Priyank Asodariya **202201470**

CODE INSPECTION:

Inspection of the First 200 Lines of Code:

Category A: Data Reference Issues

- **Uninitialized Variables:**
  Variables such as `name`, `gender`, `age`, `phone_no`, etc., are declared but may not be assigned values in all instances. This poses a risk of using these variables without proper initialization, which could lead to runtime errors.
- **Array Boundaries:**
  Arrays like `char specialization[100];` and `char name[100];` do not include explicit checks for array boundaries, which could result in buffer overflow vulnerabilities if data exceeding the allocated memory is written into them.

Category B: Data Declaration Issues

- **Implicit Declarations:**
  Variables such as adhaar and identification_id should be explicitly declared and initialized with appropriate data types before being used to avoid potential errors.

- **Array Initialization:**
  The string arrays char specialization[100]; and char gender[100]; should be explicitly initialized to prevent errors caused by undefined values.

Category C: Computation Issues

- **Mixed-Mode Operations:**
  The variables phone_no and adhaar are used for numeric input but are handled as strings. As these are numeric strings (such as phone and Aadhaar numbers), they should be consistently managed as strings instead of being treated as integers during computations.

Category E: Control Flow Issues

- **Potential Infinite Loops with goto:**
  The use of goto statements, particularly in the Aadhaar and mobile number validation logic (e.g., goto C;), can lead to infinite loops if not properly controlled. It's recommended to use a structured loop, such as a while loop, with clear exit conditions to ensure better flow control.

Category F: Interface Issues

- **Parameter Mismatch:**
  Functions like add_doctor() and display_doctor_data() should ensure the correct number and type of parameters are passed between caller and callee functions to avoid mismatches that can lead to runtime errors.

Category G: Input/Output Issues

- **File Handling Errors:**
  The system must verify that all files, such as Doctor_Data.dat, are opened successfully before being accessed and closed appropriately after use. A lack of exception handling for failed file operations can lead to crashes or data loss.

Control Flow Consideration: The use of goto statements in Aadhaar and mobile number validation processes can disrupt efficient control flow, making the code harder to maintain. Switching to loop-based structures can improve clarity and prevent potential bugs.

Second 200 Lines:

Category A: Data Reference Issues

- **File Handling:**
  Files like Doctor_Data.dat and Patient_Data.dat are used without proper error handling when attempting to open them (e.g., missing file or access permission errors). It’s important to incorporate file handling mechanisms to manage such exceptions.

Category B: Data Declaration Issues

- **Strings and Arrays:**
  The variables name[100], specialization[100], and gender[10] can pose a risk of buffer overflow if the input data exceeds the allocated space. Ensure bounds are checked to prevent overflow.

Category C: Computation Issues

- **Vaccine Stock Calculation:**
  In the display_vaccine_stock() function, the vaccine stock total is computed without validation for negative values or integer overflows. Proper validation should be in place to avoid inaccurate calculations.

Category E: Control Flow Issues

- **Repeated Use of goto:**
  Functions like add_doctor() and add_patient_data() utilize multiple goto statements for revalidating inputs like Aadhaar and mobile numbers. Replacing these with loops like while or do-while would improve code structure, readability, and maintainability.

Category F: Interface Errors

- Incorrect Data Type Comparisons:
  - In the search_doctor_data() function, the comparisons between strings such as identification_id and sidentification_id use .compare() but could also be prone to errors if not managed carefully. Ensure string handling is consistent and correct across the code.

Category G: Input/Output Errors

- Missing File Closing:
  - The files opened in search_center() and display_vaccine_stock() should always be properly closed a er reading data to avoid potential memory leaks or file lock issues.

Third 200 Lines Inspection:

Category A: Data Reference Errors

- File Handling:
  - In add_vaccine_stock() and display_vaccine_stock(), file operations for vaccine centers (center1.txt, center2.txt, etc.) should include error checking a er file opening. Always ensure that the file opens correctly before proceeding.

Category B: Data Declaration Errors

- Inconsistent Data Types:
  - The adhaar and phone_no variables are expected to be numeric strings but are handled inconsistently across various functions.
- **Category C: Computation Errors**
- **Vaccine Stock Summation:**
  In the display_vaccine_stock() function, errors may occur if vaccine quantities are either negative or not properly initialized. It is essential to ensure that all variables related to vaccine stock are initialized before being used in calculations.
- **Category E: Control-Flow Errors**
- **Use of goto Statements:**
  The use of goto statements in functions such as search_doctor_data() and add_doctor() can lead to tangled and difficult-to-maintain logic. Adopting loop-based structures like while or for would enhance both readability and control flow, reducing the risk of

infinite loops.

- **Category F: Interface Errors**
- **Parameter Mismatch:**
  Functions like search_by_aadhar() require careful attention to parameter consistency. For instance, the aadhar parameter should be passed uniformly across all functions that reference it to avoid mismatches.
- **Category G: Input/Output Errors**
- **Improper File Closure:**
  In several instances, such as in the handling of Doctor_Data.dat, files are opened for reading and writing without being consistently closed afterward. Every file operation must be followed by a closing statement to avoid resource leaks.

- **Fourth 200 Lines:**
- **Category A: Data Reference Errors**
- **Uninitialized Variables:**
  In functions like update_patient_data(), show_patient_data(), and applied_vaccine(), variables such as maadhaar and file streams should be explicitly initialized to avoid referencing uninitialized data, which could lead to unpredictable behavior.
- **Category B: Data Declaration Errors**
- **Array Length Issues:**
  The use of character arrays like sgender[10] and aadhar[12] poses a buffer overflow risk, especially if input lengths exceed the defined limits. Proper validation is necessary to ensure that input data stays within the array boundaries.
- **Category C: Computation Errors**
- **Vaccine Dose Increment:**
  The dose++ operation in the update_patient_data() function increments the dose count directly without any validation. This could result in invalid dose counts if not checked properly.
- **Category E: Control-Flow Errors**
- **Improper Use of goto:**
  Functions like search_doctor_data() and add_patient_data() continue to rely heavily on goto for control flow, which can make the logic difficult to follow and maintain. Replacing goto with structured loops, such as while or for, would improve the code's clarity and maintainability.
- **Category F: Interface Errors**
- **Incorrect String Comparisons:**
  Functions like search_by_aadhar() use direct string comparisons (e.g., aadhar.compare(sadhaar)), which may not handle all cases effectively. Ensure that string comparison logic is applied consistently and validated thoroughly.
- **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. Failure to check for errors when opening files may cause runtime issues, leading to program crashes.

- **Fifth 200 Lines:**
- **Category A: Data Reference Errors**
- **Uninitialized Variables:**
  In update_patient_data() and search_doctor_data(), variables like maadhaar must be initialized explicitly to avoid unintended use of uninitialized data.
- **Category B: Data Declaration Errors**
- **Array Boundaries:**
  Arrays such as sgender[10] may face buffer overflow risks if the input exceeds the defined limits. Input validation must be enforced to prevent this from happening.
- **Category C: Computation Errors**
- **Patient Dose Increment:**
  In update_patient_data(), the direct incrementation of dose using dose++ can result in incorrect dose counts if not properly validated. Ensure appropriate checks are in place before modifying the dose.
- **Category E: Control-Flow Errors**
- **Repeated Use of goto:**
  The goto statements in functions like search_doctor_data() and add_doctor() create complex and difficult-to-manage control flow. Replacing these statements with loop constructs (such as while or for) would greatly improve readability and maintenance.
- **Category F: Interface Errors**
- **Parameter Mismatch:**
  In functions like search_by_aadhar(), string comparisons and input/output handling should ensure correct parameter passing and consistent data types.
- **Category G: Input/Output Errors**
- **File Handling:**
  Files like Patient_Data.dat and Doctor_Data.dat are sometimes not closed correctly in certain code branches. Failure to close files can lead to resource leakage, so proper exception handling should be implemented.
- ---
- **Final 300 Lines:**
- **Category A: Data Reference Errors**
- **File Handling:**
  Files such as center1.txt, center2.txt, and center3.txt are used in the add_vaccine_stock() and display_vaccine_stock() functions without appropriate error handling. It's crucial to introduce error-handling mechanisms to manage potential file access problems.
- **Category B: Data Declaration Errors**
- **Data Initialization:**
  Variables like sum_vaccine_c1, sum_vaccine_c2, and sum_vaccine_c3 used in the

vaccine stock display function should be explicitly initialized to avoid unintended behavior due to uninitialized values.

- **Category C: Computation Errors**
- **Vaccine Stock Calculation:**
  In add_vaccine_stock(), it is important to ensure that the stock values remain positive and valid to prevent errors during the subtraction operations in display_vaccine_stock().
- **Category E: Control-Flow Errors**
- **Excessive Use of goto Statements:**
  Throughout functions such as add_doctor() and add_patient_data(), goto statements are overused. These should be replaced with loop structures (such as while or for) to enhance readability and maintainability.
- **Category G: Input/Output Errors**
- **Inconsistent File Closure:**
  Several code branches fail to properly close files after use. Ensure that all files opened in the code are closed correctly after operations to prevent resource leaks.

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("A   er: " + Arrays.toString(list));
    }
```

```
public static void mergeSort(int[] array) {
    if (array.length > 1) {
        int[] le   = le  Half(array);
        int[] right = rightHalf(array);
        mergeSort(le  );
        mergeSort(right);
        merge(array, le  , right);
    }
}

public static int[] le  Half(int[] array) {
    int size1 = array.length / 2;
    int[] le   = new int[size1];
    System.arraycopy(array, 0, le  , 0, size1);
    return le  ;
}

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[] le  , int[] right) {
        int i1 = 0, i2 = 0;
        for (int i = 0; i < result.length; i++) {
            if (i2 >= right.length || (i1 < le  .length && le  [i1] <= right[i2])) {
                result[i] = le  [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 static analysis tool for 1300 lines of code used above for program inspection.

Results:

[202201407_Lab3_2.c:1]: (information) Include file: <stdio.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_Lab3_2.c:2]: (information) Include file: <stdlib.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_Lab3_2.c:3]: (information) Include file: <sys/types.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_Lab3_2.c:4]: (information) Include file: <sys/stat.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_Lab3_2.c:5]: (information) Include file: <unistd.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_Lab3_2.c:6]: (information) Include file: <dirent.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_Lab3_2.c:7]: (information) Include file: <fcntl.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_Lab3_2.c:8]: (information) Include file: <libgen.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_Lab3_2.c:9]: (information) Include file: <errno.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_Lab3_2.c:10]: (information) Include file: <string.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_Lab3_2.c:0]: (information) Limiting analysis of branches. Use --check-level=exhaustive to analyze all branches.

[202201407_Lab3_2.c:116]: (warning) scanf() without field width limits can crash with huge input data.

[202201407_Lab3_2.c:120]: (warning) scanf() without field width limits can crash with huge input data.

[202201407_Lab3_2.c:126]: (warning) scanf() without field width limits can crash with huge input data.

[202201407_Lab3_2.c:127]: (warning) scanf() without field width limits can crash with huge input data.

[202201407_Lab3_2.c:133]: (warning) scanf() without field width limits can crash with huge input data.

[202201407_Lab3_2.c:34]: (style) The scope of the variable 'ch' can be reduced.

[202201407_Lab3_2.c:115]: (style) The scope of the variable 'path2' can be reduced.

[202201407_Lab3_2.c:16]: (style) Parameter 'file' can be declared as pointer to const

[202201407_Lab3_2.c:55]: (style) Variable 'direntp' can be declared as pointer to const

[202201407_Lab3_2.c:40]: (warning) Storing fgetc() return value in char variable and then comparing with EOF.

[202201407_Lab3_3.c:1]: (information) Include file: <stdio.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_Lab3_3.c:2]: (information) Include file: <stdlib.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_Lab3_3.c:3]: (information) Include file: <sys/types.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_Lab3_3.c:4]: (information) Include file: <sys/stat.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_Lab3_3.c:5]: (information) Include file: <unistd.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_lab3_1.c:1]: (information) Include file: <stdio.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_lab3_1.c:2]: (information) Include file: <stdlib.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_lab3_1.c:3]: (information) Include file: <sys/types.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_lab3_1.c:4]: (information) Include file: <sys/stat.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_lab3_1.c:5]: (information) Include file: <unistd.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_lab3_1.c:6]: (information) Include file: <dirent.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_lab3_1.c:7]: (information) Include file: <fcntl.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_lab3_1.c:8]: (information) Include file: <libgen.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_lab3_1.c:9]: (information) Include file: <errno.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201407_lab3_1.c:29]: (style) The scope of the variable 'ch' can be reduced.

[202201407_lab3_1.c:11]: (style) Parameter 'file' can be declared as pointer to const

[202201407_lab3_1.c:50]: (style) Variable 'direntp' can be declared as pointer to const

[202201407_lab3_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