Software Engineering Lab 7

202201029: Bhavya Kantelia

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

Program Inspection

Category A

```
2794  void CCompositor::arrangeMonitors() {
2795      static auto* const     PXWLFORCESCALEZERO = (Hyprlang::INT* const*)g_pC
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 {:j}", m->szName,
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
2819              continue;
2820          }
```

Potential array access issues: In methods like CCompositor::arrangeMonitors(), there are loops that access elements of arrays or lists (e.g., m_lMonitors). There is no clear boundary check for array indices, so there is a risk of out-of-bounds access, especially if the list is empty or shorter than expected.

```
1347  PHLWINDOW 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  }
```

The pointer PMONITOR is initialized but there is confirmation that it can not be null and so can lead to null referencing.

Category B

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

    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;
```

Variable Shadowing: In the above given snippet, the variable ‘m’ is used which is also used in multiple other instances which can cause potential issues due to shadowing of similar names in different scopes.

```
    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 Conversions: In the given snippet, when handling the variable POSTOMON, there may be implicit conversion issues if vecPosition is not compatible with the assigned type.

Category C

```
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);
    }
```

Mixed-Type Computations: Here, the function handles string-to-number conversions and operations on mixed types (like floats and ints), which may lead to unexpected rounding or truncation errors.

Category D

```
1754
1755  PHLWORKSPACE CCompositor::getWorkspaceByString(const std::string& str) {
1756      if (str.starts_with("name:")) {
1757          return getWorkspaceByName(str.substr(str.find_first_of(':') + 1));
1758      }
1759
1760      try {
1761          return getWorkspaceByID(getWorkspaceIDNameFromString(str).id);
1762      } catch (std::exception& e) { Debug::log(ERR, "Error in getWorkspaceByString, invalid id"); }
1763
1764      return nullptr;
1765  }
1766
```

Boolean Logic Errors: The logic around str.starts_with("name:") and the way it handles exceptions might fail if the string format is incorrect, leading to unexpected behaviour.

Category E

```
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
    if (m_mMonitorIDMap.contains(name) && !std::any_of(m_vRealMonitors.begin(), m_vRealMoni
        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 can be a possibility that this while block can lead to a infinite loop is the condition is never met.

```
1678  PHLWINDOW CCompositor::getNextWindowOnWorkspace(PHLWINDOW pWindow, bool focusa
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->
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_b
1701              return w;
1702      }
1703
1704      return nullptr;
1705  }
```

There are some segments of the code, such as certain debug statements, that seem to be left unreachable by early return statements, thus defeating their purpose.

Category F

```
1987  void CCompositor::swapActiveWorkspaces(CMonitor* pMonitorA,
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
2005              if (w->m_bIsFloating)
2006                  w->m_vRealPosition = w->m_vRealPosition.goa
2007
2008              if (w->isFullscreen()) {
2009                  w->m_vRealPosition = pMonitorB->vecPositior
2010                  w->m_vRealSize     = pMonitorB->vecSize;
2011              }
2012
2013              w->updateToplevel();
2014          }
2015      }
```

Mismatch in Argument Attributes: In CCompositor::swapActiveWorkspaces(), when the pMonitorA and pMonitorB workspaces are swapped, there is no type checking between workspace IDs and monitor IDs, which could lead to issues in mismatched arguments.

Category G

```
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  }
```

File Handling: In the function CCompositor::createLockFile(), there is no clear handling of potential I/O errors such as the inability to write to the file. Also, the same applies to the removeLockFile() method where file existence is checked but not error-handled in a robust way.

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)

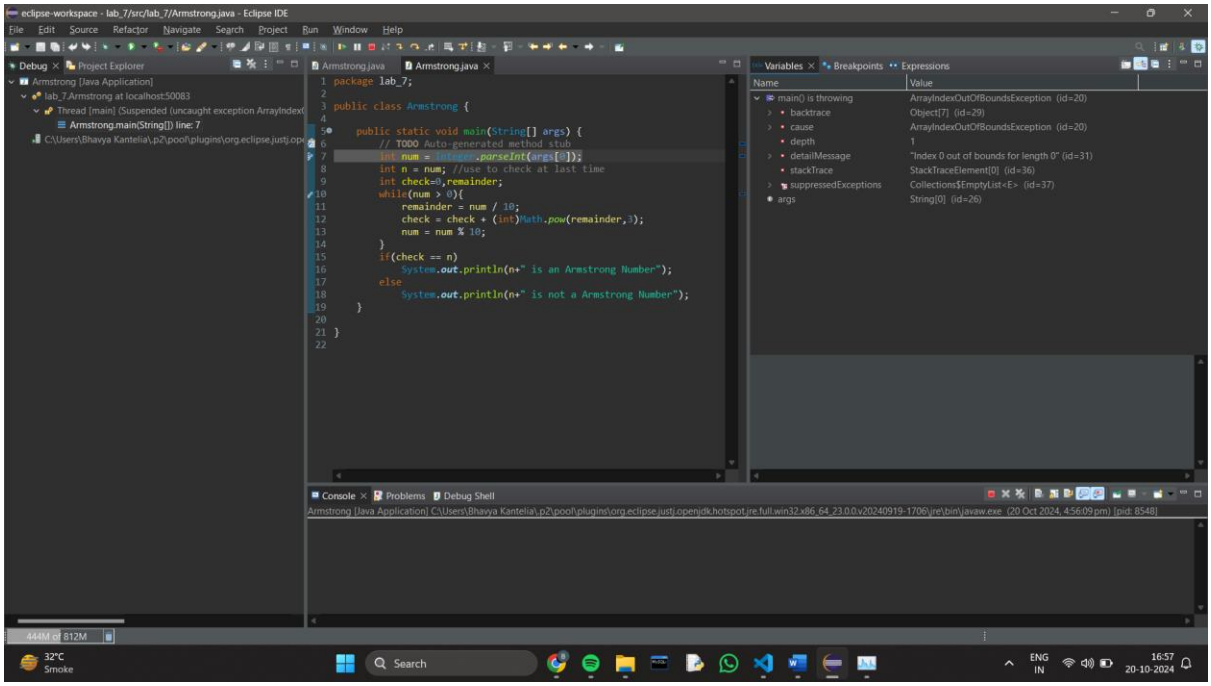
2. Performance bottlenecks (e.g., memory leaks)
3. Dynamic memory allocation failures
4. File handling and external dependency errors
5. Logic errors from unexpected user input

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.

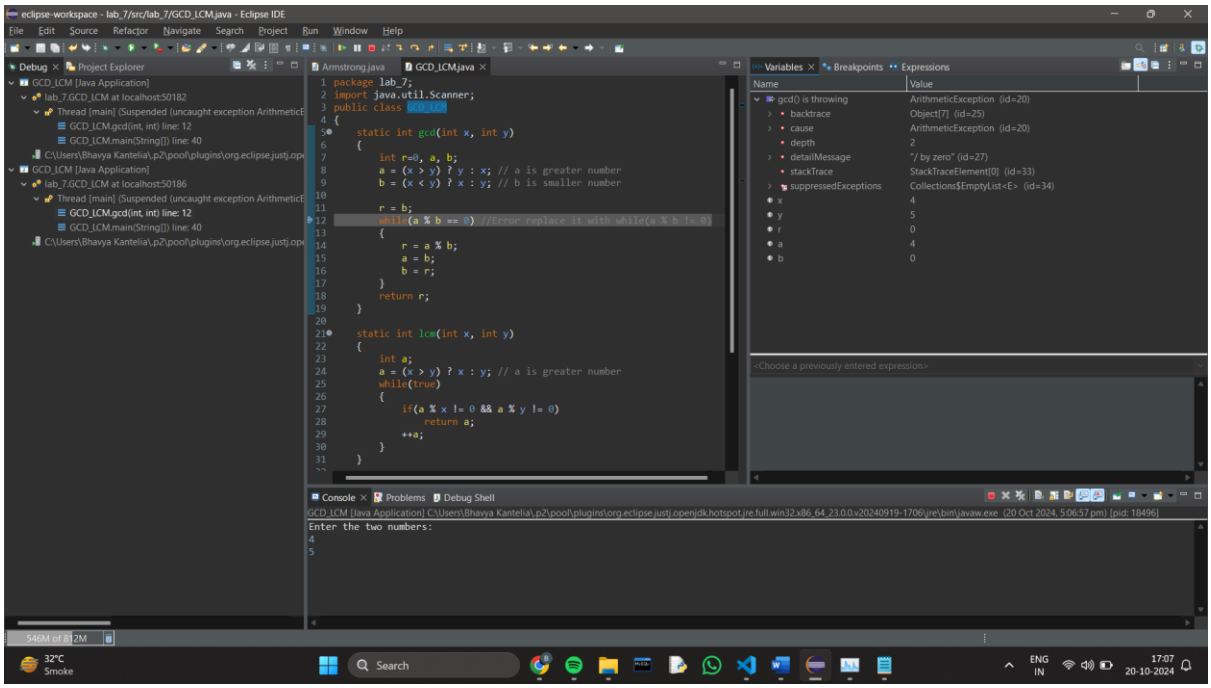
Code debugging

Armstrong



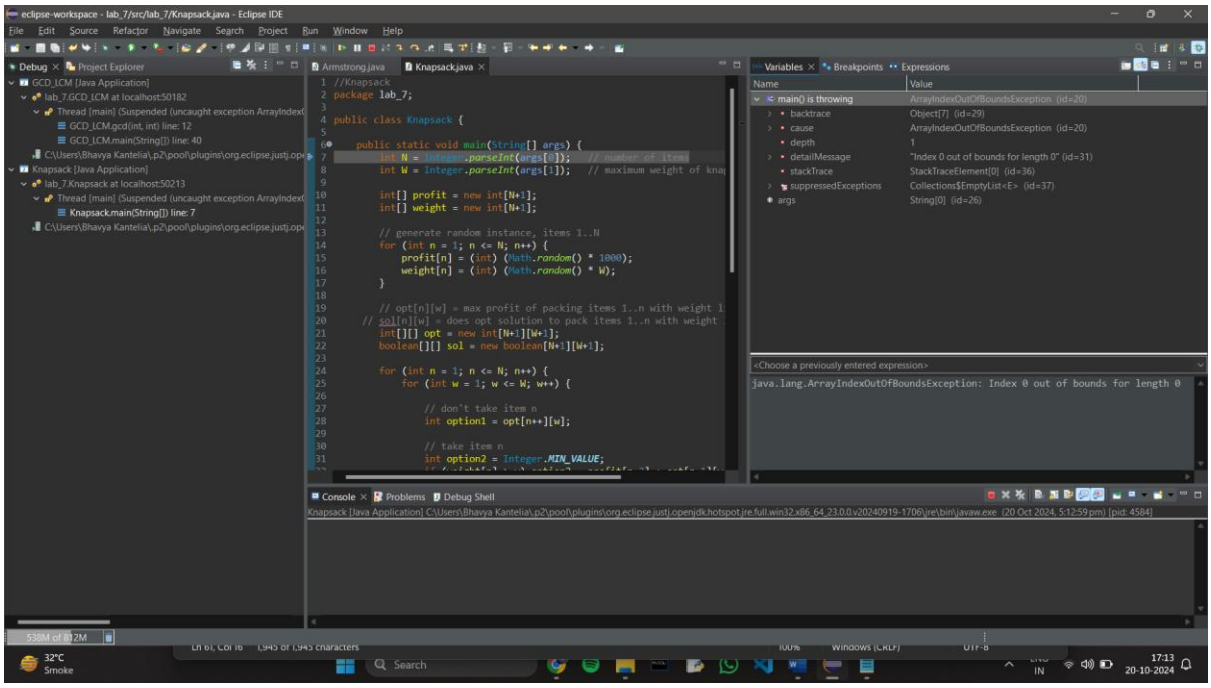
The error in this code is that remainder is calculated wrongly and this leads to error in the main().

GCD_LCM



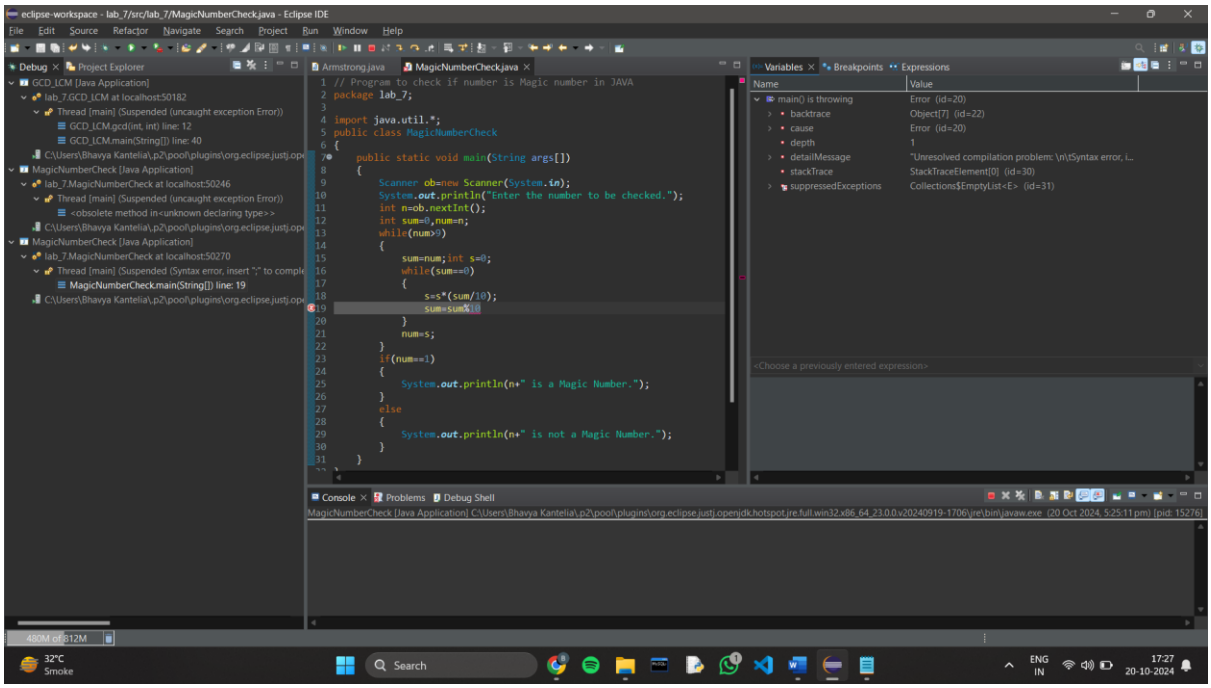
Here the while loop should be a%b!=0 instead of a%b==0 and thus it throws ArithmeticExpression error.

KnapSack



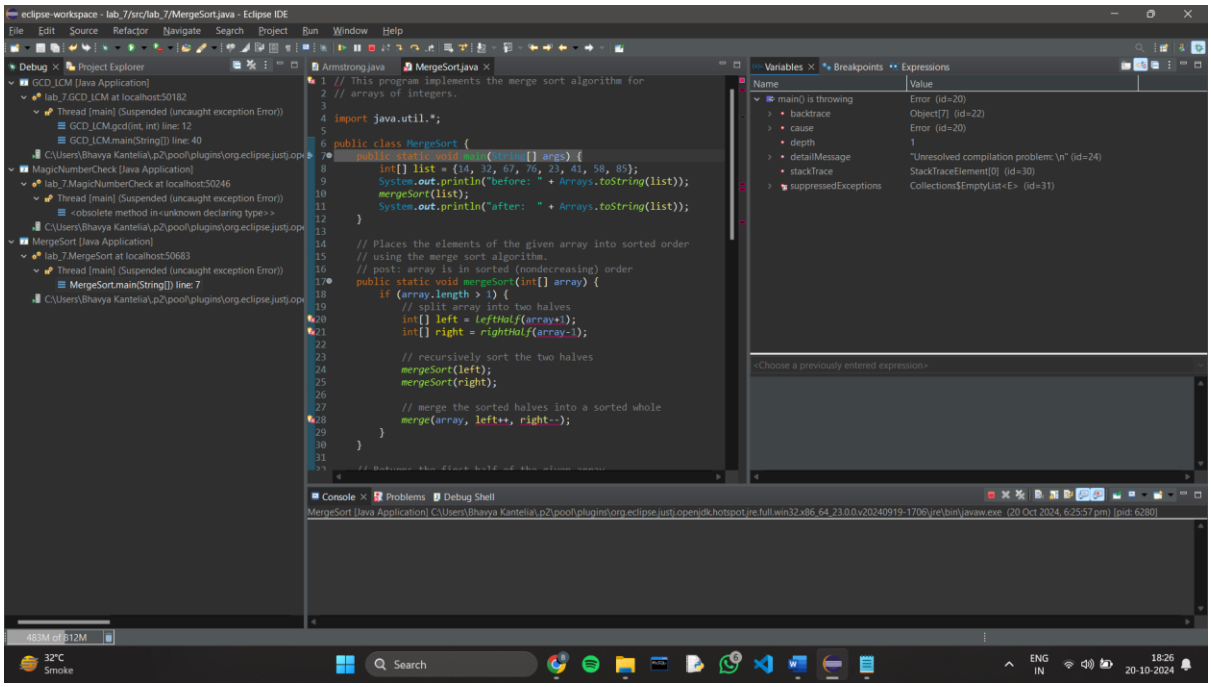
Error here is incorrect index update of n-1 which should be n++ which causes main out of bound error.

MagicNumberCheck

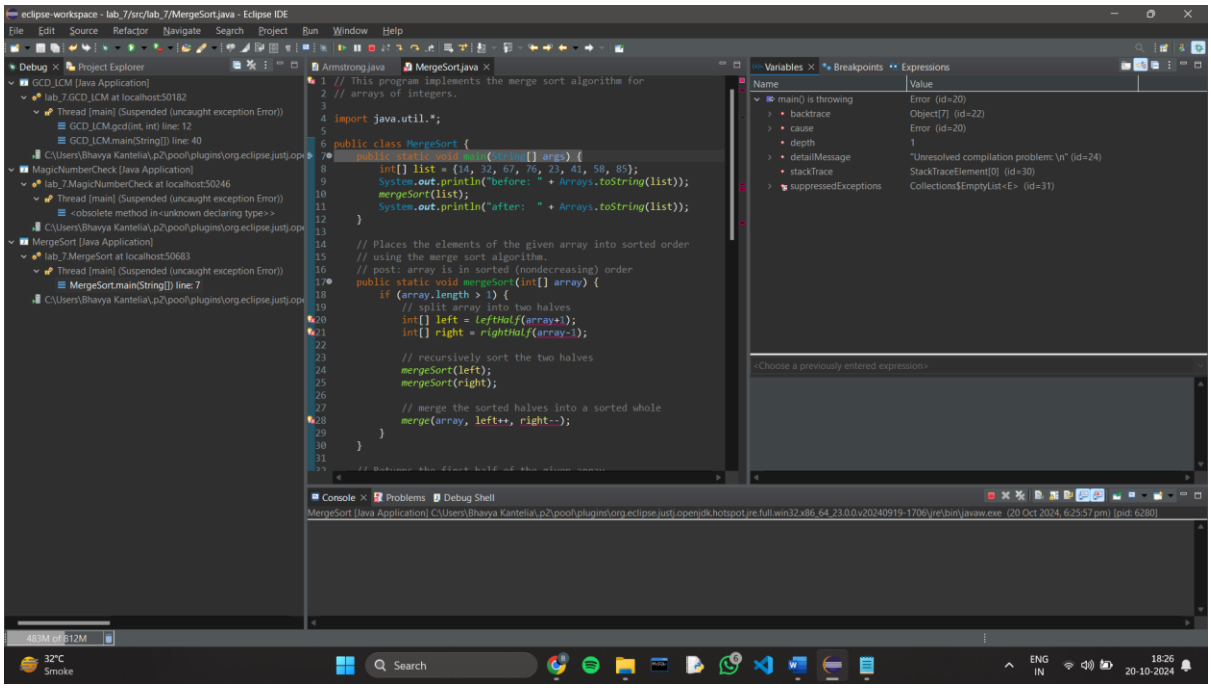


The condition while(sum == 0) is incorrect. You want to sum the digits of the number, so the condition should be while(sum != 0).

Missing a semicolon (;) in sum=sum%10.

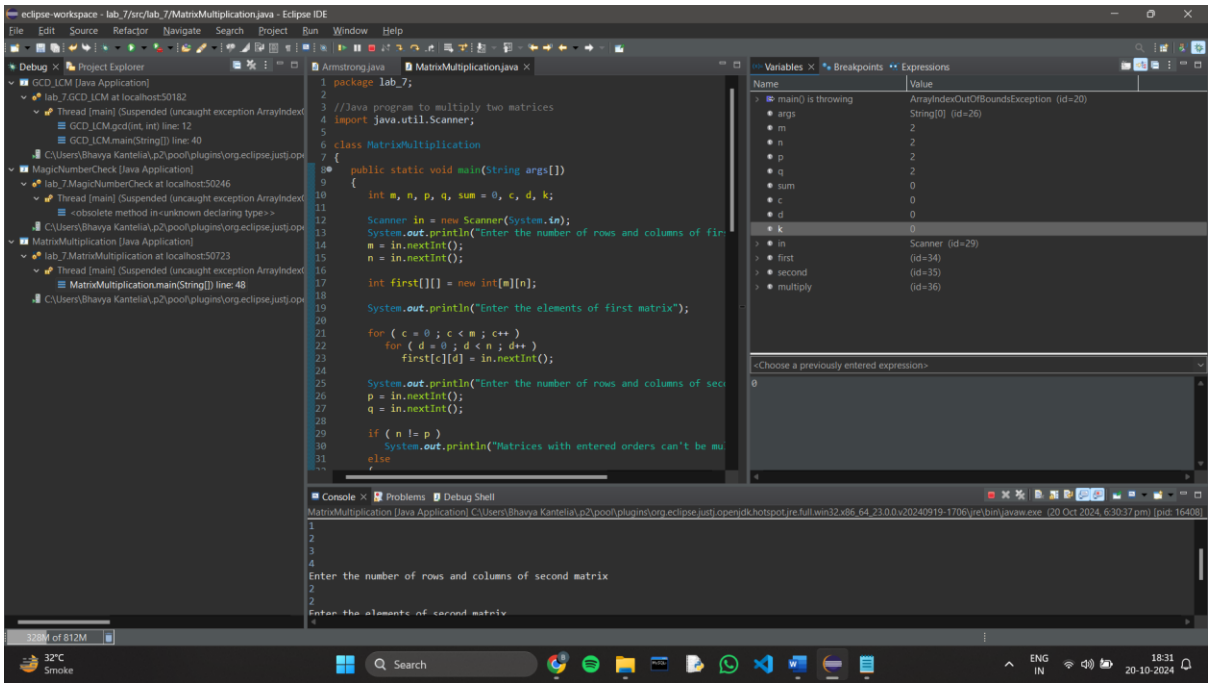


MergeSort



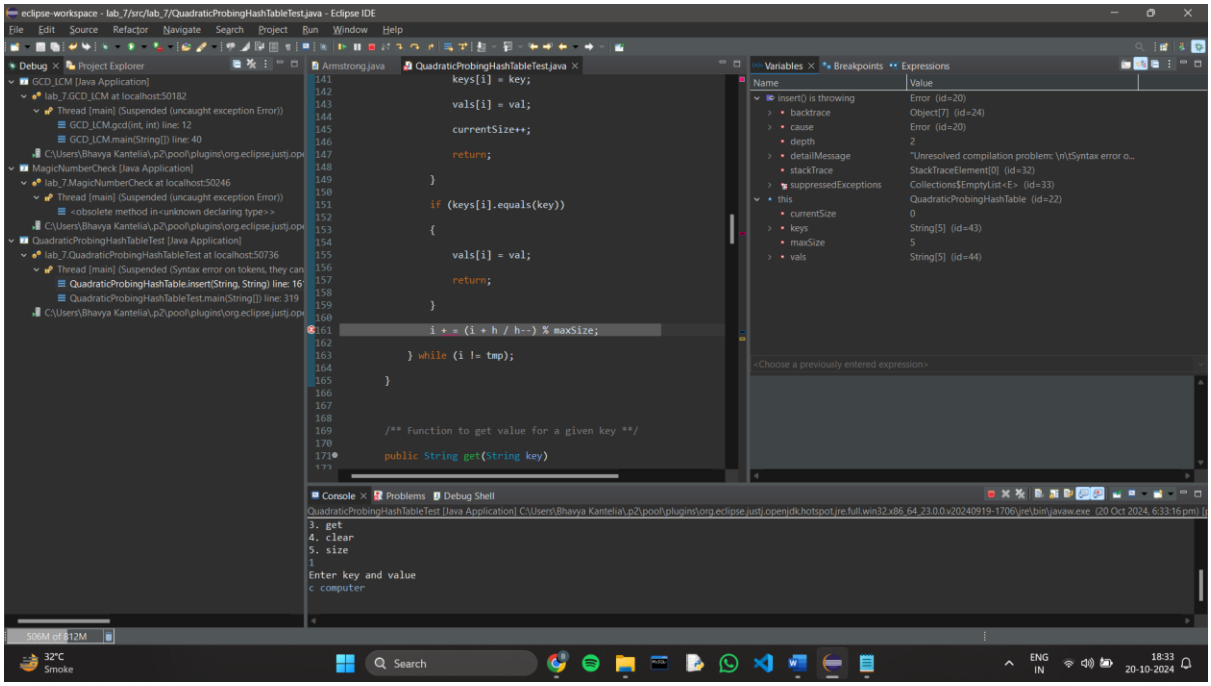
int[] left = leftHalf(array+1); should be int[] left = leftHalf(array);. The operation array+1 is invalid because you cannot add an integer to an array reference. int[] right = rightHalf(array-1); should be int[] right = rightHalf(array);. The operation array-1 is also invalid for the same reason. merge(array, left++, right--); should just be merge(array, left, right);. Increment (++) and decrement (--) operators are not needed here, as you're passing the entire arrays.

MatrixMultiplication



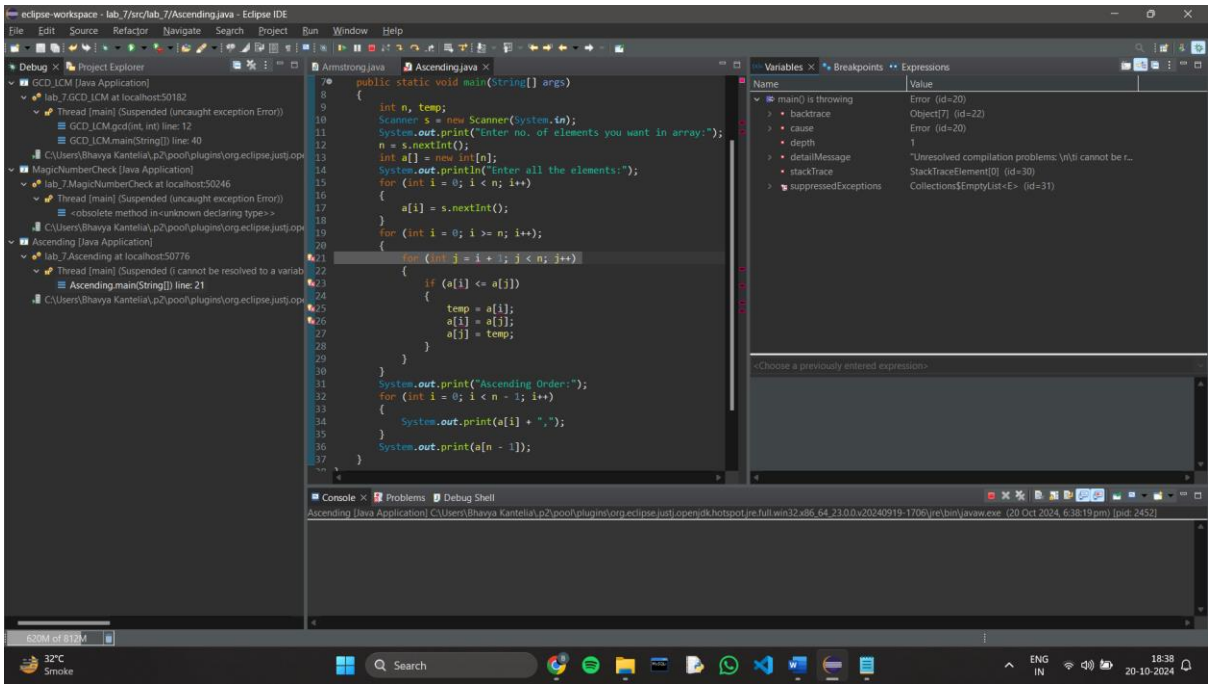
In the loop where you are multiplying the matrices, you have incorrect indices for both matrices. first[c-1][c-k] and second[k-1][k-d] are wrong because you're trying to access invalid indices with negative values or incorrect references. The correct indices should be first[c][k] and second[k][d] because you need to multiply the corresponding elements of row c of the first matrix with column d of the second matrix.

QuadraticProbing



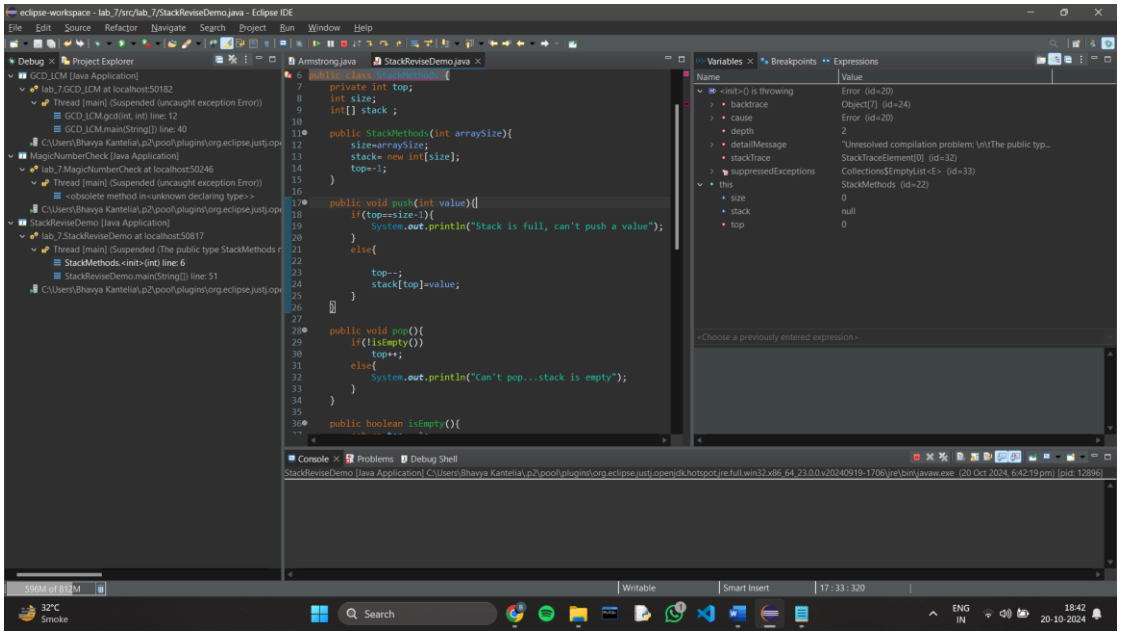
i + = (i + h / h--) % maxSize; is invalid syntax. It should be i = (i + h * h++) % maxSize;. The += operator should be properly placed, and the arithmetic operation should use * for quadratic probing, not /.

Ascending



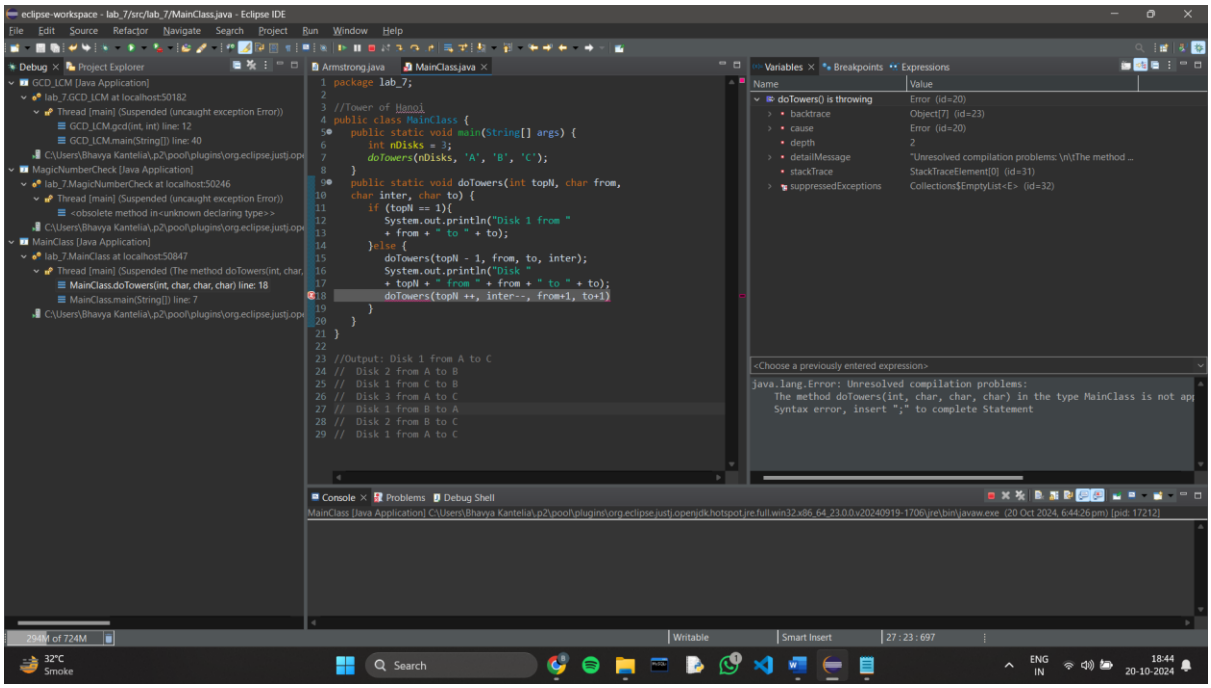
The class name Ascending _Order has a space, which is not allowed in Java. The space should be removed or replaced with an underscore (_) if you want to separate words. The condition for (int i = 0; i >= n; i++); is incorrect because i >= n means the loop will never run, and there is an unnecessary semicolon (;) at the end of the loop. The correct condition should be i < n. In the inner if condition, you are checking if (a[i] <= a[j]), which will sort the array in descending order. You should change it to if (a[i] > a[j]) to sort the array in ascending order. The final loop prints the array elements separated by commas but incorrectly leaves a trailing comma.

Stack



In the push method, top-- is used, which decrements top, but 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, it only increments the top but doesn't actually remove the element or return it. For a correct stack implementation, you should return the popped value, and it should also decrement the top pointer

TowerOfHanoi



The expressions topN++, inter--, from + 1, and to + 1 are incorrect in the context of the recursive calls. The parameters should be passed unchanged (no increment or decrement) to maintain the correct behavior of the algorithm. The topN decrement in the recursive calls should be topN - 1, not topN ++.