IT-314 Software Engineering

Lab Report - 7

Autumn Semester (AY 2024-25)

Dave Preet A.
202201072

Aim:

Program Inspection, Debugging and Static Analysis

Section 2:

Armstrong Number: Errors and Fixes

1. How many errors are there in the program?
There are two errors identified in the program.

2. How many breakpoints are needed to fix those errors?
We require two breakpoints to address these errors.

Steps Taken to Fix the Errors:

- **Error 1:** The operations for division and modulus are incorrectly swapped within the while loop.
  Fix: Adjust the code to ensure that the modulus operation retrieves the last digit, while the division operation appropriately reduces the number for subsequent iterations.
- **Error 2:** The variable used for checking is not accumulating values correctly.
  Fix: Revise the logic to guarantee that this variable accurately reflects the sum of each digit raised to the power of the total number of digits in the original number.

```
class Armstrong {
public static void main(String args[]) {
int num = Integer.parseInt(args[0]);
int n = num; // use to check at last time
int check = 0, remainder;
while (num > 0) {
    remainder = num % 10;
check = check + (int)Math.pow(remainder,
3);
num = num / 10;
}
if (check == n)
System.out.println(n + " is an Armstrong
Number");
else
System.out.println(n + " is not an
Armstrong Number");
}
}
```

GCD and LCM: Errors and Fixes

1. How many errors are there in the program?
There is one error identified in the program.

2. How many breakpoints are needed to fix this error?
We need one breakpoint to address this error.

Steps Taken to Fix the Error:

- **Error:** The condition in the while loop of the GCD method is incorrect.
  Fix: Modify the condition to while (a % b != 0) instead of while (a % b == 0). This change ensures that the loop continues until the remainder is zero, thereby correctly calculating the GCD.

```
import java.util.Scanner;
public class GCD_LCM {
static int gcd(int x, int y) {
int r = 0, a, b;
a = (x > y) ? x : y; // a is greater number
b = (x < y) ? x : y; // b is smaller number
r = b;
while (a % b != 0) {
r = a % b;
a = b;
b = r;
}
return r;
}
static int lcm(int x, int y) {
int a;
a = (x > y) ? x : y; // a is greater number
while (true) {
if (a % x == 0 && a % y == 0)
return a;
++a;
}
}
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();
}
}
```

Knapsack Problem: Errors and Fixes

1. How many errors are there in the program?
There are three errors identified in the program.

2. How many breakpoints are needed to fix these errors?
We need two breakpoints to address these errors.

Steps Taken to Fix the Errors:

- **Error:** In the "take item n" case, the condition is incorrect.
  Fix: Change if (weight[n] > w) to if (weight[n] <= w) to ensure the profit is calculated when the item can be included.
- **Error:** The profit calculation is incorrect.
  Fix: Change profit[n-2] to profit[n] to ensure the correct profit value is used.
- **Error:** In the "don't take item n" case, the indexing is incorrect.
  Fix: Change opt[n++][w] to opt[n-1][w] to properly index the item

```
public class Knapsack {
    public static void main(String[] args) {
    int N = Integer.parseInt(args[0]); // numberof items
    int W = Integer.parseInt(args[1]); // maximum
    weight of knapsack
    int[] profit = new int[N+1];
    int[] weight = new int[N+1];
    // generate random instance, items 1..N
```

```
    for (int n = 1; n <= N; n++) {
    profit[n] = (int) (Math.random() * 1000);
    weight[n] = (int) (Math.random() * W);
    }
    // opt[n][w] = max profit of packing items 1..n
    with weight limit w
    // sol[n][w] = does opt solution to pack items
    1..n with weight limit w include item n?
    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++) {
    // don't take item n
    int option1 = opt[n-1][w];
    // take item n
    int option2 = Integer.MIN_VALUE;
    if (weight[n] <= w) option2 = profit[n]
    + opt[n-1][w-weight[n]];
    // select better of two options
    opt[n][w] = Math.max(option1, option2);
    sol[n][w] = (option2 > option1)
    }
    }
    // determine which items to take
    boolean[] take = new boolean[N+1];
    for (int n = N, w = W; n > 0; n--) {
    if (sol[n][w]) { take[n] = true; w = w -
    weight[n]; }
    else { take[n] = false;
    }
    }
    // print results
    System.out.println("item" + "\t" + "profit" +
    "\t" + "weight" + "\t" + "take");
```

```
      for (int n = 1; n <= N; n++) {
      System.out.println(n + "\t" + profit[n] +
      "\t" + weight[n] + "\t" + take[n]);
      }
      }
      }
```

Magic Number Check: Errors and Fixes

1. How many errors are there in the program?
There are three errors identified in the program.

2. How many breakpoints are needed to fix these errors?
We need one breakpoint to address these errors.

Steps Taken to Fix the Errors:

- **Error:** The condition in the inner while loop is incorrect.
  Fix: Change while(sum == 0) to while(sum != 0) to ensure that the loop processes digits correctly.
- **Error:** The calculation of s in the inner loop is incorrect.
  Fix: Change s = s * (sum / 10) to s = s + (sum % 10) to correctly sum the digits.
- **Error:** The order of operations in the inner while loop is incorrect.
  Fix: Reorder the operations to s = s + (sum % 10); sum = sum / 10; to correctly accumulate the digit sum.

```
// Program to check if number is Magic number in JAVA
import java.util.*;
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);
                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.");
        }
    }
}
```

Merge Sort: Errors and Fixes

1. How many errors are there in the program?
There are three errors identified in the program.

2. How many breakpoints are needed to fix these errors?
We need two breakpoints to address these errors.

Steps Taken to Fix the Errors:

- **Error:** Incorrect array indexing when splitting the array in mergeSort.
  Fix: Change int[] left = leftHalf(array + 1) to int[] left = leftHalf(array) and int[] right = rightHalf(array - 1) to int[] right = rightHalf(array) to pass the array correctly.
- **Error:** Incorrect increment and decrement in merge.
  Fix: Remove the ++ and -- from merge(array, left++, right--) and instead use merge(array, left, right) to pass the arrays directly.
- **Error:** The array access in the merge function is incorrectly accessing beyond the array bounds.

Fix: Ensure the array boundaries are respected by adjusting the indexing in the merging logic.

```
// This program implements the merge sort algorithm for
// arrays of integers.

import java.util.*;

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));
    }

    // Places the elements of the given array into sorted order
    // using the merge sort algorithm.
    // post: array is in sorted (nondecreasing) order
    public static void mergeSort(int[] array) {
        if (array.length > 1) {
            // split array into two halves
            int mid = array.length / 2; // Find the midpoint
            int[] left = Arrays.copyOfRange(array, 0, mid); // Get the left
half
            int[] right = Arrays.copyOfRange(array, mid, array.length); //
Get the right half

            // recursively sort the two halves
            mergeSort(left);
            mergeSort(right);

            // merge the sorted halves into a sorted whole
            merge(array, left, right);
        }
```

```
    }

    // Merges the given left and right arrays into the given
    // result array.
    // pre : result is empty; left/right are sorted
    // post: result contains result of merging sorted lists;
    public static void merge(int[] result, int[] left, int[] right) {
        int i1 = 0;   // index into left array
        int i2 = 0;   // index into right array

        for (int i = 0; i < result.length; i++) {
            if (i2 >= right.length || (i1 < left.length && left[i1] <=
right[i2])) {
                result[i] = left[i1];    // take from left
                i1++;
            } else {
                result[i] = right[i2];   // take from right
                i2++;
            }
        }
    }
}

```

Matrix Multiplication: Errors and Fixes

1. **How many errors are there in the program?**
   There is **1 error** in the program.
2. **How many breakpoints do you need to fix this error?**
   We need **1 breakpoint** to fix this error.
3. **Steps Taken to Fix the Error:**
   - **Error:** Incorrect array indexing in the matrix multiplication logic.
   - **Fix:** Change first[c-1][c-k] and second[k-1][k-d] to first[c][k] and second[k][d]. These changes ensure that matrix elements are correctly referenced during multiplication.

```java
// Java program to multiply two matrices
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 first matrix");
          m = in.nextInt();
          n = in.nextInt();

          int first[][] = new int[m][n];

          System.out.println("Enter the elements of 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 second matrix");
          p = in.nextInt();
          q = in.nextInt();

          if (n != p)
              System.out.println("Matrices with entered orders can't be multiplied with each other.");
          else {
              int second[][] = new int[p][q];
              int multiply[][] = new int[m][q];

```

```
34.             System.out.println("Enter the elements of second
    matrix");
35.
36.             for (c = 0; c < p; c++)
37.                 for (d = 0; d < q; d++)
38.                     second[c][d] = in.nextInt();
39.
40.             for (c = 0; c < m; c++) {
41.                 for (d = 0; d < q; d++) {
42.                     for (k = 0; k < n; k++) { // Change p to n for
    correct iteration
43.                         sum = sum + first[c][k] * second[k][d]; //
    Corrected indexing
44.                     }
45.
46.                     multiply[c][d] = sum;
47.                     sum = 0;
48.                 }
49.             }
50.
51.             System.out.println("Product of entered matrices:-");
52.
53.             for (c = 0; c < m; c++) {
54.                 for (d = 0; d < q; d++)
55.                     System.out.print(multiply[c][d] + "\t");
56.
57.                 System.out.print("\n");
58.             }
59.         }
60.
61.         in.close(); // Close the scanner
62.     }
63. }
64.
```

Quadratic Probing Hash Table: Errors and Fixes

1. **How many errors are there in the program?**
    - There is **1 error** in the program.
2. **How many breakpoints do you need to fix this error?**
    - We need **1 breakpoint** to fix this error.
3. **Steps Taken to Fix the Error:**
    - **Error:** In the insert method, the line i += (i + h / h--) % maxSize; is incorrect.
    - **Fix:** The correct logic should be i = (i + h * h++) % maxSize; to correctly implement quadratic probing

```
/**
 * Java Program to implement Quadratic Probing Hash Table
 */

import java.util.Scanner;

    /** Class QuadraticProbingHashTable **/
    class QuadraticProbingHashTable {
        private int currentSize, maxSize;
        private String[] keys;
        private String[] vals;

        /** Constructor **/
        public QuadraticProbingHashTable(int capacity) {
            currentSize = 0;
            maxSize = capacity;
            keys = new String[maxSize];
            vals = new String[maxSize];
        }

        /** Function to clear hash table **/
```

```
25.         public void makeEmpty() {
26.             currentSize = 0;
27.             keys = new String[maxSize];
28.             vals = new String[maxSize];
29.         }
30.
31.         /** Function to get size of hash table
    **/
32.         public int getSize() {
33.             return currentSize;
34.         }
35.
36.         /** Function to check if hash table is
    full **/
37.         public boolean isFull() {
38.             return currentSize == maxSize;
39.         }
40.
41.         /** Function to check if hash table is
    empty **/
42.         public boolean isEmpty() {
43.             return getSize() == 0;
44.         }
45.
46.         /** Function to check if hash table
    contains a key **/
47.         public boolean contains(String key) {
48.             return get(key) != null;
49.         }
50.
51.         /** Function to get hash code of a given
    key **/
52.         private int hash(String key) {
```

```
53.             return (key.hashCode() % maxSize +
  maxSize) % maxSize; // Ensure positive index
54.         }
55.
56.         /** Function to insert key-value pair **/
57.         public void insert(String key, String
  val) {
58.             if (isFull()) {
59.                 System.out.println("Hash table is
  full. Cannot insert.");
60.                 return;
61.             }
62.
63.             int tmp = hash(key);
64.             int i = tmp, h = 1;
65.
66.             do {
67.                 if (keys[i] == null) {
68.                     keys[i] = key;
69.                     vals[i] = val;
70.                     currentSize++;
71.                     return;
72.                 }
73.                 if (keys[i].equals(key)) {
74.                     vals[i] = val; // Update
  existing key
75.                     return;
76.                 }
77.                 i = (tmp + h * h) % maxSize; //
  Update index for quadratic probing
78.                 h++;
79.             } while (i != tmp);
80.         }
81.
```

```
82.         /** Function to get value for a given key
  **/
83.         public String get(String key) {
84.             int i = hash(key), h = 1;
85.
86.             while (keys[i] != null) {
87.                 if (keys[i].equals(key))
88.                     return vals[i];
89.                 i = (i + h * h) % maxSize;
90.                 h++;
91.             }
92.
93.             return null;
94.         }
95.
96.         /** Function to remove key and its value
  **/
97.         public void remove(String key) {
98.             if (!contains(key)) return;
99.
100.            /** Find position key and delete **/
101.            int i = hash(key), h = 1;
102.
103.            while (!key.equals(keys[i])) {
104.                i = (i + h * h) % maxSize;
105.                h++;
106.            }
107.
108.            keys[i] = vals[i] = null;
109.
110.            /** Rehash all keys **/
111.            for (i = (i + h * h) % maxSize;
  keys[i] != null; i = (i + h * h) % maxSize) {
```

```
112.                 String tmp1 = keys[i], tmp2 =
  vals[i];
113.                 keys[i] = vals[i] = null;
114.                 currentSize--;
115.                 insert(tmp1, tmp2);
116.             }
117.             currentSize--;
118.         }
119.
120.         /** Function to print HashTable **/
121.         public void printHashTable() {
122.             System.out.println("\nHash Table: ");
123.             for (int i = 0; i < maxSize; i++)
124.                 if (keys[i] != null)
125.                     System.out.println(keys[i] +
  " " + vals[i]);
126.             System.out.println();
127.         }
128.     }
129.
130.     /** Class QuadraticProbingHashTableTest **/
131.     public class QuadraticProbingHashTableTest {
132.         public static void main(String[] args) {
133.             Scanner scan = new
  Scanner(System.in);
134.             System.out.println("Hash Table
  Test\n\n");
135.             System.out.println("Enter size");
136.
137.             /** Make object of
  QuadraticProbingHashTable **/
138.             QuadraticProbingHashTable qpht = new
  QuadraticProbingHashTable(scan.nextInt());
139.
```

```
140.            char ch;
141.            /** Perform QuadraticProbingHashTable
  operations **/
142.            do {
143.                System.out.println("\nHash Table
  Operations\n");
144.                System.out.println("1. insert ");
145.                System.out.println("2. remove");
146.                System.out.println("3. get");
147.                System.out.println("4. clear");
148.                System.out.println("5. size");
149.
150.                int choice = scan.nextInt();
151.                switch (choice) {
152.                    case 1:
153.                        System.out.println("Enter
  key and value");
154.                        qpht.insert(scan.next(),
  scan.next());
155.                        break;
156.                    case 2:
157.                        System.out.println("Enter
  key");
158.                        qpht.remove(scan.next());
159.                        break;
160.                    case 3:
161.                        System.out.println("Enter
  key");
162.                        System.out.println("Value
  = " + qpht.get(scan.next()));
163.                        break;
164.                    case 4:
165.                        qpht.makeEmpty();
```

```
166.                         System.out.println("Hash
  Table Cleared\n");
167.                         break;
168.                     case 5:
169.                         System.out.println("Size
  = " + qpht.getSize());
170.                         break;
171.                     default:
172.                         System.out.println("Wrong
  Entry \n ");
173.                         break;
174.                 }
175.                 /** Display hash table **/
176.                 qpht.printHashTable();
177.
178.                 System.out.println("\nDo you want
  to continue (Type y or n) \n");
179.                 ch = scan.next().charAt(0);
180.             } while (ch == 'Y' || ch == 'y');
181.
182.             scan.close(); // Close scanner to
  avoid resource leak
183.         }
184.     }
185.
```

Sorting Array

Identified Issues:

- **Total Errors**: The program contains **2 errors**.
- **Breakpoints Required**: To resolve these errors, **2 breakpoints** are necessary.

Error Analysis and Corrections:

1. **Error 1**: The loop initialization for iterating through the array is incorrect.

- **Original Condition**: for (int i = 0; i >= n; i++);
- **Correction**: Update it to for (int i = 0; i < n; i++) to ensure proper iteration over the array elements.

2. **Error 2**: The condition within the inner loop that determines the sorting order is reversed.
   - **Original Condition**: if (a[i] <= a[j])
   - **Correction**: Modify it to if (a[i] > a[j]) to accurately implement ascending order sorting.

```
3. // Sorting the array in ascending order
4. import java.util.Scanner;
5. 
6. public class AscendingOrder
7. {
8.     public static void main(String[] args)
9.     {
10.            int n, temp;
11.            Scanner s = new Scanner(System.in);
12.            System.out.print("Enter no. of
   elements you want in array: ");
13.            n = s.nextInt();
14.            int a[] = new int[n];
15.            System.out.println("Enter all the
   elements:");
16. 
17.            // Reading elements into the array
18.            for (int i = 0; i < n; i++)
19.            {
20.                a[i] = s.nextInt();
21.            }
22. 
23.            // Corrected loop for sorting the
   array
24.            for (int i = 0; i < n; i++) //
   Changed condition from 'i >= n' to 'i < n'
25.            {
```

```
                for (int j = i + 1; j < n; j++)
                {
                    // Corrected condition for sorting
                    if (a[i] > a[j]) // Changed from 'a[i] <= a[j]' to 'a[i] > a[j]'
                    {
                        // Swap elements
                        temp = a[i];
                        a[i] = a[j];
                        a[j] = temp;
                    }
                }
            }

            // Printing the sorted array
            System.out.print("Ascending Order: ");
            for (int i = 0; i < n - 1; i++)
            {
                System.out.print(a[i] + ", ");
            }
            System.out.print(a[n - 1]); // Print the last element without a comma
        }
    }

```

Stack Implementation Errors and Resolutions

- **Total Errors:** 2

- **Breakpoints Required:** 2

Error Analysis and Corrections

1. **Error in push Method:**
    - **Description of Error:** The statement top-- is incorrectly implemented. This operation incorrectly decrements the stack pointer, which can lead to stack underflow.
    - **Proposed Fix:** Modify the line to top++ to correctly increment the stack pointer, allowing for the addition of new elements to the stack.
2. **Error in display Method:**
    - **Description of Error:** The loop condition for (int i = 0; i > top; i++) is incorrectly set. This condition prevents any elements from being displayed if top is greater than or equal to zero.
    - **Proposed Fix:** Change the loop condition to for (int i = 0; i <= top; i++) to ensure that all elements in the stack are displayed correctly.

```
3.// Stack implementation in Java
4.import java.util.Arrays;
5.
6.public class StackMethods {
7.    private int top;
8.    int size;
9.    int[] stack;
10.
11.        public StackMethods(int arraySize) {
12.            size = arraySize;
13.            stack = new int[size];
14.            top = -1; // Initialize top to -1 to
  indicate an empty stack
15.        }
16.
17.        public void push(int value) {
18.            if (top == size - 1) {
19.                System.out.println("Stack is
  full, can't push a value");
20.            } else {
21.                top++; // Increment the top
  pointer
```

```
22.                 stack[top] = value; // Push the
  value onto the stack
23.             }
24.         }
25.
26.         public void pop() {
27.             if (!isEmpty()) {
28.                 System.out.println("Popped value:
  " + stack[top]); // Print the popped value
29.                 top--; // Decrement the top
  pointer
30.             } else {
31.                 System.out.println("Can't
  pop...stack is empty");
32.             }
33.         }
34.
35.         public boolean isEmpty() {
36.             return top == -1; // Check if the
  stack is empty
37.         }
38.
39.         public void display() {
40.             if (isEmpty()) {
41.                 System.out.println("Stack is
  empty.");
42.                 return;
43.             }
44.
45.             System.out.print("Stack elements: ");
46.             for (int i = 0; i <= top; i++) { //
  Corrected loop condition to display all elements
47.                 System.out.print(stack[i] + " ");
48.             }
```

```
            System.out.println();
        }
    }

    public class StackReviseDemo {
        public static void main(String[] args) {
            StackMethods newStack = new
  StackMethods(5);
            newStack.push(10);
            newStack.push(1);
            newStack.push(50);
            newStack.push(20);
            newStack.push(90);

            newStack.display(); // Display stack
  elements
            newStack.pop(); // Pop last value
            newStack.pop(); // Pop next value
            newStack.pop(); // Pop next value
            newStack.pop(); // Pop next value
            newStack.display(); // Display stack
  after pops
        }
    }

```

Tower of Hanoi Implementation

Errors and Fixes:

- **How many errors are there in the program?** There is 1 error in the program.

- **How many breakpoints do you need to fix this error?** We need 1 breakpoint to fix this error.
- **Steps Taken to Fix the Error:**
    - **Error:** In the recursive call doTowers(topN ++, inter--, from+1, to+1);, incorrect increments and decrements are applied to the variables.
    - **Fix:** Change the call to doTowers(topN - 1, inter, from, to); for proper recursion and to follow the Tower of Hanoi logic.

```
// Tower of Hanoi
public class MainClass {
    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); // Move top N-1 disks from source to intermediate
            System.out.println("Disk " + topN + " from " + from + " to " + to); // Move the Nth disk
            doTowers(topN - 1, inter, from, to); // Move N-1 disks from intermediate to destination
        }
    }
}

```

Section 1:

First 200 lines

Category	Description
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, leading to errors if 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.
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 arrays char specialization[100]; and char gender[100]; could benefit from explicit initialization to avoid issues with undefined values.
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.
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 safer.
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.
G: Input/Output Error	File Handling: The system should ensure all files like Doctor_Data.dat are opened before use and closed after 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

Category	Description
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.

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.
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.
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.
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.
G: Input/Output Error	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.

Third 200 lines:

Category	Description
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 after file opening. Always ensure that the file opens correctly before proceeding.
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.
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.
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..
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.
G: Input/Output Error	**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:

Category	Description

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.
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.
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.
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.
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.
G: Input/Output Error	**File Handling Issues:** The files (Patient_Data.dat, Doctor_Data.dat) are opened in various functions like add_patient_data() without proper error checking after opening. Failure to handle file opening errors may result in runtime issues.

Fifth 200 code:

Category	Description
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.
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.
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..
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.
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.
G: Input/Output Error	**File Handling:** Files like Patient_Data.dat and Doctor_Data.dat are opened but sometimes not closed properly in certain branches of the code. This can lead to resource leakage. Proper exception handling should be added to prevent this.

Final Segment:

Category	Description

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.
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.
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().
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.
G: Input/Output Error	**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.

Debuging:

STATIC ANALYSIS TOOL:

Using cppcheck, I run static analysis tool for 1300 lines of code used above

for program inspection.

Results:

[202201072_Lab7_2.c:1]: (information) Include file: <stdio.h> not found. Please note:

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

[202201072_Lab7_2.c:2]: (information) Include file: <stdlib.h> not found. Please note:

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

[202201072_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.

[202201072_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.

[202201072_Lab7_2.c:5]: (information) Include file: <unistd.h> not found. Please note:

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

[202201072_Lab7_2.c:6]: (information) Include file: <dirent.h> not found. Please note:

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

[202201072_Lab7_2.c:7]: (information) Include file: <fcntl.h> not found. Please note:

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

[202201072_Lab7_2.c:8]: (information) Include file: <libgen.h> not found. Please note:

Cppcheck does not need standard library headers to get proper results.
[202201072_Lab7_2.c:9]: (information) Include file: <errno.h> not found. Please note:
Cppcheck does not need standard library headers to get proper results.
[202201072_Lab7_2.c:10]: (information) Include file: <string.h> not found. Please note:
Cppcheck does not need standard library headers to get proper results.
[202201072_Lab7_2.c:0]: (information) Limiting analysis of branches. Use
--check-level=exhaustive to analyze all branches.
[202201072_Lab7_2.c:116]: (warning) scanf() without field width limits can crash with
huge input data.
[202201072_Lab7_2.c:120]: (warning) scanf() without field width limits can crash with
huge input data.
[202201072_Lab7_2.c:126]: (warning) scanf() without field width limits can crash with
huge input data.
[202201072_Lab7_2.c:127]: (warning) scanf() without field width limits can crash with
huge input data.
[202201072_Lab7_2.c:133]: (warning) scanf() without field width limits can crash with
huge input data.
[202201072_Lab7_2.c:34]: (style) The scope of the variable 'ch' can be reduced.
[202201072_Lab7_2.c:115]: (style) The scope of the variable 'path2' can be reduced.
[202201072_Lab7_2.c:16]: (style) Parameter 'file' can be declared as pointer to const
[202201072_Lab7_2.c:55]: (style) Variable 'direntp' can be declared as pointer to const
[202201072_Lab7_2.c:40]: (warning) Storing fgetc() return value in char variable and
then comparing with EOF.
[202201072_Lab7_3.c:1]: (information) Include file: <stdio.h> not found. Please note:
Cppcheck does not need standard library headers to get proper results.
[202201072_Lab7_3.c:2]: (information) Include file: <stdlib.h> not found. Please note:
Cppcheck does not need standard library headers to get proper results.
[202201072_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.
[202201072_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.

[202201072_Lab7_3.c:5]: (information) Include file: <unistd.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201072_lab7_1.c:1]: (information) Include file: <stdio.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201072_lab7_1.c:2]: (information) Include file: <stdlib.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201072_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.
[202201072_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.
[202201072_lab7_1.c:5]: (information) Include file: <unistd.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201072_lab7_1.c:6]: (information) Include file: <dirent.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201072_lab7_1.c:7]: (information) Include file: <fcntl.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201072_lab7_1.c:8]: (information) Include file: <libgen.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201072_lab7_1.c:9]: (information) Include file: <errno.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201072_lab7_1.c:29]: (style) The scope of the variable 'ch' can be reduced.
[202201072_lab7_1.c:11]: (style) Parameter 'file' can be declared as pointer to const
[202201072_lab7_1.c:50]: (style) Variable 'direntp' can be declared as pointer to const
[202201072_lab7_1.c:35]: (warning) Storing fgetc() return value in char variable and then comparing with EOF.
[Covid-Management-System.cpp:4]: (information) Include file: <iostream> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[Covid-Management-System.cpp:5]: (information) Include file: <cstring> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[Covid-Management-System.cpp:6]: (information) Include file: <windows.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

[Covid-Management-System.cpp:7]: (information) Include file: <fstream> not found.
Please note: Cppcheck does not need standard library headers to get proper results.
[Covid-Management-System.cpp:8]: (information) Include file: <conio.h> not found.
Please note: Cppcheck does not need standard library headers to get proper results.
[Covid-Management-System.cpp:9]: (information) Include file: <iomanip> not found.
Please note: Cppcheck does not need standard library headers to get proper results.
[Covid-Management-System.cpp:10]: (information) Include file: <cstdlib> not found.
Please note: Cppcheck does not need standard library headers to get proper results.
[Covid-Management-System.cpp:11]: (information) Include file: <string> not found.
Please note: Cppcheck does not need standard library headers to get proper results.
[Covid-Management-System.cpp:12]: (information) Include file: <unistd.h> not found.
Please note: Cppcheck does not need standard library headers to get proper results.
[Covid-Management-System.cpp:562]: (portability) fflush() called on input stream 'stdin' may result in undefined behaviour on non-linux systems.
[Covid-Management-System.cpp:565]: (portability) fflush() called on input stream 'stdin' may result in undefined behaviour on non-linux systems.
[Covid-Management-System.cpp:614]: (portability) fflush() called on input stream 'stdin' may result in undefined behaviour on non-linux systems.
[Covid-Management-System.cpp:1121]: (portability) fflush() called on input stream 'stdin' may result in undefined behaviour on non-linux systems.
[Covid-Management-System.cpp:538]: (style) C-style pointer casting
[Covid-Management-System.cpp:619]: (style) C-style pointer casting
[Covid-Management-System.cpp:641]: (style) C-style pointer casting
[Covid-Management-System.cpp:646]: (style) C-style pointer casting
[Covid-Management-System.cpp:749]: (style) C-style pointer casting
[Covid-Management-System.cpp:758]: (style) C-style pointer casting
[Covid-Management-System.cpp:788]: (style) C-style pointer casting
[Covid-Management-System.cpp:797]: (style) C-style pointer casting
[Covid-Management-System.cpp:827]: (style) C-style pointer casting
[Covid-Management-System.cpp:836]: (style) C-style pointer casting
[Covid-Management-System.cpp:866]: (style) C-style pointer casting

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

passed by const reference.

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

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