IT-314

Software Engineering

Jal Dani

202201315

Lab 7

TASK:1 Program Inspection

→ I did the program Inspection of map_benchmark code provided in the given link:

Github code link:

https://github.com/martinus/robin-hood-hashing/blob/master/src/include/robin_hood.h

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

—>

Dangling Pointers:

In the `BulkPoolAllocator`, the `reset()` method releases memory but does not set the pointer to `nullptr`.

```
std::free(mListForFree);
// Should be followed by `mListForFree = nullptr;` to avoid dangling pointer access.
```

Type Mismatches:

In `reinterpret_cast_no_cast_align_warning`, memory is cast without verifying types or attributes, which can introduce subtle bugs.

```
T* obj = static_cast<T*>(std::malloc(...));  // The memory may not have the correct type or attributes.
```

Uninitialized Variables:

`mHead` and `mListForFree` are initialized to `nullptr`, but after deallocation, they are not consistently reset, leading to possible dangling pointers or uninitialized variable access.

```
T* tmp = mHead;
if (!tmp) {
    tmp = performAllocation();
} // If performAllocation fails or `mHead` is improperly initialized later, `tmp` may be null.
```

Array Bound Violations:

In the `shiftUp` and `shiftDown` operations, there are no checks to ensure that the index is within valid array bounds.

```
while (--idx != insertion_idx) {
    mKeyVals[idx] = std::move(mKeyVals[idx - 1]);
}
```

Category B: Data-Declaration Errors:

Potential Data Type Mismatches:

In `hash_bytes`, the hashing process involves several castings between different data types, which could cause unexpected behavior if the sizes or attributes of these types differ.

```
auto k = detail::unaligned_load<uint64_t>(data64 + i);  // Type mismatches in memory.
```

Similar Variable Names:

Variables such as `mHead`, `mListForFree`, and `mKeyVals` have similar names, which could create confusion during code modifications or debugging.

```
while (--idx != insertion_idx);  // Risk of off-by-one errors when shifting elements.
```

Incorrect Boolean Comparisons:

In functions like `findIdx`, improper use of logical operators (`&&` and `||`) can lead to incorrect evaluations when multiple conditions are combined.

```
if (info == mInfo[idx] &&
    ROBIN_HOOD_LIKELY(WKeyEqual::operator()(key, mKeyVals[idx].getFirst()))) {
    return idx;
}
```

Category C: Computation Errors:

Integer Overflow:

In `hash_bytes`, the hash function involves multiple shifts and multiplications on large integers, which could cause overflow if the results exceed the maximum representable value.

```
h ^= h >> r;
h *= m;
```

Category D: Comparison Errors:

Incorrect Boolean Comparisons:
In functions like `findIdx`, improper handling of logical operators (`&&` and `||`) when combining multiple conditions can result in incorrect evaluations.

```
if (info == mInfo[idx] &&
    ROBIN_HOOD_LIKELY(WKeyEqual::operator()(key, mKeyVals[idx].getFirst()))) {
    return idx;
}
```

Mixed Comparisons:

In certain cases, comparisons are made between different types (e.g., signed and unsigned integers), which may lead to incorrect results depending on the system or compiler.

Category E: Control-Flow Errors:

Potential Infinite Loop:

In loops such as `shiftUp` and `shiftDown`, there is a risk of the loop failing to terminate properly if the termination condition is not met.

```
while (--idx != insertion_idx);  // Risk of off-by-one errors when shifting elements.
```

Unnecessary Loop Executions:

In some cases, loops may either run an extra iteration or not execute at all due to incorrect initialization or condition checks.

```
for (size_t idx = start; idx != end; ++idx) {  // If `start` or `end` are incorrectly set, the loop might iterate incorrectly.
```

Category F: Interface Errors:

Mismatched Parameter Attributes:

In functions like `insert_move`, there is potential for parameter mismatches, where the arguments passed might not match the expected data type or size.

Global Variables:

When global variables are referenced across multiple functions, care must be taken to ensure consistent usage and proper initialization, as inconsistent handling could lead to errors when the code is expanded.

Category G: Input/Output Errors:

Missing File Handling:

Although the current code does not deal with file I/O, future extensions could introduce typical file handling errors, such as failing to close files, missing end-of-file checks, or improper error handling.

```
if (info == mInfo[idx] &&
    ROBIN_HOOD_LIKELY(WKeyEqual::operator()(key, mKeyVals[idx].getFirst()))) {
    return idx;
}
```

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

Category A: Data Reference Errors is the most effective in this case due to the extensive use of manual memory management, pointers, and dynamic data structures. Errors in pointer dereferencing and memory allocation/deallocation can easily result in critical issues like crashes, segmentation faults, or memory leaks, making this category essential. Other significant categories include **Computation Errors** and **Control-Flow Errors**, particularly in larger projects.

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

Concurrency Issues: The inspection does not account for multi-threading or concurrency-related issues, such as race conditions or deadlocks. If this program were expanded to handle multiple threads, issues related to shared resources, locks, and thread safety would need to be addressed.

Dynamic Errors: Some errors, such as those related to memory overflow, underflow, or runtime environment behavior, may not be caught until the code is executed in a real-world scenario.

4. Is the program inspection technique worth applying?

Yes, the program inspection technique is valuable, particularly for detecting static errors that might not be caught by compilers, such as pointer mismanagement, array bound violations, and improper control flow. Although it may not catch every dynamic issue or concurrency-related bug, it’s an essential step to ensure code quality, especially in memory-critical applications like this C++ implementation of hash tables. This approach improves the code's reliability and helps maintain best practices in memory handling, control flow, and computational logic.

TASK-2 Code Debugging

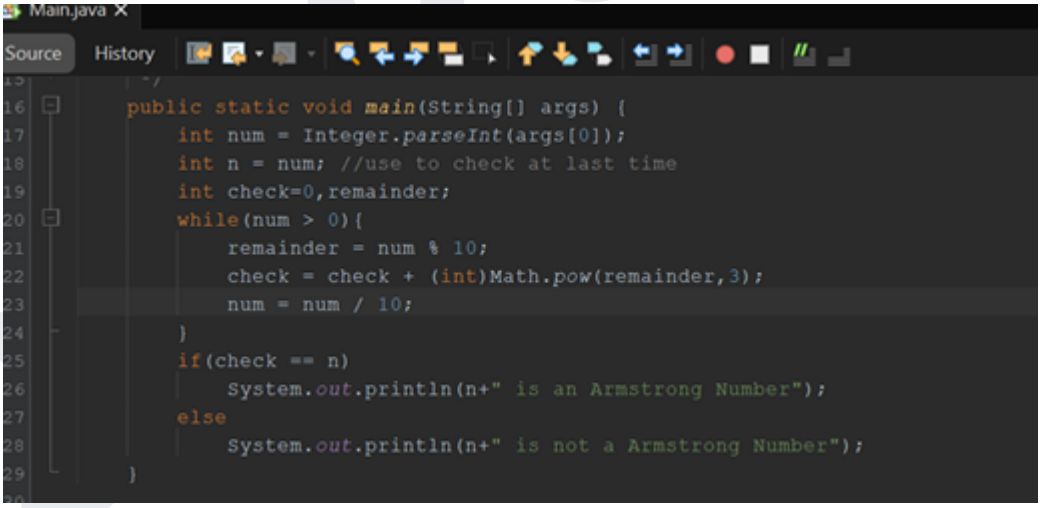
➔ Code Debugging for given Java files

➔ Note: All the executable files are in separate folder

1: **Armstrong Program**

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

- **Incorrect Calculation of Remainder:**
  ➔ The line `remainder = num / 10;` should be `remainder = num % 10;` because the goal is to extract the last digit of the number.
- **Updating num Incorrectly:**
  ➔ The line `num = num % 10;` should be `num = num / 10;`. We need to remove the last digit from `num` after processing it, not take its remainder again.



2. **How many breakpoints do you need to fix those errors?**
   ➔ Two breakpoints:
3. On the line where the remainder is calculated (`remainder = num / 10;`).
4. On the line where `num` is updated (`num = num % 10;`).

a. What are the steps you have taken to fix the error you identified in the code fragment?

- **Step 1:** Fix the calculation of the remainder to correctly extract the last digit by changing the line to `remainder = num % 10;`.
- **Step 2:** Correctly update `num` to remove the last digit by changing the line to `num = num / 10;`.

2) GCD and LCM

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

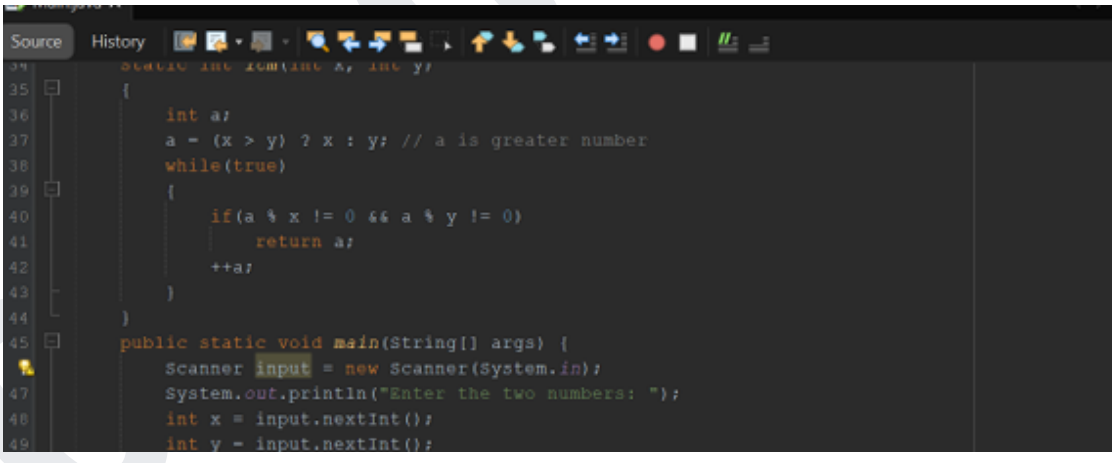
There are **two errors** in the program:

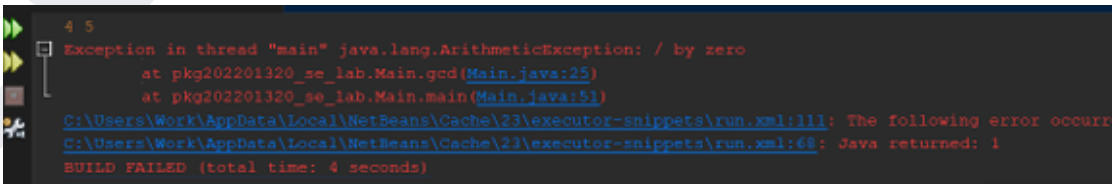
1. **Logical Error in the gcd Method:**
   The condition in the `while` loop is incorrect. It should be `while (a % b != 0)` instead of `while (a % b == 0)`. The original condition can lead to an infinite loop if `b` is not a divisor of `a`.
2. **Logical Error in the lcm Method:**
   The condition to check whether `a` is a multiple of both `x` and `y` is incorrect. It should be `if (a % x == 0 && a % y == 0)` instead of `if (a % x != 0 && a % y != 0)`.
3. **How many breakpoints do you need to fix those errors?**
   You need **two breakpoints** to debug and fix the identified errors:

4. A breakpoint at the beginning of the `gcd` method to monitor the values of `a`, `b`, and `r`.
5. A breakpoint at the beginning of the `lcm` method to check the initial value of `a` and how it increments during the loop.

a. What are the steps you have taken to fix the errors you identified in the code fragment?

1. **Fixing the gcd Method:**
   ➔ Changed the condition in the `while` loop from `while (a % b == 0)` to `while (a % b != 0)` to correctly implement the Euclidean algorithm for calculating the GCD.
2. **Fixing the lcm Method:**
   ➔ Modified the condition in the `if` statement from `if (a % x != 0 && a % y != 0)` to `if (a % x == 0 && a % y == 0)` to ensure that the method correctly identifies when `a` is a multiple of both `x` and `y`.





```
        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;
        }
    }

    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter the two numbers: ");
        int x = input.nextInt();
        int y = input.nextInt();

        System.out.println("The GCD of two numbers is:
" + gcd(x, y));
        System.out.println("The LCM of two numbers is:
" + lcm(x, y));
        input.close();
    }
}
```

3) Knapsack Problem:

Main.java

Source History

```
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


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

Source History

```
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] + opt[n-1][w-weight[n]];

        // select better of two options
        opt[n][w] = Math.max(option1, option2);
        sol[n][w] = (option2 > option1);
    }
}
```

```
        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]
+ opt[n-1][w-weight[n]];

                // select better of two options
                opt[n][w] = Math.max(option1, option2);
                sol[n][w] = (option2 > option1);
```

```
            }
        }

        // determine which items to take
        boolean[] take = new boolean[N+1];
        for (int n = N, w = W; n > 0; n--) {
            if (sol[n][w]) { take[n] = true;  w = w -
weight[n]; }
            else           { take[n] = false;
}
        }

        // print results
        System.out.println("item" + "\t" + "profit" +
"\t" + "weight" + "\t" + "take");
        for (int n = 1; n <= N; n++) {
            System.out.println(n + "\t" + profit[n] +
"\t" + weight[n] + "\t" + take[n]);
        }
    }
}
```

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

There are three main errors in the program:

1. Array Indexing Issue: The line int option1 = opt[n++][w]; incorrectly increments n, which can lead to out-of-bounds access in subsequent iterations. It should simply be int option1 = opt[n][w];.
2. Wrong Profit Calculation: In the line int option2 = profit[n-2] + opt[n 1][w-weight[n]];, the program incorrectly uses profit[n-2] instead of profit[n] to calculate the profit of the current item.

3. Weight Condition Logic: The condition for taking the item is correct, but the logic for option2 should only be calculated if the item's weight does not exceed the current weight limit (w).

2. How many breakpoints do you need to fix those errors?

You would need three breakpoints to debug and fix the errors:

1. Set a breakpoint at the beginning of the nested loop to check the values of n, w, opt[n][w], and other variables.

2. Set a breakpoint right before the assignment of option1 to monitor how n is changing.

3. Set a breakpoint after the assignment of option2 to verify the calculations for both option1 and option2.

a. What are the steps you have taken to fix the error you identified in the code fragment?

1. Correcting Array Indexing:

Changed int option1 = opt[n++][w]; to int option1 = opt[n][w]; to prevent n from being incremented incorrectly.

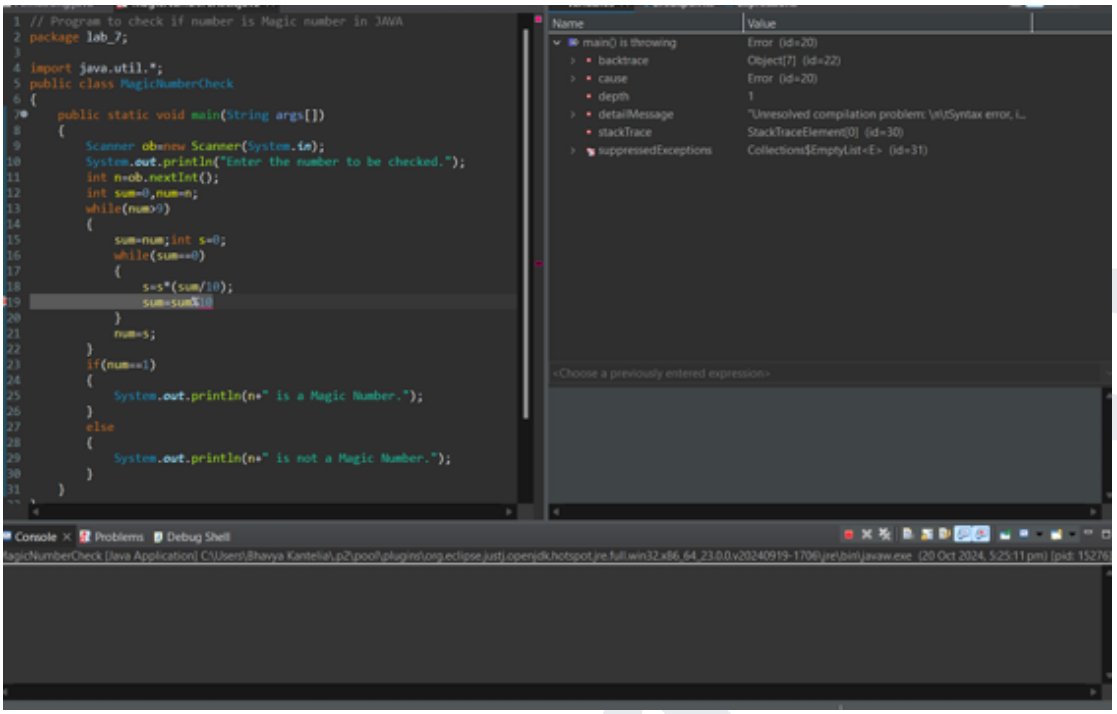
2. Correcting Profit Calculation:

Modified the line int option2 = profit[n-2] + opt[n-1][w-weight[n]]; to int option2 = profit[n] + opt[n-1][w-weight[n]]; to reference the correct item profit.

3. Adjusting Weight Condition Logic:

Added a condition to ensure that option2 is only calculated if the current item's weight does not exceed w. This prevents erroneous profit calculations for items that can't be added.

4) Magic Number:



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

There are four errors in the program:

1. **Logical Error in the Inner Loop:** The condition in the line while(sum==0) should be while(sum!=0). The current condition will not enter the loop when the sum is zero, which is incorrect.

2. Incorrect Calculation in the Inner Loop: The line s=s*(sum/10); should be s = s + (sum % 10); to correctly accumulate the sum of the digits.

3. Missing Semicolon: The line sum=sum%10 should have a semicolon at the end: sum = sum % 10;.

4. Logical Error in the While Loop: The outer loop condition while(num>9) should be while(num>9 || num == 0) to account for the scenario where the number becomes zero.

5) Merge Sort:

Main.java ×

Source History

```
        for (int i = 0; i < size2; i++) {
            right[i] = array[i + size1];
        }
        return right;
    }
    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--);
        }
```

```
run:
before: [14, 32, 67, 76, 23, 41, 58, 85]
after:  [14, 32, 67, 76, 23, 41, 58, 85]
BUILD SUCCESSFUL (total time: 1 second)
```

```java
import java.util.*;

public class MergeSort {
    public static void main(String[] args) {
        int[] list = {14, 32, 67, 76, 23, 41, 58, 85};
        System.out.println("before: " +
Arrays.toString(list));
        mergeSort(list);
        System.out.println("after:  " +
Arrays.toString(list));
    }

    public static void mergeSort(int[] array) {
        if (array.length > 1) {
            int[] left = leftHalf(array);
            int[] right = rightHalf(array);

            mergeSort(left);
            mergeSort(right);

            merge(array, left, right);
        }
    }

    public static int[] leftHalf(int[] array) {
        int size1 = array.length / 2;
        int[] left = new int[size1];
        for (int i = 0; i < size1; i++) {
            left[i] = array[i];
```

```
        }
        return left;
    }

    public static int[] rightHalf(int[] array) {
        int size1 = (array.length + 1) / 2;
        int size2 = array.length - size1;
        int[] right = new int[size2];
        for (int i = 0; i < size2; i++) {
            right[i] = array[i + size1];
        }
        return right;
    }

    public static void merge(int[] result,
                             int[] left, int[] right) {
        int i1 = 0;
        int i2 = 0;

        for (int i = 0; i < result.length; i++) {
            if (i2 >= right.length || (i1 < left.length
&&
                    left[i1] <= right[i2])) {
                result[i] = left[i1];
                i1++;
            } else {
                result[i] = right[i2];
                i2++;
            }
```

Errors Identified:

1. Incorrect Array Slicing: Array slicing with `array + 1` and `array - 1` is wrong. Use proper array splitting methods.
2. Incorrect Parameters in Recursive Calls: Increment/decrement operators (`++`, `--`) shouldn't be used on arrays during merge calls.
3. Incorrect Size Calculation: Left-half size should be `(array.length + 1) / 2` to handle odd-length arrays correctly.
4. Merging Logic Issue: The merge method should modify the original array correctly.

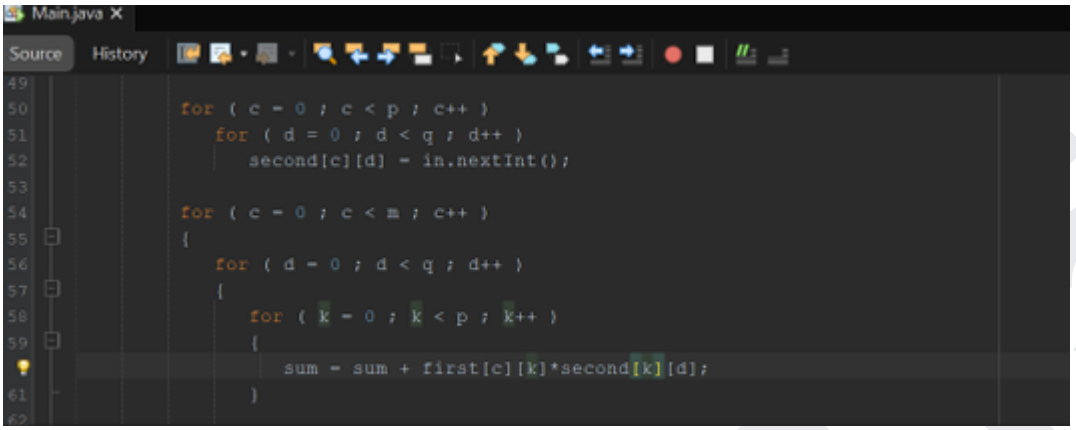
Breakpoints Needed:

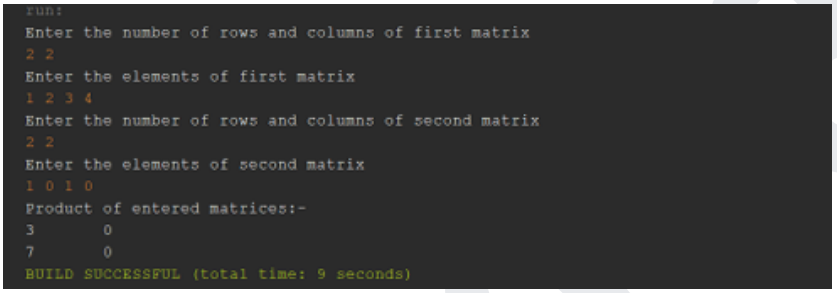
1. At the start of `mergeSort` to inspect how the array is split.
2. Before the merge operation to check left and right arrays.
3. Inside the merge method to observe how merging occurs.

Steps to Fix Errors:

1. Correct Array Slicing: Use `Arrays.copyOfRange` for proper array splitting.
2. Fix Parameters: Pass arrays directly in `merge(array, left, right)`.
3. Adjust Size Calculation: Update the size logic to `(array.length + 1) / 2` for the left half.
4. Fix Merging Logic: Ensure the merge method modifies the original array correctly.

6) Multiply Metrics:





Errors Identified:

1. Array Indexing Errors: Incorrect indices in `sum = sum + first[c-1][c-k] * second[k-1][k-d];`. Use `first[c][k] * second[k][d]` for correct access.
2. Uninitialized Variables: `sum` is not reset properly in the inner loop. Reset to 0 at the start of each `c` and `d` iteration.
3. Wrong Input Prompt: Incorrect prompt for the second matrix. It should ask for "rows and columns of second matrix."
4. Multiplication Logic: Matrix multiplication logic is incorrect. Use `first[c][k] * second[k][d]` formula.
5. Output Readability: Output format lacks proper headers or clarity for the product matrix.

Breakpoints Needed:

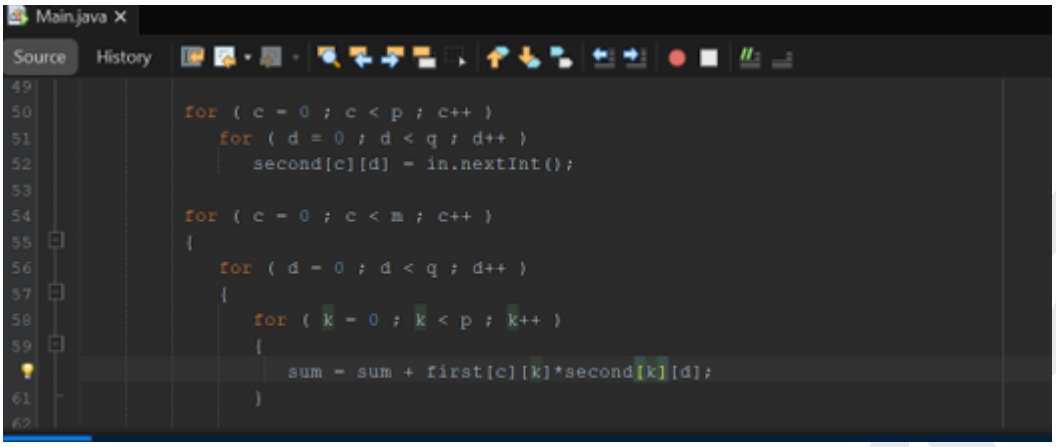
1. Inside the multiplication loop to check indices and values.
2. Before printing the result to check the product matrix.

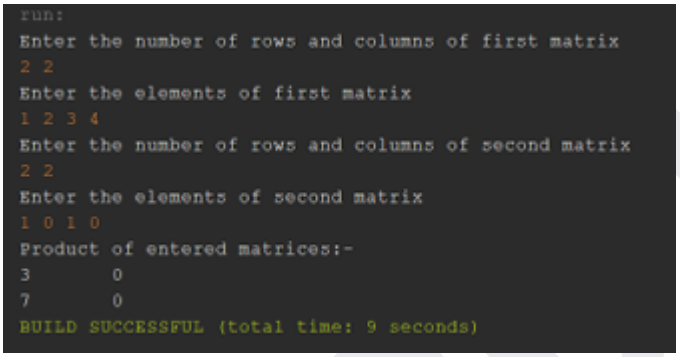
3. After reading the second matrix to verify inputs.

Steps to Fix Errors:

1. Correct Array Indexing: Change to `sum = sum + first[c][k] * second[k][d]` for proper indexing.
2. Reset Variables: Reset `sum = 0;` at the start of the `d` loop.
3. Fix Input Prompts: Correct the second matrix prompt to be clear.
4. Adjust Output Formatting: Add headers for clarity in the output.

7) Quadratic Probing:





Errors Identified:

1. Syntax Error: In the insert method, `i + =` should be `i +=` to fix the syntax error.
2. Incorrect Hashing Logic: Modifying `h` in the line `i = (i + h * h++) % maxSize;` can cause an infinite loop. This needs correction.
3. Key Removal Logic: `currentSize--` is decremented twice in the remove method, leading to size mismanagement.
4. Uninitialized Value Printing: The print method might display null or improperly formatted outputs.
5. Clear Method Logic: The `makeEmpty` method doesn't clear the actual elements in the arrays, potentially causing memory issues.

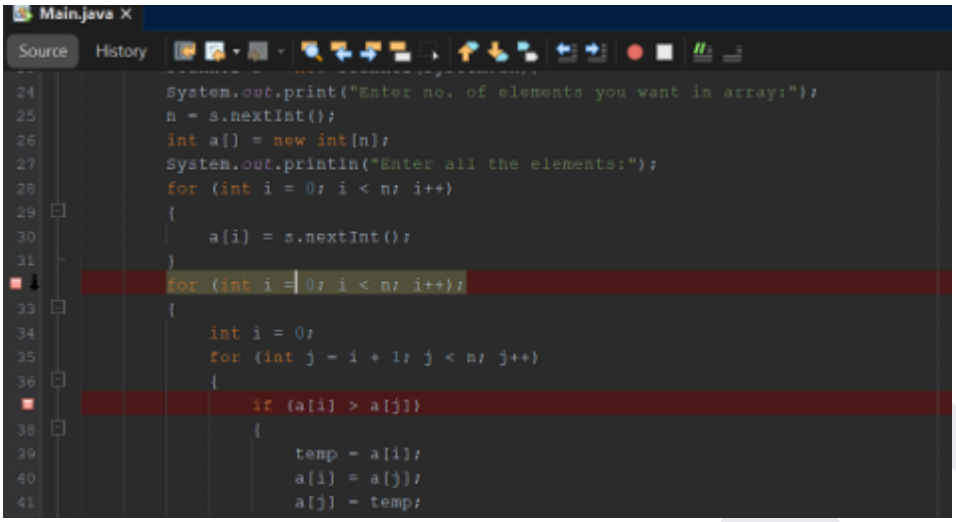
Breakpoints Needed:

1. Insert Method: Before the line with `i +=` to check `i`'s value.
2. Hash Method: To inspect hash value calculations for different keys.
3. Remove Method: To ensure correct key removal and check hash table state after removal.
4. Print Method: To verify the values being printed are correct.

Steps to Fix Errors:

1. Fix Insert Method: Correct the syntax by removing the space and adjust logic for `h`.
2. Fix Hashing Logic: Ensure `h` isn't modified within the loop to prevent infinite loops.
3. Correct Remove Logic: Ensure `currentSize--` is decremented only once after removal.
4. Fix Print Method: Prevent printing of null values and improve output formatting.
5. Improve Clear Logic: Adjust `makeEmpty` to clear the contents of both keys and values arrays properly.

8) Sorting Array:




```
run:
Enter no. of elements you want in array:5
Enter all the elements:
9 8 7 6 4
Ascending Order:4,6,7,8,9BUILD SUCCESSFUL (total time: 5 seconds)
```

Errors Identified:

1. Class Name Error: `Ascending _Order` contains a space, which is invalid in Java. It should be `AscendingOrder`.
2. Incorrect Loop Condition: The outer loop `for (int i = 0; i >= n; i++);` will never run because of the wrong condition (`i >= n`). It should be `i < n`.
3. Unnecessary Semicolon: The semicolon after the outer loop (`for (int i = 0; i >= n; i++);`) ends the loop prematurely.
4. Sorting Logic: The condition `if (a[i] < a[j])` is wrong for ascending order. It should be `if (a[i] > a[j])`.
5. Output Formatting: Printing the array directly can result in an extra comma at the end. It needs proper formatting to avoid this.

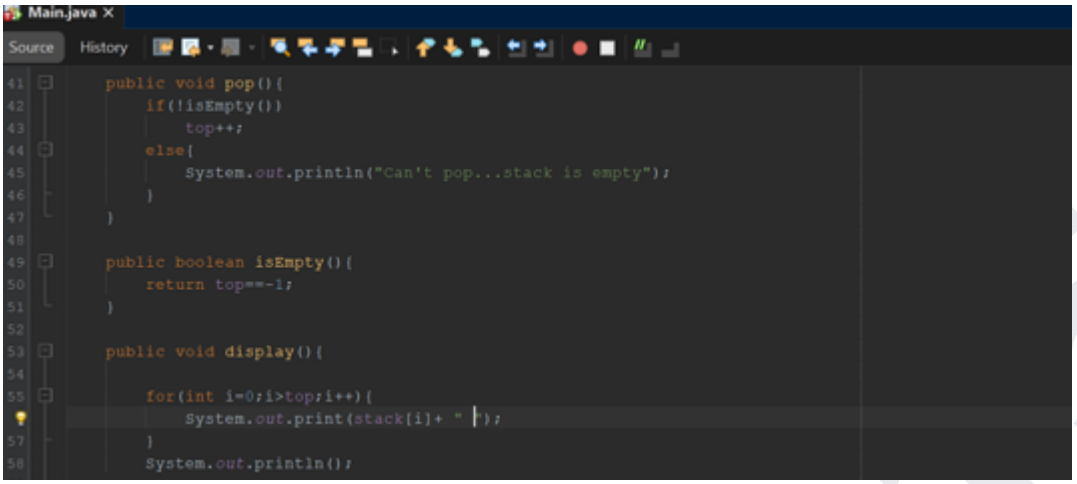
Breakpoints Needed:

1. Breakpoint at the Class Declaration: To check the proper class name.
2. Breakpoint on Outer Loop: To ensure the loop starts and runs correctly.
3. Breakpoint at Sorting Logic: To inspect `a[i]` and `a[j]` values during sorting.
4. Breakpoint at Output: To verify the formatting and avoid extra commas.

Steps to Fix Errors:

1. Rename the Class: Change `Ascending _Order` to `AscendingOrder`.
2. Fix Loop Condition: Correct the loop condition to `i < n`.
3. Remove Semicolon: Delete the unnecessary semicolon after the loop declaration.
4. Correct Sorting Logic: Change the comparison to `if (a[i] > a[j])` for proper sorting.
5. Fix Output Formatting: Adjust the output to avoid trailing commas.

9) Stack:



Errors Identified:

1. Incorrect Logic in `push` Method: `top--` should be `top++` to correctly push an element onto the stack.
2. Incorrect Logic in `pop` Method: `top++` should be `top--` to remove the top element of the stack.
3. Incorrect Condition in `display` Method: The loop condition `for (int i = 0; i > top; i++)` should be `i <= top` to display all elements.
4. Handling Stack Underflow: The `pop` method should return the value being popped before decrementing `top`.
5. Displaying the Stack: The display logic needs adjustment to correctly show the stack after popping.

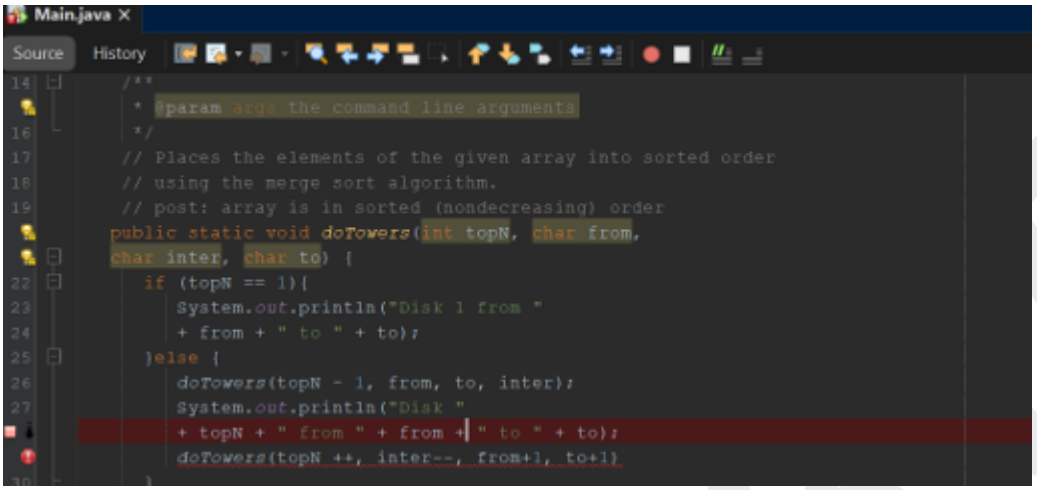
Breakpoints Needed:

1. Breakpoint in `push` Method: To observe the value of `top` before and after pushing.
2. Breakpoint in `pop` Method: To check the value being popped and the state of `top`.
3. Breakpoint in `display` Method: To verify that all elements in the stack are correctly displayed.

Steps to Fix Errors:

1. Fix `push` Logic: Change `top--` to `top++` to push elements at the correct index.
2. Fix `pop` Logic: Change `top++` to `top--` to properly remove elements from the stack.
3. Update Loop in `display`: Change `i > top` to `i <= top` to display all stack elements.
4. Handle Stack Underflow: Modify `pop` to return the value before decrementing `top`.
5. Adjust Display Logic: Ensure the display method accurately shows the stack after popping elements.

10) Tower Of Hanoi:




```
run:
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
BUILD SUCCESSFUL (total time: 0 seconds)
```

Errors Identified:

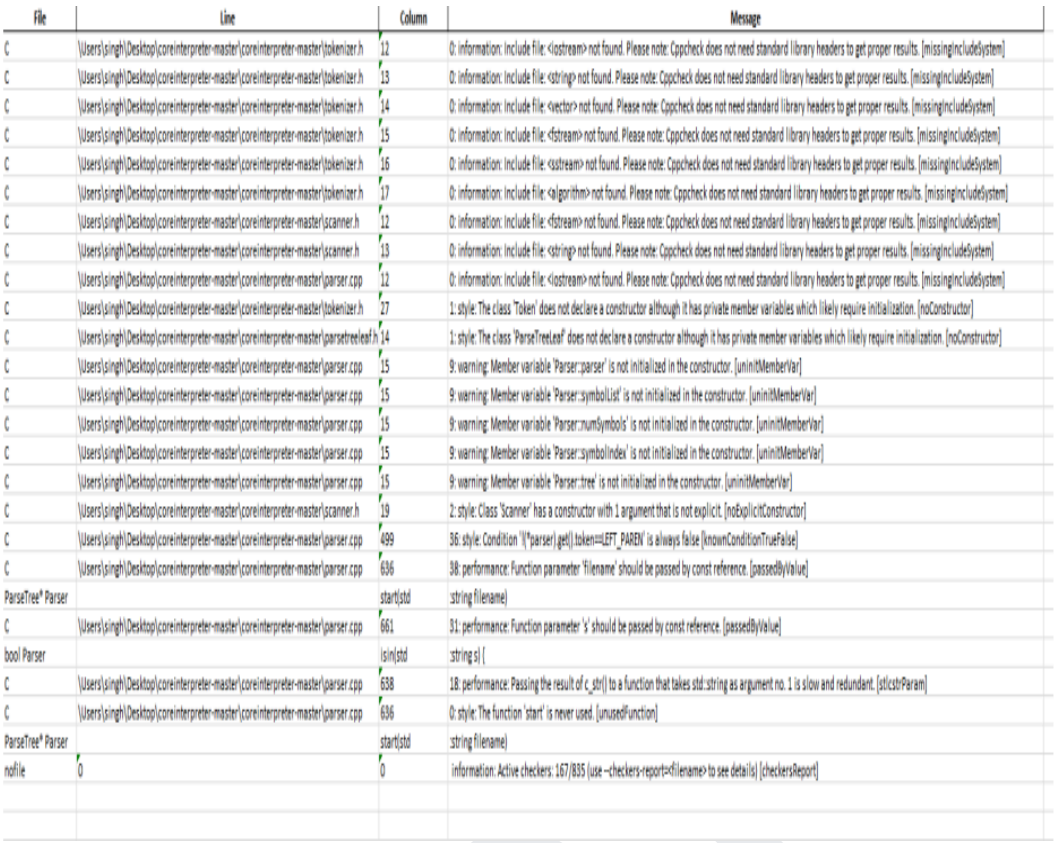
1. Incorrect Increment/Decrement in Recursive Call: The line `doTowers(topN++, inter--, from+1, to+1)` is incorrect because post-increment (`++`) and post-decrement (`--`) are wrongly applied. They don't modify the values passed to the function.
2. Missing Recursive Call for Disk Movement: The logic for handling disk movements in the recursive calls is incorrect, leading to errors in the Tower of Hanoi calculations.
3. Printing Issues: The output is incorrect due to improper handling of parameters, causing wrong disk movement instructions.

Breakpoints Needed:

1. Breakpoint on the first `doTowers` call: To verify the values of `topN`, `from`, `inter`, and `to` before the recursive calls.
2. Breakpoint before the print statement: To ensure the correct flow of disk movements.
3. Breakpoint on the second `doTowers` call: To check the parameters being passed after the first recursive call.

Steps to Fix Errors:

1. Fix Recursive Call: Change `doTowers(topN++, inter--, from+1, to+1)` to `doTowers(topN - 1, inter, from, to)` to handle disk movement correctly.
2. Remove Invalid Modifications: Stop using `++` and `--` on `from`, `inter`, and `to`; pass the original variables directly.
3. Correct Disk Movement Logic: Ensure that the recursive logic follows the correct Tower of Hanoi algorithm for accurate disk movements.



File	Line	Column	Message
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\tokenizer.h	12	0: information: Include file: <iostream> not found. Please note: Cppcheck does not need standard library headers to get proper results. [missingIncludeSystem]
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\tokenizer.h	13	0: information: Include file: <string> not found. Please note: Cppcheck does not need standard library headers to get proper results. [missingIncludeSystem]
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\tokenizer.h	14	0: information: Include file: <vector> not found. Please note: Cppcheck does not need standard library headers to get proper results. [missingIncludeSystem]
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\tokenizer.h	15	0: information: Include file: <fstream> not found. Please note: Cppcheck does not need standard library headers to get proper results. [missingIncludeSystem]
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\tokenizer.h	16	0: information: Include file: <sstream> not found. Please note: Cppcheck does not need standard library headers to get proper results. [missingIncludeSystem]
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\tokenizer.h	17	0: information: Include file: <algorithm> not found. Please note: Cppcheck does not need standard library headers to get proper results. [missingIncludeSystem]
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\scanner.h	12	0: information: Include file: <fstream> not found. Please note: Cppcheck does not need standard library headers to get proper results. [missingIncludeSystem]
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\scanner.h	13	0: information: Include file: <string> not found. Please note: Cppcheck does not need standard library headers to get proper results. [missingIncludeSystem]
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\parser.cpp	12	0: information: Include file: <iostream> not found. Please note: Cppcheck does not need standard library headers to get proper results. [missingIncludeSystem]
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\tokenizer.h	27	1: style: The class 'Token' does not declare a constructor although it has private member variables which likely require initialization. [noConstructor]
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\parsetreeleaf.h	14	1: style: The class 'ParseTreeLeaf' does not declare a constructor although it has private member variables which likely require initialization. [noConstructor]
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\parser.cpp	15	9: warning: Member variable 'Parser::parser' is not initialized in the constructor. [uninitMemberVar]
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\parser.cpp	15	9: warning: Member variable 'Parser::symbolList' is not initialized in the constructor. [uninitMemberVar]
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\parser.cpp	15	9: warning: Member variable 'Parser::numSymbols' is not initialized in the constructor. [uninitMemberVar]
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\parser.cpp	15	9: warning: Member variable 'Parser::symbolIndex' is not initialized in the constructor. [uninitMemberVar]
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\parser.cpp	15	9: warning: Member variable 'Parser::tree' is not initialized in the constructor. [uninitMemberVar]
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\scanner.h	19	2: style: Class 'Scanner' has a constructor with 1 argument that is not explicit. [noExplicitConstructor]
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\parser.cpp	499	36: style: Condition '!(*parser).get().token==LEFT_PAREN' is always false [knownConditionTrueFalse]
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\parser.cpp	636	38: performance: Function parameter 'filename' should be passed by const reference. [passedByValue]
ParseTree* Parser		start(std	:string filename)
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\parser.cpp	661	31: performance: Function parameter 's' should be passed by const reference. [passedByValue]
bool Parser		isin(std	:string s) {
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\parser.cpp	638	18: performance: Passing the result of c_str() to a function that takes std::string as argument no. 1 is slow and redundant. [stlcstrParam]
C	\Users\singh\Desktop\coreinterpreter-master\coreinterpreter-master\parser.cpp	636	0: style: The function 'start' is never used. [unusedFunction]
ParseTree* Parser		start(std	:string filename)
nofile	0	0	information: Active checkers: 167/835 (use --checkers-report=<filename> to see details) [checkersReport]

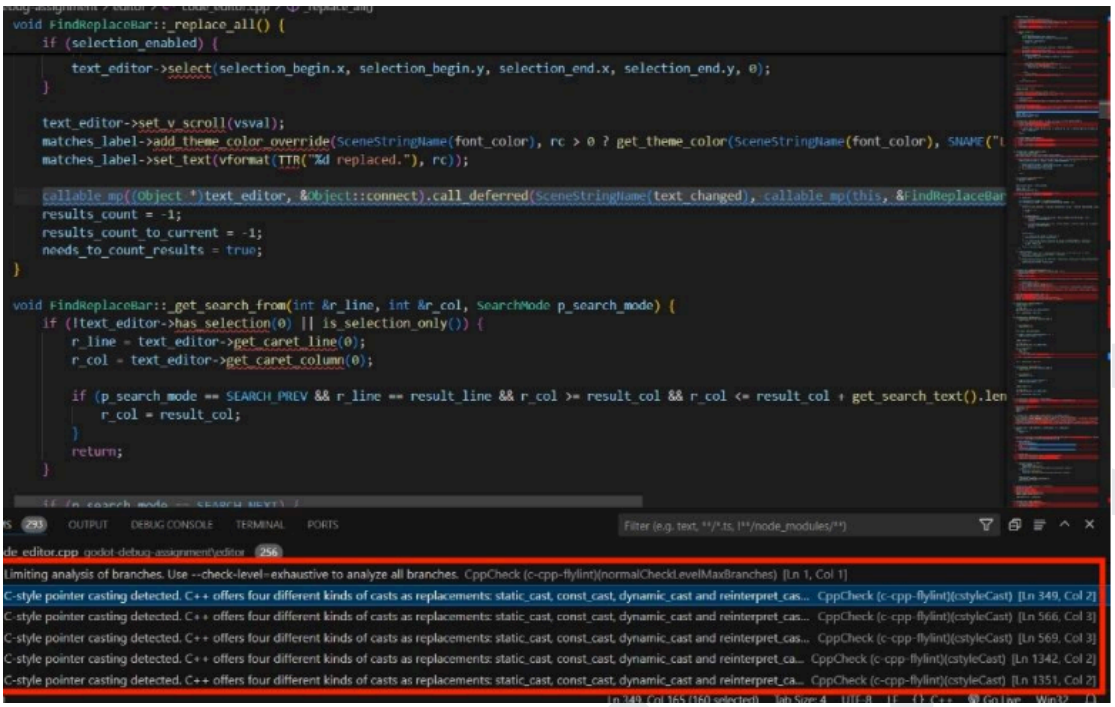
nofile:0:0: information: Active checkers: 167/835 (use--checkers-report= to see details) [checkersReport] In Excel format: (as above)

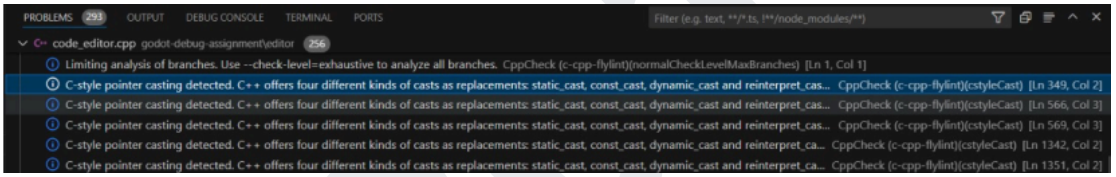
III.STATIC ANALYSIS TOOLS

Choose a static analysis tool (in Java, Python, C, C++) in any programming language of your interest and identify the defects. You can also choose your own code fragment from GitHub (more than 2000 LOC) in any programming language to perform static analysis.

Programming Language :

C++ Static Analysis Tool Used : CppCheck The static analysis tool gives 6 errors from CppCheck while Intellisense from VSCode gives 250 errors which are mostly due to lack of header file access permissions. The results of static analysis tools are given in the Excel file with the submissions


void FindReplaceBar::_replace_all() {
if (selection_enabled) {
text_editor->select(selection_begin.x, selection_begin.y, selection_end.x, selection_end.y, 0);
}
text_editor->set_v_scroll(vsval);
matches_label->add_theme_color_override(SceneStringName(font_color), rc > 0 ? get_theme_color(SceneStringName(font_color), SNAME("
matches_label->set_text(vformat(TTR("%d replaced."), rc));
callable_mp((Object *)text_editor, &Object::connect).call_deferred(SceneStringName(text_changed), callable_mp(this, &FindReplaceBar
results_count = -1;
results_count_to_current = -1;
needs_to_count_results = true;
}
void FindReplaceBar::_get_search_from(int &r_line, int &r_col, SearchMode p_search_mode) {
if (!text_editor->has_selection(0) || is_selection_only()) {
r_line = text_editor->get_caret_line(0);
r_col = text_editor->get_caret_column(0);
if (p_search_mode == SEARCH_PREV && r_line == result_line && r_col >= result_col && r_col <= result_col + get_search_text().len
r_col = result_col;
}
return;
}
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Filter (e.g. text, **/*.ts, !**/node_modules/**)
Limiting analysis of branches. Use --check-level=exhaustive to analyze all branches. CppCheck (c-cpp-flylint)(normalCheckLevelMaxBranches) [Ln 1, Col 1]
C-style pointer casting detected. C++ offers four different kinds of casts as replacements: static_cast, const_cast, dynamic_cast and reinterpret_cas... CppCheck (c-cpp-flylint)(cstyleCast) [Ln 349, Col 2]
C-style pointer casting detected. C++ offers four different kinds of casts as replacements: static_cast, const_cast, dynamic_cast and reinterpret_cas... CppCheck (c-cpp-flylint)(cstyleCast) [Ln 566, Col 3]
C-style pointer casting detected. C++ offers four different kinds of casts as replacements: static_cast, const_cast, dynamic_cast and reinterpret_cas... CppCheck (c-cpp-flylint)(cstyleCast) [Ln 569, Col 3]
C-style pointer casting detected. C++ offers four different kinds of casts as replacements: static_cast, const_cast, dynamic_cast and reinterpret_ca... CppCheck (c-cpp-flylint)(cstyleCast) [Ln 1342, Col 2]
C-style pointer casting detected. C++ offers four different kinds of casts as replacements: static_cast, const_cast, dynamic_cast and reinterpret_ca... CppCheck (c-cpp-flylint)(cstyleCast) [Ln 1351, Col 2]


PROBLEMS 293
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Filter (e.g. text, **/*.ts, !**/node_modules/**)
code_editor.cpp godot-debug-assignment\editor 256
Limiting analysis of branches. Use --check-level=exhaustive to analyze all branches. CppCheck (c-cpp-flylint)(normalCheckLevelMaxBranches) [Ln 1, Col 1]
C-style pointer casting detected. C++ offers four different kinds of casts as replacements: static_cast, const_cast, dynamic_cast and reinterpret_cas... CppCheck (c-cpp-flylint)(cstyleCast) [Ln 349, Col 2]
C-style pointer casting detected. C++ offers four different kinds of casts as replacements: static_cast, const_cast, dynamic_cast and reinterpret_cas... CppCheck (c-cpp-flylint)(cstyleCast) [Ln 566, Col 3]
C-style pointer casting detected. C++ offers four different kinds of casts as replacements: static_cast, const_cast, dynamic_cast and reinterpret_cas... CppCheck (c-cpp-flylint)(cstyleCast) [Ln 569, Col 3]
C-style pointer casting detected. C++ offers four different kinds of casts as replacements: static_cast, const_cast, dynamic_cast and reinterpret_ca... CppCheck (c-cpp-flylint)(cstyleCast) [Ln 1342, Col 2]
C-style pointer casting detected. C++ offers four different kinds of casts as replacements: static_cast, const_cast, dynamic_cast and reinterpret_ca... CppCheck (c-cpp-flylint)(cstyleCast) [Ln 1351, Col 2]