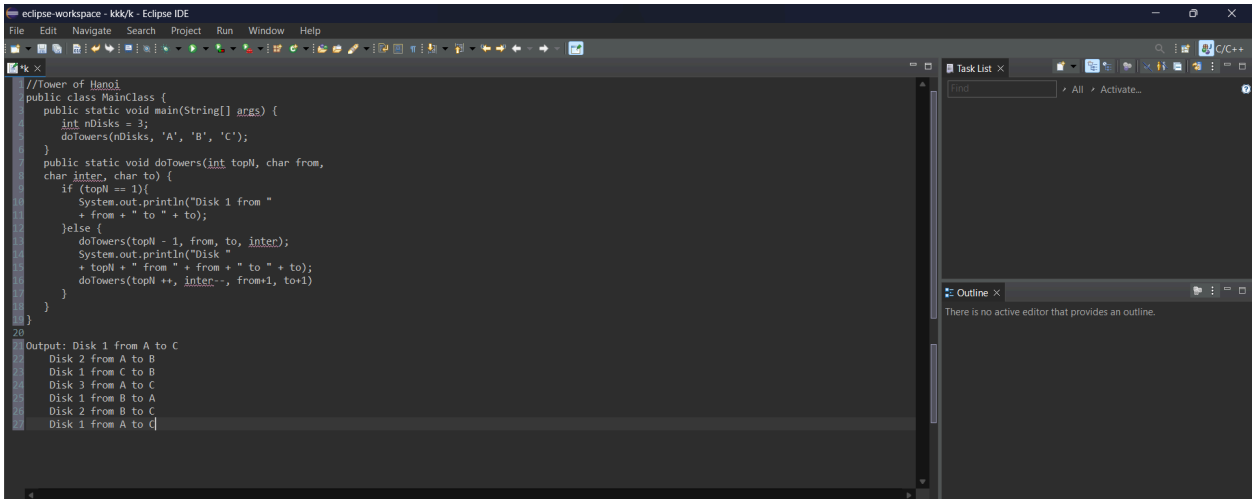
3. Type of Error Not Identified by Program Inspection:

- **Stack Underflow Handling:** The current implementation doesn't check for underflow conditions before displaying elements

4. Applicability of Program Inspection:

- **Yes**

Code 10:



1. Errors in the Program:

- **Error in the Recursive Call (Line 14):** In the recursive call `doTowers(topN ++, inter--, from+1, to+1)`, the increment (`++`) and decrement (`--`) operators are incorrectly applied.

  Error in the Recursive Call (Line 14): The correct call should pass the original rod characters directly as they represent the movement of disks between rods. Also, `topN ++` should simply be `topN - 1` in the second recursive call, just like the first one.

2. Effective Category of Program Inspection:

- **Control-flow errors (Category E)** are the most relevant here. The incorrect manipulation of parameters during the recursive calls causes logical errors.

3. Type of Error Not Identified by Program Inspection:

- **Input Validation:** The code doesn't handle cases where `nDisks <= 0`. This won't cause a runtime error but should be checked.

4. Applicability of Program Inspection:

- **Yes**