GITHUB CODES & JAVA PROGRAMS

CODE INSPECTION, DEBUGGING & STATIC ANALYSIS TOOL

Name: Dev Darpesh Soni

Id: 202201418

CODE INSPECTION:

I had done the inspection of 1300 Lines of Code in pieces of 200. For each segment, I wrote category wise erroneous lines of code.

→ First 200 Lines Inspection:

Category A: Data Reference Errors

- **Uninitialized Variables:** Variables like `name`, `gender`, `age`, and `phone_no` are declared but might not be initialized before use, leading to potential errors if they are referenced without prior assignment.
- **Array Bounds:** Arrays such as `char specialization[100]` and `char name[100]` lack explicit bounds checking, which could result in buffer overflow issues.

Category B: Data Declaration Errors

- **Implicit Declarations:** Variables like `adhaar` and `identification_id` need to be explicitly declared and initialized with appropriate data types to avoid usage errors.
- **Array Initialization:** Arrays like `char specialization[100]` and `char gender[100]` would benefit from explicit initialization to prevent undefined values.

Category C: Computation Errors

- **Mixed-mode Computations:** `phone_no` and `adhaar`, which are numeric strings, should be treated as strings rather than integers in computations to avoid misinterpretations.

Category E: Control-Flow Errors

- **Infinite Loops with goto:** The use of `goto` statements in Aadhaar and mobile number validation (e.g., `goto C;`) is risky and can lead to infinite loops if conditions aren't properly handled. A structured `while` loop with defined exit conditions would be safer.

Category F: Interface Errors

- **Parameter Mismatch:** Functions like `add_doctor()` and `display_doctor_data()` need to ensure that their parameters align correctly with the caller functions.

Category G: Input/Output Errors

- **File Handling:** Files such as `Doctor_Data.dat` should be opened before use and closed after to avoid file access issues. Additionally, there is no error handling for failed file operations, which could lead to runtime errors.

Control-Flow Issue:

- The `goto` statements used in Aadhaar and mobile number validation could create inefficient control flow, making the code harder to debug. It is advisable to replace them with loops for improved structure.

→ Second 200 Lines Inspection:

Category A: Data Reference Errors

- **File Handling:** Files like `Doctor_Data.dat` and `Patient_Data.dat` are used without proper error handling during file opening (e.g., file not found or access issues). Proper file handling is necessary to prevent system crashes.

Category B: Data Declaration Errors

- **Strings and Arrays:** Variables such as `name[100]`, `specialization[100]`, and `gender[10]` may cause buffer overflows if input exceeds the defined limits.

Category C: Computation Errors

- **Vaccine Stock Calculation:** In the `display_vaccine_stock()` method, vaccine totals are calculated without checks for negative values or integer overflow. These cases should be handled to prevent miscalculations.

Category E: Control-Flow Errors

- **Repetitive Use of goto:** Multiple `goto` statements for Aadhaar and mobile number validation in functions like `add_doctor()` and `add_patient_data()` should be replaced with proper loop constructs (e.g., `while` or `do-while`) to enhance code readability and maintainability.

Category F: Interface Errors

- **Incorrect Data Type Comparisons:** In the `search_doctor_data()` function, string comparisons using `.compare()` may lead to errors if not carefully managed. Ensure consistency in string handling throughout the code.

Category G: Input/Output Errors

- **Missing File Closing:** Files opened in `search_center()` and `display_vaccine_stock()` should always be closed properly after reading to prevent memory leaks or file lock issues.

→ Third 200 Lines Inspection:

Category A: Data Reference Errors

- **File Handling:** In the `add_vaccine_stock()` and `display_vaccine_stock()` functions, error checking is necessary after opening files for vaccine centers (e.g., `center1.txt`, `center2.txt`). Make sure files are correctly opened before proceeding.

Category B: Data Declaration Errors

- **Inconsistent Data Types:** Variables like `adhaar` and `phone_no` are used inconsistently across different functions. They should be treated as numeric strings in all functions to ensure uniform handling.

Category C: Computation Errors

- **Vaccine Stock Summation:** In the `display_vaccine_stock()` function, total stock calculations could produce errors if vaccine counts are negative or uninitialized. Ensure all vaccine stock variables are initialized before use.

Category E: Control-Flow Errors

- **Use of goto:** `goto` statements in functions like `search_doctor_data()` and `add_doctor()` can lead to tangled logic. Using loop structures such as `while` or `for` would improve readability and help avoid infinite loops.

Category F: Interface Errors

- **Parameter Mismatch:** Ensure consistency of parameters, such as in `search_by_aadhar()`, where the `adhaar` parameter should be uniform across all subroutines that use it.

Category G: Input/Output Errors

- **File Access Without Proper Closing:** Files like `Doctor_Data.dat` are opened frequently for reading and writing but are not always properly closed. Ensure all file operations are followed by closing statements to avoid resource leaks.

→ Fourth 200 Lines Inspection:

Category A: Data Reference Errors

- **Uninitialized Variables:** In functions such as `update_patient_data()`, `show_patient_data()`, and `applied_vaccine()`, variables like `maadhaar` and file streams should be explicitly initialized to prevent the use of uninitialized data.

Category B: Data Declaration Errors

- **Array Length Issues:** Character arrays like `sgender[10]` and `adhaar[12]` pose a risk of buffer overflows as input lengths are not validated against the array sizes.

Category C: Computation Errors

- **Vaccine Doses:** In `update_patient_data()`, the `dose++` operation increments the dose directly, which could result in an invalid dose count if no validation is applied.

Category E: Control-Flow Errors

- **Improper Use of goto:** Functions like `search_doctor_data()` and `add_patient_data()` still rely heavily on `goto` for control flow, which complicates the logic. Using loops would improve readability and maintainability.

Category F: Interface Errors

- **Incorrect String Comparisons:** In functions such as `search_by_aadhar()`, string comparisons (e.g., `adhaar.compare(sadhaar)`) might not handle all cases correctly. Consistent and proper string validation logic is required.

Category G: Input/Output Errors

- **File Handling Issues:** Files like `Patient_Data.dat` and `Doctor_Data.dat` are opened in functions like `add_patient_data()` without proper error handling for failed file operations. This can lead to runtime issues.

→ Fifth 200 Lines Inspection:

Category A: Data Reference Errors

- **Uninitialized Variables:** In `update_patient_data()` and `search_doctor_data()`, variables such as `maadhaar` and other fields should be explicitly initialized to avoid using uninitialized data.

Category B: Data Declaration Errors

- **Array Boundaries:** Arrays like `sgender[10]` are vulnerable to buffer overflows if input exceeds the allowed length. Ensure that input validation is in place to prevent this.

Category C: Computation Errors

- **Patient Dose Incrementation:** In `update_patient_data()`, the dose is incremented with `dose++` without range checks or validation, which could result in incorrect dose counts if not handled correctly.

Category E: Control-Flow Errors

- **Repetitive Use of goto:** Functions such as `search_doctor_data()` and `add_doctor()` have multiple `goto` statements that complicate the control flow. Replacing these with structured loops like `while` or `for` would improve readability and maintainability.

Category F: Interface Errors

- **Parameter Mismatch:** Functions like `search_by_aadhar()` perform string comparisons and I/O operations. Ensure that parameters are consistently passed with the expected types in all functions.

Category G: Input/Output Errors

- **File Handling:** Files like `Patient_Data.dat` and `Doctor_Data.dat` are opened but not always properly closed in certain code branches, potentially leading to resource leakage. Exception handling should be added to manage this.

→ Final 300 Lines Inspection:

Category A: Data Reference Errors

- **File Handling:** Files such as `center1.txt`, `center2.txt`, and `center3.txt` are accessed in `add_vaccine_stock()` and `display_vaccine_stock()` without proper error handling. Make sure to include error handling mechanisms in case file access fails.

Category B: Data Declaration Errors

- **Data Initialization:** Variables like `sum_vaccine_c1`, `sum_vaccine_c2`, and `sum_vaccine_c3` in the vaccine stock display functions should be explicitly initialized to avoid unexpected behavior from uninitialized values.

Category C: Computation Errors

- **Vaccine Stock Calculation:** In `add_vaccine_stock()`, ensure that stock values are validated to be positive and valid to prevent errors during subtraction in `display_vaccine_stock()`.

Category E: Control-Flow Errors

- **Excessive Use of goto Statements:** The use of `goto` dominates control flow in functions like `add_doctor()` and `add_patient_data()`. Replace these with loops to improve code clarity and maintainability.

Category G: Input/Output Errors

- **Inconsistent File Closing:** Some branches of the code do not properly close files after operations. Ensure that all opened files are correctly closed to prevent resource leakage.

DEBUGGING:

1. Armstrong Number Program

- Error: Incorrect computation of the remainder.
- Fix: Use breakpoints to check the remainder calculation.

Corrected Code:

```
class Armstrong {
    public static void main(String args[]) {
        int num = Integer.parseInt(args[0]);
        int n = num, check = 0, remainder;
        while (num > 0) {
            remainder = num % 10;
            check += Math.pow(remainder, 3);
            num /= 10;
        }
        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 Program

- Errors:
    1. Incorrect while loop condition in GCD.
    2. Incorrect LCM calculation logic.
- Fix: Breakpoints at the GCD loop and LCM logic.

Corrected Code:

```
import java.util.Scanner;
public class GCD_LCM {
   static int gcd(int x, int y) {
      while (y != 0) {
         int temp = y;
         y = x % y;
         x = temp;
      }
      return x;
   }
   static int lcm(int x, int y) {
      return (x * y) / gcd(x, y);
   }
   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 Program

- Error: Incrementing n inappropriately in the loop.
- Fix: Breakpoint to check loop behavior.

Corrected Code:

```
public class Knapsack {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        int W = Integer.parseInt(args[1]);
        int[] profit = new int[N + 1], weight = new int[N + 1];
        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++) {
                int option1 = opt[n - 1][w];
                int option2 = (weight[n] <= w) ? profit[n] + opt[n - 1][w - weight[n]] :
Integer.MIN_VALUE;
                opt[n][w] = Math.max(option1, option2);
                sol[n][w] = (option2 > option1);
            }
        }
    }
}
```

4. Magic Number Program

- Errors:
    1. Incorrect condition in the inner while loop.
    2. Missing semicolons in expressions.
- Fix: Set breakpoints at the inner while loop and check variable values.

Corrected Code:

```
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);  // Fixed missing semicolon
            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.");
        }
    }
}
```

5. Merge Sort Program

- Errors:
    1. Incorrect array splitting logic.
    2. Incorrect inputs for the merge method.
- Fix: Breakpoints at array split and merge operations.

Corrected Code:

```
import java.util.Scanner;
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];
    System.arraycopy(array, 0, left, 0, size1);
    return left;
}

public static int[] rightHalf(int[] array) {
    int size1 = array.length / 2;
    int size2 = array.length - size1;
    int[] right = new int[size2];
    System.arraycopy(array, size1, right, 0, size2);
    return right;
}

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

```
            result[i] = right[i2];
            i2++;
        }
    }
  }
}
```

6. Multiply Matrices Program

- Errors:
    1. Incorrect loop indices.
    2. Wrong error message.
- Fix: Set breakpoints to check matrix multiplication and correct messages.

Corrected Code:

```
import java.util.Scanner;
class MatrixMultiplication {
   public static void main(String args[]) {
      int m, n, p, q, sum = 0, c, d, k;
      Scanner in = new Scanner(System.in);
      System.out.println("Enter the number of rows and columns of the first matrix");
      m = in.nextInt();
      n = in.nextInt();
      int first[][] = new int[m][n];
      System.out.println("Enter the elements of the 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 the second matrix");
p = in.nextInt();
q = in.nextInt();
if (n != p)
   System.out.println("Matrices with entered orders can't be multiplied.");
else {
   int second[][] = new int[p][q];
   int multiply[][] = new int[m][q];
   System.out.println("Enter the elements of the 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++) {
         for (k = 0; k < p; k++) {
            sum += first[c][k] * second[k][d];
         }
         multiply[c][d] = sum;
         sum = 0;
      }
   }
   System.out.println("Product of entered matrices:");
   for (c = 0; c < m; c++) {
      for (d = 0; d < q; d++)
         System.out.print(multiply[c][d] + "\t");
      System.out.print("\n");
```

```
            }
        }
    }
}
```

7. Quadratic Probing Hash Table Program

- Errors:
    1. Typos in insert, remove, and get methods.
    2. Incorrect logic for rehashing.
- Fix: Set breakpoints and step through logic for insert, remove, and get methods.

Corrected Code:

```
import java.util.Scanner;
class QuadraticProbingHashTable {
    private int currentSize, maxSize;
    private String[] keys, vals;
    public QuadraticProbingHashTable(int capacity) {
        currentSize = 0;
        maxSize = capacity;
```

```
    keys = new String[maxSize];
    vals = new String[maxSize];
}
public void insert(String key, String val) {
    int tmp = hash(key), 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 += (h * h++) % maxSize;
    } 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;
    }

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

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

public class HashTableTest {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        QuadraticProbingHashTable hashTable = new
QuadraticProbingHashTable(scan.nextInt());
        hashTable.insert("key1", "value1");
```

```
        System.out.println("Value: " + hashTable.get("key1"));
    }
}
```

8. Sorting Array Program

- Errors:
    1. Incorrect class name with an extra space.
    2. Incorrect loop condition and extra semicolon.
- Fix: Set breakpoints to check the loop and class name.

Corrected Code:

```
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: ");
        n = s.nextInt();
        int[] a = new int[n];
        System.out.println("Enter all the elements:");
        for (int i = 0; i < n; i++) a[i] = s.nextInt();
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (a[i] > a[j]) {
                    temp = a[i];
                    a[i] = a[j];
                    a[j] = temp;
                }
```

```
            }
        }
        System.out.println("Sorted Array: " + Arrays.toString(a));
    }
}
```

9. Stack Implementation Program

- Errors:
    1. Incorrect top-- instead of top++ in push.
    2. Incorrect loop condition in display.
    3. Missing pop method.
- Fix: Add breakpoints to check push, pop, and display methods.

Corrected Code:

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

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

    public void push(int value) {
        if (top == stack.length - 1) {
            System.out.println("Stack full");
        } else {
            stack[++top] = value;
```

```
        }
    }

    public void pop() {
        if (top == -1) {
            System.out.println("Stack empty");
        } else {
            top--;
        }
    }

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

10. Tower of Hanoi Program

- Error: Incorrect increment/decrement in recursive call.
- Fix: Breakpoints at the recursive calls to verify logic.

Corrected Code:

```
public class TowerOfHanoi {
```

```
    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);
        }
    }
}
```

STATIC ANALYSIS TOOL:

Using cppcheck, I run a static analysis tool for 1300 lines of code used above for program inspection.

Results:

[202201418_Lab7_2.c:1]: (information) Include file: <stdio.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_2.c:2]: (information) Include file: <stdlib.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_2.c:3]: (information) Include file: <sys/types.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_2.c:4]: (information) Include file: <sys/stat.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_2.c:5]: (information) Include file: <unistd.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_2.c:6]: (information) Include file: <dirent.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_2.c:7]: (information) Include file: <fcntl.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_2.c:8]: (information) Include file: <libgen.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_2.c:9]: (information) Include file: <errno.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_2.c:10]: (information) Include file: <string.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_2.c:0]: (information) Limiting analysis of branches. Use --check-level=exhaustive to analyze all branches.

[202201418_Lab7_2.c:116]: (warning) scanf() without field width limits can crash with huge input data.

[202201418_Lab7_2.c:120]: (warning) scanf() without field width limits can crash with huge input data.

[202201418_Lab7_2.c:126]: (warning) scanf() without field width limits can crash with huge input data.

[202201418_Lab7_2.c:127]: (warning) scanf() without field width limits can crash with huge input data.

[202201418_Lab7_2.c:133]: (warning) scanf() without field width limits can crash with huge input data.

[202201418_Lab7_2.c:34]: (style) The scope of the variable 'ch' can be reduced.

[202201418_Lab7_2.c:115]: (style) The scope of the variable 'path2' can be reduced.

[202201418_Lab7_2.c:16]: (style) Parameter 'file' can be declared as pointer to const

[202201418_Lab7_2.c:55]: (style) Variable 'direntp' can be declared as pointer to const

[202201418_Lab7_2.c:40]: (warning) Storing fgetc() return value in char variable and then comparing with EOF.

[202201418_Lab7_3.c:1]: (information) Include file: <stdio.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_3.c:2]: (information) Include file: <stdlib.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_3.c:3]: (information) Include file: <sys/types.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_3.c:4]: (information) Include file: <sys/stat.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_3.c:5]: (information) Include file: <unistd.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_1.c:1]: (information) Include file: <stdio.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_1.c:2]: (information) Include file: <stdlib.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_1.c:3]: (information) Include file: <sys/types.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_1.c:4]: (information) Include file: <sys/stat.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_1.c:5]: (information) Include file: <unistd.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_1.c:6]: (information) Include file: <dirent.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_1.c:7]: (information) Include file: <fcntl.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_1.c:8]: (information) Include file: <libgen.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_1.c:9]: (information) Include file: <errno.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201418_Lab7_1.c:29]: (style) The scope of the variable 'ch' can be reduced.

[202201418_Lab7_1.c:11]: (style) Parameter 'file' can be declared as pointer to const

[202201418_Lab7_1.c:50]: (style) Variable 'direntp' can be declared as pointer to const

[202201418_Lab7_1.c:35]: (warning) Storing fgetc() return value in char variable and then comparing with EOF.

[Covid-Management-System.cpp:4]: (information) Include file: <iostream> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[Covid-Management-System.cpp:5]: (information) Include file: <cstring> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[Covid-Management-System.cpp:6]: (information) Include file: <windows.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[Covid-Management-System.cpp:7]: (information) Include file: <fstream> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[Covid-Management-System.cpp:8]: (information) Include file: <conio.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[Covid-Management-System.cpp:9]: (information) Include file: <iomanip> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[Covid-Management-System.cpp:10]: (information) Include file: <cstdlib> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[Covid-Management-System.cpp:11]: (information) Include file: <string> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[Covid-Management-System.cpp:12]: (information) Include file: <unistd.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[Covid-Management-System.cpp:562]: (portability) fflush() called on input stream 'stdin' may result in undefined behaviour on non-linux systems.

[Covid-Management-System.cpp:565]: (portability) fflush() called on input stream 'stdin' may result in undefined behaviour on non-linux systems.

[Covid-Management-System.cpp:614]: (portability) fflush() called on input stream 'stdin' may result in undefined behaviour on non-linux systems.

[Covid-Management-System.cpp:1121]: (portability) fflush() called on input stream 'stdin' may result in undefined behaviour on non-linux systems.

[Covid-Management-System.cpp:538]: (style) C-style pointer casting

[Covid-Management-System.cpp:619]: (style) C-style pointer casting

[Covid-Management-System.cpp:641]: (style) C-style pointer casting

[Covid-Management-System.cpp:646]: (style) C-style pointer casting

[Covid-Management-System.cpp:749]: (style) C-style pointer casting

[Covid-Management-System.cpp:758]: (style) C-style pointer casting

[Covid-Management-System.cpp:788]: (style) C-style pointer casting

[Covid-Management-System.cpp:797]: (style) C-style pointer casting

[Covid-Management-System.cpp:827]: (style) C-style pointer casting

[Covid-Management-System.cpp:836]: (style) C-style pointer casting

[Covid-Management-System.cpp:866]: (style) C-style pointer casting

[Covid-Management-System.cpp:875]: (style) C-style pointer casting

[Covid-Management-System.cpp:907]: (style) C-style pointer casting

[Covid-Management-System.cpp:973]: (style) C-style pointer casting

[Covid-Management-System.cpp:982]: (style) C-style pointer casting

[Covid-Management-System.cpp:1012]: (style) C-style pointer casting

[Covid-Management-System.cpp:1021]: (style) C-style pointer casting

[Covid-Management-System.cpp:1051]: (style) C-style pointer casting

[Covid-Management-System.cpp:1060]: (style) C-style pointer casting

[Covid-Management-System.cpp:1090]: (style) C-style pointer casting

[Covid-Management-System.cpp:1099]: (style) C-style pointer casting

[Covid-Management-System.cpp:1181]: (style) C-style pointer casting

[Covid-Management-System.cpp:1207]: (style) C-style pointer casting

[Covid-Management-System.cpp:1216]: (style) C-style pointer casting

[Covid-Management-System.cpp:1307]: (style) C-style pointer casting

[Covid-Management-System.cpp:1317]: (style) C-style pointer casting

[Covid-Management-System.cpp:1320]: (style) C-style pointer casting

[Covid-Management-System.cpp:427]: (style) Consecutive return, break, continue, goto or throw statements are unnecessary.

[Covid-Management-System.cpp:443]: (style) Consecutive return, break, continue, goto or throw statements are unnecessary.

[Covid-Management-System.cpp:459]: (style) Consecutive return, break, continue, goto or throw statements are unnecessary.

[Covid-Management-System.cpp:892]: (style) Consecutive return, break, continue, goto or throw statements are unnecessary.

[Covid-Management-System.cpp:306]: (style) The scope of the variable 'usern' can be reduced.

[Covid-Management-System.cpp:48] -> [Covid-Management-System.cpp:277]: (style) Local variable 'user' shadows outer function

[Covid-Management-System.cpp:40] -> [Covid-Management-System.cpp:304]: (style) Local variable 'c' shadows outer variable

[Covid-Management-System.cpp:275]: (performance) Function parameter 'str' should be passed by const reference.

[Covid-Management-System.cpp:277]: (style) Unused variable: user

[Covid-Management-System.cpp:304]: (style) Unused variable: c