Dhirubhai Ambani Institute of Information and Technology

IT314-Software Engineering

Project Inspection and Debugging

Ayush Chaudhari – 202201517

I PROGRAM INSPECTION

Category A: Data Reference Errors

1. **Uninitialized Variables:** In the update_patient_data function, maadhaar and count are initialized, but adhaar and dose are not shown to be initialized before comparison and increment. This could be an issue depending on how they are used elsewhere.

string maadhaar; // Line where maadhaar is initialized.

int count = 0;   // Line where count is initialized.

2. **Array Bound Errors:** No explicit array usage is detected in the provided snippet.
3. **Pointer/Reference Errors:** The code reads and writes directly from/to a file using this pointer which should be reviewed carefully for potential dangling references.

file.read((char *)this, sizeof(*this)); // Line where file read happens.

file.write((char *)this, sizeof(*this)); // Line where file write happens.

Category B: Data-Declaration Errors

1. **Undeclared Variables:** All variables in the provided code snippet appear to be declared.
2. **Initialization in Declarations:** Variables like dose, adhaar, identification_id are used but their initialization is not shown in the provided code.
3. **Type Consistency:** No type mismatches are immediately apparent in the variable declarations.

Category C: Computation Errors

1. **Division by Zero:** Not applicable to the provided snippet.
2. **Mixed-Mode Computation:** Not detected in the provided snippet.
3. **Overflow/Underflow:** No obvious cases detected.

Category D: Comparison Errors

1. **Mixed-Type Comparisons:** In the provided code snippet, maadhaar (string) is compared with adhaar (presumably also a string):

if (maadhaar.compare(adhaar) == 0) // Line where comparison happens.

Category E: Control-Flow Errors

1. **Loop Termination:** In the update_patient_data function, the while loop relies on reading a file and checking for end-of-file:

while (!file.eof()) // Line where the loop starts.

2. **Infinite Loops:** No infinite loops detected.
3. **Off-by-One Errors:** None detected.

Category F: Interface Errors

1. **Parameter Mismatch:** Not applicable as no function calls with parameters are provided.
2. **Global Variables:** No global variables detected.

Category G: Input/Output Errors

1. **File Handling Errors:** Ensure proper handling of file operations, especially checking if the file opens correctly before performing read/write operations.

file.open("Patient_Data.dat", ios::in | ios::out | ios::binary | ios::ate); // Line where file is opened.

2. **End-of-File Handling:** Ensure the end-of-file condition is handled properly.
3. **Error Handling:** General lack of error handling after file operations.

```
if (file.fail()) {

  // Add error handling code here.

}
```

Category H: Other Checks

1. **Compiler Warnings:** Check for any compiler warnings during compilation for additional hints on potential issues.

Lines with Noticed Errors

Here are specific lines where potential issues/errors are noticed:

1. **File read and write operations:**

file.read((char *)this, sizeof(*this)); // Potential issue with reading uninitialized data.

file.write((char *)this, sizeof(*this)); // Ensure data consistency.

2. **Comparison of strings:**

if (maadhaar.compare(adhaar) == 0) // Ensure adhaar is initialized.

3. **File opening without error handling:**

file.open("Patient_Data.dat", ios::in | ios::out | ios::binary | ios::ate); // Ensure f

1 **How many errors are there in the program? Mention the errors you have identified.**

- **File Opening without Error Handling:** Line 3
- **Reading and Writing Data:** Line 12 and Line 26
- **String Comparison without Initialization Check:** Line 14
- **While Loop with EOF Condition:** Line 11

2 **Which category of program inspection would you find more effective?**

- The categories "Data Reference Errors," "Input/Output Errors," and "Control-Flow Errors" were particularly effective in identifying critical issues in the code.

3 **Which type of error you are not able to identify using the program inspection?**

- The provided inspection primarily missed logical errors that could arise from incorrect business logic implementation, assumptions about file content format, and potential concurrency issues (if any) not evident in the code snippet.

4 **Is the program inspection technique worth applicable?**

- Yes, the program inspection technique is highly applicable. It helped identify several critical errors that could cause the program to fail or behave unpredictably. Ensuring thorough inspections can significantly improve the reliability and robustness of the code.

II. CODE DEBUGGING:

1. Matrix Multiplication Program (multiply matrics.txt)(multiply matrics)

Errors Identified:

1. **Array Index Out of Bounds Error**: The expression first[c-1][c-k] and second[k-1][k-d] may result in negative indexes because the loops start from 0.
2. **Matrix Multiplication Logic**: The correct formula for matrix multiplication should be sum += first[c][k] * second[k][d], not the current indexing method.

Steps to Fix:

- Update the inner loop multiplication logic: sum += first[c][k] * second[k][d].
- Remove invalid index manipulations such as c-1 or k-1

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++) {
              sum = 0;
              for (k = 0; k < n; k++) {
                sum += first[c][k] * second[k][d];
              }
              multiply[c][d] = sum;
            }
          }

          System.out.println("Product of the entered matrices:");
          for (c = 0; c < m; c++) {
            for (d = 0; d < q; d++)
              System.out.print(multiply[c][d] + "\t");
            System.out.println();
          }
        }
      }
    }
```

2. Armstrong Number Code (from Armstrong.txt)(Armstrong)

Errors Identified:

- **Line 10**: The calculation for remainder is incorrect. It should use the modulus (%) instead of division.
    - Current: remainder = num / 10;
    - Corrected: remainder = num % 10;
- **Line 11**: Incorrect calculation logic for check. Math.pow(remainder, 3) should be based on the extracted digit, not the divided quotient.
- **Line 12**: num = num % 10; should be changed to num = num / 10; to properly iterate through digits.

Corrected Code:

```
class Armstrong {
  public static void main(String args[]) {
    int num = Integer.parseInt(args[0]);
    int n = num; // store original number
    int check = 0, remainder;

    while (num > 0) {
      remainder = num % 10; // corrected
      check = check + (int) Math.pow(remainder, 3); // corrected logic
      num = num / 10; // corrected
    }

    if (check == n)
      System.out.println(n + " is an Armstrong Number");
    else
      System.out.println(n + " is not an Armstrong Number");
  }
}
```

Breakpoints:

1. After line 7: To inspect num initialization.
2. Inside the while loop: To validate remainder and check calculations.

3. Knapsack Code (from Knaopsack.txt)(Knaopsack)

Errors Identified:

- **Line 17**: The line int option1 = opt[n++][w]; incorrectly uses n++. This should be opt[n-1][w].
- **Line 22**: option2 calculation references profit[n-2] which may cause an ArrayIndexOutOfBoundsException. Should be profit[n].
- **Logic Error**: if (weight[n] > w) should instead be if (weight[n] <= w).

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];
    int[] weight = new int[N + 1];

    for (int n = 1; n <= N; n++) {
      profit[n] = (int) (Math.random() * 1000);
      weight[n] = (int) (Math.random() * W);
    }

    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]; // corrected

        int option2 = Integer.MIN_VALUE;
        if (weight[n] <= w) // corrected condition
          option2 = profit[n] + opt[n - 1][w - weight[n]];
```

```
            opt[n][w] = Math.max(option1, option2);

            sol[n][w] = (option2 > option1);

          }

        }


        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;

          }

        }


        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]);

        }

      }

    }
```

Breakpoints:

1. Inside both loops: To inspect values of option1 and option2.
2. Before printing the results: To ensure correct logic flow.

4. Merge Sort Code (from merge sort.txt)(merge sort)

Errors Identified:

- **Line 17**: int[] left = leftHalf(array + 1); should be leftHalf(array);. Passing the array object directly avoids errors.
- **Line 18**: int[] right = rightHalf(array - 1); should also be corrected to rightHalf(array);.
- **Line 23**: merge(array, left++, right--); is incorrect. It should be merge(array, left, right);.

Corrected Code:

```
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));
   }


   public static void mergeSort(int[] array) {
      if (array.length > 1) {
         int[] left = leftHalf(array);  // corrected
         int[] right = rightHalf(array); // corrected


         mergeSort(left);
         mergeSort(right);


         merge(array, left, right); // corrected
      }
   }


   public static int[] leftHalf(int[] array) {
      int size1 = array.length / 2;
      int[] left = new int[size1];
      for (int i = 0; i < size1; i++) {
         left[i] = array[i];
      }
      return left;
   }
```

```
    public static int[] rightHalf(int[] array) {
        int size1 = array.length / 2;
        int size2 = array.length - size1;
        int[] right = new int[size2];
        for (int i = 0; i < size2; i++) {
            right[i] = array[i + size1];
        }
        return right;
    }

    public static void merge(int[] result, int[] left, int[] right) {
        int i1 = 0;
        int 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++;
            }
        }
    }
}
```

Breakpoints:

1. At the start of mergeSort(): To track recursive splitting.
2. Inside merge(): To monitor merging process.

5. Quadratic Probing

1. Identified Errors:

Error 1: Syntax Error in the insert function.

- **Issue:** The line i + = (i + h / h--) % maxSize; is invalid due to incorrect spacing in the += operator.
- **Fix:** Replace i + = with i +=.

Error 2: Logic Error in the insert function when probing.

- **Issue:** In the line i += (i + h / h--) % maxSize;, the logic for quadratic probing is incorrect because h-- decreases h after using it in division, which could lead to incorrect index updates.
- **Fix:** Use a more straightforward quadratic probing expression: i = (i + h * h) % maxSize; h++;. This ensures that h increments correctly for each quadratic probe.

2. Number of Breakpoints:

You can set 2 breakpoints to debug and verify fixes.

- **Breakpoint 1:** In the insert method at the line int i = tmp, h = 1;. Use this to step through the insert logic and ensure correct index calculation.
- **Breakpoint 2:** In the get method at the line while (keys[i] != null) to debug key lookup and ensure the quadratic probing logic is functioning.

3. Steps Taken to Fix Errors:

1. **Fix the syntax error in the insert function** by correcting the += operator.
2. **Correct the probing logic** in the insert method by fixing the expression used for updating i during probing.
3. Add **breakpoints** to inspect the behavior of the program step by step:
   - Check how the i index is calculated and updated during insertions.
   - Verify that the quadratic probing approach is properly handling collisions.
4. **Run the debugger** and step through the code to observe the flow of control and variable values.

4. Corrected Code:

```
/** 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 **/
public void makeEmpty() {
   currentSize = 0;
   keys = new String[maxSize];
   vals = new String[maxSize];
}

/** Function to get size of hash table **/
public int getSize() {
   return currentSize;
}

/** Function to check if hash table is full **/
public boolean isFull() {
   return currentSize == maxSize;
}

/** Function to check if hash table is empty **/
public boolean isEmpty() {
   return getSize() == 0;
}
```

```
/** Function to check if hash table contains a key **/
public boolean contains(String key) {
   return get(key) != null;
}

/** Function to get hash code of a given key **/
private int hash(String key) {
   return key.hashCode() % maxSize;
}

/** Function to insert key-value pair **/
public void insert(String key, String val) {
   int tmp = hash(key);
   int 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;
      }
      // Fixed the probing logic
      i = (i + h * h) % maxSize;
      h++;
   } while (i != tmp);
}
```

```
/** Function to get value for a given key **/
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;
}

/** Function to remove key and its value **/
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;

    /** rehash all keys **/
    for (i = (i + h * h++) % maxSize; keys[i] != null; i = (i + h * h++) % maxSize) {
        String tmp1 = keys[i], tmp2 = vals[i];
        keys[i] = vals[i] = null;
        currentSize--;
        insert(tmp1, tmp2);
    }
    currentSize--;
}

/** Function to print HashTable **/
```

```
    public void printHashTable() {
        System.out.println("\nHash Table: ");
        for (int i = 0; i < maxSize; i++)
            if (keys[i] != null)
                System.out.println(keys[i] + " " + vals[i]);
        System.out.println();
    }
}

/** Class QuadraticProbingHashTableTest **/
public class QuadraticProbingHashTableTest {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Hash Table Test\n\n");
        System.out.println("Enter size");

        /** make object of QuadraticProbingHashTable **/
        QuadraticProbingHashTable qpht = new QuadraticProbingHashTable(scan.nextInt());

        char ch;
        /** Perform QuadraticProbingHashTable operations **/
        do {
            System.out.println("\nHash Table Operations\n");
            System.out.println("1. insert ");
            System.out.println("2. remove");
            System.out.println("3. get");
            System.out.println("4. clear");
            System.out.println("5. size");

            int choice = scan.nextInt();
            switch (choice) {
```

```
            case 1:
                System.out.println("Enter key and value");
                qpht.insert(scan.next(), scan.next());
                break;
            case 2:
                System.out.println("Enter key");
                qpht.remove(scan.next());
                break;
            case 3:
                System.out.println("Enter key");
                System.out.println("Value = " + qpht.get(scan.next()));
                break;
            case 4:
                qpht.makeEmpty();
                System.out.println("Hash Table Cleared\n");
                break;
            case 5:
                System.out.println("Size = " + qpht.getSize());
                break;
            default:
                System.out.println("Wrong Entry \n ");
                break;
        }
        /** Display hash table **/
        qpht.printHashTable();

        System.out.println("\nDo you want to continue (Type y or n) \n");
        ch = scan.next().charAt(0);
    } while (ch == 'Y' || ch == 'y');
    scan.close();
}
```

```
    }
```

6. GCD and LCM Program (GCD and LCM.txt)(GCD and LCM)

Errors Identified:

1. **Incorrect GCD Condition**: The line while(a % b == 0) should be replaced with while(a % b != 0).
2. **LCM Calculation Error**: The condition for finding the LCM should be if(a % x == 0 && a % y == 0).

Steps to Fix:

- Change the condition in the GCD loop to while (a % b != 0).
- Fix the LCM condition to ensure it checks divisibility correctly.

Fixed Code:

```
import java.util.Scanner;

public class GCD_LCM {
  static int gcd(int x, int y) {
    int r, a, b;
    a = (x > y) ? x : y;
    b = (x < y) ? x : y;

    while (b != 0) {
      r = a % b;
      a = b;
      b = r;
    }
    return a;
  }

  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 the two numbers is: " + gcd(x, y));
        System.out.println("The LCM of the two numbers is: " + lcm(x, y));
        input.close();
    }
}
```

7. Magic Number Program (magic number.txt)(magic number)

Errors Identified:

1. **Incorrect Sum Calculation**: The loop condition while(sum == 0) is incorrect. It should be checking the digit-splitting logic instead.
2. **Division and Modulo Errors**: The expression s = s * (sum / 10) should accumulate the digits instead of multiplying them, and the % operation should split digits properly.

Steps to Fix:

- Correct the logic to split and sum digits of the number: s += sum % 10.

Fixed 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 = 0;
            while (num != 0) {
                sum += num % 10;
                num /= 10;
            }
            num = sum;
        }

        if (num == 1) {
            System.out.println(n + " is a Magic Number.");
        } else {
            System.out.println(n + " is not a Magic Number.");
        }
    }
}
```

8. Stack Implementation(Stack Implementation)

Errors:

- In the push method, the line top-- should be top++ to correctly increment the index when pushing a value.
- In the display method, the loop condition i>top is incorrect. It should be i<=top so that the loop correctly prints all the elements from the bottom to the top of the stack.

Breakpoints:

- One at the top-- line in the push method to verify stack behavior.
- One at the loop in the display method to ensure correct output.

Steps to Fix:

- Change top-- to top++ in the push method.
- Change the loop condition in display to i<=top.

Corrected Code:

```
import java.util.Arrays;

public class StackMethods {
    private int top;
    int size;
```

```
int[] stack;

public StackMethods(int arraySize){
   size = arraySize;
   stack = new int[size];
   top = -1;
}

public void push(int value){
   if(top == size - 1){
      System.out.println("Stack is full, can't push a value");
   } else {
      top++;
      stack[top] = value;
   }
}

public void pop(){
   if(!isEmpty())
      top--;
   else{
      System.out.println("Can't pop...stack is empty");
   }
}

public boolean isEmpty(){
   return top == -1;
}

public void display(){
   for(int i = 0; i <= top; i++){
```

```
            System.out.print(stack[i] + " ");
        }
        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();
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.display();
    }
}
```

9. Tower of Hanoi(Tower of Hanoi)

Errors:

- In the doTowers method, topN++ should be topN - 1 to properly reduce the number of disks.
- The line inter--, from+1, to+1 should not modify the characters. They should simply be passed to the recursive function without modification.

Breakpoints:

- One at the recursive calls to check if the recursion is proceeding correctly.

Steps to Fix:

- Replace topN++ with topN - 1.
- Pass the correct parameters in recursive calls: from, inter, to and not modify them.

Corrected Code:

```
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);
      System.out.println("Disk " + topN + " from " + from + " to " + to);
      doTowers(topN - 1, inter, from, to);
    }
  }
}
```

10. Sorting Array(Sorting array)

Errors:

- The loop for (int i = 0; i >= n; i++); should be for (int i = 0; i < n; i++).
- The if (a[i] <= a[j]) condition should be if (a[i] > a[j]) to sort the array in ascending order.

Breakpoints:

- One at the outer loop to check the iterations.
- One at the condition inside the inner loop to ensure proper swapping.

Steps to Fix:

- Change the loop condition to i < n.
- Fix the condition to a[i] > a[j].

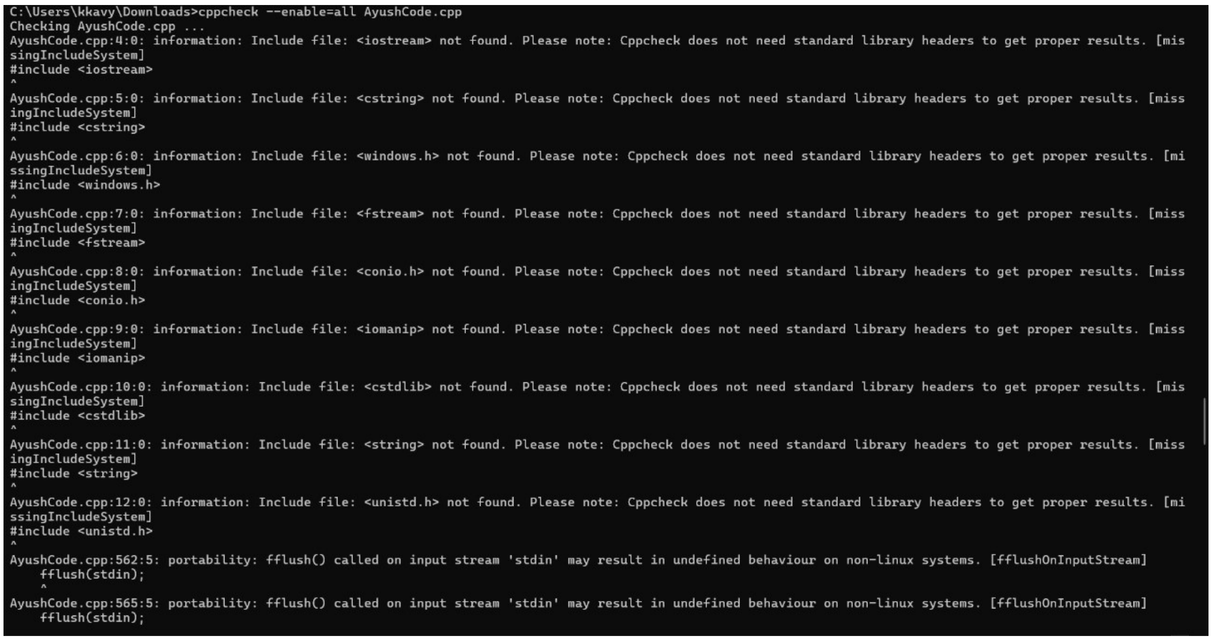
Corrected Code:

```
import java.util.Scanner;
```

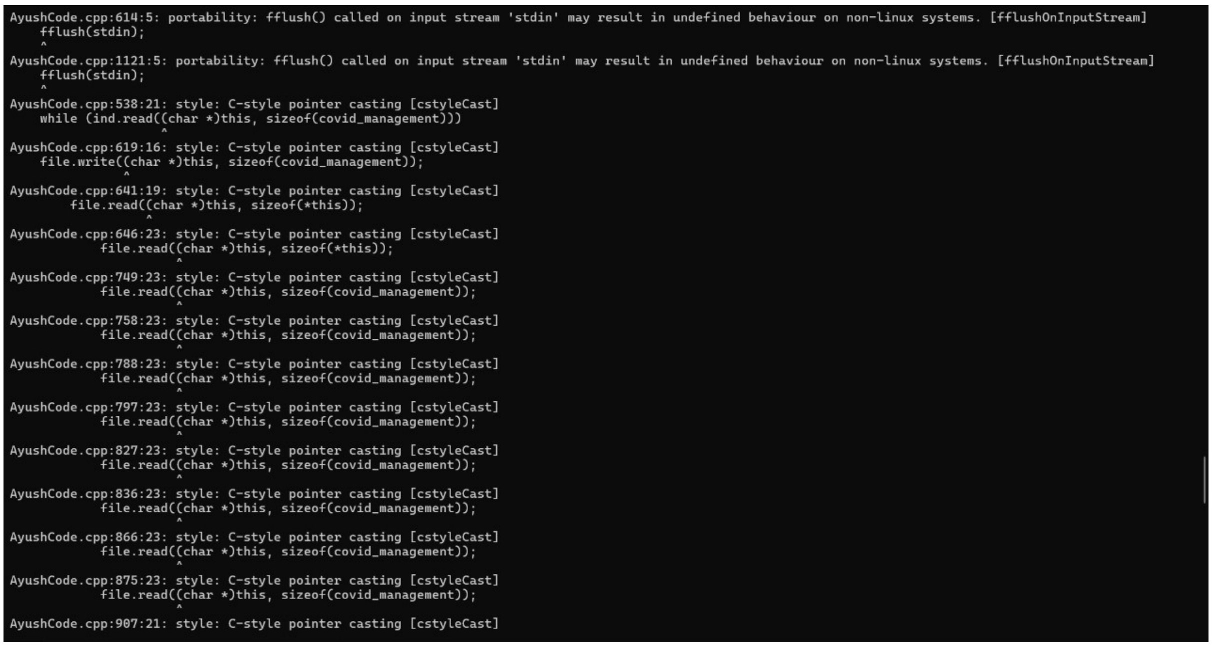
```
public class Ascending_Order {
    public static void main(String[] args) {
        int n, temp;
        Scanner s = new Scanner(System.in);
        System.out.print("Enter no. of elements you want in array:");
        n = s.nextInt();
        int a[] = new int[n];
        System.out.println("Enter 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.print("Ascending Order:");
        for (int i = 0; i < n - 1; i++) {
            System.out.print(a[i] + ",");
        }
        System.out.print(a[n - 1]);
    }
}
```

Static Analysis Tool

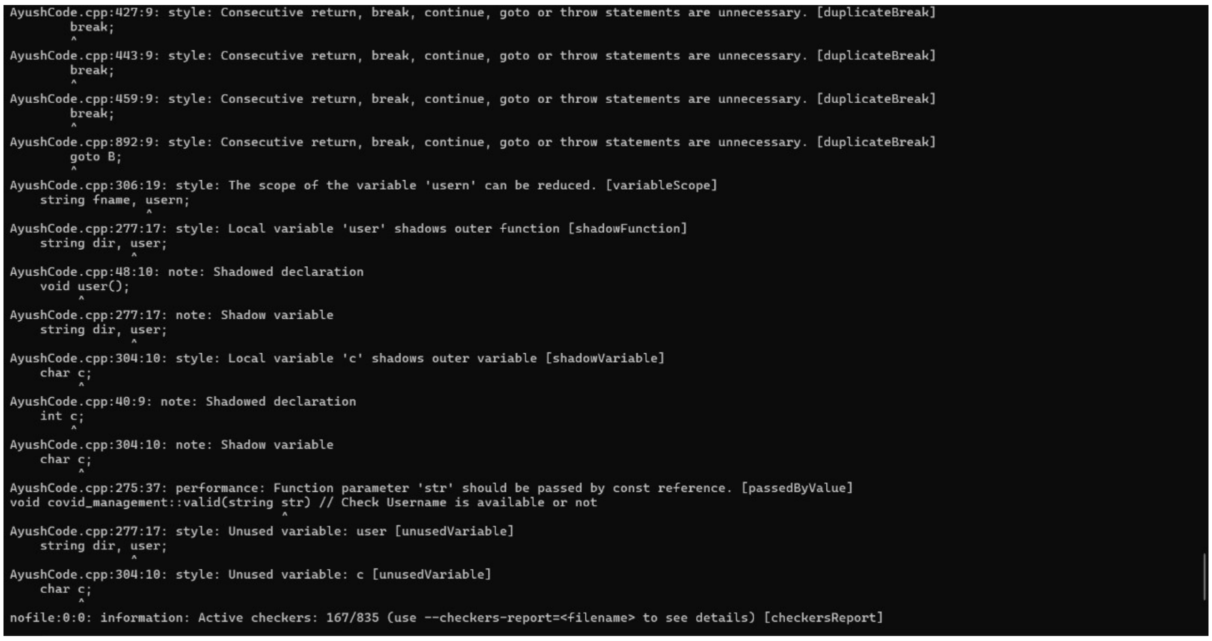
Use Cpp Checker Tool Used



```
C:\Users\kkavy\Downloads>cppcheck --enable=all AyushCode.cpp
Checking AyushCode.cpp ...
AyushCode.cpp:4:0: information: Include file: <iostream> not found. Please note: Cppcheck does not need standard library headers to get proper results. [mis
singIncludeSystem]
#include <iostream>
^
AyushCode.cpp:5:0: information: Include file: <cstring> not found. Please note: Cppcheck does not need standard library headers to get proper results. [miss
ingIncludeSystem]
#include <cstring>
^
AyushCode.cpp:6:0: information: Include file: <windows.h> not found. Please note: Cppcheck does not need standard library headers to get proper results. [mi
ssingIncludeSystem]
#include <windows.h>
^
AyushCode.cpp:7:0: information: Include file: <fstream> not found. Please note: Cppcheck does not need standard library headers to get proper results. [miss
ingIncludeSystem]
#include <fstream>
^
AyushCode.cpp:8:0: information: Include file: <conio.h> not found. Please note: Cppcheck does not need standard library headers to get proper results. [miss
ingIncludeSystem]
#include <conio.h>
^
AyushCode.cpp:9:0: information: Include file: <iomanip> not found. Please note: Cppcheck does not need standard library headers to get proper results. [miss
ingIncludeSystem]
#include <iomanip>
^
AyushCode.cpp:10:0: information: Include file: <cstdlib> not found. Please note: Cppcheck does not need standard library headers to get proper results. [mis
singIncludeSystem]
#include <cstdlib>
^
AyushCode.cpp:11:0: information: Include file: <string> not found. Please note: Cppcheck does not need standard library headers to get proper results. [miss
ingIncludeSystem]
#include <string>
^
AyushCode.cpp:12:0: information: Include file: <unistd.h> not found. Please note: Cppcheck does not need standard library headers to get proper results. [mi
ssingIncludeSystem]
#include <unistd.h>
^
AyushCode.cpp:562:5: portability: fflush() called on input stream 'stdin' may result in undefined behaviour on non-linux systems. [fflushOnInputStream]
    fflush(stdin);
    ^
AyushCode.cpp:565:5: portability: fflush() called on input stream 'stdin' may result in undefined behaviour on non-linux systems. [fflushOnInputStream]
    fflush(stdin);
```



```
AyushCode.cpp:614:5: portability: fflush() called on input stream 'stdin' may result in undefined behaviour on non-linux systems. [fflushOnInputStream]
    fflush(stdin);
    ^
AyushCode.cpp:1121:5: portability: fflush() called on input stream 'stdin' may result in undefined behaviour on non-linux systems. [fflushOnInputStream]
    fflush(stdin);
    ^
AyushCode.cpp:538:21: style: C-style pointer casting [cstyleCast]
    while (ind.read((char *)this, sizeof(covid_management)))
                    ^
AyushCode.cpp:619:16: style: C-style pointer casting [cstyleCast]
    file.write((char *)this, sizeof(covid_management));
               ^
AyushCode.cpp:641:19: style: C-style pointer casting [cstyleCast]
        file.read((char *)this, sizeof(*this));
                  ^
AyushCode.cpp:646:23: style: C-style pointer casting [cstyleCast]
            file.read((char *)this, sizeof(*this));
                      ^
AyushCode.cpp:749:23: style: C-style pointer casting [cstyleCast]
            file.read((char *)this, sizeof(covid_management));
                      ^
AyushCode.cpp:758:23: style: C-style pointer casting [cstyleCast]
            file.read((char *)this, sizeof(covid_management));
                      ^
AyushCode.cpp:788:23: style: C-style pointer casting [cstyleCast]
            file.read((char *)this, sizeof(covid_management));
                      ^
AyushCode.cpp:797:23: style: C-style pointer casting [cstyleCast]
            file.read((char *)this, sizeof(covid_management));
                      ^
AyushCode.cpp:827:23: style: C-style pointer casting [cstyleCast]
            file.read((char *)this, sizeof(covid_management));
                      ^
AyushCode.cpp:836:23: style: C-style pointer casting [cstyleCast]
            file.read((char *)this, sizeof(covid_management));
                      ^
AyushCode.cpp:866:23: style: C-style pointer casting [cstyleCast]
            file.read((char *)this, sizeof(covid_management));
                      ^
AyushCode.cpp:875:23: style: C-style pointer casting [cstyleCast]
            file.read((char *)this, sizeof(covid_management));
                      ^
AyushCode.cpp:907:21: style: C-style pointer casting [cstyleCast]
```



```
AyushCode.cpp:427:9: style: Consecutive return, break, continue, goto or throw statements are unnecessary. [duplicateBreak]
        break;
        ^
AyushCode.cpp:443:9: style: Consecutive return, break, continue, goto or throw statements are unnecessary. [duplicateBreak]
        break;
        ^
AyushCode.cpp:459:9: style: Consecutive return, break, continue, goto or throw statements are unnecessary. [duplicateBreak]
        break;
        ^
AyushCode.cpp:892:9: style: Consecutive return, break, continue, goto or throw statements are unnecessary. [duplicateBreak]
        goto B;
        ^
AyushCode.cpp:306:19: style: The scope of the variable 'usern' can be reduced. [variableScope]
    string fname, usern;
                  ^
AyushCode.cpp:277:17: style: Local variable 'user' shadows outer function [shadowFunction]
    string dir, user;
                ^
AyushCode.cpp:48:10: note: Shadowed declaration
    void user();
         ^
AyushCode.cpp:277:17: note: Shadow variable
    string dir, user;
                ^
AyushCode.cpp:304:10: style: Local variable 'c' shadows outer variable [shadowVariable]
    char c;
         ^
AyushCode.cpp:40:9: note: Shadowed declaration
    int c;
        ^
AyushCode.cpp:304:10: note: Shadow variable
    char c;
         ^
AyushCode.cpp:275:37: performance: Function parameter 'str' should be passed by const reference. [passedByValue]
void covid_management::valid(string str) // Check Username is available or not
                                    ^
AyushCode.cpp:277:17: style: Unused variable: user [unusedVariable]
    string dir, user;
                ^
AyushCode.cpp:304:10: style: Unused variable: c [unusedVariable]
    char c;
         ^
nofile:0:0: information: Active checkers: 167/835 (use --checkers-report=<filename> to see details) [checkersReport]
```