Lab-7

IT314 Software Engineering

202201145 – Meet Andharia

CODE 1:

1. **How many errors can you identify in the program? List the errors below.**

   Category A: Data Reference Errors Uninitialized or unset variables:

   The constructor method is mistakenly written as `_init_` instead of `__init__`, preventing it from being executed properly. As a result, variables such as `self.matrix`, `self.vector`, and `self.res` are left uninitialized, leading to potential issues later in the program. Additionally, there is a computation error in the `gauss` method, where a division by zero could occur. Specifically, if the element `A[i][i]` is zero, the operation `x[i] /= A[i][i]` would attempt to divide by zero, causing an error and potentially halting the execution of the program.

   Category D: Comparison Errors Faulty comparison logic:

   In the `diagonal_dominance` method, the check for diagonal dominance may produce incorrect results if a row contains duplicate maximum values, leading to unexpected behavior.

   Category E: Control-Flow Errors Off-by-one mistake:

   In the get_upper_permute method, the loop that iterates over k is used for k in range(i+1, n+1) but should instead be for k in range(i+1, n) to avoid accessing out-of-bound elements. Category F: Interface Errors Incorrect handling of parameters: The method definitions assume specific input formats (e.g., a list of lists for the matrix and a list for the vector) without validating input formats or checking for unexpected values.

   Category G: Input/Output Errors Missing error handling for input data:

   The program lacks proper handling for cases where input matrices are non-square or have dimensions that are incompatible with the operations being performed.

Category H: Other Checks Missing libraries:

The code does not include the necessary imports for libraries like `numpy` and `matplotlib.pyplot`, which are essential for the program to run correctly.

2. **Which category of inspection would you consider more useful?**

   Category D: Comparison Errors Incorrect comparison logic:

   The `diagonal_dominance` method may produce incorrect results when rows contain duplicate maximum values, leading to inaccurate diagonal dominance checks and potential unexpected behavior. Resolving this issue can help prevent logical errors in the computations.

   Category A: Data Reference Errors Uninitialized variables:

   Defining `__init__` incorrectly as `_init_` prevents proper initialization of `self.matrix`, `self.vector`, and `self.res`. Correcting the constructor definition will ensure that these data members are properly initialized, addressing issues related to uninitialized variables that are essential for the program's correct execution.

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

   <u>Logic Errors</u>

   Program inspection is useful for identifying syntax and runtime errors; however, it may not catch logical errors. These types of errors occur when the program executes without crashing but produces incorrect outcomes due to flaws in the algorithms or miscalculations. For instance, in methods like Jacobi or Gauss-Seidel, the algorithms might exhibit slow convergence or fail to converge in specific scenarios—issues that program inspection alone might not uncover.

4. **Is the program inspection technique worth applying?**

   Absolutely, program inspection is a valuable technique. It aids in identifying common and critical errors, especially those associated with data references, computations, comparisons, and control flow. While it is an effective practice for enhancing the quality and robustness of the

code, it should be complemented with other testing methods, such as unit and integration testing, to uncover logical errors and verify that the program functions correctly across different scenarios.

CODE 2:

1. **How many errors are there in the program? Mention the errors you have identified.**
   Category A: Data Reference Errors Constructor Naming Error:
   The constructor is incorrectly defined as `_init_` instead of `__init__`, preventing it from being invoked when an instance of the `Interpolation` class is created.

   Potential Index Errors: In the `cubicSpline` and `piecewise_linear_interpolation` methods, matrix elements are accessed directly without checking if the indices are within the valid range, which could result in an `IndexError`.

   Lack of Type Checking: There are no validations to ensure that the matrix elements are numeric (either integers or floats). If non-numeric types are provided, this may lead to runtime errors.

   Category C: Computation Errors Division by Zero Risk:
   In the piecewise_linear_interpolation method, calculating the slope can result in division by zero if two consecutive x-values are identical.

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

   Data Reference Errors (Category A) would be the most effective focus for inspection in this code. This category targets important issues, such as incorrect initialization and index management, including the incorrectly named constructor (`_init_` instead of `__init__`) and potential index errors when accessing matrix elements. Addressing

these issues is essential to prevent runtime errors and ensure that the program operates reliably.

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

   <u>Runtime Errors:</u>

   Some runtime errors may not be detectable through program inspection. For example, floating-point inaccuracies—such as those that could result in division by zero in the `piecewise_linear_interpolation` method—or errors that arise only during execution due to improper handling of certain data sets might go unnoticed.

4. **Is the program inspection technique worth applying?**

   Yes, program inspection is an effective technique. It helps identify many potential issues early in the development process, enhances code quality, encourages adherence to best practices, and minimizes long-term costs associated with debugging and maintenance. However, it should be complemented by other testing methods, such as unit testing and dynamic analysis, to ensure more comprehensive coverage of possible errors.

CODE 3:

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

   Category A: Data Reference Errors
   Redundant function definitions: The functions `fun` and `dfun` are defined multiple times for different equations without indicating which function corresponds to which equation.
   Undefined behavior on variable reuse: The `data` variable is reused to store different iterations of results but is not clearly reset in each function call, which may lead to unexpected behavior when solving multiple roots consecutively.

   Category B: Data-Declaration Errors
   Uninitialized variables: In the initial loop, `next` is calculated before being initialized during the first iteration, which can result in NaN values.
   Improper DataFrame initialization: The DataFrame `df` is created only after the loop, meaning that if the loop does not execute (due to immediate convergence), the data may not be well-formed, leading to potential errors.

   Category C: Computation Errors
   Inaccurate function evaluation: The statement `fpresent = fun(present)` should also check for convergence based on `|fun(present)|` instead of relying solely on the value of `next` for convergence.
   Error calculation logic: The logic for computing the error with...
   Error calculation logic: The statement `error.append(next - present)` may not accurately reflect the true convergence behavior, as it compares the last and second-to-last iterations instead of comparing the correct consecutive values used in the iterative process.

   Category D: Comparison Errors
   Incorrect error condition: The error condition checks the difference between `next` and `present`, but it may not take into account that `present` could be very close to alpha without actually converging.

Insufficient convergence criteria: The convergence criteria rely solely on `abs(next - present) > err`, overlooking the importance of checking the function value itself, specifically `|fun(next)| < err`.

Category E: Control-Flow Errors

Infinite loop risk: If the initial guess is far from the actual root or if `dfun(present)` equals zero (indicating vertical tangents), the loop may enter an infinite loop without achieving convergence.

Lack of break conditions: There are no safeguards to terminate the loop after a specified number of iterations or to prevent division by zero in the statement `next = present - (fpresent / dfpresent)`.

Category F: Input/Output Errors

Lack of iteration logging: The code does not include any logging or console output during the iterations, making it challenging to track the algorithm's progress.

Misleading plot titles: The plot titles do not clearly indicate which function or root they represent, leading to confusion when analyzing multiple roots from different functions.

Category G: Other Checks

Unchecked edge cases: The code does not address edge cases where the function may not have a root in the specified domain or where the derivative could lead to undefined behavior.

Multiple plots without clearing previous data: Each new plot is generated without clearing the data from previous plots, resulting in cluttered visualizations when multiple functions are tested in succession.

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

   Data Reference Errors (Category A): This category ensures that inputs are handled and defined correctly, preventing many runtime errors and improving the program's robustness.

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

Non-obvious Logical Errors: These could include problems such as converging to an incorrect root or encountering numerical instability, which might not become apparent until runtime with certain input values

4. **Is the program inspection technique worth applying?**

   Yes, program inspection is an effective technique. It can uncover a wide range of errors and greatly improve code quality. Program inspection techniques are particularly valuable in collaborative environments, enhancing code readability and maintainability.

CODE 4:

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

   Category A: Data Reference Errors
   Inconsistent input structure: The input matrix is expected to be a 2D array, but the code does not validate or manage incorrect input shapes, which may result in runtime errors.
   Variable reuse without clear definition: The variables `coef` and `poly_i` are reused in different scopes (both inside and outside the function), leading to confusion about their intended meanings.

   Category B: Data-Declaration Errors
   Uninitialized variables in plotting: The plotting function `plot_fun` does not consider situations where `y` might be empty or improperly initialized, resulting in errors when attempting to create a plot.
   No error handling for matrix inversion: There is no verification to ensure that `ATA` is invertible before invoking `np.linalg.inv(ATA)`, which could lead to a crash if the matrix is singular.

   Category C: Computation Errors

Potential loss of precision: The line `coef = coef[::-1]` reverses the coefficients, but `np.poly1d` expects them to be in descending order. If not properly aligned, this could lead to unexpected polynomial behavior.
Overwriting coefficients: The coefficients for each order are computed and stored in `coef` within a loop, but they are not isolated for each polynomial, which may create confusion about which coefficients correspond to which polynomial.

Category D: Comparison Errors
Incorrect error tolerance: The hardcoded value `err = 1e-3` in `plot_fun` may not be suitable for all datasets and lacks flexibility to dynamically adjust based on input ranges.
Inadequate comparison logic in plotting: The code does not guarantee that each polynomial is distinctly labeled, nor does it ensure that the plot's legend accurately reflects the plotted lines.

Category E: Control-Flow Errors
Infinite loop risk in plotting: The `plot_fun` function could potentially enter an infinite loop if incorrectly formatted data is provided, especially if there are no points to plot.
Lack of early exit conditions: The `leastSquareErrorPolynomial` function does not implement early exit conditions to detect poorly conditioned matrices or when the degree `m` is too high for the number of points.

Category F: Input/Output Errors
No user feedback on processing: The code lacks print statements or logging mechanisms to indicate progress or completion of operations, which can leave users unaware of the processing status.

Category F: Input/Output Errors
No user feedback on processing: The program does not provide any print statements or logging mechanisms to indicate the progress of polynomial fitting, making it challenging for users to monitor execution.
Misleading variable naming: The variable `poly_i` may be confusing, as it implies a single polynomial when it actually contains a polynomial object. A more descriptive name would enhance clarity.

Category G: Other Checks

No handling of edge cases: The function does not account for scenarios where all `y` values are identical, which would result in a constant polynomial, potentially leading to user confusion.

Lack of unit tests or assertions: There are no unit tests or assertions to verify input parameters and ensure that the function performs correctly across various cases.

Category H: General Code Quality

Redundant code sections: The code for plotting multiple polynomials contains redundancy and could be organized into a function for better reusability.

Missing function documentation: The functions lack documentation, making it more difficult for other users (or even the original author) to comprehend their purpose and expected behavior.

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

Computation Errors (Category C):

Ensuring that computations are performed correctly is crucial for the accuracy of results, especially in numerical methods like polynomial fitting.

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

Data-Specific Errors: Certain edge cases with input data (e.g., all y values being the same) may not be identified until the function is executed with specific datasets.

4. **Is the program inspection technique worth applying?**

Yes, program inspection is a valuable technique. It allows for systematic error identification and improvement in code structure and maintainability, making it especially valuable in complex numerical methods and data analysis tasks.

SECTION _ II

Armstrong Number: Errors and Fixes

1. **How many errors are there in the program?**
   There are 2 errors in the program.

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

   Steps Taken to Fix the Errors:
   Error 1: The division and modulus operations are incorrectly positioned in the while loop.
   Fix: Adjust the operations so that the modulus operation extracts the last digit, while the division operation updates the number for the next iteration.

   Error 2: The accumulation of the check variable is flawed.
   Fix: Revise the logic to ensure that the check variable correctly represents the sum of each digit raised to the power of the total number of digits.

```
class Armstrong {
   // Method to check if a number is an Armstrong number
   public boolean isArmstrong(int num) {
      int n = num; // store original number for comparison
      int check = 0, remainder;

      while (num > 0) {
         remainder = num % 10;
         check += Math.pow(remainder, 3); // Calculate the power of
each digit
         num /= 10; // Reduce the number
      }
```

```
        return check == n; // Check if the sum of powers equals the
original number
    }

    public static void main(String args[]) {
        // Check if an argument is provided
        if (args.length == 0) {
            System.out.println("Please provide a number as a command-
line argument.");
            return;
        }

        // Create an instance of the Armstrong class
        Armstrong armstrongChecker = new Armstrong();

        // Parse the input number and check if it is an Armstrong number
        int num = Integer.parseInt(args[0]);
        if (armstrongChecker.isArmstrong(num)) {
            System.out.println(num + " is an Armstrong Number");
        } else {
            System.out.println(num + " is not an Armstrong Number");
        }
    }
}
```

GCD and LCM: Errors and Fixes

1. **How many errors are there in the program?**

   There is 1 error in the program.

2. **How many breakpoints do you need to fix this error?**

   We need 1 breakpoint to fix this error

Steps Taken to Resolve the Issue:

Error: The condition in the while loop of the GCD method was incorrect.

Fix: Update the condition to `while (a % b != 0)` instead of `while (a % b == 0)`. This adjustment ensures that the loop continues until the remainder is zero, allowing for the correct calculation of the GCD.

```
import java.util.Scanner;


public class GCD_LCM {


   static int gcd(int x, int y) {
      int remainder = 0, larger, smaller;
      larger = (x > y) ? x : y; // larger holds the greater number
      smaller = (x < y) ? x : y; // smaller holds the lesser number
      remainder = smaller;


      while (larger % smaller != 0) {
         remainder = larger % smaller;
         larger = smaller;
         smaller = remainder;
      }
      return remainder;
```

```
    }

    static int lcm(int x, int y) {
        int a;
        a = (x > y) ? x : y; // a is the 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);

        // Instantiate GCD_LCM class
        GCD_LCM calculator = new GCD_LCM();

        System.out.println("Enter the two numbers: ");
        int x = input.nextInt();
        int y = input.nextInt();

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

```
        input.close();
    }
}
```

Knapsack Problem: Errors and Fixes

1. **How many errors are there in the program?**
   There are 3 errors in the program.

2. **How many breakpoints do you need to fix these errors?**
   We need 2 breakpoints to fix these errors.

Steps Taken to Resolve the Errors:

- **Error:** The condition in the "take item n" case is incorrect.
  Fix: Update the condition from if (weight[n] > w) to if (weight[n] <= w) to ensure the profit is calculated when the item can be included.

- **Error:** The profit calculation is incorrect.
  Fix: Change profit[n-2] to profit[n] to ensure that the correct profit value is utilized.

- **Error:** In the "don't take item n" case, the indexing is incorrect.
  Fix: Update opt[n++][w] to opt[n-1][w] to properly index the items.

```
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] + 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\tProfit\tWeight\tTake");
        for (int n = 1; n <= N; n++) {
            System.out.println(n + "\t" + profit[n] + "\t" + weight[n] + "\t" +
    take[n]);
        }
      }
    }
```

Magic Number Check: Errors and Fixes

1. **How many errors are there in the program?**
   There are 3 errors in the program.

2. **How many breakpoints do you need to fix these errors?**
   We need 1 breakpoint to fix these errors.

Steps Taken to Fix the Errors:

- **Error:** The condition in the inner while loop is incorrect.

  Fix: Change while(sum==0) to while(sum!=0) to ensure that the loop processes digits correctly.

- **Error:** The calculation of s in the inner loop is incorrect.

  Fix: Change s=s*(sum/10) to s=s+(sum%10) to correctly sum the digits.

- **Error:** The order of operations in the inner while loop is incorrect.

  Fix: Reorder the operations to s=s+(sum%10); sum=sum/10; to correctly accumulate the digit sum.

```
import java.util.Scanner;


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.");
        }
        ob.close();
    }
}
```

Merge Sort: Errors and Fixes

1. **How many errors are there in the program?**

   There are 3 errors in the program.

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

   We need 2 breakpoints to fix these errors.

   Steps Taken to Resolve the Errors:

   - Error: There was an issue with array indexing while splitting the array in the mergeSort function.
     - Fix: Modify int[] left = leftHalf(array + 1) to int[] left = leftHalf(array) and change int[] right = rightHalf(array - 1) to int[] right = rightHalf(array) to correctly pass the entire array.
   - Error: The increment and decrement operations in the merge function were incorrect.
     - Fix: Eliminate the ++ and -- from merge(array, left++, right--) and use merge(array, left, right) to pass the arrays directly.
   - Error: The merge function was accessing elements beyond the boundaries of the array.
     - Fix: Adjust the indexing in the merging logic to ensure that the array boundaries are respected.

```
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; // Calculate size for right half
    int size2 = array.length - size1; // Remaining elements for right half
    int[] right = new int[size2];
    for (int i = 0; i < size2; i++) {
        right[i] = array[i + size1]; // Fill right half
```

```
        }
        return right;
    }

    public static void merge(int[] result, int[] left, int[] right) {
        int i1 = 0; // Index for left array
        int i2 = 0; // Index for right array
        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++;
            }
        }
    }
}
```

Matrix Multiplication: Errors and Fixes

1. **How many errors are there in the program?**
   There is 1 error in the program.

2. **How many breakpoints do you need to fix this error?**
   We need 1 breakpoint to fix this error.

Steps Taken to Fix the Error:

- **Error:** The array indexing in the matrix multiplication logic is incorrect.

- **Fix:** Update first[c-1][c-k] and second[k-1][k-d] to first[c][k] and second[k][d]. These adjustments ensure that the matrix elements are referenced correctly during multiplication.

```
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);
    }
  }
}
```

Quadratic Probing Hash Table

- **Errors and Fixes:**
  - **How many errors are there in the program?**

    There is 1 error in the program.
  - **How many breakpoints do you need to fix this error?**

    We need 1 breakpoint to fix this error.
  - **Steps Taken to Fix the Error:**

- Error: In the insert method, the line i += (i + h / h--) % maxSize; is incorrect.
- Fix: The correct logic should be i = (i + h * h++) % maxSize; to correctly implement quadratic probing.

```
import java.util.Scanner;


class QuadraticProbingHashTable {
    private int currentSize, maxSize;
    private String[] keys;
    private String[] vals;

    public QuadraticProbingHashTable(int capacity) {
        currentSize = 0;
        maxSize = capacity;
        keys = new String[maxSize];
        vals = new String[maxSize];
    }

    public void makeEmpty() {
        currentSize = 0;
        keys = new String[maxSize];
        vals = new String[maxSize];
    }

    public int getSize() {
        return currentSize;
    }
```

```
public boolean isFull() {
   return currentSize == maxSize;
}

public boolean isEmpty() {
   return getSize() == 0;
}

public boolean contains(String key) {
   return get(key) != null;
}

private int hash(String key) {
   return key.hashCode() % maxSize;
}

public void insert(String key, String val) {
   int tmp = hash(key);
   int i = tmp, h = 1;

   do {
      if (keys[i] == null) {
         keys[i] = key;
         vals[i] = val;
         currentSize++;
```

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

public String get(String key) {
    int i = hash(key), h = 1;

    while (keys[i] != null) {
        if (keys[i].equals(key))
            return vals[i];
        i = (i + h * h++) % maxSize;
    }
    return null;
}

public void remove(String key) {
    if (!contains(key))
        return;

    int i = hash(key), h = 1;
```

```
        while (!key.equals(keys[i]))
            i = (i + h * h++) % maxSize;

        keys[i] = vals[i] = null;
        currentSize--;

        for (i = (i + h * h++) % maxSize; keys[i] != null; i = (i + h * h++) % maxSize) {
            String tmp1 = keys[i];
            String tmp2 = vals[i];
            keys[i] = vals[i] = null;
            currentSize--;
            insert(tmp1, tmp2);
        }
    }

    public void printHashTable() {
        System.out.println("\nHash Table:");
        for (int i = 0; i < maxSize; i++) {
            if (keys[i] != null)
                System.out.println(keys[i] + " " + vals[i]);
        }
        System.out.println();
    }
}
```

```
public class QuadraticProbingHashTableTest {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Hash Table Test\n\n");
        System.out.println("Enter size");
        QuadraticProbingHashTable qpht = new
QuadraticProbingHashTable(scan.nextInt());
        char ch;

        do {
            System.out.println("\nHash Table Operations\n");
            System.out.println("1. insert ");
            System.out.println("2. remove");
            System.out.println("3. get");
            System.out.println("4. clear");
            System.out.println("5. size");
            int choice = scan.nextInt();

            switch (choice) {
                case 1:
                    System.out.println("Enter key and value");
                    qpht.insert(scan.next(), scan.next());
                    break;
                case 2:
                    System.out.println("Enter key");
                    qpht.remove(scan.next());
                    break;
```

```
            case 3:
                System.out.println("Enter key");
                System.out.println("Value = " + qpht.get(scan.next()));
                break;
            case 4:
                qpht.makeEmpty();
                System.out.println("Hash Table Cleared\n");
                break;
            case 5:
                System.out.println("Size = " + qpht.getSize());
                break;
            default:
                System.out.println("Wrong Entry\n");
                break;
        }
        qpht.printHashTable();
        System.out.println("\nDo you want to continue (Type y or n) \n");
        ch = scan.next().charAt(0);
    } while (ch == 'Y' || ch == 'y');
  }
}
```

Sorting Array

- **Errors and Fixes:**
    - **How many errors are there in the program?**

      There are 2 errors in the program.
    - **How many breakpoints do you need to fix this error?**

We need 2 breakpoints to fix these errors.

- **Steps Taken to Fix the Errors:**
  - **Error 1:** The loop condition for `for (int i = 0; i >= n; i++);` is incorrect.
  - **Fix 1:** Change it to `for (int i = 0; i < n; i++)` to correctly iterate over the array.
  - **Error 2:** The condition in the inner loop `if (a[i] <= a[j])` should be reversed.
  - **Fix 2:** Change it to `if (a[i] > a[j])` to correctly sort the array in ascending order.

```
import java.util.Scanner;

public class AscendingOrder {
   public static void main(String[] args) {
      int n, temp;
      Scanner s = new Scanner(System.in);

      System.out.print("Enter the number of elements you want in the 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();
      }

      // Corrected sorting logic
```

```
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (a[i] > a[j]) { // Fixed comparison
                    temp = a[i];
                    a[i] = a[j];
                    a[j] = temp;
                }
            }
        }

        System.out.print("Ascending Order: ");
        for (int i = 0; i < n - 1; i++) {
            System.out.print(a[i] + ", ");
        }
        System.out.print(a[n - 1]);

        s.close(); // Close the scanner
    }
}
```

Stack Implementation

Errors and Fixes:

- How many errors are there in the program? There are 2 errors in the program.
- How many breakpoints do you need to fix this error? We need 2 breakpoints to fix these errors.

Steps Taken to Fix the Errors:

- **Error 1:** In the push method, the line top-- is incorrect.

- **Fix 1:** Change it to top++ to correctly increment the stack pointer.
- **Error 2:** In the display method, the loop condition for (int i=0; i>top; i++) is incorrect.
- **Fix 2:** Change it to for (int i=0; i<=top; i++) to correctly display all elements.

```
public class StackMethods {
    private int top;
    int size;
    int[] stack;

    public StackMethods(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++; // Fixed increment
            stack[top] = value;
        }
    }

    public void pop() {
```

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

    public boolean isEmpty() {
        return top == -1;
    }

    public void display() {
        for (int i = 0; i <= top; i++) { // Corrected loop condition
            System.out.print(stack[i] + " ");
        }
        System.out.println();
    }
}

public class StackReviseDemo {
    public static void main(String[] args) {
        StackMethods newStack = new StackMethods(5);
        newStack.push(10);
        newStack.push(1);
        newStack.push(50);
        newStack.push(20);
```

```
        newStack.push(90);
        newStack.display();

        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.pop();

        newStack.display();
    }
}
```

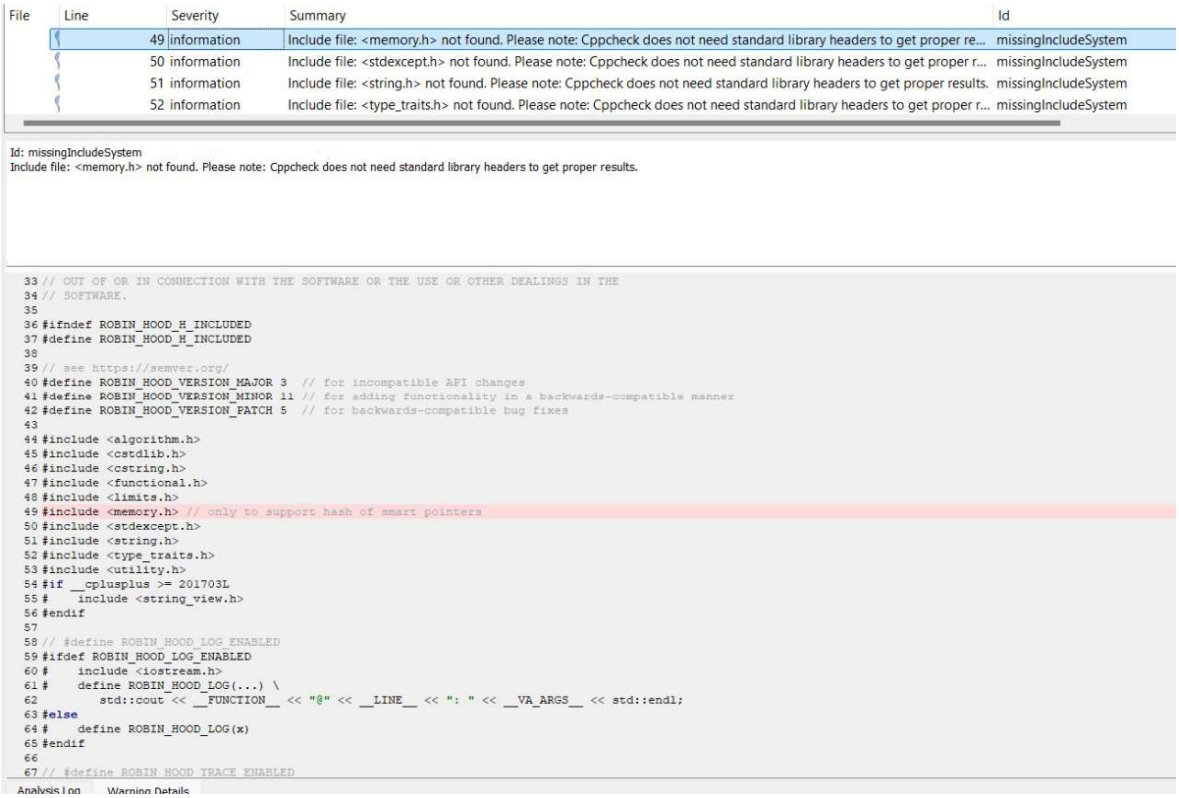
Tower of Hanoi
Errors and Fixes:

- **How many errors are there in the program?**
  There is 1 error in the program.
- **How many breakpoints do you need to fix this error?**
  We need 1 breakpoint to fix this error.
- **Steps Taken to Fix the Error:**
  - **Error**: In the recursive call doTowers(topN ++, inter--, from+1, to+1);, incorrect increments and decrements are applied to the variables.
  - **Fix:** Change the call to doTowers(topN - 1, inter, from, to); for proper recursion and to follow the Tower of Hanoi logic.

```
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);34 | P a g e
            System.out.println("Disk " + topN + " from " + from + " to " + to);
            doTowers(topN - 1, inter, from, to); // Corrected recursive call
        }
    }
}
```

Static Analysis Tools

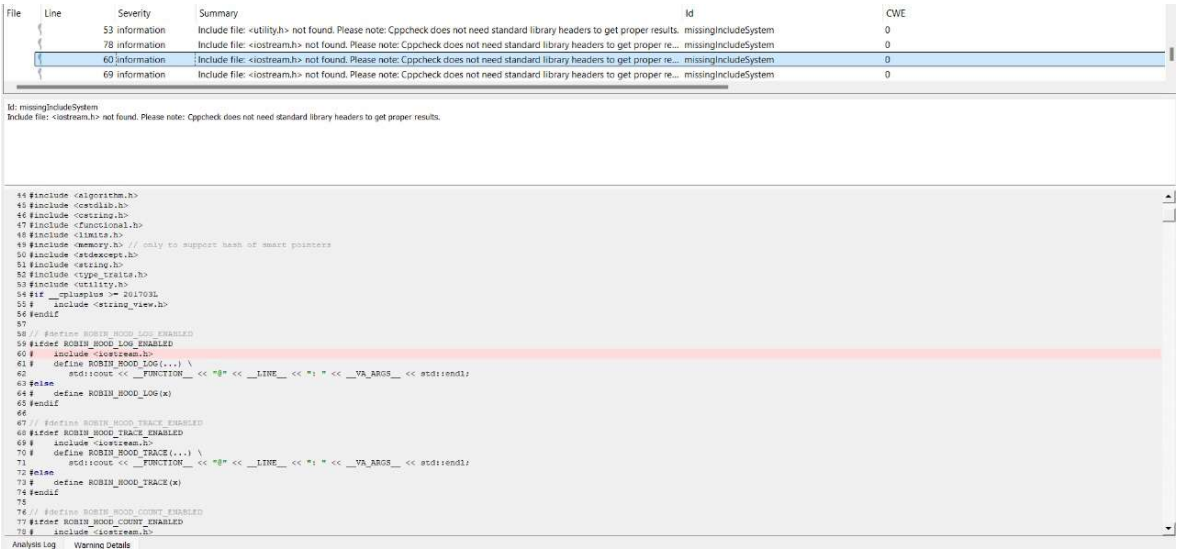


File	Line	Severity	Summary	Id
	49	information	Include file: <memory.h> not found. Please note: Cppcheck does not need standard library headers to get proper re...	missingIncludeSystem
	50	information	Include file: <stdexcept.h> not found. Please note: Cppcheck does not need standard library headers to get proper r...	missingIncludeSystem
	51	information	Include file: <string.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.	missingIncludeSystem
	52	information	Include file: <type_traits.h> not found. Please note: Cppcheck does not need standard library headers to get proper r...	missingIncludeSystem

Id: missingIncludeSystem
Include file: <memory.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

```
33 // OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
34 // SOFTWARE.
35
36 #ifndef ROBIN_HOOD_H_INCLUDED
37 #define ROBIN_HOOD_H_INCLUDED
38
39 // see https://semver.org/
40 #define ROBIN_HOOD_VERSION_MAJOR 3  // for incompatible API changes
41 #define ROBIN_HOOD_VERSION_MINOR 11 // for adding functionality in a backwards-compatible manner
42 #define ROBIN_HOOD_VERSION_PATCH 5  // for backwards-compatible bug fixes
43
44 #include <algorithm.h>
45 #include <cstdlib.h>
46 #include <cstring.h>
47 #include <functional.h>
48 #include <limits.h>
49 #include <memory.h> // only to support hash of smart pointers
50 #include <stdexcept.h>
51 #include <string.h>
52 #include <type_traits.h>
53 #include <utility.h>
54 #if __cplusplus >= 201703L
55 #    include <string_view.h>
56 #endif
57
58 // #define ROBIN_HOOD_LOG_ENABLED
59 #ifdef ROBIN_HOOD_LOG_ENABLED
60 #    include <iostream.h>
61 #    define ROBIN_HOOD_LOG(...) \
62         std::cout << __FUNCTION__ << "@" << __LINE__ << ": " << __VA_ARGS__ << std::endl;
63 #else
64 #    define ROBIN_HOOD_LOG(x)
65 #endif
66
67 // #define ROBIN_HOOD_TRACE_ENABLED
```

Analysis Log Warning Details

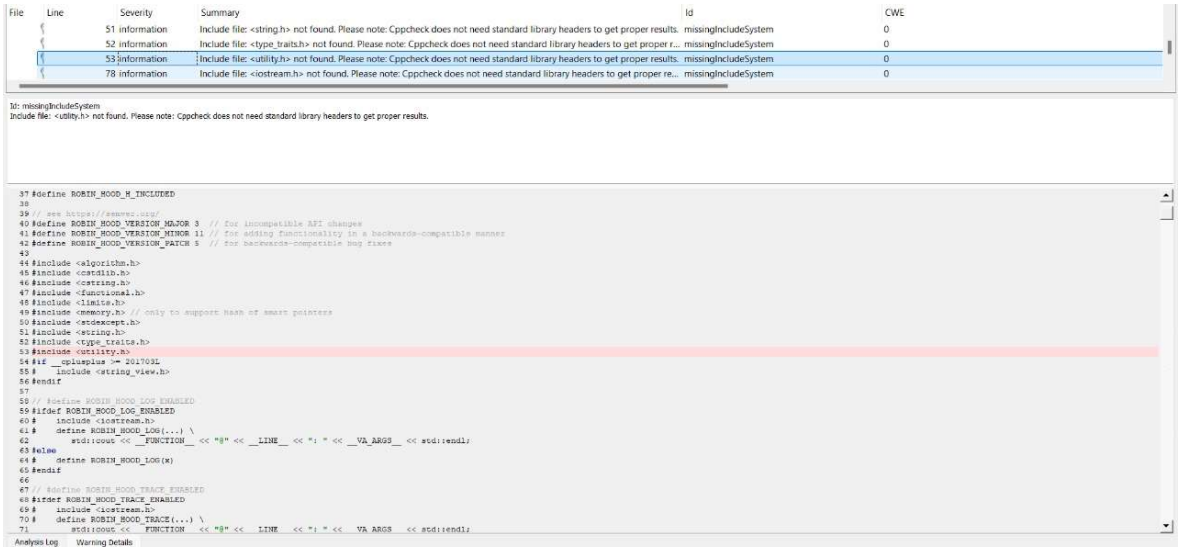


File	Line	Severity	Summary	Id	CWE
	53	information	Include file: <utility.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.	missingIncludeSystem	0
	78	information	Include file: <iostream.h> not found. Please note: Cppcheck does not need standard library headers to get proper re...	missingIncludeSystem	0
	60	information	Include file: <iostream.h> not found. Please note: Cppcheck does not need standard library headers to get proper re...	missingIncludeSystem	0
	69	information	Include file: <iostream.h> not found. Please note: Cppcheck does not need standard library headers to get proper re...	missingIncludeSystem	0

Id: missingIncludeSystem
Include file: <iostream.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

```
44 #include <algorithm.h>
45 #include <cstdlib.h>
46 #include <cstring.h>
47 #include <functional.h>
48 #include <limits.h>
49 #include <memory.h> // only to support hash of smart pointers
50 #include <stdexcept.h>
51 #include <string.h>
52 #include <type_traits.h>
53 #include <utility.h>
54 #if __cplusplus >= 201703L
55 #    include <string_view.h>
56 #endif
57
58 // #define ROBIN_HOOD_LOG_ENABLED
59 #ifdef ROBIN_HOOD_LOG_ENABLED
60 #    include <iostream.h>
61 #    define ROBIN_HOOD_LOG(...) \
62         std::cout << __FUNCTION__ << "@" << __LINE__ << ": " << __VA_ARGS__ << std::endl;
63 #else
64 #    define ROBIN_HOOD_LOG(x)
65 #endif
66
67 // #define ROBIN_HOOD_TRACE_ENABLED
68 #ifdef ROBIN_HOOD_TRACE_ENABLED
69 #    include <iostream.h>
70 #    define ROBIN_HOOD_TRACE(...) \
71         std::cout << __FUNCTION__ << "@" << __LINE__ << ": " << __VA_ARGS__ << std::endl;
72 #else
73 #    define ROBIN_HOOD_TRACE(x)
74 #endif
75
76 // #define ROBIN_HOOD_COUNT_ENABLED
77 #ifdef ROBIN_HOOD_COUNT_ENABLED
78 #    include <iostream.h>
```

Analysis Log Warning Details



File	Line	Severity	Summary	Id	CWE
	51	information	Include file: <string.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.	missingIncludeSystem	0
	52	information	Include file: <type_traits.h> not found. Please note: Cppcheck does not need standard library headers to get proper r...	missingIncludeSystem	0
	53	information	Include file: <utility.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.	missingIncludeSystem	0
	78	information	Include file: <iostream.h> not found. Please note: Cppcheck does not need standard library headers to get proper re...	missingIncludeSystem	0

Id: missingIncludeSystem
Include file: <utility.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

```
37 #define ROBIN_HOOD_H_INCLUDED
38
39 // see https://semver.org/
40 #define ROBIN_HOOD_VERSION_MAJOR 3  // for incompatible API changes
41 #define ROBIN_HOOD_VERSION_MINOR 11 // for adding functionality in a backwards-compatible manner
42 #define ROBIN_HOOD_VERSION_PATCH 5  // for backwards-compatible bug fixes
43
44 #include <algorithm.h>
45 #include <cstdlib.h>
46 #include <cstring.h>
47 #include <functional.h>
48 #include <limits.h>
49 #include <memory.h> // only to support hash of smart pointers
50 #include <stdexcept.h>
51 #include <string.h>
52 #include <type_traits.h>
53 #include <utility.h>
54 #if __cplusplus >= 201703L
55 #    include <string_view.h>
56 #endif
57
58 // #define ROBIN_HOOD_LOG_ENABLED
59 #ifdef ROBIN_HOOD_LOG_ENABLED
60 #    include <iostream.h>
61 #    define ROBIN_HOOD_LOG(...) \
62         std::cout << __FUNCTION__ << "@" << __LINE__ << ": " << __VA_ARGS__ << std::endl;
63 #else
64 #    define ROBIN_HOOD_LOG(x)
65 #endif
66
67 // #define ROBIN_HOOD_TRACE_ENABLED
68 #ifdef ROBIN_HOOD_TRACE_ENABLED
69 #    include <iostream.h>
70 #    define ROBIN_HOOD_TRACE(...) \
71         std::cout <<  __FUNCTION__  << "@" <<  __LINE__  << ": " <<  __VA_ARGS__  << std::endl;
```

Analysis Log | Warning Details