IT314 Software Engineering

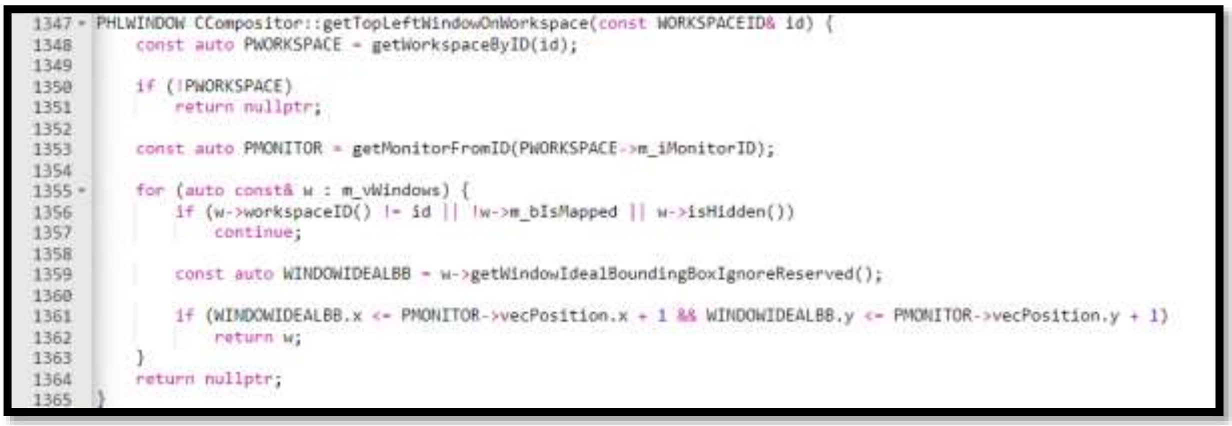
Lab 7: Program Inspection, Debugging and Static Analysis

Name: Rit Rajendra Trambadia **Id: 202201424**

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

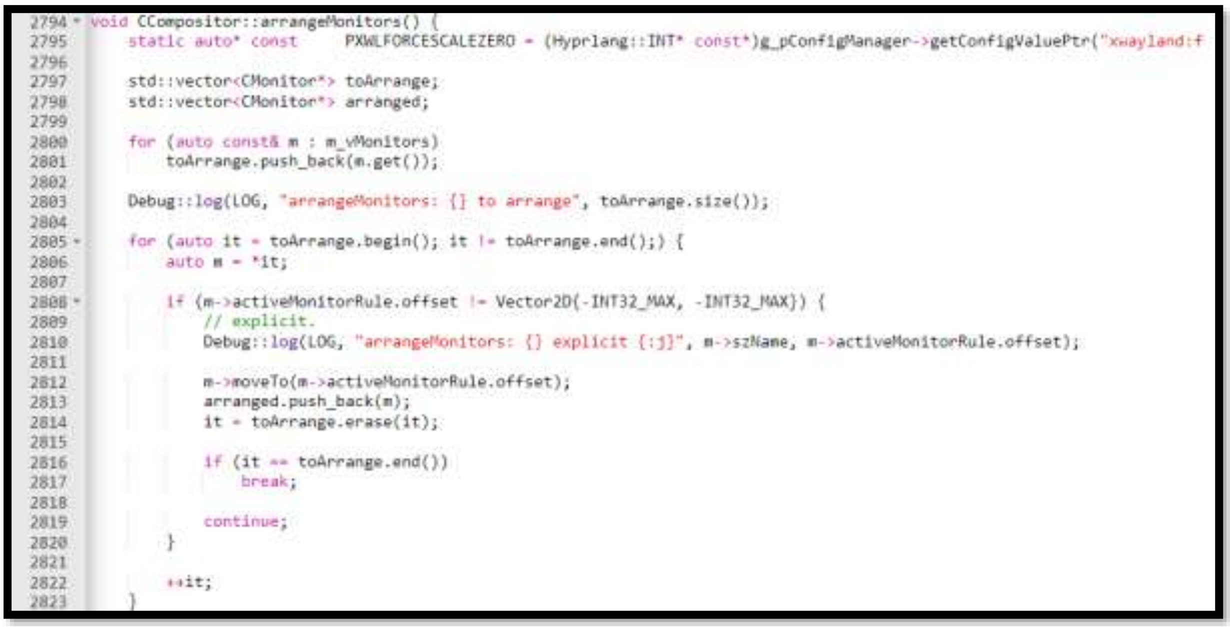
Program Inspection

- Category A



```cpp
PHLWINDOW CCompositor::getTopLeftWindowOnWorkspace(const WORKSPACEID& id) {
    const auto PWORKSPACE = getWorkspaceByID(id);

    if (!PWORKSPACE)
        return nullptr;

    const auto PMONITOR = getMonitorFromID(PWORKSPACE->m_iMonitorID);

    for (auto const& w : m_vWindows) {
        if (w->workspaceID() != id || !w->m_bIsMapped || w->isHidden())
            continue;

        const auto WINDOWIDEALBB = w->getWindowIdealBoundingBoxIgnoreReserved();

        if (WINDOWIDEALBB.x <= PMONITOR->vecPosition.x + 1 && WINDOWIDEALBB.y <= PMONITOR->vecPosition.y + 1)
            return w;
    }
    return nullptr;
}
```

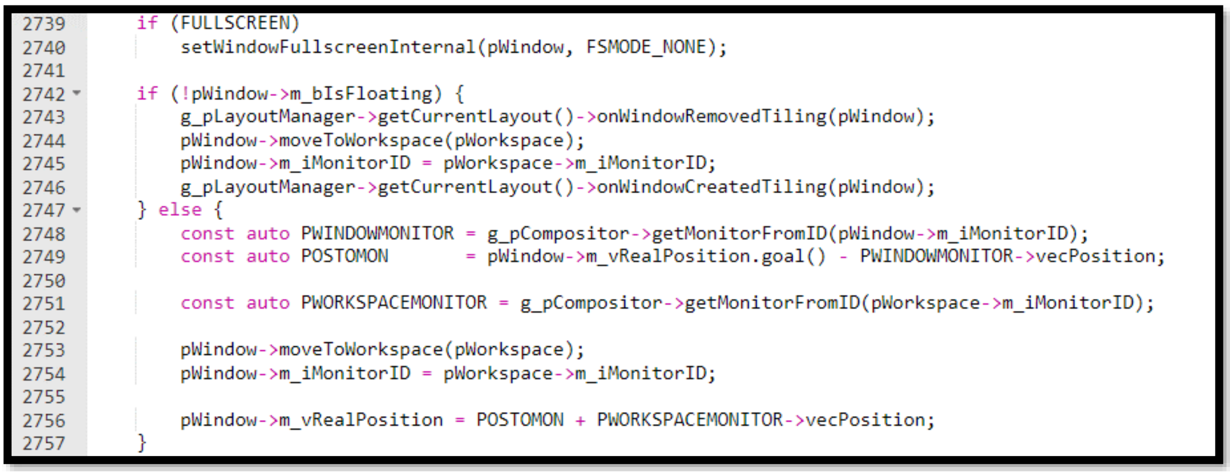
The pointer PMONITOR is initialized, but there is no guarantee that it won't be null, which may result in a null reference.



```cpp
void CCompositor::arrangeMonitors() {
    static auto* const     PXWLFORCESCALEZERO = (Hyprlang::INT* const*)g_pConfigManager->getConfigValuePtr("xwayland:f

    std::vector<CMonitor*> toArrange;
    std::vector<CMonitor*> arranged;

    for (auto const& m : m_vMonitors)
        toArrange.push_back(m.get());

    Debug::log(LOG, "arrangeMonitors: {} to arrange", toArrange.size());

    for (auto it = toArrange.begin(); it != toArrange.end();) {
        auto m = *it;

        if (m->activeMonitorRule.offset != Vector2D(-INT32_MAX, -INT32_MAX)) {
            // explicit.
            Debug::log(LOG, "arrangeMonitors: {} explicit {:j}", m->szName, m->activeMonitorRule.offset);

            m->moveTo(m->activeMonitorRule.offset);
            arranged.push_back(m);
            it = toArrange.erase(it);

            if (it == toArrange.end())
                break;

            continue;
        }

        ++it;
    }
```

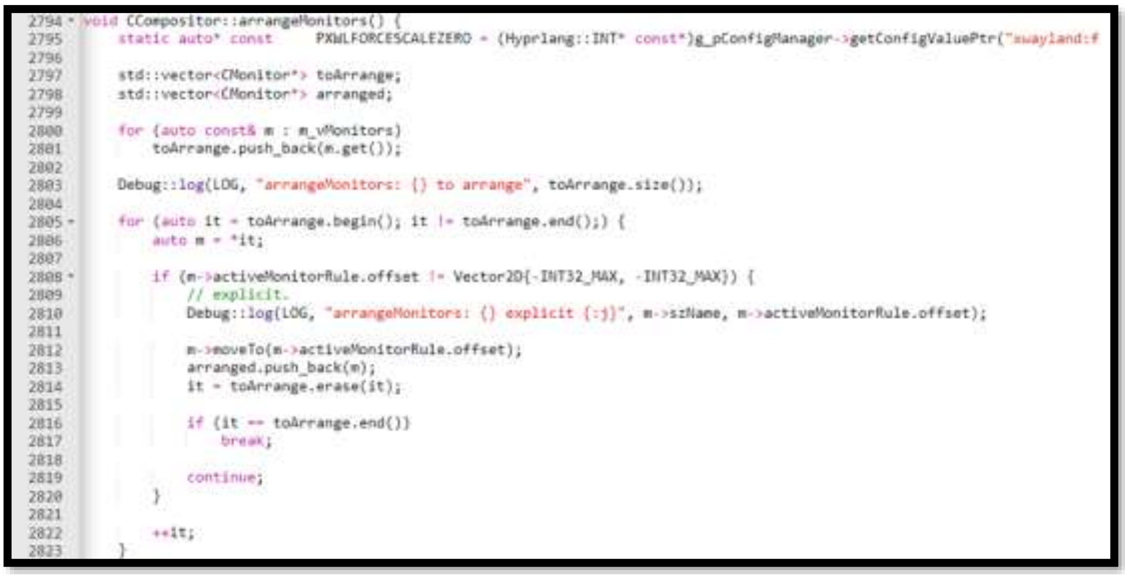
In methods such as CCompositor::arrangeMonitors(), loops access elements in arrays or lists (like m_lMonitors). Since there are no explicit checks on array boundaries, there is a potential risk of out-of-bounds access, particularly if the list is empty or smaller than anticipated.

- Category B



```
    if (FULLSCREEN)
        setWindowFullscreenInternal(pWindow, FSMODE_NONE);

    if (!pWindow->m_bIsFloating) {
        g_pLayoutManager->getCurrentLayout()->onWindowRemovedTiling(pWindow);
        pWindow->moveToWorkspace(pWorkspace);
        pWindow->m_iMonitorID = pWorkspace->m_iMonitorID;
        g_pLayoutManager->getCurrentLayout()->onWindowCreatedTiling(pWindow);
    } else {
        const auto PWINDOWMONITOR = g_pCompositor->getMonitorFromID(pWindow->m_iMonitorID);
        const auto POSTOMON       = pWindow->m_vRealPosition.goal() - PWINDOWMONITOR->vecPosition;

        const auto PWORKSPACEMONITOR = g_pCompositor->getMonitorFromID(pWorkspace->m_iMonitorID);

        pWindow->moveToWorkspace(pWorkspace);
        pWindow->m_iMonitorID = pWorkspace->m_iMonitorID;

        pWindow->m_vRealPosition = POSTOMON + PWORKSPACEMONITOR->vecPosition;
    }
```

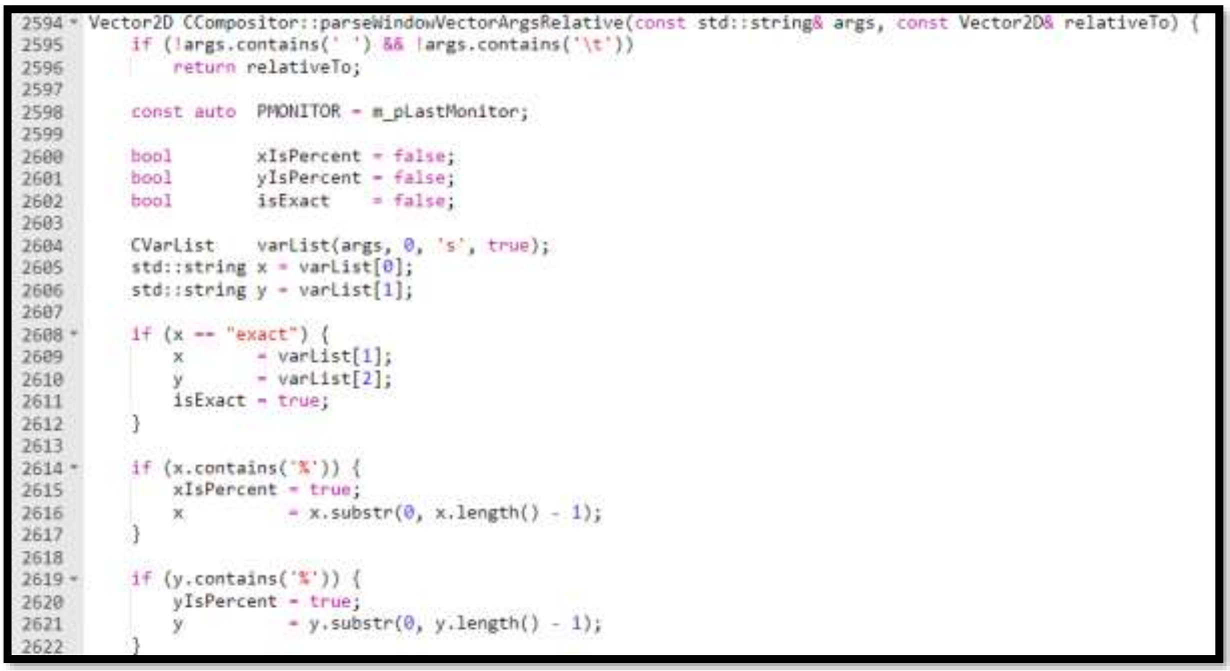
Implicit conversion issues might arise in the given snippet when handling the POSTOMON variable, especially if vecPosition is not fully compatible with the assigned type.



```
void CCompositor::arrangeMonitors() {
    static auto* const      PXWLFORCESCALEZERO = (Hyprlang::INT* const*)g_pConfigManager->getConfigValuePtr("xwayland:f

    std::vector<CMonitor*> toArrange;
    std::vector<CMonitor*> arranged;

    for (auto const& m : m_vMonitors)
        toArrange.push_back(m.get());

    Debug::log(LOG, "arrangeMonitors: {} to arrange", toArrange.size());

    for (auto it = toArrange.begin(); it != toArrange.end();) {
        auto m = *it;

        if (m->activeMonitorRule.offset != Vector2D{-INT32_MAX, -INT32_MAX}) {
            // explicit.
            Debug::log(LOG, "arrangeMonitors: {} explicit {:j}", m->szName, m->activeMonitorRule.offset);

            m->moveTo(m->activeMonitorRule.offset);
            arranged.push_back(m);
            it = toArrange.erase(it);

            if (it == toArrange.end())
                break;

            continue;
        }

        ++it;
    }
```

The variable `m` is used in the given snippet, but since it is also used in several other places, it could lead to potential issues due to variable shadowing in different scopes.

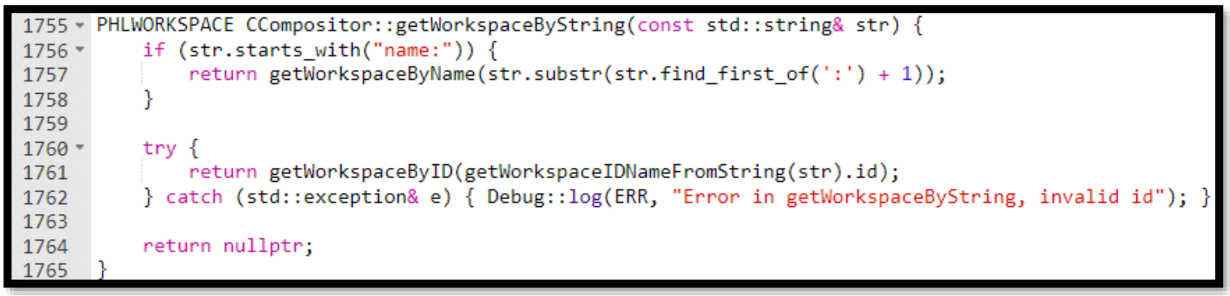
- Category C



```cpp
Vector2D CCompositor::parseWindowVectorArgsRelative(const std::string& args, const Vector2D& relativeTo) {
    if (!args.contains(' ') && !args.contains('\t'))
        return relativeTo;

    const auto  PMONITOR = m_pLastMonitor;

    bool        xIsPercent = false;
    bool        yIsPercent = false;
    bool        isExact    = false;

    CVarList    varList(args, 0, 's', true);
    std::string x = varList[0];
    std::string y = varList[1];

    if (x == "exact") {
        x       = varList[1];
        y       = varList[2];
        isExact = true;
    }

    if (x.contains('%')) {
        xIsPercent = true;
        x          = x.substr(0, x.length() - 1);
    }

    if (y.contains('%')) {
        yIsPercent = true;
        y          = y.substr(0, y.length() - 1);
    }
```

The function involves string-to-number conversions and performs operations on mixed types (such as floats and integers), which could result in unintended rounding or truncation errors.

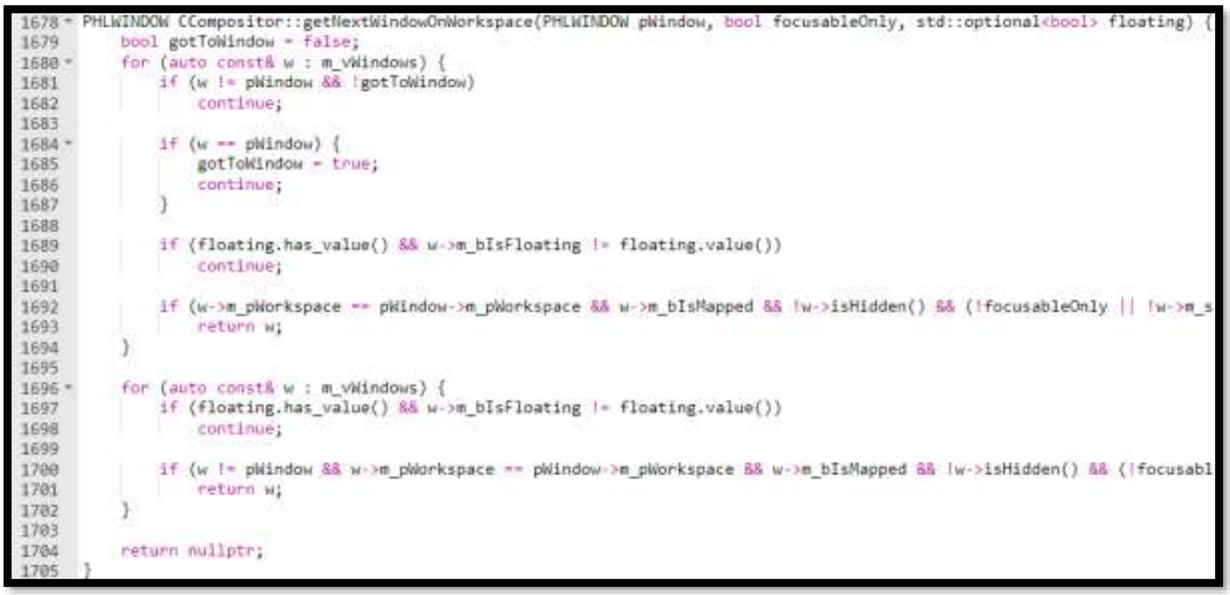
- Category D



```cpp
PHLWORKSPACE CCompositor::getWorkspaceByString(const std::string& str) {
    if (str.starts_with("name:")) {
        return getWorkspaceByName(str.substr(str.find_first_of(':') + 1));
    }

    try {
        return getWorkspaceByID(getWorkspaceIDNameFromString(str).id);
    } catch (std::exception& e) { Debug::log(ERR, "Error in getWorkspaceByString, invalid id"); }

    return nullptr;
}
```

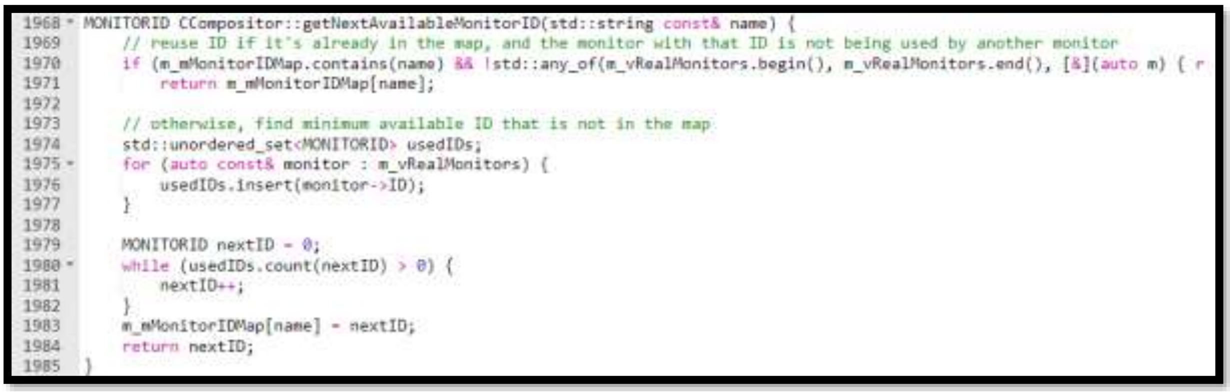
The logic involving str.starts_with("name:") and its exception handling may fail if the string format is incorrect, potentially causing unexpected behavior.

- Category E



```cpp
PHLWINDOW CCompositor::getNextWindowOnWorkspace(PHLWINDOW pWindow, bool focusableOnly, std::optional<bool> floating) {
    bool gotToWindow = false;
    for (auto const& w : m_vWindows) {
        if (w != pWindow && !gotToWindow)
            continue;

        if (w == pWindow) {
            gotToWindow = true;
            continue;
        }

        if (floating.has_value() && w->m_bIsFloating != floating.value())
            continue;

        if (w->m_pWorkspace == pWindow->m_pWorkspace && w->m_bIsMapped && !w->isHidden() && (!focusableOnly || !w->m_s
            return w;
    }

    for (auto const& w : m_vWindows) {
        if (floating.has_value() && w->m_bIsFloating != floating.value())
            continue;

        if (w != pWindow && w->m_pWorkspace == pWindow->m_pWorkspace && w->m_bIsMapped && !w->isHidden() && (!focusabl
            return w;
    }

    return nullptr;
}
```

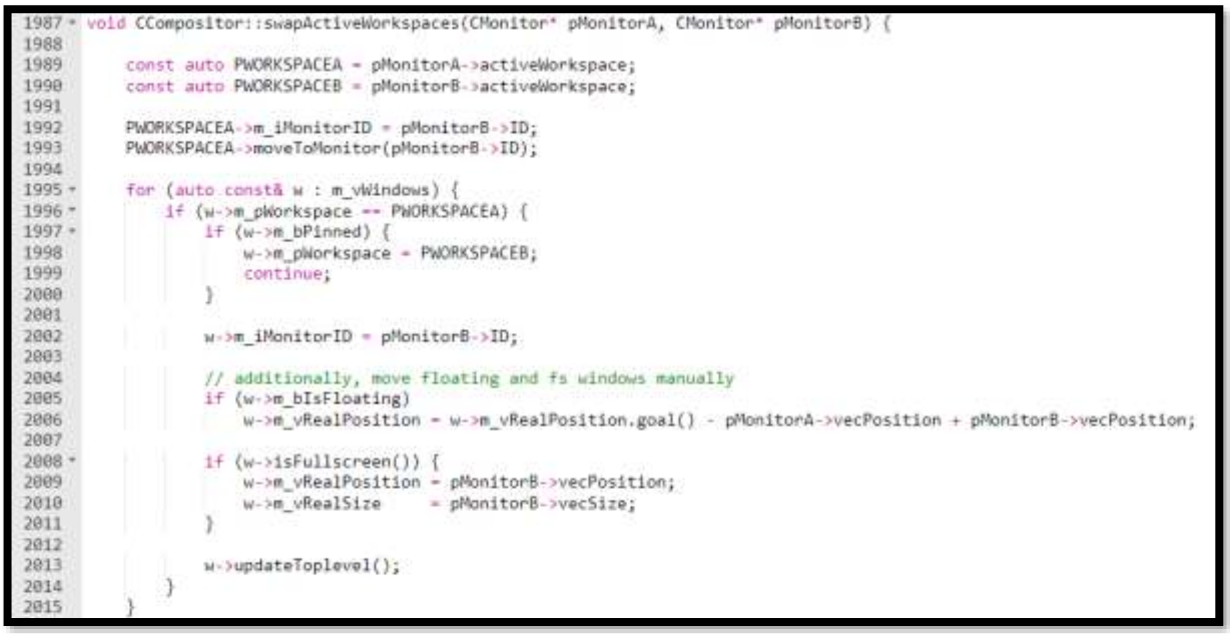
Some sections of the code, like certain debug statements, appear to be rendered unreachable due to early return statements, which negates their intended purpose.



```cpp
MONITORID CCompositor::getNextAvailableMonitorID(std::string const& name) {
    // reuse ID if it's already in the map, and the monitor with that ID is not being used by another monitor
    if (m_mMonitorIDMap.contains(name) && !std::any_of(m_vRealMonitors.begin(), m_vRealMonitors.end(), [&](auto m) { r
        return m_mMonitorIDMap[name];

    // otherwise, find minimum available ID that is not in the map
    std::unordered_set<MONITORID> usedIDs;
    for (auto const& monitor : m_vRealMonitors) {
        usedIDs.insert(monitor->ID);
    }

    MONITORID nextID = 0;
    while (usedIDs.count(nextID) > 0) {
        nextID++;
    }
    m_mMonitorIDMap[name] = nextID;
    return nextID;
}
```

There is a potential risk that the while loop could result in an infinite loop if the exit condition is never satisfied.

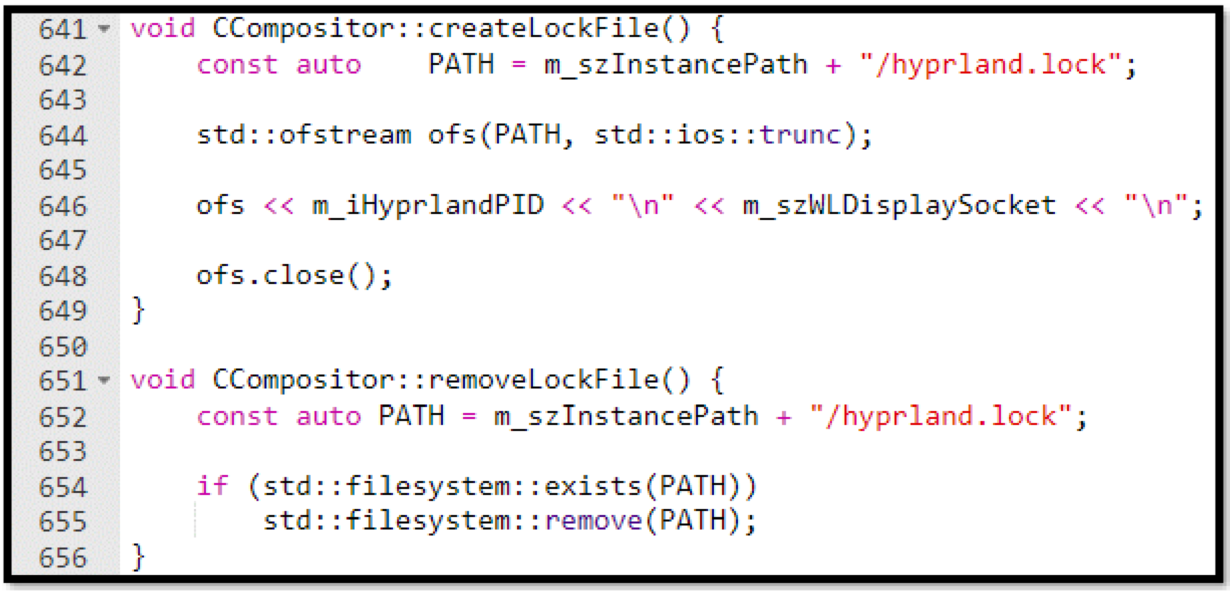
- Category F



```
void CCompositor::swapActiveWorkspaces(CMonitor* pMonitorA, CMonitor* pMonitorB) {

    const auto PWORKSPACEA = pMonitorA->activeWorkspace;
    const auto PWORKSPACEB = pMonitorB->activeWorkspace;

    PWORKSPACEA->m_iMonitorID = pMonitorB->ID;
    PWORKSPACEA->moveToMonitor(pMonitorB->ID);

    for (auto const& w : m_vWindows) {
        if (w->m_pWorkspace == PWORKSPACEA) {
            if (w->m_bPinned) {
                w->m_pWorkspace = PWORKSPACEB;
                continue;
            }

            w->m_iMonitorID = pMonitorB->ID;

            // additionally, move floating and fs windows manually
            if (w->m_bIsFloating)
                w->m_vRealPosition = w->m_vRealPosition.goal() - pMonitorA->vecPosition + pMonitorB->vecPosition;

            if (w->isFullscreen()) {
                w->m_vRealPosition = pMonitorB->vecPosition;
                w->m_vRealSize     = pMonitorB->vecSize;
            }

            w->updateToplevel();
        }
    }
```

In CCompositor::swapActiveWorkspaces(), when the pMonitorA and pMonitorB workspaces are swapped, the absence of type checking between workspace IDs and monitor IDs may result in issues due to mismatched arguments.

- Category G



```
void CCompositor::createLockFile() {
    const auto    PATH = m_szInstancePath + "/hyprland.lock";

    std::ofstream ofs(PATH, std::ios::trunc);

    ofs << m_iHyprlandPID << "\n" << m_szWLDisplaySocket << "\n";

    ofs.close();
}

void CCompositor::removeLockFile() {
    const auto PATH = m_szInstancePath + "/hyprland.lock";

    if (std::filesystem::exists(PATH))
        std::filesystem::remove(PATH);
}
```

In the function CCompositor::createLockFile(), potential I/O errors, such as being unable to write to the file, are not clearly addressed. Similarly, the removeLockFile() method checks for file existence but lacks robust error handling.

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

Based on the analysis, Category A: Data Reference Errors is particularly effective for program inspection in the context of C++ because:

1. Frequent in C++: C++ heavily relies on pointers, dynamic memory allocation, and object references, making it prone to data reference issues such as uninitialized variables, null pointer dereferencing, and memory leaks.
2. Hard-to-Detect Bugs: These types of errors can be subtle and often do not cause immediate crashes. Instead, they lead to undefined behaviour that may only manifest under specific conditions or after prolonged use, making them critical to catch during inspection.
3. Broad Impact: Errors related to data references can have wide-reaching effects across the entire program. A single uninitialized variable or dangling pointer can compromise multiple areas of the code.

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

The errors not easily identified through program inspection are runtime errors, such as:

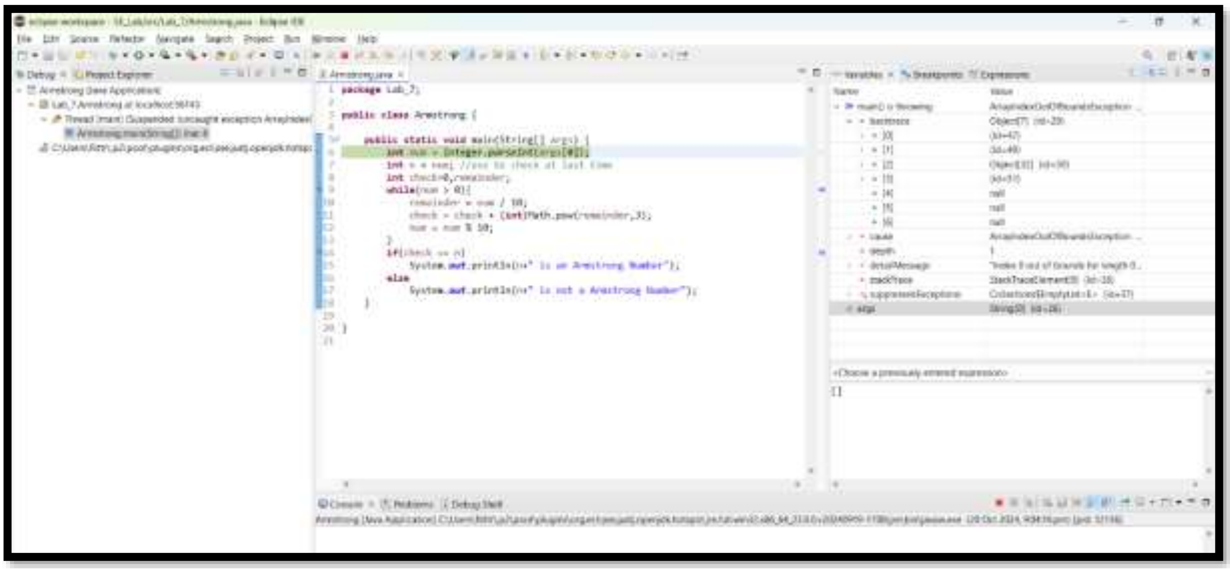
1. Concurrency issues (e.g., race conditions, deadlocks)
2. Performance bottlenecks (e.g., memory leaks)
3. Dynamic memory allocation failures
4. File handling and external dependency errors
5. Logic errors from unexpected user input

4. Is the program inspection technique is worth applicable?

Yes, the program inspection technique is worth applying. It helps identify many common issues, such as data reference errors, variable initialization issues, control-flow mistakes, and logical errors at an early stage. By reviewing code systematically, inspection can prevent bugs before they manifest during runtime, reducing debugging time and improving code quality. However, it is most effective when combined with dynamic testing to catch runtime-specific issues.

➢ Code debugging

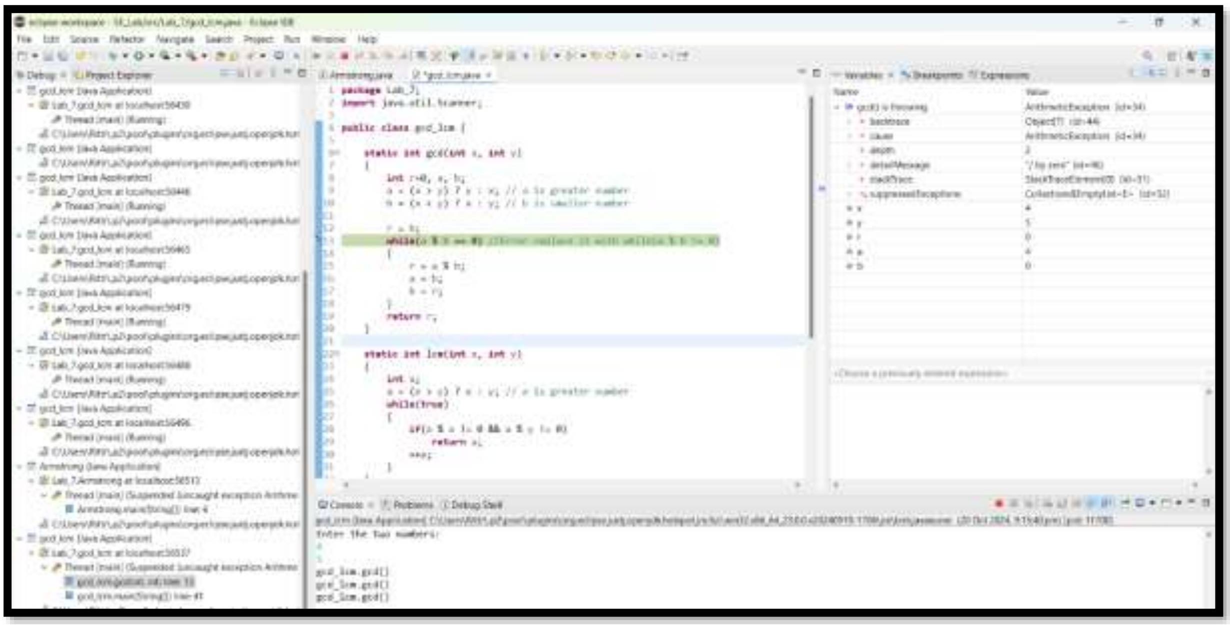
- **Armstrong**



```java
package Lab_7;

public 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");
    }

}
```

The error in this code arises from an incorrect calculation of the remainder, which causes issues in the main() function.

- **GCD_LCM**

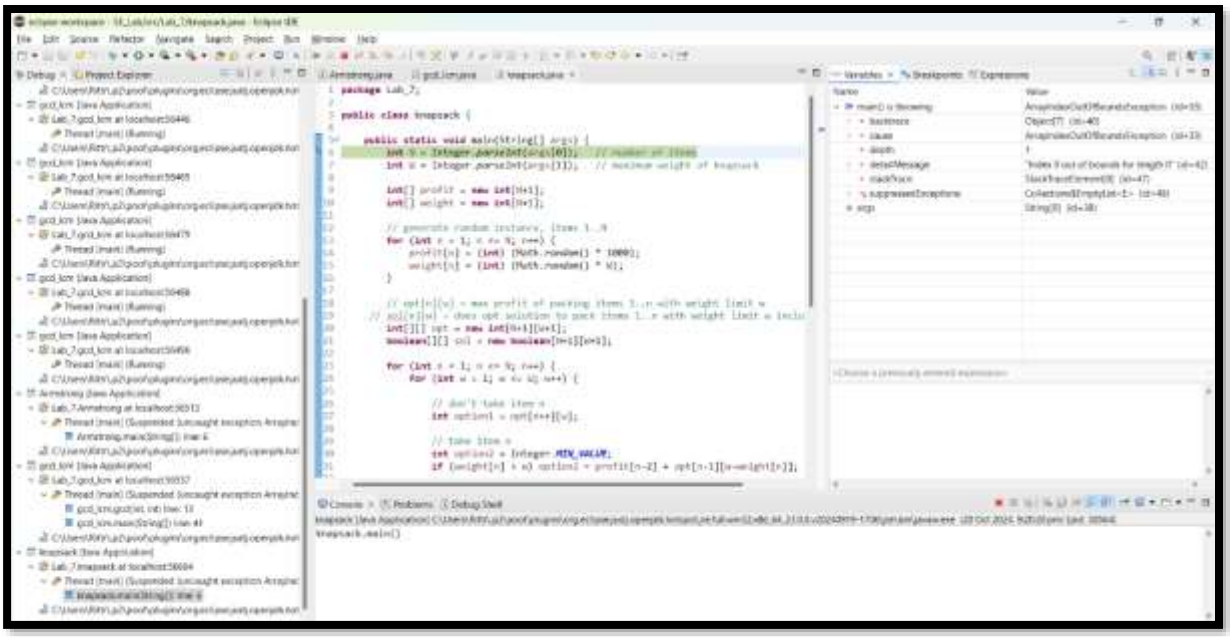


```java
package Lab_7;
import java.util.Scanner;

public class gcd_lcm {

    static int gcd(int x, int y)
    {
        int r=0, a, b;
        a = (x > y) ? x : y; // 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;
        }
```

```
Enter the two numbers:
4
5
gcd_lcm.gcd()
gcd_lcm.gcd()
gcd_lcm.gcd()
```

The condition for the while loop should be a % b != 0 instead of a % b == 0, which results in an ArithmeticExpression error.

- **KnapSack**

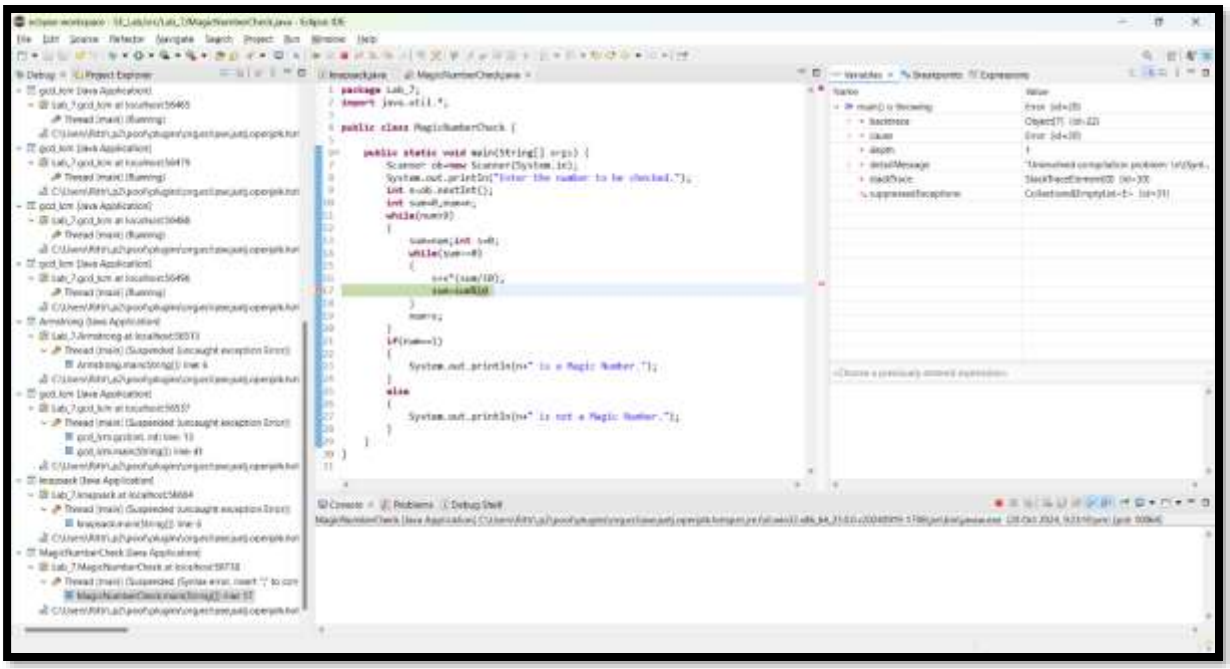


```java
package Lab_7;

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 inclu
        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++][w];

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

```
knapsack.main()
```

The error here is due to an incorrect index update of n-1, which should be n++, leading to an out-of-bounds error in the main function.

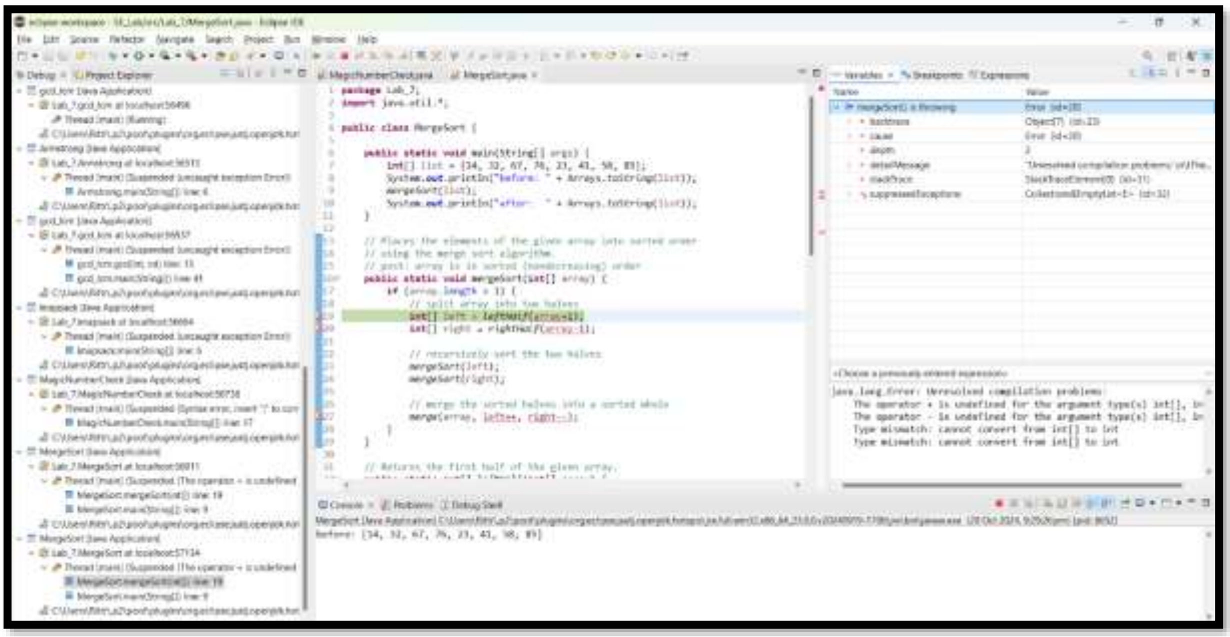
- **MagicNumberCheck**



```java
package Lab_7;
import java.util.*;

public class MagicNumberCheck {

    public static void main(String[] args) {
        Scanner ob=new Scanner(System.in);
        System.out.println("Enter the number to be checked.");
        int n=ob.nextInt();
        int sum=0,num=n;
        while(num>9)
        {
            sum=num;int s=0;
            while(sum==0)
            {
                s=s*(sum/10);
                sum=sum%10
            }
            num=s;
        }
        if(num==1)
        {
            System.out.println(n+" is a Magic Number.");
        }
        else
        {
            System.out.println(n+" is not a Magic Number.");
        }
    }
}
```

The condition while(sum == 0) is incorrect; since you want to sum the digits of the number, it should be while(sum != 0). Additionally, there is a missing semicolon (;) in the line sum = sum % 10.

- **MergeSort**

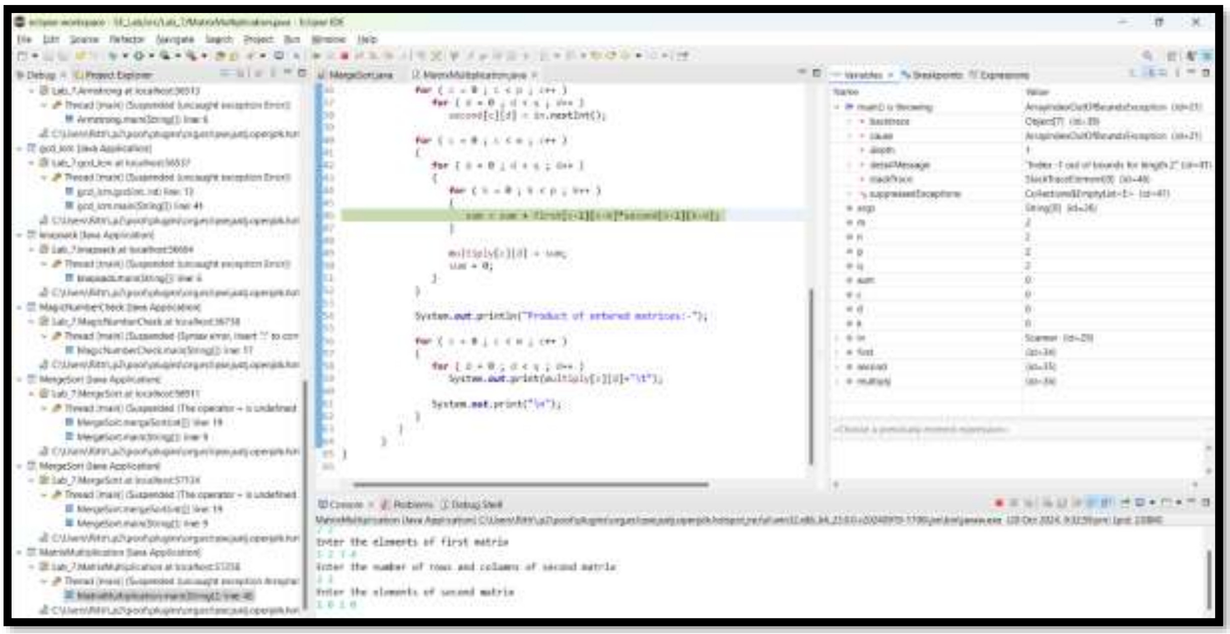


```
package Lab_7;
import java.util.*;

public class MergeSort {

    public static void main(String[] args) {
        int[] list = {34, 32, 67, 76, 23, 41, 58, 85};
        System.out.println("before: " + Arrays.toString(list));
        mergeSort(list);
        System.out.println("after:  " + Arrays.toString(list));
    }

    // Places the elements of the given array into sorted order
    // using the merge sort algorithm.
    // post: array is in sorted (nondecreasing) order
    public static void mergeSort(int[] array) {
        if (array.length > 1) {
            // split array into two halves
            int[] left = leftHalf(array+1);
            int[] right = rightHalf(array-1);

            // recursively sort the two halves
            mergeSort(left);
            mergeSort(right);

            // merge the sorted halves into a sorted whole
            merge(array, left++, right--);
        }
    }

    // Returns the first half of the given array.
```

```
before: [34, 32, 67, 76, 23, 41, 58, 85]
```

The line int[] left = leftHalf(array + 1); should be corrected to int[] left = leftHalf(array); because adding an integer to an array reference is invalid. Similarly, int[] right = rightHalf(array - 1); should be changed to int[] right = rightHalf(array); for the same reason. The call merge(array, left++, right--); should simply be merge(array, left, right); since the increment (++) and decrement (--) operators are unnecessary when passing the entire arrays.

- **MatrixMultiplication**

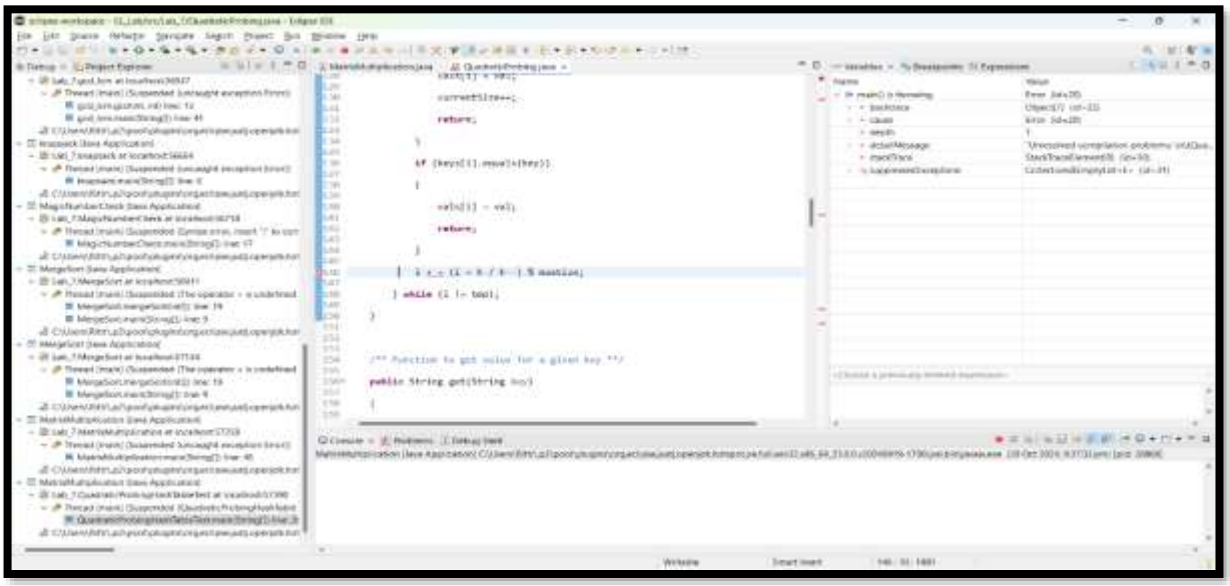


```
            for ( c = 0 ; c < p ; c++ )
               for ( d = 0 ; d < q ; d++ )
                  second[c][d] = in.nextInt();

            for ( c = 0 ; c < m ; c++ )
            {
               for ( d = 0 ; d < q ; d++ )
               {
                  for ( k = 0 ; k < p ; k++ )
                  {
                     sum = sum + first[c-1][c-k]*second[k-1][k-d];
                  }

                  multiply[c][d] = sum;
                  sum = 0;
               }
            }

            System.out.println("Product of entered matrices:-");

            for ( c = 0 ; c < m ; c++ )
            {
               for ( d = 0 ; d < q ; d++ )
                  System.out.print(multiply[c][d]+"\t");

               System.out.print("\n");
            }
         }
      }
}
```

```
Enter the elements of first matrix
1 2 3 4
Enter the number of rows and columns of second matrix
2 2
Enter the elements of second matrix
1 0 1 0
```

In the matrix multiplication loop, the indices for both matrices are incorrect. Accessing first[c-1][c-k] and second[k-1][k-d] leads to invalid indices due to negative values or incorrect references. The correct indices should be first[c][k] and second[k][d], as you need to multiply the corresponding elements from row c of the first matrix with column d of the second matrix.

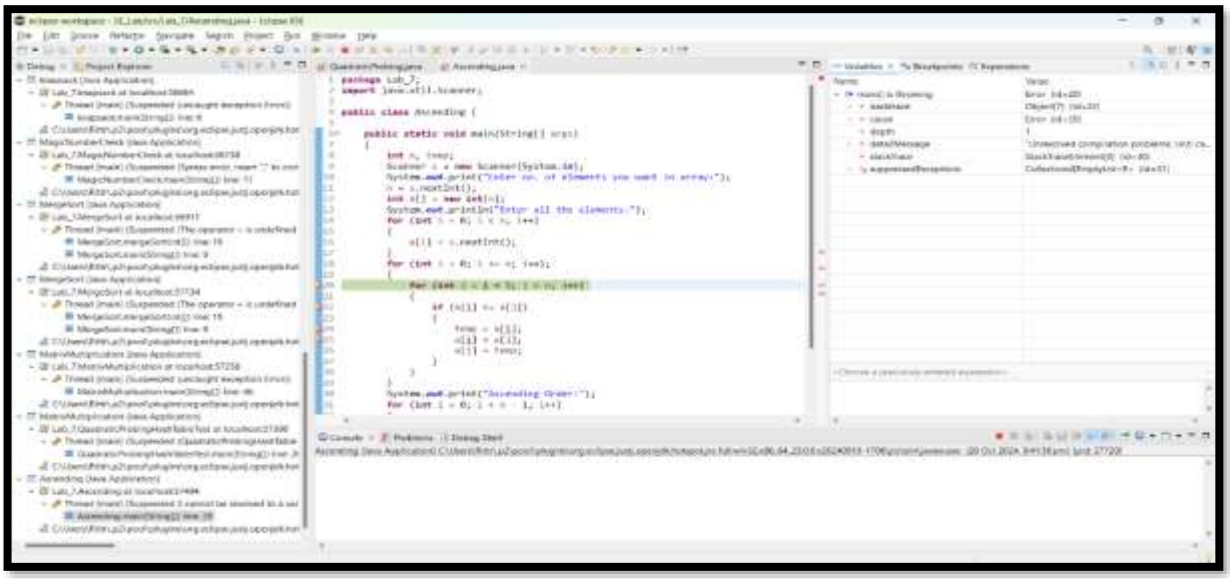
- **QuadraticProbing**



```
                keys[i] = key;
                currentSize++;
                return;
            }
            if (keys[i].equals(key))
            {
                vals[i] = val;
                return;
            }
        i += (i + h / h--) % maxSize;
    } while (i != tmp);
}

/** Function to get value for a given key **/
public String get(String key)
{
```

The syntax i += (i + h / h--); is invalid. It should be corrected to i = (i + h * h++); since the += operator is not properly placed, and the arithmetic operation should use * for quadratic probing instead of /.

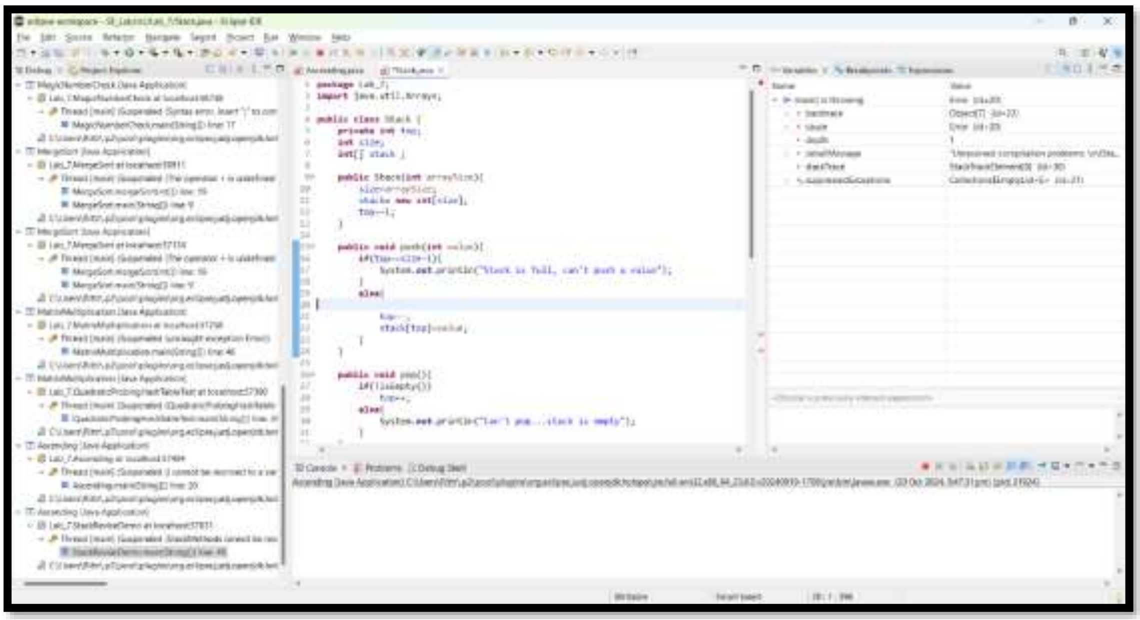
- **Ascending**



```
package Lab_7;
import java.util.Scanner;

public class Ascending {

    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++)
```

The class name Ascending _Order contains a space, which is not permitted in Java. You should either remove the space or replace it with an underscore (_) to separate the words. The condition (int i = 0; i >= n; i++); is incorrect because i >= n will prevent the loop from executing; also, there is an unnecessary semicolon (;) at the end of the loop. The correct condition should be i < n. Additionally, in the inner if condition, checking if (a[i] <= a[j]) will sort the array in descending order; it should be changed to if (a[i] > a[j]) to sort the array in ascending order. Finally, the last loop prints the array elements separated by commas but incorrectly leaves a trailing comma.

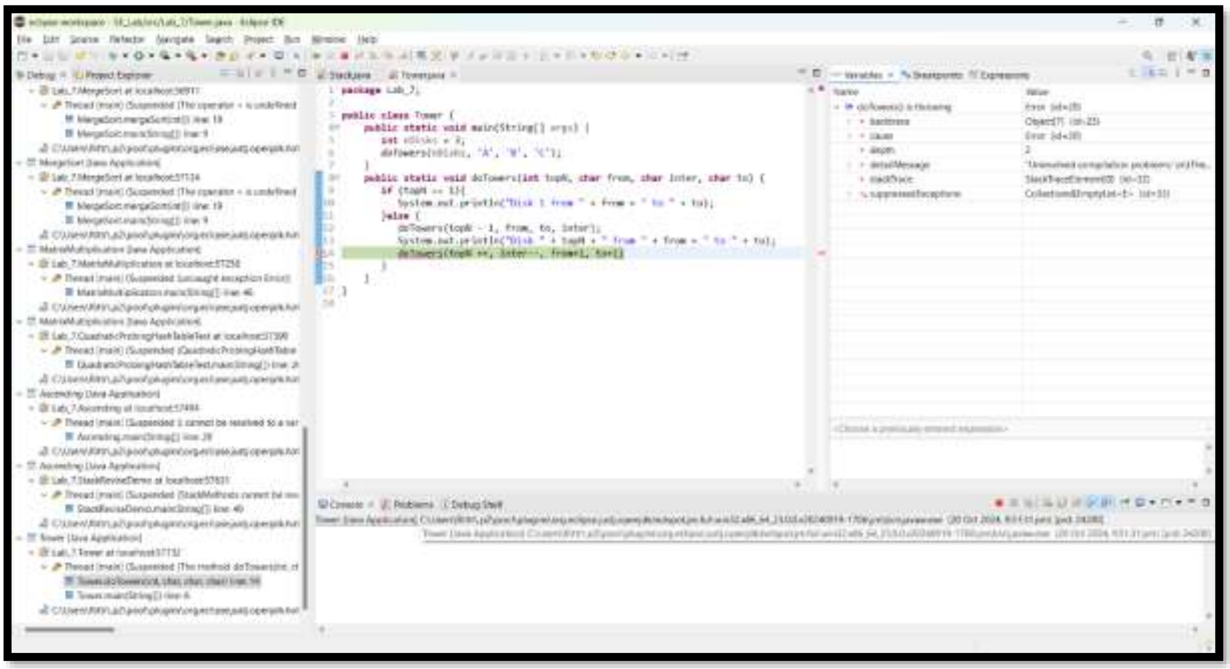
- **Stack**



```
package Lab_7;
import java.util.Arrays;

public class Stack {
    private int top;
    int size;
    int[] stack ;

    public Stack(int arraySize){
        size=arraySize;
        stack= new int[size];
        top=-1;
    }

    public void push(int value){
        if(top==size-1){
            System.out.println("Stack is full, can't push a value");
        }
        else{

            top--;
            stack[top]=value;
        }
    }

    public void pop(){
        if(!isEmpty())
            top++;
        else{
            System.out.println("Can't pop...stack is empty");
        }
```

In the push method, top-- is used, which decrements top; instead, it should be top++ to increment the position for inserting a new value. In the display method, the condition for(int i=0; i > top; i++) is incorrect, as it will never execute. The condition should be i <= top to display all elements from index 0 to top. In the pop method, the code only increments top without actually removing or returning the element. For a proper stack implementation, you should return the popped value and also decrement the top pointer.

- **TowerOfHanoi**



```
package Lab_7;

public class Tower {
    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 ++, inter--, from+1, to+1);
        }
    }
}
```

The expressions topN++, inter--, from + 1, and to + 1 are incorrect in the context of the recursive calls. The parameters should be passed without any increments or decrements to ensure the algorithm functions correctly. Additionally, the decrement for topN in the recursive calls should be topN - 1, not topN++.

➢ Static Testing

On static testing of the c++ code using cppcheck application the following result was obtained:

Errors : 0

Warnings : 0

Style warnings : 30

Portability warnings : 0

Performance warnings : 1

Information messages : 37