IT-314

Software Engineering Lab - 7

202201219
Chaudhari Chirag

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:
  - The variables name, gender, age, phone_no, etc., are declared but may not have values initialized at all points of reference, which can lead to errors if they are used before assignment.
- Array Bounds:
  - Arrays like char specialization[100]; and char name[100]; do not have explicit bounds checking, which could lead to buffer overflow errors.

Category B: Data Declaration Errors

- Implicit Declarations:
  - Ensure all variables like adhaar and identification_id are explicitly declared and initialized with the correct data types before usage.
- Array Initialization:

- The string array char specialization[100]; and char gender[100]; could benefit from explicit initialization to avoid issues with undefined values.

Category C: Computation Errors

- Mixed-mode computations:
  - The phone_no and adhaar strings are used for numeric input. Since phone numbers and Aadhaar numbers are numeric strings, ensure they are appropriately handled as strings rather than integers in calculations.

Category E: Control-Flow Errors

- Infinite Loops with goto:
  - The use of goto statements in the Aadhaar and mobile number validation sections (e.g., goto C;) is a dangerous practice and could result in infinite loops if conditions are not properly managed. A while loop with well-defined exit conditions might be a safer alternative.

Category F: Interface Errors

- Parameter Mismatch:
  - Ensure that functions like add_doctor() or display_doctor_data() have a well-matched number of parameters and attributes with the caller functions.

Category G: Input/Output Errors

- File Handling:
  - The system should ensure all files like Doctor_Data.dat are opened before use and closed a er use to avoid file access errors. No exception handling is seen for failed file operations, which can lead to runtime errors.

Control-Flow Issue:

The goto statements used for Aadhaar and mobile number validation can cause inefficient flow control and lead to hard-to-trace bugs. Consider replacing them with loops.

Second 200 Lines Inspection:

Category A: Data Reference Errors

- File Handling:
  - Files such as Doctor_Data.dat and Patient_Data.dat are used frequently without proper exception handling when opening files (e.g., file not found or access issues). Ensure proper file handling mechanisms are in place to prevent crashes.

Category B: Data Declaration Errors

- Strings and Arrays:
  - Variables such as name[100], specialization[100], and gender[10] could potentially lead to buffer overflow issues if inputs exceed defined lengths.

Category C: Computation Errors

- Vaccine Stock Calculation:
  - In the display_vaccine_stock() method, the sum of vaccines across different centers is calculated without checks for negative values or integer overflows. Ensure these cases are handled to avoid potential miscalculations.

Category E: Control-Flow Errors

- Repetitive Use of goto:
  - In functions like add_doctor() and add_patient_data(),

there are multiple goto statements used for revalidation (e.g., Aadhaar or mobile number). These should be replaced with proper loop constructs like while or do-while to improve control flow 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.
    - Make sure that all functions handling these strings treat them as such and do not inadvertently treat them as integers.

Category C: Computation Errors

- Vaccine Stock Summation:
    - In display_vaccine_stock(), the total stock calculation can result in errors if vaccine numbers are negative or not properly initialized. Ensure that all vaccine stock variables are initialized before use.

Category E: Control-Flow Errors

- Use of goto:
    - goto statements appear again in functions like search_doctor_data() and add_doctor(), which could lead to tangled logic. Using loop-based structures such as while or for can improve readability and avoid potential issues with infinite
      loops.

Category F: Interface Errors

- Parameter Mismatch:
    - Check the consistency of parameters, such as in search_by_aadhar(), where the function expects the adhaar parameter to be consistent across all subroutines that reference it.

Category G: Input/Output Errors

- File Access Without Proper Closing:
    - Files like Doctor_Data.dat are frequently opened for reading and writing, but without proper closing in certain branches of the code. Ensure every file

operation is followed by a closing statement to prevent resource leakage.

- Fourth 200 Lines Inspection :

Category A: Data Reference Errors

- Uninitialized Variables:
  - In functions like update_patient_data(), show_patient_data(), and applied_vaccine(), variables like maadhaar and file streams could benefit from explicit initialization to avoid referencing unset or uninitialized data.

Category B: Data Declaration Errors

- Array Length Issues:
  - The usage of character arrays like sgender[10] and adhaar[12] poses a risk of buffer overflows, especially since input length is not validated against the array size.

Category C: Computation Errors

- Vaccine Doses:
  - In update_patient_data(), the dose++ operation increments the dose directly, which could potentially result in an invalid dose count if not checked properly.

Category E: Control-Flow Errors

- Improper Use of goto:
  - Functions like search_doctor_data() and add_patient_data() still heavily rely on goto for control flow, making the logic difficult to follow and maintain. Loops should be used instead to ensure better readability and control.

Category F: Interface Errors

- Incorrect String Comparisons:

- Functions like search_by_aadhar() compare string variables directly (e.g., adhaar.compare(sadhaar)), which may not handle all cases properly. Ensure proper validation and matching logic is used consistently.

Category G: Input/Output Errors

- File Handling Issues:
  - The files (Patient_Data.dat, Doctor_Data.dat) are opened in various functions like add_patient_data() without proper error checking a er opening. Failure to handle file opening errors may result in runtime issues.

Fifih 200 Lines Inspection:

Category A: Data Reference Errors

- Uninitialized Variables:
  - In update_patient_data() and search_doctor_data(), variables like maadhaar and other fields should be explicitly initialized to avoid using uninitialized values.

Category B: Data Declaration Errors

- Array Boundaries:
  - Arrays like sgender[10] are prone to buffer overflows if input exceeds the defined limit. Ensure string length validation to avoid this issue.

Category C: Computation Errors

- Patient Dose Incrementation:
  - In update_patient_data(), the dose value is incremented directly with dose++ without any range checks or validation. This can lead to incorrect dose counts if not handled properly.

Category E: Control-Flow Errors

- Repetitive Use of goto:
- In both search_doctor_data() and add_doctor(), there are several goto statements that make the control flow complex and difficult to maintain. Replacing them with structured loops (while or for) would be a better approach for readability and maintainability.

Category F: Interface Errors

- Parameter Mismatch:
  - Functions like search_by_aadhar() perform string comparisons and handle input/output operations. Ensure parameters are passed correctly and with expected types in all functions.

Category G: Input/Output Errors

- File Handling:
  - Files like Patient_Data.dat and Doctor_Data.dat are opened but sometimes not closed properly in case of certain branches of the code. This can lead to resource leakage. Proper exception handling should be added to prevent this.

Final 300 Lines Inspection:

Category A: Data Reference Errors

- File Handling:
  - Files like center1.txt, center2.txt, and center3.txt are used across the add_vaccine_stock() and display_vaccine_stock() functions without proper error handling. Ensure error handling mechanisms are added in case of file access issues.

Category B: Data Declaration Errors

- Data Initialization:
    - Variables such as sum_vaccine_c1, sum_vaccine_c2, and sum_vaccine_c3 used in vaccine stock display should be
    - initialized explicitly to avoid unintended behavior if le uninitialized.

Category C: Computation Errors

- Vaccine Stock Calculation:
    - In functions like add_vaccine_stock(), ensure that stock values are always positive and valid to avoid potential errors during subtraction in display_vaccine_stock().

Category E: Control-Flow Errors

- Excessive Use of goto Statements:
    - Throughout functions like add_doctor() and add_patient_data(), goto statements dominate the control flow. These should be replaced with loop constructs (while, for) for better readability and maintainability.

Category G: Input/Output Errors

- Inconsistent File Closing:
    - Several branches of file-handling code don't always close files correctly. Ensure every opened file is properly closed a er 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(ke
            y)) { 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(k
            ey)) 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:

[202201219_Lab_3.c:1]: (information) Include file: <stdio.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201219_Lab_3.c:2]: (information) Include file: <stdlib.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201219_Lab_3.c:3]: (information) Include file: <sys/types.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201219_Lab_3.c:4]: (information) Include file: <sys/stat.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201219_Lab_3.c:5]: (information) Include file: <unistd.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201219_Lab_3.c:6]: (information) Include file: <dirent.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201219_Lab_3.c:7]: (information) Include file: <fcntl.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201219_Lab_3.c:8]: (information) Include file: <libgen.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201219_Lab_3.c:9]: (information) Include file: <errno.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201219_Lab_3.c:10]: (information) Include file: <string.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201219_Lab_3.c:0]: (information) Limiting analysis of branches. Use --check-level=exhaustive to analyze all branches.

[202201219_Lab_3.c:116]: (warning) scanf() without field width limits can

crash with huge input data.

[202201219_Lab_3.c:120]: (warning) scanf() without field width limits can crash with huge input data.

[202201219_Lab_3.c:126]: (warning) scanf() without field width limits can crash with huge input data.

[202201219_Lab_3.c:127]: (warning) scanf() without field width limits can crash with huge input data.

[202201219_Lab_3.c:133]: (warning) scanf() without field width limits can crash with huge input data.

[202201219_Lab_3.c:34]: (style) The scope of the variable 'ch' can be

reduced. [202201219_Lab_3.c:115]: (style) The scope of the variable 'path2'

can be reduced. [202201219_Lab_3.c:16]: (style) Parameter 'file' can be

declared as pointer to const [202201219_Lab_3.c:55]: (style) Variable

'direntp' can be declared as pointer to const

[202201219_Lab_3.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