



IT314:Lab 7

USING GITHUB CODES & JAVA PROGRAMS
CODE INSPECTION, DEBUGGING & STATIC
ANALYSIS TOOL

Mayank Parmar
202201464

CODE INSPECTION

I conducted the inspection of 1400 lines of code in sets of 200. I typed erroneous lines of code in each section based on categorization.

First 200 Lines Inspection:

Category A: Data Reference Errors

Uninitialized Variables: Variables like name, gender, age, and phone_no are declared but might not always be initialized when referenced, potentially causing issues if used before being assigned values.

Array Bounds: Arrays such as char specialization[100] and char name[100] lack explicit boundary checks, risking buffer overflow errors.

Category B: Data Declaration Errors

Implicit Declarations: Variables like adhaar and identification_id should be clearly declared with the correct data types before use.

Array Initialization: Arrays like char specialization[100] and char gender[100] would benefit from explicit initialization to avoid the use of undefined values.

Category C: Computation Errors

Mixed-mode Computations: Numeric strings like phone_no and adhaar are treated inconsistently. These should be handled as strings rather than integers to avoid computational errors.

Category E: Control-Flow Errors

Risky goto Usage: The goto statements in Aadhaar and mobile number validation could result in infinite loops if exit conditions aren't well defined. Replacing them with well-structured loops would be a safer approach.

Category F: Interface Errors

Parameter Mismatch: Functions like `add_doctor()` and `display_doctor_data()` should ensure proper matching of parameters with the caller functions.

Category G: Input/Output Errors

File Handling: Files like `Doctor_Data.dat` are opened without ensuring proper error handling or closure, which could lead to file access issues and runtime errors.

Second 200 Lines Inspection

Category A: Data Reference Errors

File Handling: Files such as `Doctor_Data.dat` and `Patient_Data.dat` are accessed without proper exception handling for file opening errors, like missing files or access issues.

Category B: Data Declaration Errors

Buffer Overflow Risks: Arrays like `name[100]`, `specialization[100]`, and `gender[10]` could overflow if inputs exceed their maximum size limits.

Category C: Computation Errors

Vaccine Stock Calculation: In the `display_vaccine_stock()` function, the total stock calculation lacks checks for negative values or overflows, which could result in miscalculations.

Category E: Control-Flow Errors

Excessive goto Usage: Functions like `add_doctor()` and `add_patient_data()` use goto statements for validation. These should be replaced with loops to enhance readability and control flow.

Category F: Interface Errors

String Comparison Issues: In the `search_doctor_data()` function, string comparisons could introduce errors if not managed correctly. Ensure consistent and proper string handling.

Category G: Input/Output Errors

File Closure: Files opened in functions like `search_center()` and `display_vaccine_stock()` are not always closed, which could lead to memory leaks or file lock issues.

Third 200 Lines Inspection

Category A: Data Reference Errors

File Error Handling: In functions like `add_vaccine_stock()` and `display_vaccine_stock()`, file operations for various centers (`center1.txt`, etc.) lack error handling after file opening. Always check that the file is successfully opened before proceeding.

Category B: Data Declaration Errors

Inconsistent Data Handling: Variables like `adhaar` and `phone_no` are inconsistently treated as numeric strings or integers across different functions, which can lead to errors.

Category C: Computation Errors

Vaccine Stock Calculation: In `display_vaccine_stock()`, stock calculations can fail if values are negative or uninitialized. Ensure all stock variables are initialized before use.

Category E: Control-Flow Errors

`goto` Usage: `goto` statements in functions like `search_doctor_data()` and `add_doctor()` complicate logic. Structured loops should replace these for better readability.

Category F: Interface Errors

Parameter Consistency: Ensure parameters like `adhaar` are passed consistently in functions like `search_by_aadhar()` across all subroutines.

Category G: Input/Output Errors

File Closure: Files like `Doctor_Data.dat` are not always closed in all branches, risking resource leaks.

Fourth 200 Lines Inspection

Category A: Data Reference Errors

Uninitialized Variables: Variables like `maadhaar` and file streams in functions like `update_patient_data()` and `applied_vaccine()` should be explicitly initialized to avoid issues with unset data.

Category B: Data Declaration Errors

Buffer Overflow Risk: Character arrays like `sgender[10]` and `adhaar[12]` risk overflow if input length isn't validated.

Category C: Computation Errors

Dose Incrementation: The `dose++` operation in `update_patient_data()` could lead to invalid dose counts if not properly validated.

Category E: Control-Flow Errors

Overuse of `goto`: Heavy reliance on `goto` in functions like `search_doctor_data()` and `add_patient_data()` makes the control flow complex. Replacing `goto` with structured loops improves maintainability.

Category F: Interface Errors

String Comparison Issues: String comparisons in functions like `search_by_aadhar()` may not handle all cases properly. Ensure consistent validation logic for string operations.

Category G: Input/Output Errors

File Handling: Files like `Patient_Data.dat` and `Doctor_Data.dat` should include error checks when opened to avoid runtime failures.

Fifth 200 Lines Inspection

Category A: Data Reference Errors

Uninitialized Variables: In functions like `update_patient_data()` and `search_doctor_data()`, variables such as `maadhaar` should be properly initialized to avoid uninitialized value usage.

Category B: Data Declaration Errors

Buffer Overflow Risk: Arrays like `sgender[10]` are susceptible to overflow if input validation is not performed.

Category C: Computation Errors

Dose Increment Issues: The direct increment (`dose++`) in `update_patient_data()` needs validation to avoid erroneous dose counts.

Category E: Control-Flow Errors

Complex goto Statements: The use of goto in functions like search_doctor_data() and add_doctor() makes the code difficult to follow. Consider using loops for improved control flow.

Category F: Interface Errors

Parameter Handling: Functions like search_by_aadhar() should ensure correct and consistent parameter types are passed between functions.

Category G: Input/Output Errors

File Closure Issues: Files such as Patient_Data.dat and Doctor_Data.dat are not always properly closed, leading to resource management problems.

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 left 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 after operations to prevent resource leaks.

Code Debugging

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[] 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 1200+ lines of code used above for program inspection.

Results:

[202201464_lab7_2.c:1]: (information) Include file: <stdio.h> not found.

Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_2.c:2]: (information) Include file: <stdlib.h> not found.

Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_2.c:3]: (information) Include file: <sys/types.h> not found.

Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_2.c:4]: (information) Include file: <sys/stat.h> not found.

Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_2.c:5]: (information) Include file: <unistd.h> not found.

Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_2.c:6]: (information) Include file: <dirent.h> not found.

Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_2.c:7]: (information) Include file: <fcntl.h> not found.

Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_2.c:8]: (information) Include file: <libgen.h> not found.

Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_2.c:9]: (information) Include file: <errno.h> not found.

Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_2.c:10]: (information) Include file: <string.h> not found.
Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_2.c:0]: (information) Limiting analysis of branches. Use --check-level=exhaustive to analyze all branches.

[202201464_lab7_2.c:116]: (warning) scanf() without field width limits can crash with huge input data.

[202201464_lab7_2.c:120]: (warning) scanf() without field width limits can crash with huge input data.

[202201464_lab7_2.c:126]: (warning) scanf() without field width limits can crash with huge input data.

[202201464_lab7_2.c:127]: (warning) scanf() without field width limits can crash with huge input data.

[202201464_lab7_2.c:133]: (warning) scanf() without field width limits can crash with huge input data.

[202201464_lab7_2.c:34]: (style) The scope of the variable 'ch' can be reduced. [202201464_lab7_2.c:115]: (style) The scope of the variable 'path2' can be reduced. [202201464_lab7_2.c:16]: (style) Parameter 'file' can be declared as pointer to const [202201464_lab7_2.c:55]: (style) Variable 'direntp' can be declared as pointer to const

[202201464_lab7_2.c:40]: (warning) Storing fgetc() return value in char variable and then comparing with EOF.

[202201464_lab7_3.c:1]: (information) Include file: <stdio.h> not found.
Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_3.c:2]: (information) Include file: <stdlib.h> not found.
Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_3.c:3]: (information) Include file: <sys/types.h> not found.
Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_3.c:4]: (information) Include file: <sys/stat.h> not found.
Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_3.c:5]: (information) Include file: <unistd.h> not found.
Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_1.c:1]: (information) Include file: <stdio.h> not found.
Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_1.c:2]: (information) Include file: <stdlib.h> not found.
Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_1.c:3]: (information) Include file: <sys/types.h> not found.
Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_1.c:4]: (information) Include file: <sys/stat.h> not found.
Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_1.c:5]: (information) Include file: <unistd.h> not found.
Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_1.c:6]: (information) Include file: <dirent.h> not found.
Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_1.c:7]: (information) Include file: <fcntl.h> not found.
Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_1.c:8]: (information) Include file: <libgen.h> not found.
Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_1.c:9]: (information) Include file: <errno.h> not found.
Please note: Cppcheck does not need standard library headers to get proper results.

[202201464_lab7_1.c:29]: (style) The scope of the variable 'ch' can be reduced. [202201464_lab7_1.c:11]: (style) Parameter 'file' can be declared as pointer to const [202201464_lab7_1.c:50]: (style) Variable 'direntp' can be declared as pointer to const

[202201464_lab7_1.c:35]: (warning) Storing fgetc() return value in char variable and then comparing with EOF.

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

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

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

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

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

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

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

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

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

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

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

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

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

[Hospital-Management-System.cpp:538]: (style) C-style pointer casting
[Hospital-Management-System.cpp:619]: (style) C-style pointer casting
[Hospital-Management-System.cpp:641]: (style) C-style pointer casting
[Hospital-Management-System.cpp:646]: (style) C-style pointer casting
[Hospital-Management-System.cpp:749]: (style) C-style pointer casting
[Hospital-Management-System.cpp:758]: (style) C-style pointer casting
[Hospital-Management-System.cpp:788]: (style) C-style pointer casting
[Hospital-Management-System.cpp:797]: (style) C-style pointer casting
[Hospital-Management-System.cpp:827]: (style) C-style pointer casting
[Hospital-Management-System.cpp:836]: (style) C-style pointer casting
[Hospital-Management-System.cpp:866]: (style) C-style pointer casting
[Hospital-Management-System.cpp:875]: (style) C-style pointer casting
[Hospital-Management-System.cpp:907]: (style) C-style pointer casting
[Hospital-Management-System.cpp:973]: (style) C-style pointer casting
[Hospital-Management-System.cpp:982]: (style) C-style pointer casting
[Hospital-Management-System.cpp:1012]: (style) C-style pointer casting
[Hospital-Management-System.cpp:1021]: (style) C-style pointer casting
[Hospital-Management-System.cpp:1051]: (style) C-style pointer casting
[Hospital-Management-System.cpp:1060]: (style) C-style pointer casting
[Hospital-Management-System.cpp:1090]: (style) C-style pointer casting
[Hospital-Management-System.cpp:1099]: (style) C-style pointer casting
[Hospital-Management-System.cpp:1181]: (style) C-style pointer casting
[Hospital-Management-System.cpp:1207]: (style) C-style pointer casting
[Hospital-Management-System.cpp:1216]: (style) C-style pointer casting
[Hospital-Management-System.cpp:1307]: (style) C-style pointer casting
[Hospital-Management-System.cpp:1317]: (style) C-style pointer casting

[Hospital-Management-System.cpp:1320]: (style) C-style pointer casting
[Hospital-Management-System.cpp:427]: (style) Consecutive return, break, continue, goto or throw statements are unnecessary.
[Hospital-Management-System.cpp:443]: (style) Consecutive return, break, continue, goto or throw statements are unnecessary.
[Hospital-Management-System.cpp:459]: (style) Consecutive return, break, continue, goto or throw statements are unnecessary.
[Hospital-Management-System.cpp:892]: (style) Consecutive return, break, continue, goto or throw statements are unnecessary.

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

[Hospital-Management-System.cpp:48] -> [Hospital-Management-System.cpp:277]: (style)

Local variable 'user' shadows outer function

[Hospital-Management-System.cpp:40] -> [Hospital-Management-System.cpp:304]: (style)

Local variable 'c' shadows outer variable

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

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

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

