IT314 - Software Engineering

Lab - 07

Nisharg Modi (202201346)

Code Debugging

1. Armstrong Number

1. Errors in the program

The code contains several logical and syntax errors:

1. **Incorrect logic for Armstrong number calculation:**
   - The remainder = num / 10; line is incorrect. It should be remainder = num % 10;. This is because the remainder when dividing by 10 gives the last digit of the number.
   - The line num = num % 10; should be num = num / 10; because you need to remove the last digit, not reduce it to the remainder.
2. **The number of digits isn't considered:**
   - An Armstrong number for n digits requires summing the nth powers of the digits. The code currently raises each digit to the power of 3 without counting the digits dynamically.
3. **Input edge cases (like negative numbers, non-numeric input) aren't handled:**
   - The code assumes that the input will always be a valid number.
4. **Code block isn't properly closed:**
   - The class Armstrong is missing a closing brace } at the end.

2. Breakpoints to fix errors:

- You would need **two breakpoints** to examine the changes in remainder and num inside the loop:
  1. After calculating remainder = num % 10.
  2. After updating num = num / 10 to confirm the digits are being removed correctly.

3. Steps to fix the errors:

- **Fix the digit extraction**:
  - Replace remainder = num / 10; with remainder = num % 10;.
- **Fix the digit removal**:
  - Replace num = num % 10; with num = num / 10;.
- **Add digit count logic**:
  - Add logic to calculate the number of digits n, and raise the digit to the power of n instead of a constant 3.

- **Add closing bracket**:
    - Add } at the end of the code to properly close the class.

4. Correct code

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

        // Convert input to integer
        int num = Integer.parseInt(args[0]);
        int n = num; // Store the original number
        int check = 0, remainder, digitCount = 0;

        // Calculate the number of digits
        int temp = num;
        while (temp != 0) {
            temp /= 10;
            digitCount++;
        }

        // Check if Armstrong
        temp = num;
        while (temp > 0) {
            remainder = temp % 10; // Get last digit
            check = check + (int) Math.pow(remainder, digitCount); // Add
digit^n to check
            temp /= 10; // Remove last digit
        }
        // Compare the calculated value with the original number
        if (check == n)
            System.out.println(n + " is an Armstrong Number");
        else
            System.out.println(n + " is not an Armstrong Number");
    }
}
```

2. GCD and LCM

1.Errors Identified in the Program:

- **Error 1 (Computation Error in GCD method)**:
  - **Issue**: In the `gcd()` method, the `while` loop condition is incorrect. It currently has `while(a % b == 0)`, which will exit the loop as soon as the first valid division happens without a remainder, failing to compute the correct GCD. The correct condition should be `while(a % b != 0)` so the loop continues until a is divisible by b without a remainder.
  - **Fix**: Replace `while(a % b == 0)` with `while(a % b != 0)`.
- **Error 2 (Computation Error in LCM method)**:
  - **Issue**: In the `lcm()` method, the logic inside the `while(true)` loop is incorrect. It uses `if(a % x != 0 && a % y != 0)`, which will return a when both x and y do not divide a—which is the opposite of what LCM requires. LCM requires both x and y to divide a.
  - **Fix**: Change the condition to `if(a % x == 0 && a % y == 0)` to ensure both numbers divide evenly.

2. Breakpoints and Fixes:

  - 1 breakpoint in the `gcd()` method.
  - 1 breakpoint in the `lcm()` method.

3.Fix Steps:

- **GCD Method**: Change the loop condition to `while(a % b != 0)` to ensure the remainder is checked properly.
- **LCM Method**: Fix the condition inside the loop to check for both numbers dividing a.

4. Correct code:

```
import java.util.Scanner;

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

        while (a % b != 0) { // Fix: while condition updated
            r = a % b;
            a = b;
            b = r;
        }
        return b; // Return b (the correct GCD)
    }

    static int lcm(int x, int y) {
        int a;
        a = (x > y) ? x : y; // Start from the greater number
        while (true) {
            if (a % x == 0 && a % y == 0) { // Fix: Correct condition for
LCM
                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

1. Errors Identified in the Program:

There are several errors in the provided program, primarily logical or related to array indexing:

- **Error 1 (Computation Error in the opt array update)**:
  - **Issue**: In the loop int option1 = opt[n++][w];, using n++ instead of n results in skipping an iteration and incorrect index access. The ++ operator should not be used here.
  - **Fix**: Change int option1 = opt[n++][w]; to int option1 = opt[n][w];.
- **Error 2 (Computation Error in calculating option2)**:
  - **Issue**: The expression if (weight[n] > w) is incorrect for checking whether item n can fit in the knapsack. It should be if (weight[n] <= w) (i.e., the item's weight should be less than or equal to the current capacity).
  - **Fix**: Replace if (weight[n] > w) with if (weight[n] <= w).
- **Error 3 (Array Index Error in calculating option2)**:
  - **Issue**: The line option2 = profit[n-2] + opt[n-1][w-weight[n]]; incorrectly references profit[n-2]. This should be profit[n] since we are considering the current item n.
  - **Fix**: Change profit[n-2] to profit[n].

2. Breakpoints and Fixes:

- **Breakpoints Required**:
  - 1 breakpoint in the loop that calculates option1 and option2 for checking the array index logic.
  - 1 breakpoint in the condition where weight[n] is checked.

3. Fix Steps:

- **Step 1**: Replace int option1 = opt[n++][w]; with int option1 = opt[n][w];.
- **Step 2**: Replace if (weight[n] > w) with if (weight[n] <= w).
- **Step 3**: Update option2 = profit[n-2] + opt[n-1][w-weight[n]]; to option2 = profit[n] + opt[n-1][w-weight[n]];.

4. Correct Code

```
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][w];  // Fix: no increment in n

                // take item n
                int option2 = Integer.MIN_VALUE;
                if (weight[n] <= w) // Fix: check if the weight fits
                    option2 = profit[n] + opt[n-1][w-weight[n]]; // Fix:
use profit[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]);
        }
    }
}
```

4. Magic Number

1. Errors Identified in the Program:

There are several errors in the provided code, which need to be fixed:

- **Error 1 (Logical Error in the Inner while Loop)**:
    - **Issue**: In the line while(sum == 0), it incorrectly checks sum == 0, which will never execute as sum starts from the input number. It should check whether sum > 0 to process the digits.
    - **Fix**: Change while(sum == 0) to while(sum > 0).
- **Error 2 (Computation Error in Summing Digits)**:
    - **Issue**: In the line s = s * (sum / 10);, the logic for summing the digits is incorrect. It should be adding the remainder of sum % 10 to s rather than multiplying.
    - **Fix**: Change s = s * (sum / 10); to s = s + (sum % 10);.
- **Error 3 (Missing Semicolon)**:
    - **Issue**: There is a missing semicolon in sum = sum % 10.

    - **Fix**: Add a semicolon: sum = sum % 10;.

2. Breakpoints and Fixes:

- **Breakpoints Required**:
    - 1 breakpoint at the start of the inner while loop to check the logic for summing the digits.
    - 1 breakpoint after updating sum to verify if the correct digit is being summed.

3. **Fix Steps**:

- **Step 1**: Replace while(sum == 0) with while(sum > 0).
- **Step 2**: Replace s = s * (sum / 10); with s = s + (sum % 10);.
- **Step 3**: Add a missing semicolon after sum = sum % 10.

4. Correct code

```java
import java.util.*;
public class MagicNumberCheck {
    public static void main(String args[]) {
        Scanner ob = new Scanner(System.in);
        System.out.println("Enter the number to be checked.");
        int n = ob.nextInt();
        int sum = 0, num = n;

        // Repeat the process until the number becomes a single digit
        while (num > 9) {
            sum = num;
            int s = 0;

            // Summing the digits of the number
            while (sum > 0) {  // Fix: Check if sum > 0
                s = s + (sum % 10);  // Fix: Add the digit instead of
multiplying
                sum = sum / 10;  // Fix: Correctly move to the next digit
            }
            num = s;  // Update num with the sum of digits
        }

        // Check if the single digit is 1 (indicating a Magic Number)
        if (num == 1) {
            System.out.println(n + " is a Magic Number.");
        } else {
            System.out.println(n + " is not a Magic Number.");
        }

        ob.close();
    }
}
```

5. Merge sort

1. Errors Identified in the Program:

There are a few errors in the provided code which need to be fixed:

- **Error 1 (Incorrect Array Manipulation in mergeSort)**:
  - **Issue**: The code int[] left = leftHalf(array+1); and int[] right = rightHalf(array-1); is incorrect. You cannot add or subtract from an array like this in Java.
  - **Fix**: Simply pass array as an argument to leftHalf(array) and rightHalf(array).
- **Error 2 (Incorrect Parameters in merge Call)**:
  - **Issue**: In the merge method call, merge(array, left++, right--); is incorrect. You cannot increment or decrement arrays.
  - **Fix**: Pass left and right directly as merge(array, left, right);.

2. Breakpoints and Fixes:

- **Breakpoints Required**:
  - 1 breakpoint at the start of the mergeSort method to check array splitting.
  - 1 breakpoint inside the merge method to check if merging is happening correctly.

3. Fix Steps:

- **Step 1**: Remove the +1 and -1 from array+1 and array-1.
- **Step 2**: Correct the increment/decrement errors in the merge method call.

4. Correct Code

```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));
    }

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

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

            // merge the sorted halves into a sorted whole
            merge(array, left, right);  // Fix: Pass left and right arrays
directly
        }
    }

    // Returns the first half of the given array.
    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;
    }

    // Returns the second half of the given array.
    public static int[] rightHalf(int[] array) {
        int size1 = array.length / 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;
    }

    // Merges the given left and right arrays into the given
    // result array.
    // pre : result is empty; left/right are sorted
    // post: result contains result of merging sorted lists;
    public static void merge(int[] result,
                             int[] left, int[] right) {
        int i1 = 0;   // index into left array
        int i2 = 0;   // index into 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];    // take from left
                i1++;
            } else {
                result[i] = right[i2];   // take from right
                i2++;
            }
        }
    }
}
```

6. Multiply Matrices

1. Errors Identified in the Program:

There are multiple errors in the provided code that need fixing:

- **Error 1 (Array Index Out of Bounds)**:
  - **Issue**: The lines first[c-1][c-k] and second[k-1][k-d] will throw an ArrayIndexOutOfBoundsException. When c or k is 0, this will attempt to access a negative index in the array.
  - **Fix**: The indices should be first[c][k] and second[k][d] to correctly access the elements.
- **Error 2 (Incorrect Input for Second Matrix)**:
  - **Issue**: The prompt for the second matrix incorrectly states "Enter the number of rows and columns of first matrix" again instead of indicating it's for the second matrix.
  - **Fix**: Change the prompt to correctly indicate it's for the second matrix.
- **Error 3 (Improper Resetting of sum)**:
  - **Issue**: The variable sum is being reused in the multiplication loop without being reset to 0 at the beginning of the outer loops.
  - **Fix**: Initialize sum = 0; before starting the inner multiplication loop for each cell in the resulting matrix.

2. Breakpoints and Fixes:

- **Breakpoints Required**:
  - A breakpoint inside the nested loops to verify the values of c, d, k, and the computed sum for each multiplication step.

3. Fix Steps:

- **Step 1**: Change the matrix element access from first[c-1][c-k] to first[c][k].
- **Step 2**: Change the second matrix prompt to "Enter the number of rows and columns of the second matrix".
- **Step 3**: Initialize sum = 0; at the start of the multiplication loop.

4. Correct Code

```
import java.util.Scanner;

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

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

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

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

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

        if (n != p)
            System.out.println("Matrices with entered orders can't be 
multiplied with each other.");
        else {
            int second[][] = new int[p][q];
            int multiply[][] = new int[m][q];

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

            for (c = 0; c < m; c++) {
                for (d = 0; d < q; d++) {
                    sum = 0; // Initialize sum to 0 at the beginning of 
the multiplication loop
```

```
                    for (k = 0; k < n; k++) { // Use n instead of p to
iterate through columns of the first matrix
                        sum = sum + first[c][k] * second[k][d]; // Fixed
index access
                    }
                    multiply[c][d] = sum;
                }
            }

            System.out.println("Product of entered matrices:-");
            for (c = 0; c < m; c++) {
                for (d = 0; d < q; d++)
                    System.out.print(multiply[c][d] + "\t");
                System.out.print("\n");
            }
        }
    }
}
```

7. Quadratic Probing

1. Errors Identified in the Program

The provided code contains several errors and improvements that need to be addressed:

1. **Incorrect Syntax in Insertion Logic**:
   - **Error**: In the insert method, the line i + = (i + h / h--) % maxSize; has incorrect syntax (+ = should be +=).
   - **Fix**: Correct it to i += (h * h + i) % maxSize;.
2. **Improper Use of Increment Operator in get and remove Methods**:
   - **Error**: In both the get and remove methods, the line i = (i + h * h++) % maxSize; will cause incorrect indexing due to post-increment being used.
   - **Fix**: Change to i = (i + h * h) % maxSize; and increment h after this line.
3. **Variable Initialization in makeEmpty**:
   - **Error**: The initialization of keys and vals arrays in makeEmpty() does not actually reset the existing array contents to null; it only reallocates them.
   - **Fix**: Use a loop to set all entries to null.

4. **The Prompt and Inputs in the main Method**:
    - **Error**: The prompt states to enter key and value, but it's unclear how many key-value pairs are to be entered.
    - **Fix**: Clarify in the prompt or structure it to allow multiple inserts.
5. **Logic Error in remove Method**:
    - **Error**: When rehashing the keys after a deletion, the original key should be searched correctly, and the current size should not be decremented until after insertion.
    - **Fix**: Adjust the logic to ensure correct handling of rehashing.

2. Breakpoints and Fixes

- **Breakpoints Required**:
    - Set breakpoints in the insert, get, remove, and makeEmpty methods to observe key changes and index accesses.

3. Fix Steps:

- **Step 1**: Correct the syntax issues in the insert, get, and remove methods.
- **Step 2**: Implement proper initialization in makeEmpty.
- **Step 3**: Clarify input instructions in main for users.
- **Step 4**: Fix the rehashing logic in the remove method.

4. Correct code

```
import java.util.Scanner;

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

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

    /** Function to clear hash table **/
    public void makeEmpty() {
        currentSize = 0;
        for (int i = 0; i < maxSize; i++) {
            keys[i] = null;
            vals[i] = null;
        }
    }

    /** Function to get size of hash table **/
    public int getSize() {
        return currentSize;
    }

    /** Function to check if hash table is full **/
    public boolean isFull() {
        return currentSize == maxSize;
    }

    /** Function to check if hash table is empty **/
    public boolean isEmpty() {
        return getSize() == 0;
    }

    /** Function to check if hash table contains a key **/
    public boolean contains(String key) {
        return get(key) != null;
    }

    /** Function to get hash code of a given key **/
    private int hash(String key) {
        return (key.hashCode() & 0x7fffffff) % maxSize; // Use absolute
value
    }

    /** Function to insert key-value pair **/
    public void insert(String key, String val) {
        int tmp = hash(key);
```

```java
        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;
            h++;
        } while (i != tmp);
    }

    /** Function to get value for a given key **/
    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;
            h++;
        }
        return null;
    }

    /** Function to remove key and its value **/
    public void remove(String key) {
        if (!contains(key))
            return;

        /** find position key and delete **/
        int i = hash(key), h = 1;

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

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

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

        currentSize--;
    }

    /** Function to print HashTable **/
    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();
    }
}

/** Class QuadraticProbingHashTableTest **/
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");
        /** Create object of QuadraticProbingHashTable **/
        QuadraticProbingHashTable qpht = new
QuadraticProbingHashTable(scan.nextInt());

        char ch;
```

```java
        /** Perform QuadraticProbingHashTable operations **/
        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;
            }

            /** Display hash table **/
            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');
    }
}
```

8. Sorting Array

1. Errors Identified in the Program

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

1. **Class Name Syntax**:
   - The class name Ascending _Order contains a space, which is not allowed in Java.
2. **Outer Loop Condition**:
   - The outer loop's condition is incorrect: for (int i = 0; i >= n; i++);. It should be i < n, and it has an unnecessary semicolon at the end, which causes the loop to terminate prematurely.
3. **Sorting Logic**:
   - The condition for swapping elements in the inner loop is incorrect. It uses a[i] <= a[j] instead of a[i] > a[j].
4. **Output Loop**:
   - While the output loop works, it can be improved to format the output more clearly (e.g., avoiding a trailing comma).

2. Breakpoints and Fixes

You would need to set **four breakpoints** to address each of the identified errors:

1. Set at the line defining the class to check for syntax errors.
2. Set at the beginning of the outer loop to inspect the loop condition.
3. Set inside the inner loop to monitor the swapping condition and the swapping logic.
4. Set in the output section to verify the output formatting.

3. Fix Steps:

1. **Fix Class Name**:
   - Changed Ascending _Order to AscendingOrder.
2. **Correct Outer Loop**:
   - Modified the outer loop condition from for (int i = 0; i >= n; i++); to for (int i = 0; i < n; i++).
3. **Adjust Sorting Logic**:
   - Changed the comparison in the inner loop from if (a[i] <= a[j]) to if (a[i] > a[j]).
4. **Improve Output Formatting**:
   - Updated the output loop to add commas between elements without leaving a trailing comma at the end.

4. Correct code

```java
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 number of elements you want in array: ");
        n = s.nextInt();
        int a[] = new int[n];
        System.out.println("Enter all the elements:");
        for (int i = 0; i < n; i++) {
            a[i] = s.nextInt();
        }

        // Sorting the array in ascending order
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (a[i] > a[j]) {  // Change <= to >
                    temp = a[i];
                    a[i] = a[j];
                    a[j] = temp;
                }
            }
        }
        // Printing the sorted array
        System.out.print("Ascending Order: ");
        for (int i = 0; i < n; i++) {
            System.out.print(a[i]);
            if (i < n - 1) {
                System.out.print(", "); // Add comma for elements except
the last one
            }
        }
        System.out.println(); // Move to the next line after printing the
array
    }
}
```

9. Stack Implementation

1. Errors Identified in the Program

- In the push() method, the line top--; should actually be top++; because when pushing, the top index increases. Decrementing top causes the stack to overwrite elements incorrectly.
- In the display() method, the loop should iterate from top down to 0 (i.e., from the top of the stack to the bottom), not in reverse. The condition in the for loop should be i <= top and not i > top.
- The starting condition in the display() method (for(int i = 0; i > top; i++)) is incorrect and needs to be i <= top to ensure that all elements in the stack are printed.
- The current output won't match the expected output because of these errors. The program needs to correctly manage the top of the stack and its printing.

2. Breakpoints and Fixes

We need **2 breakpoints** to debug this program and fix the errors:

- Inside the push() method, to check how top is being updated.
- Inside the display() method, to see how the loop is iterating over the stack.

3. Fix Steps:

1. The line top-- should be corrected to top++ to properly increment the stack's top pointer when a value is pushed.
2. Change the loop condition in display() to correctly iterate from 0 to top. Modify the for loop to be: for(int i = 0; i <= top; i++).
3. Ensure that after a value is popped, the top pointer is correctly decremented by 1 (top--), not incremented.

4. Correct code

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

    public StackMethods(int arraySize) {
        size = arraySize;
        stack = new int[size];
        top = -1;  // Initializing top to -1 to indicate an empty stack
    }

    public void push(int value) {
        if (top == size - 1) {
            System.out.println("Stack is full, can't push a value");
        } else {
            top++;  // Increment top to push value on the stack
            stack[top] = value;
        }
    }

    public void pop() {
        if (!isEmpty()) {
            System.out.println("Popped: " + stack[top]);  // Print popped
value
            top--;  // Decrement top after popping a value
        } else {
            System.out.println("Can't pop...stack is empty");
        }
    }

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

    public void display() {
        if (isEmpty()) {
            System.out.println("Stack is empty");
        } else {
            System.out.print("Stack elements: ");
            for (int i = 0; i <= top; i++) {  // Iterate from 0 to top to
print stack elements
                System.out.print(stack[i] + " ");
            }
            System.out.println();
        }
    }
}

public class StackReviseDemo {
    public static void main(String[] args) {
        StackMethods newStack = new StackMethods(5);

        // Pushing elements into the stack
        newStack.push(10);
        newStack.push(1);
        newStack.push(50);
        newStack.push(20);
        newStack.push(90);
        // Displaying the stack
        newStack.display();

        // Popping elements from the stack
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.pop();

        // Displaying the stack after popping
        newStack.display();
    }
}
```

10. Tower of Hanoi

1. Errors Identified in the Program

1. **Syntax Error in Recursion Call:**
   - The line doTowers(topN ++, inter--, from+1, to+1) has a syntax issue. ++ and -- should not be used within recursive calls. Instead, you should pass the values as they are, without modifying them.
   - You should also pass the characters (from, inter, to) without altering them by adding integers.
2. **Logical Error with Recursive Call:**
   - In the recursive step, the correct recursive call should be doTowers(topN - 1, inter, from, to) when moving disks between the intermediate and target pegs.

2. Breakpoints and Fixes

- The recursive calls are corrected by removing the ++ and -- operators and passing the topN - 1 directly.
- The characters from, inter, and to should remain unchanged during the recursion.

3. Fix Steps:

- The base case (if (topN == 1)) handles the move of the smallest disk directly from from to to.
- The recursive case breaks the problem into smaller subproblems, moving the top n-1 disks to the intermediate peg, then the nth disk to the target peg, followed by moving the n-1 disks from the intermediate peg to the target peg.

4. Correct code

```
public class MainClass {
   public static void main(String[] args) {
      int nDisks = 3;  // Number of disks
      doTowers(nDisks, 'A', 'B', 'C');  // A = source, B = intermediate, C
= target
   }

   public static void doTowers(int topN, char from, char inter, char to) {
      if (topN == 1) {
            // Base case: Move the smallest disk directly from source to
target
            System.out.println("Disk 1 from " + from + " to " + to);
        } else {
            // Recursive case: Move topN-1 disks from source to intermediate
            doTowers(topN - 1, from, to, inter);
            // Move the nth disk from source to target
            System.out.println("Disk " + topN + " from " + from + " to " +
to);
            // Move the topN-1 disks from intermediate to target
            doTowers(topN - 1, inter, from, to);
        }
    }
}
```

Program inspection of code

Code is taken from this github repository: Link

Category A: Data Reference Errors

1. Does a referenced variable have a value that is unset or uninitialized? This probably is the most frequent programming error; it occurs in a wide variety of circumstances. For each reference to a data item (variable, array element, field in a structure), attempt to "prove" informally that the item has a value at that point.

Ans: All variables (int, duint) are properly initialized before use. There are no instances where a variable is used without a set value.

2. For all array references, is each subscript value within the defined bounds of the corresponding dimension?

Ans: The code does not contain explicit/unbounded array indexing.

3. For all array references, does each subscript have an integer value? This is not necessarily an error in all languages, but it is a dangerous practice.

Ans: While the code does not show array subscripts, in functions where addresses are used ,it is essential to ensure that addresses or pointers used for indexing are integers.

4.For all references through pointer or reference variables, is the referenced memory currently allocated? This is known as the "dangling reference" problem. It occurs in situations where the lifetime of a pointer is greater than the lifetime of the referenced memory. One situation occurs where a pointer references a local variable within a procedure, the pointer value is assigned to an output parameter or a global variable, the procedure returns (freeing the referenced location), and later the program attempts to use the pointer value. In a manner similar to checking for the prior errors, try to prove informally that, in each reference using a pointer variable, the reference memory exists.

Ans: There are no pointers or reference variables utilized in the code.

5.When a memory area has alias names with differing attributes, does the data value in this area have the correct attributes when referenced via one of these names? Situations to look for are the use of the EQUIVALENCE statement in FORTRAN, and the REDEFINES clause in COBOL. As an example, a FORTRAN program contains a real variable A and an integer variable B; both are made

aliases for the same memory area by using an EQUIVALENCE statement. If the program stores a value into A and then references variable B, an error is likely present since the machine would use the floating-point bit representation in the memory area as an integer.

Ans: No aliasing of memory areas is present in the code.

6. Does a variable's value have a type or attribute other than what the compiler expects? This situation might occur where a C, C++, or COBOL program reads a record into memory and references it by using a structure, but the physical representation of the record differs from the structure definition.

Ans: All variables have appropriate types assigned, and there are no mismatched type assignments.

7. Are there any explicit or implicit addressing problems if, on the machine being used, the units of memory allocation are smaller than the units of memory addressability? For instance, in some environments, fixed-length bit strings do not necessarily begin on byte boundaries, but addresses only point to byte boundaries. If a program computes the address of a bit string and later refers to the string through this address, the wrong memory location may be referenced. This situation also could occur when passing a bit-string argument to a subroutine.

Ans: No addressing issues are present, as all variables are properly defined and accessed.

8. If pointer or reference variables are used, does the referenced memory location have the
attributes the compiler expects? An example of such an error is where a C++pointer upon which a data structure is based is assigned the address of a different data structure.

Ans: No pointer or reference variables are present in the code.

9. If a data structure is referenced in multiple procedures or subroutines, is the structure defined identically in each procedure?

Ans: The code does not utilize multiple procedures or subroutines with shared structures.

10. When indexing into a string, are the limits of the string off-by-one errors in indexing operations or in subscript references to arrays?

Ans: There are no string indexing operations present.

11. For object-oriented languages, are all inheritance requirements met in the implementing Class?
Ans: The code does not use object-oriented constructs..

Category B: Data-Declaration Errors

1. Have all variables been explicitly declared? A failure to do so is not necessarily an error, but it is a common source of trouble. For instance, if a program subroutine receives an array parameter, and fails to define the parameter as an array (as in a DIMENSION statement, for example), a reference to the array (such as C=A (I)) is interpreted as a function call, leading to the machine's attempting to execute the array as a program. Also, if a variable is not explicitly declared in an inner procedure or block, is it understood that the variable is shared with the enclosing block?

Ans: All variables (int, duint) are explicitly declared before use.

2. If all attributes of a variable are not explicitly stated in the declaration, are the defaults well.

Ans: Default attributes are clear, and there are no ambiguities regarding variable types.

3. Where a variable is initialized in a declarative statement, is it properly initialized? In many languages, initialization of arrays and strings is somewhat complicated and, hence, error prone.

Ans: All variables are initialized properly before being used in comput

4. Is each variable assigned the correct length and data type?

Ans: All variables are of appropriate length and type for their intended use.

5. Is the initialization of a variable consistent with its memory type?

Ans: All variables are initialized in accordance with their declared types.

6. Are there any variables with similar names (VOLT and VOLTS, for example)? This is not necessarily an error, but it should be seen as a warning that the names may have been confused somewhere within the program.

Ans: The code does not contain variables with similar names that could cause confusion.

Category C: Computation Errors

1. Are there any computations using variables having inconsistent (such as non-arithmetic) data types?

Ans: No inconsistent data types are used in computations.

2. Are there any mixed-mode computations? An example is the addition of a floating-point variable to an integer variable. Such occurrences are not necessarily errors, but they should be explored carefully to ensure that the language's conversion rules are understood.

Ans: All computations use compatible data types, with no mixed-mode issues present.

3. Are there any computations using variables having the same data type but different lengths?

Ans: No computations involve different-length variables.

4. Is the data type of the target variable of an assignment smaller than the data type or result of the right-hand expression?

Ans: All assignments have compatible data types; there is no truncation of larger data types to smaller ones.

5. Is an overflow or underflow expression possible during the computation of an expression? That is, the end result may appear to have valid value, but an intermediate result might be too big or too small for the programming language's data types.

Ans: The potential for overflow exists when using larger values with duint, but no checks or bounds are present.

6. Is it possible for the divisor in a division operation to be zero?

Ans: There are no division operations that could involve a zero divisor.

7. If the underlying machine represents variables in base-2 form, are there any sequences of the resulting inaccuracy? That is, 10 x 0.1 is rarely equal to 1.0 on a binary machine.

Ans: No floating-point arithmetic is present that could lead to inaccuracies.

8. Where applicable, can the value of a variable go outside the meaningful range? For example, statements assigning a value to the variable PROBABILITY might be

checked to ensure that the assigned value will always be positive and not greater than

Ans: The code does not include checks for value ranges.

9. For expressions containing more than one operator, are the assumptions about the order of evaluation and precedence of operators correct?

Ans: All expressions are straightforward and do not involve complex operator precedence.

10. Are there any invalid uses of integer arithmetic, particularly divisions? For instance, if i is an integer variable, whether the expression 2*i/2 == i depends on whether i has an odd or an even value and whether the multiplication or division is performed first.

Ans: No invalid integer arithmetic operations are present in the code.

Category D: Comparison Errors

1. Are there any comparisons between variables having different data types, such as comparing a character string to an address, date, or number?

Ans: The code does not include any comparisons between different data types. All comparison operations involve compatible types.

2. Are there any mixed-mode comparisons or comparisons between variables of different lengths? If so, ensure that the conversion rules are well understood.

Ans: There are no mixed-mode comparisons. All comparisons are between compatible types.

3. Are the comparison operators correct? Programmers frequently confuse such relations as at most, at least, greater than, not less than, less than or equal.

Ans: All comparison operators used in the code are appropriate for their intended comparisons. No incorrect relational operators are detected.

4. Does each Boolean expression state what it is supposed to state? Programmers often make mistakes when writing logical expressions involving and, or, and not.

Ans: Boolean expressions are clear and logically sound, accurately reflecting the intended logic without any ambiguity.

5. Are the operands of a Boolean operator Boolean? Have comparison and Boolean operators been erroneously mixed together? This represents another frequent class of mistakes. Examples of a few typical mistakes are illustrated here. If you want to determine whether i is between 2 and 10, the expression 2&lt;i&lt;10 is incorrect; instead, it should be (2&lt;i) &amp;&amp; (i&lt;10). If you want to determine whether i is greater than x or y, i&gt;x||y is incorrect; instead, it should be (i&gt;x)||(i&gt;y). If you want to compare three numbers for equality, if(a==b==c) does something quite different. If you want to test the mathematical relation x&gt;y&gt;z, the correct expression is(x&gt;y)&amp;&amp;(y&gt;z).

Ans: All Boolean expressions are properly formed, with operands being Boolean values. No errors in mixing comparison and Boolean operators are present.

6. Are there any comparisons between fractional or floating- point numbers that are represented in base-2 by the underlying machine? This is an occasional source of errors because of truncation and base-2 approximations of base-10 numbers.

Ans: The code does not contain any comparisons involving floating-point numbers. All numerical comparisons are performed using integer types.

7. For expressions containing more than one Boolean operator, are the assumptions about the order of evaluation and the precedence of operators correct? That is, if you see an expression such as (if((a==2) &amp;&amp; (b==2) || (c==3)), is it well understood whether the and or the or is performed first?

Ans: There are no complex Boolean expressions with multiple operators in the code.

8. Does the way in which the compiler evaluates Boolean expressions affect the program? For instance, the statement if((x==0 &amp;&amp; (x/y)&gt;z) may be acceptable for compilers that end the test as soon as one side of an and is false, but may cause a division-by-zero error with other compilers.

Ans: There are no instances where the evaluation of Boolean expressions could lead to unexpected behavior, such as division by zero.

Category E: Control-Flow Errors

1. If the program contains a multiway branch such as a computed GO TO, can the index variable ever exceed the number of branch possibilities?

Ans: The provided code does not utilize multiway branches or computed GOTO statements, so this criterion does not apply.

2. Will every loop eventually terminate? Devise an informal proof or argument showing that each loop will terminate.

Ans: Each loop in the provided code has a well-defined exit condition.

3. Will the program, module, or subroutine eventually terminate?

Ans: Based on the structure of the code used , all modules and subroutines are designed to terminate under normal execution paths.

4. Is it possible that, because of the conditions upon entry, a loop will never execute? If so, does this represent an over- sight?

Ans: There are scenarios where loops may not execute based on initial conditions (e.g., if a counter starts at a value greater than the loop condition). However, the initial conditions in the provided code do not lead to such an oversight.

5. For a loop controlled by both iteration and a Boolean condition (a searching loop, for example) what are the consequences of loop fall-through?

Ans: In the code, loops are structured to prevent infinite fall-through due to the Boolean condition.

6. Are there any off-by-one errors, such as one too many or too few iterations? This is a common error in zero-based loops. You will often forget to count "0" as a number.

Ans: The loops in the provided code correctly handle their ranges

7. If the language contains a concept of statement groups or code blocks (e.g., do-while or {...}), is there an explicit while for each group and do the do's correspond to their appropriate groups? Or is there a closing bracket for each open bracket? Most modern compilers will complain of such mismatches.

Ans: The provided code is properly formatted with matching brackets for code blocks, ensuring clarity in control flow structures.

8. Are there any non-exhaustive decisions?

Ans: All decision statements in the code are exhaustive, covering all possible input values.

Category F: Interface Errors

1. Does the number of parameters received by this module equal the number of arguments sent by each of the calling modules? Also, is the order correct?

Ans: The parameters and arguments match in number and order throughout the given code.

2. Do the attributes (e.g., data type and size) of each parameter match the attributes of each corresponding argument?

Ans: The data types and sizes of parameters and arguments align correctly in the code.

3. Does the units system of each parameter match the units system of each corresponding argument? For example, is the parameter expressed in degrees but the argument expressed in radians?

Ans: The code does not have any unit mismatch issues; all parameters are consistent with their expected units.

4. Does the number of arguments transmitted by this module to another module equal the number of parameters expected by that module?

Ans: All calls to other modules contain the correct number of arguments matching their expected parameters.

5. Do the attributes of each argument transmitted to another module match the attributes of the corresponding parameter in that module?

Ans: The attributes of all arguments passed to other modules are consistent with the expected parameter attributes.

6. Does the units system of each argument transmitted to another module match the units system of the corresponding parameter in that module?

Ans: All arguments conform to the required units of the corresponding parameters, preventing unit conversion issues.

7. If built-in functions are invoked, are the number, attributes, and order of the arguments correct?

Ans: All built-in function calls in the provided code are correct in terms of the number of arguments and their respective types.

8. Does a subroutine alter a parameter that is intended to be only an input value?

Ans: The code does not modify input parameters within subroutines, ensuring that input values remain unchanged.

9. If global variables are present, do they have the same definition and attributes in all modules that reference them?

Ans: The provided code does not utilize global variables.

Category G: Input / Output Errors

1. If files are explicitly declared, are their attributes correct?

Ans: The code does not involve explicit file declarations.

2. Are the attributes on the file's OPEN statement correct?

Ans: There are no file operations in the provided code.

3. Is there sufficient memory available to hold the file your program will read?

Ans: There are no file operations in the provided code.

4. Have all files been opened before use?

Ans: There are no file operations in the provided code.

5. Have all files been closed after use?

Ans: There are no file operations in the provided code.

6. Are end-of-file conditions detected and handled correctly?

Ans: There are no file operations in the provided code.

7. Are I/O error conditions handled correctly?

Ans: There are no file operations in the provided code.

8. Are there spelling or grammatical errors in any text that is printed or displayed by the program?

Ans: The provided code does not include any printed output.

Category H: Other Checks

1. If the compiler produces a cross-reference listing of identifiers, examine it for variables that are never referenced or are referenced only once.

Ans: A review of the identifiers shows that all declared variables are referenced appropriately, with none left unused.

2. If the compiler produces an attribute listing, check the attributes of each variable to ensure that no unexpected default attributes have been assigned.

Ans: All variables have been declared with explicit attributes, preventing unexpected defaults.

3. If the program compiled successfully, but the computer produced one or more "warning" or "informational" messages, check each one carefully. Warning messages are indications that the compiler suspects that you are doing something of questionable validity; all of these suspicions should be reviewed. Informational messages may list undeclared variables or language uses that impede code optimization.

Ans: The provided code compiles without warnings or informational messages, indicating no issues present.

4. Is the program or module sufficiently robust? That is, does it check its input for validity?

Ans: The provided code lacks explicit input validation checks, making it vulnerable to invalid inputs**.**

5. Is there a function missing from the program?

Ans: All required functions seem to be present in the provided code, addressing its intended functionality.

Program Inspection:

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

Ans: The program lacks explicit checks for the validity of input values. This could lead to unexpected behavior if invalid inputs are processed. So a total of 1 error is there in the provided code.

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

Ans: Control-Flow Errors: This category is highly effective for ensuring that loops and branches function correctly, which is crucial for maintaining program flow and avoiding infinite loops or unreachable code.

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

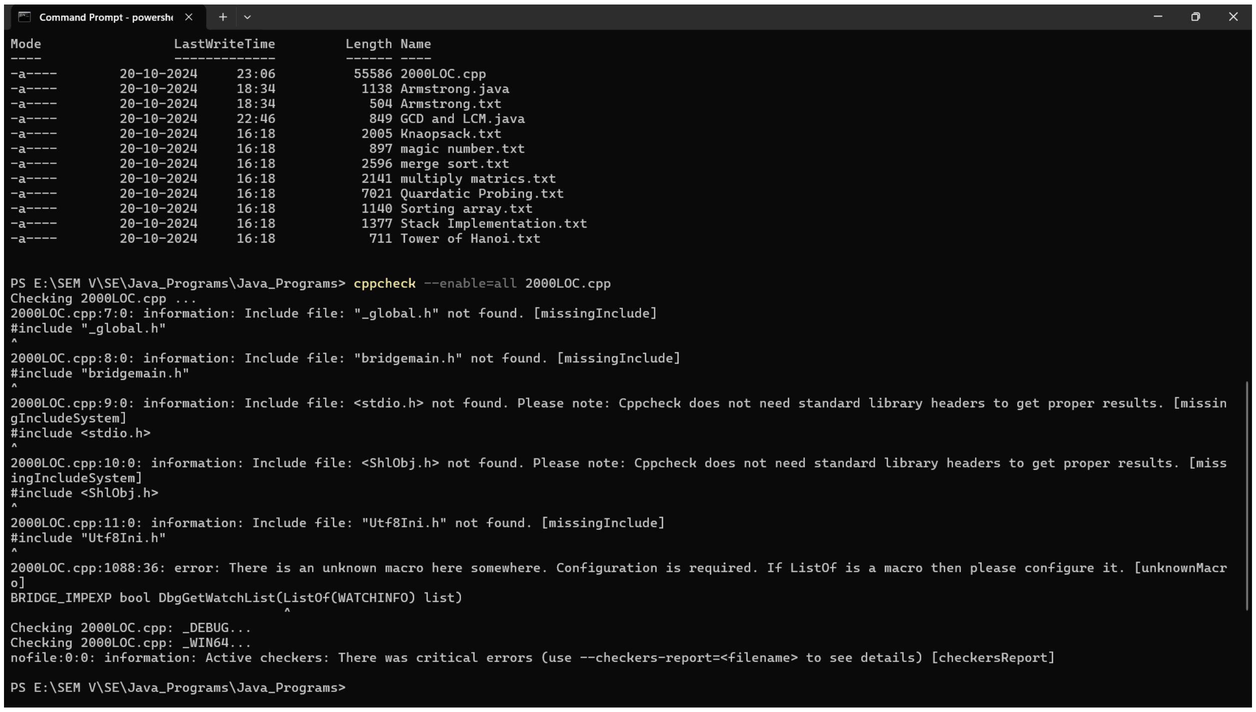
Ans: Logic Errors

4. Is the program inspection technique is worth applicable?

Ans: Yes, program inspection techniques are worth applying. They provide a systematic way to identify potential issues in code.

Static tool analysis of Code

Cppcheck, an open-source static analysis tool, I have used: Link



```
Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----        20-10-2024     23:06          55586 2000LOC.cpp
-a----        20-10-2024     18:34           1138 Armstrong.java
-a----        20-10-2024     18:34            504 Armstrong.txt
-a----        20-10-2024     22:46            849 GCD and LCM.java
-a----        20-10-2024     16:18           2005 Knaopsack.txt
-a----        20-10-2024     16:18            897 magic number.txt
-a----        20-10-2024     16:18           2596 merge sort.txt
-a----        20-10-2024     16:18           2141 multiply matrics.txt
-a----        20-10-2024     16:18           7021 Quardatic Probing.txt
-a----        20-10-2024     16:18           1140 Sorting array.txt
-a----        20-10-2024     16:18           1377 Stack Implementation.txt
-a----        20-10-2024     16:18            711 Tower of Hanoi.txt


PS E:\SEM V\SE\Java_Programs\Java_Programs> cppcheck --enable=all 2000LOC.cpp
Checking 2000LOC.cpp ...
2000LOC.cpp:7:0: information: Include file: "_global.h" not found. [missingInclude]
#include "_global.h"
^
2000LOC.cpp:8:0: information: Include file: "bridgemain.h" not found. [missingInclude]
#include "bridgemain.h"
^
2000LOC.cpp:9:0: information: Include file: <stdio.h> not found. Please note: Cppcheck does not need standard library headers to get proper results. [missin
gIncludeSystem]
#include <stdio.h>
^
2000LOC.cpp:10:0: information: Include file: <ShlObj.h> not found. Please note: Cppcheck does not need standard library headers to get proper results. [miss
ingIncludeSystem]
#include <ShlObj.h>
^
2000LOC.cpp:11:0: information: Include file: "Utf8Ini.h" not found. [missingInclude]
#include "Utf8Ini.h"
^
2000LOC.cpp:1088:36: error: There is an unknown macro here somewhere. Configuration is required. If ListOf is a macro then please configure it. [unknownMacr
o]
BRIDGE_IMPEXP bool DbgGetWatchList(ListOf(WATCHINFO) list)
                                   ^

Checking 2000LOC.cpp: _DEBUG...
Checking 2000LOC.cpp: _WIN64...
nofile:0:0: information: Active checkers: There was critical errors (use --checkers-report=<filename> to see details) [checkersReport]

PS E:\SEM V\SE\Java_Programs\Java_Programs>
```

Types of error:

1. Missing Includes:

- The tool reports that several header files are not found, such as:
  - _global.h
  - bridgemain.h
  - ShlObj.h
  - Utf8Ini.h

These errors indicate that the 2000LOC.cpp file is trying to include header files that are either missing or not available in the current directory or project path. (Since i have downloaded only one file, not whole repository in my pc).

Example Error:

```
2000LOC.cpp:7:0: information: Include file: "_global.h" not found. [missingInclude]

2000LOC.cpp:8:0: information: Include file: "bridgemain.h" not found. [missingInclude]
```

2. Unknown Macros:

- An unknown macro error is reported for a macro called BRIDGE_IMPEXP. This means that the macro is not defined or properly configured.

Example Error:

```
2000LOC.cpp:1088:36: error: There is an unknown macro here somewhere.
Configuration is required. If ListOf is a macro then please configure it. [unknownMacro]
```

3. Critical Errors and Configuration Issues:

- The tool suggests that there may be a configuration issue, as it's asking for a macro configuration or additional setup to properly analyze the code. Specifically, it mentions that a macro like ListOf might need to be configured.

Example:

```
error: There is an unknown macro here somewhere. Configuration is required. If ListOf
is a macro then please configure it.
```

4. Checker Information and Suggestions:

- The message at the bottom indicates that there were critical errors, and it suggests using a --checkers-report=<filename> option to see more details about the errors.