

Lab Assignment

Harshal Patel - 202201070

I. PROGRAM INSPECTION:

Debugging 2000 lines of

code:<https://github.com/hyprwm/Hyprland/blob/main/src/Compositor.cpp>

Category A:

```
C++ Compositor[1].cpp 9+ X
C: > Users > Harshal > Desktop > New folder > code C++ > C++ Compositor[1].cpp > getNextWindowOnWorkspace(PHWINDOW, bool, std::optional<bool>)

2794 void CCompositor::arrangeMonitors() {
2795     static auto* const PXMWFORCESCALEZERO = (Hyprlang::INT* const*)g_pConfigManager->getConfigValuePtr("xwayland:force_zero_scal
2796
2797     std::vector<CMonitor*> toArrange;
2798     std::vector<CMonitor*> arranged;
2799
2800     for (auto const& m : m_vMonitors)
2801         toArrange.push_back(m.get());
2802
2803     Debug::log(LOG, "arrangeMonitors: {} to arrange", toArrange.size());
2804
2805     for (auto it = toArrange.begin(); it != toArrange.end(); ) {
2806         auto m = *it;
2807
2808         if (m->activeMonitorRule.offset != Vector2D(-INT32_MAX, -INT32_MAX)) {
2809             // explicit.
2810             Debug::log(LOG, "arrangeMonitors: {} explicit {:?}", m->szName, m->activeMonitorRule.offset);
2811
2812             m->moveTo(m->activeMonitorRule.offset);
2813             arranged.push_back(m);
2814             it = toArrange.erase(it);
2815
2816             if (it == toArrange.end())
2817                 break;
2818             continue;
2819 }
```

```
C++ Compositor[1].cpp 9+ X
C: > Users > Harshal > Desktop > New folder > code C++ > C++ Compositor[1].cpp > getNextWindowOnWorkspace(PHWINDOW, bool, std::optional<bool>)

1347 PHWINDOW CCompositor::getTopLeftWindowOnWorkspace(const WORKSPACEID& id) {
1348     const auto PWORKSPACE = getWorkspaceByID(id);
1349
1350     if (!PWORKSPACE)
1351         return nullptr;
1352
1353     const auto PMONITOR = getMonitorFromID(PWORKSPACE->m_iMonitorID);
1354
1355     for (auto const& w : m_vWindows) {
1356         if (w->workspaceID() != id || !w->m_bIsMapped || w->isHidden())
1357             continue;
1358
1359         const auto WINDOWIDEALBB = w->getWindowIdealBoundingBoxIgnoreReserved();
1360
1361         if (WINDOWIDEALBB.x <= PMONITOR->vecPosition.x + 1 && WINDOWIDEALBB.y <= PMONITOR->vecPosition.y + 1)
1362             return w;
1363     }
1364     return nullptr;
1365 }
1366
1367 bool CCompositor::isWindowActive(PHWINDOW pWindow) {
1368     if (m_plastWindow.expired() && !m_plastFocus)
1369         return false;
1370
1371     if (!pWindow->m_bIsMapped)
1372         return false;
```

```
G++ Compositor[1].cpp 9+ X
C: > Users > Harshal > Desktop > New folder > code C++ > G++ Compositor[1].cpp > arrangeMonitors()
2729 void CCompositor::moveWindowToWorkspaceSafe(PHLOWINDOW pwindow, PHLWORKSPACE
> void CCompositor::arr Aa .ab . 1 of 1 ↑ ↓ ≡ ×

2739     if (FULLSCREEN)
2740     {
2741         setWindowFullscreenInternal(pwindow, FSMODE_NONE);
2742     }
2743     if (!pwindow->m_bIsFloating) {
2744         g_playoutManager->getCurrentLayout()->onWindowRemovedTiling(pwindow);
2745         pwindow->moveToWorkspace(pWorkspace);
2746         pwindow->m_iMonitorID = pWorkspace->m_iMonitorID;
2747         g_playoutManager->getCurrentLayout()->onWindowCreatedTiling(pwindow);
2748     } else {
2749         const auto PWINDOWMONITOR = g_pCompositor->getMonitorFromID(pwindow->m_iMonitorID);
2750         const auto POSTOMON = pwindow->m_vRealPosition.goal() - PWINDOWMONITOR->vecPosition;
2751
2752         const auto PWORKSPACEMONITOR = g_pCompositor->getMonitorFromID(pWorkspace->m_iMonitorID);
2753
2754         pwindow->moveToWorkspace(pWorkspace);
2755         pwindow->m_iMonitorID = pWorkspace->m_iMonitorID;
2756
2757         pwindow->m_vRealPosition = POSTOMON + PWORKSPACEMONITOR->vecPosition;
2758     }
2759
2760     pwindow->updateTopLevel();
2761     pwindow->updateDynamicRules();
2762     pwindow->uncacheWindowDecos();
2763
2764     if (!pwindow->m_sGroupData.pNextWindow.expired()) {
```

```
C++ Compositor[1].cpp 9+ X
C: > Users > Harshal > Desktop > New folder > code C++ > C++ Compositor[1].cpp > getNextAvailableMonitorID(std::string const &)
1678 PHLWINDOW CCompositor::getNextWindowOnWorkspace(PHLWINDOW pwindow, bool focus) {
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_bMapped && !w->isHidden() && (!focusableOnly || !w->m_sWindowData.noFocus))
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_bMapped && !w->isHidden() && (!focusableOnly || !w->m_sWindowData.noFocus))
1701             return w;
1702     }
1703 }
1704
#include "Compositor.h"
```

```
C++ Compositor[1].cpp 9+ X
C: > Users > Harshal > Desktop > New folder > code C++ > C++ Compositor[1].cpp > getNextAvailableMonitorID(std::string const &)
1987 void CCompositor::swapActiveWorkspaces(CMonitor* pMonitorA, CMonitor* pMonitorB) {
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         }
2013     }
2014 }
```

```
641 void CCompositor::createLockFile() {
642     const auto PATH = m_szInstancePath + "/hyprland.lock";
643
644     std::ofstream ofs(PATH, std::ios::trunc);
645
646     ofs << m_iHyprlandPID << "\n" << m_szWlDisplaySocket << "\n";
647
648     ofs.close();
649 }
650
651 void CCompositor::removeLockFile() {
652     const auto PATH = m_szInstancePath + "/hyprland.lock";
653
654     if (std::filesystem::exists(PATH))
655         std::filesystem::remove(PATH);
656 }
657
658 void CCompositor::prepareFallbackOutput() {
659     // create a backup monitor
660     SP<Aquamarine::IBackendImplementation> headless;
661     for (auto const& impl : m_pAqBackend->getImplementations()) {
662         if (impl->type() == Aquamarine::AQ_BACKEND_HEADLESS) {
663             headless = impl;
664             break;
665         }
666     }
```

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:

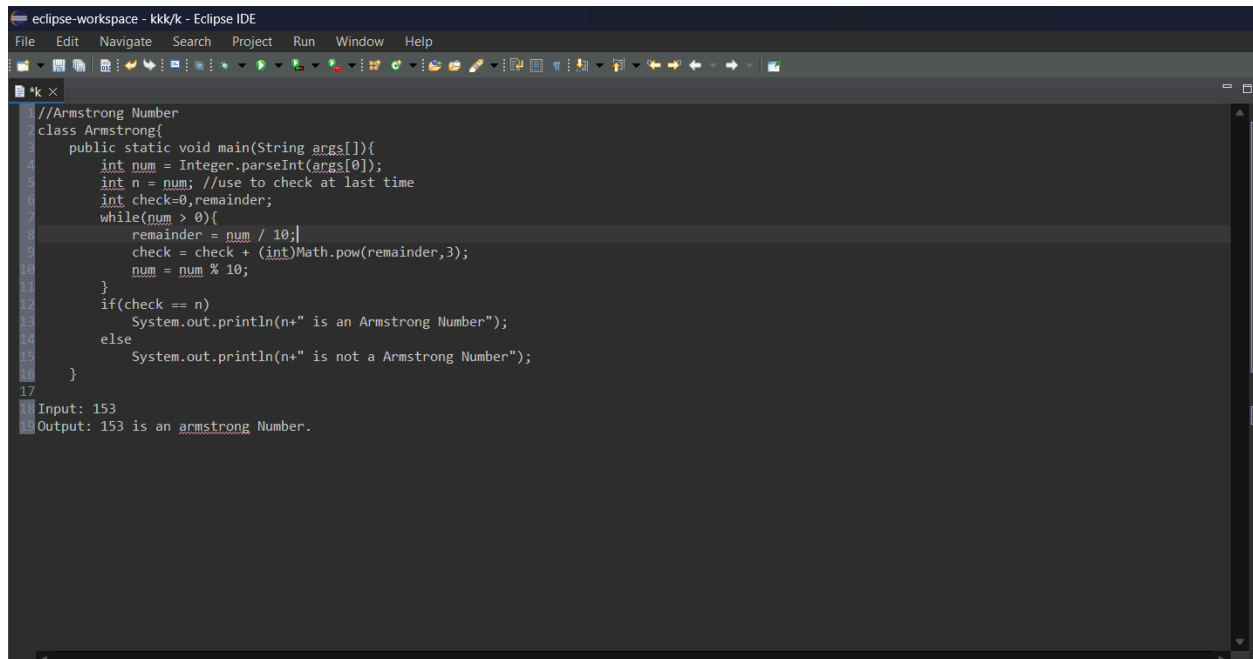
- Concurrency Issues: Race conditions and deadlocks.
- Performance Bottlenecks: Memory leaks affecting performance.
- Dynamic Memory Allocation Failures.
- File Handling Errors: Issues with external dependencies.
- Logic Errors: Resulting from unexpected user input.

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: **Lines represent the break points**

Code 1 : Line 5



```
//Armstrong Number
class Armstrong{
    public static void main(String args[]){
        int num = Integer.parseInt(args[0]);
        int n = num; //use to check at last time
        int check=0,remainder;
        while(num > 0){
            remainder = num / 10;
            check = check + (int)Math.pow(remainder,3);
            num = num % 10;
        }
        if(check == n)
            System.out.println(n+" is an Armstrong Number");
        else
            System.out.println(n+" is not a Armstrong Number");
    }
}
Input: 153
Output: 153 is an armstrong Number.
```

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**

```
//program to calculate the GCD and LCM of two given numbers
import java.util.Scanner;

public class GCD_LCM
{
    static int gcd(int x, int y)
    {
        int r=0, a, b;
        a = (x > y) ? y : x; // a is greater number
        b = (x < y) ? x : y; // b is smaller number

        r = b;
        while(a % b == 0) //Error replace it with while(a % b != 0)
        {
            r = a % b;
            a = b;
            b = r;
        }
        return r;
    }

    static int lcm(int x, int y)
    {
        int a;
        a = (x > y) ? x : y; // a is greater number
        while(true)
        {
            if(a % x != 0 && a % y != 0)
                return a;
            ++a;
        }
    }
}
```

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.

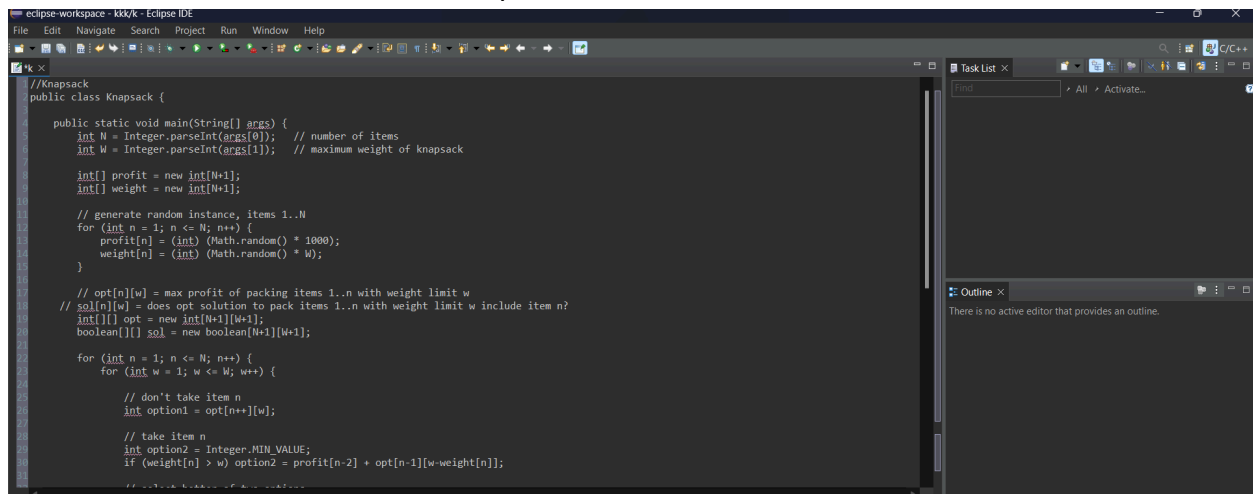
3. Type of Error Not Identified by Program Inspection:

- **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



```
//Knapsack
public class Knapsack {

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]); // number of items
        int W = Integer.parseInt(args[1]); // maximum weight of knapsack

        int[] profit = new int[N+1];
        int[] weight = new int[N+1];

        // generate random instance, items 1..N
        for (int n = 1; n <= N; n++) {
            profit[n] = (int) (Math.random() * 1000);
            weight[n] = (int) (Math.random() * W);
        }

        // opt[n][w] = max profit of packing items 1..n with weight limit w
        // sol[n][w] = does opt solution to pack items 1..n with weight limit w include item n?
        int[][] opt = new int[N+1][W+1];
        boolean[][] sol = new boolean[N+1][W+1];

        for (int n = 1; n <= N; n++) {
            for (int w = 1; w <= W; w++) {

                // don't take item n
                int option1 = opt[n+1][w];

                // take item n
                int option2 = Integer.MIN_VALUE;
                if (weight[n] > w) option2 = profit[n-2] + opt[n-1][w-weight[n]];

                // if (weight[n] <= w) option2 = profit[n] + opt[n-1][w-weight[n]];

                if (option1 > option2) {
                    opt[n][w] = option1;
                    sol[n][w] = false;
                } else {
                    opt[n][w] = option2;
                    sol[n][w] = true;
                }
            }
        }

        // print the results of the solution
    }
}
```

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.

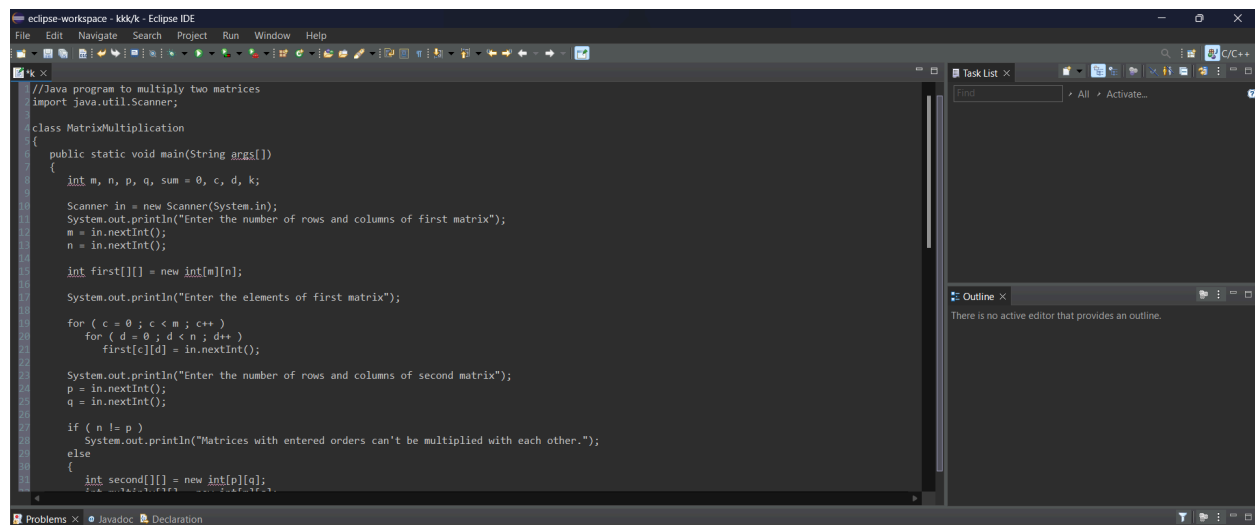
3. Type of Error Not Identified by Program Inspection:

- **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:



```
//Java program to multiply two matrices
import java.util.Scanner;

class MatrixMultiplication
{
    public static void main(String args[])
    {
        int m, n, p, q, sum = 0, c, d, k;

        Scanner in = new Scanner(System.in);
        System.out.println("Enter the number of rows and columns of first matrix");
        m = in.nextInt();
        n = in.nextInt();

        int first[][] = new int[m][n];

        System.out.println("Enter the elements of first matrix");
        for ( c = 0 ; c < m ; c++)
            for ( d = 0 ; d < n ; d++)
                first[c][d] = in.nextInt();

        System.out.println("Enter the number of rows and columns of second matrix");
        p = in.nextInt();
        q = in.nextInt();

        if ( n != p )
            System.out.println("Matrices with entered orders can't be multiplied with each other.");
        else
        {
            int second[][] = new int[p][q];
            // ... (rest of the code is partially obscured)
        }
    }
}
```

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.

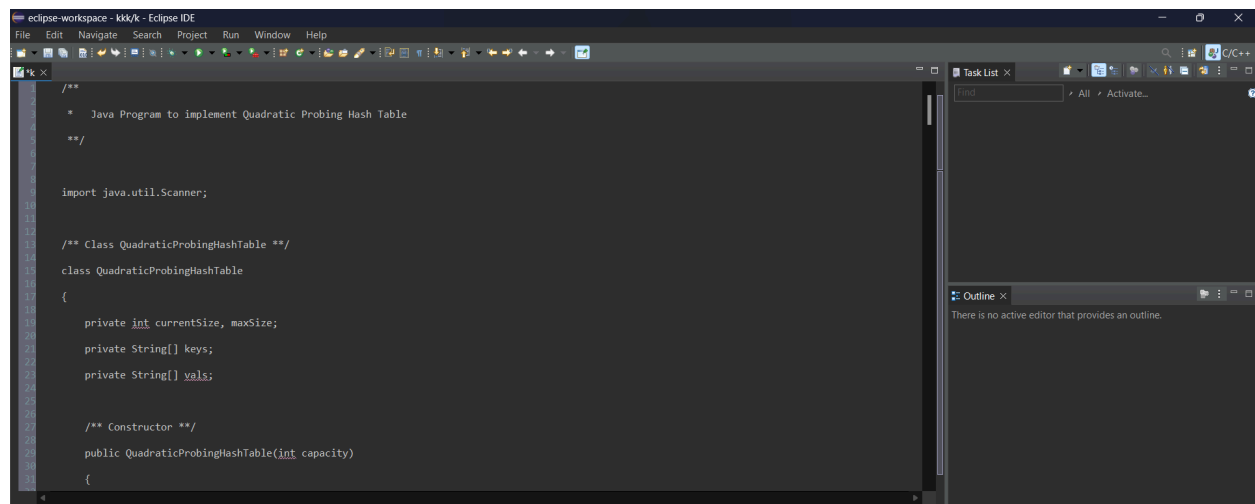
3. Type of Error Not Identified by Program Inspection:

- **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 :

The screenshot shows the Eclipse IDE interface. The main editor window displays a Java file named 'QuadraticProbingHashTable.java'. The code includes a package declaration, imports, a class declaration, and a constructor. The code is as follows:

```
1  /**  
2   * Java Program to implement Quadratic Probing Hash Table  
3   **/  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  import java.util.Scanner;  
14  
15  
16  
17  /** Class QuadraticProbingHashTable **/  
18  
19  class QuadraticProbingHashTable  
20  {  
21  
22      private int currentSize, maxSize;  
23      private String[] keys;  
24      private String[] vals;  
25  
26  
27      /** Constructor **/  
28      public QuadraticProbingHashTable(int capacity)  
29      {  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

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.

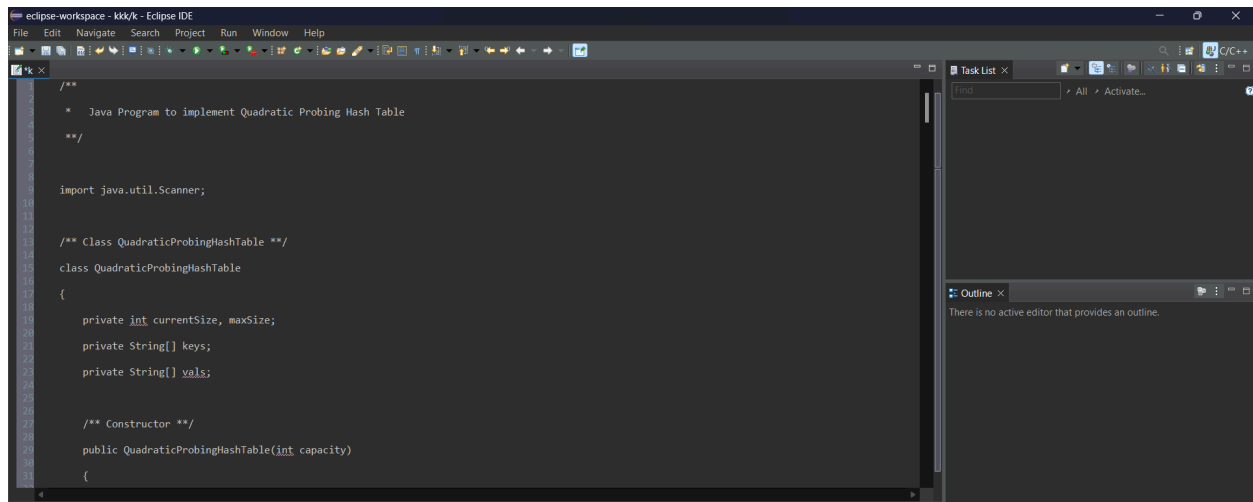
3. Type of Error Not Identified by Program Inspection:

- **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:

The screenshot shows the Eclipse IDE interface. The main editor window displays a Java file named 'QuadraticProbingHashTable.java'. The code includes a package declaration, imports, and a class definition. The class has private attributes for 'currentSize', 'maxSize', 'keys', and 'vals'. It also has a constructor 'QuadraticProbingHashTable(int capacity)'. The code is partially visible, showing lines from 1 to 31. The right-hand side of the IDE shows the 'Task List' and 'Outline' views, both of which are currently empty.

```
1  /**
2   * Java Program to implement Quadratic Probing Hash Table
3   */
4
5
6
7
8
9
10
11
12
13 import java.util.Scanner;
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
```

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.

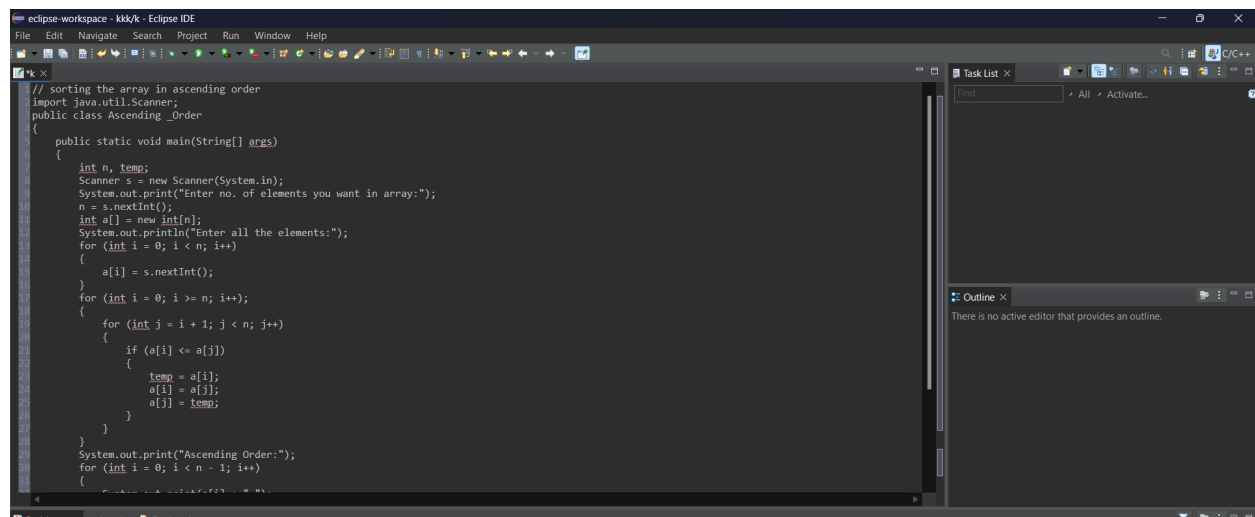
3. Type of Error Not Identified by Program Inspection:

- **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:



```
// Sorting the array in ascending order
import java.util.Scanner;
public class Ascending_Order
{
    public static void main(String[] args)
    {
        int n, temp;
        Scanner s = new Scanner(System.in);
        System.out.print("Enter no. of elements you want in array:");
        n = s.nextInt();
        int a[] = new int[n];
        System.out.println("Enter all the elements:");
        for (int i = 0; i < n; i++)
        {
            a[i] = s.nextInt();
        }
        for (int i = 0; i >= n; i++)
        {
            for (int j = i + 1; j < n; j++)
            {
                if (a[i] <= a[j])
                {
                    temp = a[i];
                    a[i] = a[j];
                    a[j] = temp;
                }
            }
        }
        System.out.print("Ascending Order:");
        for (int i = 0; i < n - 1; i++)
        {
            // ...
        }
    }
}
```

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.

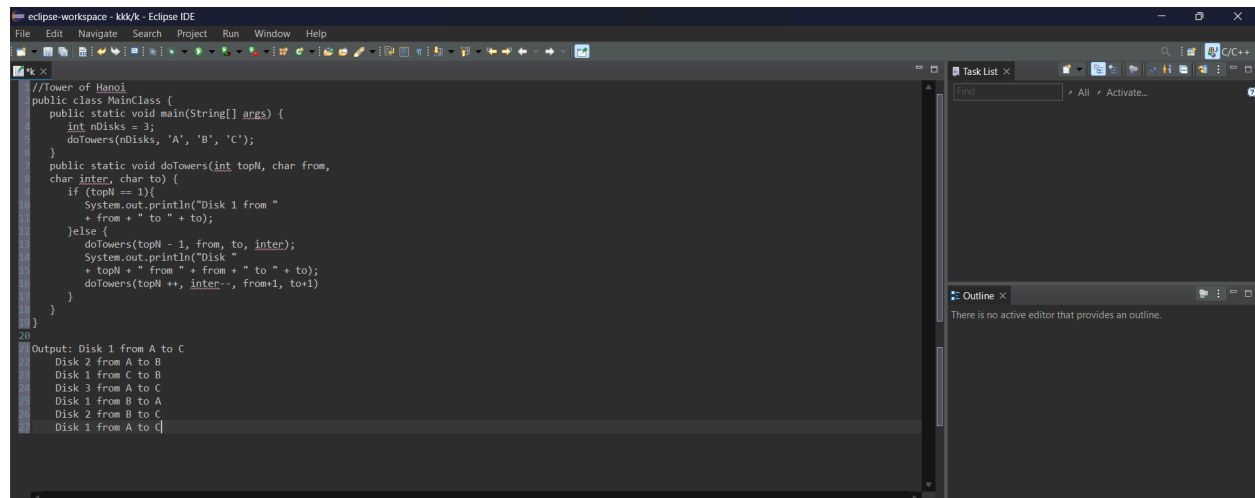
3. Type of Error Not Identified by Program Inspection:

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

4. Applicability of Program Inspection:

- Yes

Code 10:



```
//Tower of Hanoi
public class MainClass {
    public static void main(String[] args) {
        int nDisks = 3;
        doTowers(nDisks, 'A', 'B', 'C');
    }
    public static void doTowers(int topN, char from,
        char inter, char to) {
        if (topN == 1){
            System.out.println("Disk 1 from "
                + from + " to " + to);
        }
        else {
            doTowers(topN - 1, from, to, inter);
            System.out.println("Disk "
                + topN + " from " + from + " to " + to);
            doTowers(topN - 1, inter, from, to);
        }
    }
}

Output: Disk 1 from A to C
        Disk 2 from A to B
        Disk 1 from C to B
        Disk 3 from A to C
        Disk 1 from B to A
        Disk 2 from B to C
        Disk 1 from A to C
```

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