IT314
LAB: 07

Name: Maunil Modi
S.Id: 202201490.

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 such as name, gender, age, and ph_no are declared but might not be initialized before every reference, which could cause errors if they are accessed before being assigned a value.
- Array Bounds:
  Arrays like 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:
  Make sure that variables like `adhaar` and `identification_id` are explicitly declared and properly initialized with the correct data types before they are used.
- Array Initialization:
  String arrays such as `char specialization[100];` and `char gender[100];` would benefit from explicit initialization to prevent potential issues with uninitialized values.

Category C: Computation Errors

- Mixed-mode computations:
  The ph_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:
Ensure that all files, such as `Doctor_Data.dat`, are opened before use and properly closed afterward to prevent file access errors. Additionally, there is no exception handling for failed file operations, which could result in runtime errors.

Control-Flow Issue:
The use of `goto` statements for Aadhaar and mobile number validation can create inefficient control flow and may lead to bugs that are hard to trace. It is recommended to replace them with loops.

<u>Second 200 Lines Inspection:</u>

Category A: Data Reference Errors
● File Handling:
Files like `Doctor_Data.dat` and `Patient_Data.dat` are frequently accessed without proper exception handling (e.g., for cases where files may not be found or there are

access issues). Ensure robust file handling mechanisms are in place to avoid crashes.

Category B: Data Declaration Errors

● Strings and Arrays:
Variables such as `name[100]`, `specialization[100]`, and `gender[10]` may cause buffer overflow issues if the input exceeds the defined lengths.

Category C: Computation Errors

● Vaccine Stock Calculation:
In the `display_vaccine_stock()` function, the total number of vaccines across centers is calculated without validating for negative values or guarding against integer overflows. These cases should be handled to prevent incorrect calculations.

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 after reading data to avoid potential memory leaks or file lock issues.

<u>Third 200 Lines Inspection:</u>

Category A: Data Reference Errors

- File Handling:
  In `add_vaccine_stock()` and `display_vaccine_stock()`, file operations involving vaccine centers (e.g., `center1.txt`, `center2.txt`) should include error handling immediately after the file is opened. Always verify that the file was successfully opened before proceeding.

Category B: Data Declaration Errors

- Inconsistent Data Types:
  The variables `adhaar` and `phone_no` are meant to be numeric strings but are inconsistently treated in different functions. Ensure that all functions handle these values as strings rather than as integers.

Category C: Computation Errors

- Vaccine Stock Summation:
  In the `display_vaccine_stock()` function, errors may occur if vaccine stock numbers are negative or uninitialized. Make sure all vaccine stock variables are initialized before they are used.

Category E: Control-Flow Errors

- Use of goto:
  `goto` statements reappear in functions such as `search_doctor_data()` and `add_doctor()`, leading to complex control flow. Replacing these with loops, such as `while` or `for`, would improve readability and reduce the risk of infinite loops.

Category F: Interface Errors

- Parameter Mismatch:

Check the parameter consistency in functions like `search_by_aadhar()`, ensuring that the `adhaar` parameter is handled consistently across all related subroutines.

Category G: Input/Output Errors

- File Access Without Proper Closing:
  Files like `Doctor_Data.dat` are opened for reading and writing but are sometimes not properly closed in certain code paths. Ensure that all file operations are followed by a closing statement to avoid resource leakage.

<u>Fourth 200 Lines Inspection:-</u>

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 avoid accessing uninitialized data.

Category B: Data Declaration Errors

- Array Length Issues:
  Character arrays like `sgender[10]` and `adhaar[12]` could face buffer overflow issues if the input exceeds

the defined array length. Ensure input is validated against the array size.

Category C: Computation Errors

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

Category E: Control-Flow Errors

- Improper Use of goto:
  Functions like `search_doctor_data()` and `add_patient_data()` rely heavily on `goto` statements, making the control flow harder to follow and maintain. Replacing these with loop structures will enhance both readability and control**.**

Category F: Interface Errors

- Incorrect String Comparisons:
  In functions like `search_by_aadhar()`, string comparisons (e.g., `adhaar.compare(sadhaar)`) may not be handled consistently. Ensure proper string validation and comparison logic across all cases.

Category G: Input/Output Errors

- File Handling Issues:

In functions like `add_patient_data()`, files like `Patient_Data.dat` and `Doctor_Data.dat` are opened without proper error checking. Failing to handle file access errors could lead to runtime issues.

Fifth 200 Lines Inspection:-

Category A: Data Reference Errors

- Uninitialized Variables:
  In functions such as `update_patient_data()` and `search_doctor_data()`, variables like `maadhaar` and other fields should be explicitly initialized to prevent errors due to uninitialized values.

Category B: Data Declaration Errors

- Array Boundaries:
  Arrays such as `sgender[10]` are prone to buffer overflow if the input exceeds the array's size limit. Input length validation should be added to avoid such issues.

Category C: Computation Errors

- Patient Dose Incrementation:
  In `update_patient_data()`, the dose value is incremented with `dose++` without range checks or validation, which may lead to incorrect dose counts if not handled carefully.

Category E: Control-Flow Errors

- Repetitive Use of goto:
  Functions like `search_doctor_data()` and `add_doctor()` rely on multiple `goto` statements, which make the control flow more complex. Using structured loops like `while` or `for` would enhance code readability and maintainability.

Category F: Interface Errors

- Parameter Mismatch:
  Functions like `search_by_aadhar()` handle string comparisons and I/O operations. Ensure that the parameters are passed correctly and match the expected types in all relevant functions.

Category G: Input/Output Errors

- File Handling:
  Files such as `Patient_Data.dat` and `Doctor_Data.dat` are not always properly closed after certain branches of code execution, leading to potential resource leaks. Add proper exception handling to close files and avoid leaks.

<u>Final 300 Lines Inspection:-</u>

Category A: Data Reference Errors

- File Handling:

Files like `center1.txt`, `center2.txt`, and `center3.txt` are used in the functions `add_vaccine_stock()` and `display_vaccine_stock()` without proper error checking. Add error handling to ensure file access issues are managed.

Category B: Data Declaration Errors

- Data Initialization:
  Variables like `sum_vaccine_c1`, `sum_vaccine_c2`, and `sum_vaccine_c3` used in vaccine stock displays should be explicitly initialized to prevent unintended behavior from uninitialized values.

Category C: Computation Errors

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

Category E: Control-Flow Errors

- Excessive Use of goto Statements:
  In functions like `add_doctor()` and `add_patient_data()`, `goto` statements are heavily used, making the control flow more complex. It is advisable to replace these with loop constructs such as `while` or `for` to improve readability and maintainability**.**

Category G: Input/Output Errors

- Inconsistent File Closing:
  Several branches of code that involve file operations do not always close files properly. Ensure that all files are closed after use 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 static analysis tool for 1300 lines of code used above for program inspection.

Results:

[202201490_Lab3_2.c:1]: (information) Include file: <stdio.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201490_Lab3_2.c:2]: (information) Include file: <stdlib.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201490_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.
[202201490_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.
[202201490_Lab3_2.c:5]: (information) Include file: <unistd.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201490_Lab3_2.c:6]: (information) Include file: <dirent.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201490_Lab3_2.c:7]: (information) Include file: <fcntl.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[202201490_Lab3_2.c:8]: (information) Include file: <libgen.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201490_Lab3_2.c:9]: (information) Include file: <errno.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201490_Lab3_2.c:10]: (information) Include file: <string.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201490_Lab3_2.c:0]: (information) Limiting analysis of branches. Use--check-level=exhaustive to analyze all branches.
[202201490_Lab3_2.c:116]: (warning) scanf() without field width limits can crash with huge input data.
[202201490_Lab3_2.c:120]: (warning) scanf() without field width limits can crash with huge input data.
[202201490_Lab3_2.c:126]: (warning) scanf() without field width limits can crash with huge input data.
[202201490_Lab3_2.c:127]: (warning) scanf() without field width limits can crash with huge input data.
[202201490_Lab3_2.c:133]: (warning) scanf() without field width limits can crash with huge input data.
[202201490_Lab3_2.c:34]: (style) The scope of the variable 'ch' can be reduced.
[202201490_Lab3_2.c:115]: (style) The scope of the variable 'path2' can be reduced.
[202201490_Lab3_2.c:16]: (style) Parameter 'file' can be declared as pointer to const
[202201490_Lab3_2.c:55]: (style) Variable 'direntp' can be declared as pointer to const

[202201490_Lab3_2.c:40]: (warning) Storing fgetc() return value in char variable and then comparing with EOF.
[202201490_Lab3_3.c:1]: (information) Include file: <stdio.h> not found. Please note:
Cppcheck does not need standard library headers to get proper results.
[202201490_Lab3_3.c:2]: (information) Include file: <stdlib.h> not found. Please note:
Cppcheck does not need standard library headers to get proper results.
[202201490_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.
[202201490_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.
[202201490_Lab3_3.c:5]: (information) Include file: <unistd.h> not found. Please note:
Cppcheck does not need standard library headers to get proper results.
[202201490_lab3_1.c:1]: (information) Include file: <stdio.h> not found. Please note:
Cppcheck does not need standard library headers to get proper results.
[202201490_lab3_1.c:2]: (information) Include file: <stdlib.h> not found. Please note:
Cppcheck does not need standard library headers to get proper results.

[202201490_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.
[202201490_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.
[202201490_lab3_1.c:5]: (information) Include file: <unistd.h> not found. Please note:
Cppcheck does not need standard library headers to get proper results.
[202201490_lab3_1.c:6]: (information) Include file: <dirent.h> not found. Please note:
Cppcheck does not need standard library headers to get proper results.
[202201490_lab3_1.c:7]: (information) Include file: <fcntl.h> not found. Please note:
Cppcheck does not need standard library headers to get proper results.
[202201490_lab3_1.c:8]: (information) Include file: <libgen.h> not found. Please note:
Cppcheck does not need standard library headers to get proper results.
[202201490_lab3_1.c:9]: (information) Include file: <errno.h> not found. Please note:
Cppcheck does not need standard library headers to get proper results.
[202201490_lab3_1.c:29]: (style) The scope of the variable 'ch' can be reduced.

[202201490_lab3_1.c:11]: (style) Parameter 'file' can be declared as pointer to const
[202201490_lab3_1.c:50]: (style) Variable 'direntp' can be declared as pointer to const
[202201490_lab3_1.c:35]: (warning) Storing fgetc() return value in char variable and then
comparing with EOF.
[Healthguard.cpp:4]: (information) Include file: <iostream> not found.
Please note: Cppcheck does not need standard library headers to get proper results.
[Healthguard.cpp:5]: (information) Include file: <cstring> not found.
Please note: Cppcheck does not need standard library headers to get proper results.
[Healthguard.cpp:6]: (information) Include file: <windows.h> not found.
Please note: Cppcheck does not need standard library headers to get proper results.
[Healthguard.cpp:7]: (information) Include file: <fstream> not found.
Please note: Cppcheck does not need standard library headers to get proper results.
[Healthguard.cpp:8]: (information) Include file: <conio.h> not found.
Please note: Cppcheck does not need standard library headers to get proper results.
[Healthguard.cpp:9]: (information) Include file: <iomanip> not found.
Please note: Cppcheck does not need standard library headers to get proper results.

[Healthguard.cpp:10]: (information) Include file: <cstdlib> not found.
Please note: Cppcheck does not need standard library headers to get proper results.
[Healthguard.cpp:11]: (information) Include file: <string> not found.
Please note: Cppcheck does not need standard library headers to get proper results.
[Healthguard.cpp:12]: (information) Include file: <unistd.h> not found.
Please note: Cppcheck does not need standard library headers to get proper results.
[Healthguard.cpp:562]: (portability) fflush() called on input stream 'stdin'
may result in undefined behaviour on non-linux systems.
[Healthguard.cpp:565]: (portability) fflush() called on input stream 'stdin'
may result in undefined behaviour on non-linux systems.
[Healthguard.cpp:614]: (portability) fflush() called on input stream 'stdin'
may result in undefined behaviour on non-linux systems.
[Healthguard.cpp:1121]: (portability) fflush() called on input stream
'stdin' may result in undefined behaviour on non-linux systems.
[Healthguard.cpp:538]: (style) C-style pointer casting
[Healthguard.cpp:619]: (style) C-style pointer casting
[Healthguard.cpp:641]: (style) C-style pointer casting
[Healthguard.cpp:646]: (style) C-style pointer casting
[Healthguard.cpp:749]: (style) C-style pointer casting
[Healthguard.cpp:758]: (style) C-style pointer casting

[Healthguard.cpp:788]: (style) C-style pointer casting
[Healthguard.cpp:797]: (style) C-style pointer casting
[Healthguard.cpp:827]: (style) C-style pointer casting
[Healthguard.cpp:836]: (style) C-style pointer casting
[Healthguard.cpp:866]: (style) C-style pointer casting
[Healthguard.cpp:875]: (style) C-style pointer casting
[Healthguard.cpp:907]: (style) C-style pointer casting
[Healthguard.cpp:973]: (style) C-style pointer casting
[Healthguard.cpp:982]: (style) C-style pointer casting
[Healthguard.cpp:1012]: (style) C-style pointer casting
[Healthguard.cpp:1021]: (style) C-style pointer casting
[Healthguard.cpp:1051]: (style) C-style pointer casting
[Healthguard.cpp:1060]: (style) C-style pointer casting
[Healthguard.cpp:1090]: (style) C-style pointer casting
[Healthguard.cpp:1099]: (style) C-style pointer casting
[Healthguard.cpp:1181]: (style) C-style pointer casting
[Healthguard.cpp:1207]: (style) C-style pointer casting
[Healthguard.cpp:1216]: (style) C-style pointer casting
[Healthguard.cpp:1307]: (style) C-style pointer casting
[Healthguard.cpp:1317]: (style) C-style pointer casting
[Healthguard.cpp:1320]: (style) C-style pointer casting
[Healthguard.cpp:427]: (style) Consecutive return, break, continue, goto or throw statements are unnecessary.
[Healthguard.cpp:443]: (style) Consecutive return, break, continue, goto or throw statements are unnecessary.
[Healthguard.cpp:459]: (style) Consecutive return, break, continue, goto or throw statements are unnecessary.
[Healthguard.cpp:892]: (style) Consecutive return, break, continue, goto or throw statements are unnecessary.
[Healthguard.cpp:306]: (style) The scope of the variable 'usern' can be reduced.

[Healthguard.cpp:48] -> [Healthguard.cpp:277]: (style) Local variable 'user' shadows outer function
[Healthguard.cpp:40] -> [Healthguard.cpp:304]: (style) Local variable 'c' shadows outer variable
[Healthguard.cpp:275]: (performance) Function parameter 'str' should be passed by const reference.
[Healthguard.cpp:277]: (style) Unused variable: user
[Healthguard.cpp:304]: (style) Unused variable: c