Lab7
IT-314
Jish Chanchapra
202201501

Part 1 : PROGRAM INSPECTION

- **Github Code Link :** https://github.com/JOSMAN-UE/ADB.-SQL-like-DB./blob/main/adb.c

1. Errors Identified in the Program :

Category A: Data Reference Errors

1. Uninitialized variables (e.g., MABUF)
2. Array boundary issues (e.g., TOKS[MAXWORDS][IDSZX])
3. Pointer memory management (e.g., invalid qq in hexdmp())
4. Uninitialized variables (e.g., DBDIN, RECUNI)
5. Array boundary issues (e.g., TADIN[], RECUNI[] without bounds checking)
6. Invalid file handle in HDLUopen() if open() fails
7. Pointer dereferencing issues (e.g., p in existfile())
8. Uninitialized variables (e.g., TADIN[tt].SEQ, RECUNI[tt].crc)
9. Array boundary issues (e.g., buf[nw] in setCRC() and evalCRC())
10. Pointer memory management issues (e.g., p in setRECdflt() and printREC())
11. Pointer dereferencing issues with pf in sortfcmp2
12. No array boundary checks for IXDB and IXDIN
13. Potential buffer overflow in printix with IXDB[ii].RBA
14. Pointer dereferencing issues in tabfullscan with memmove and array KPAGE.R[].
15. Lack of bounds checks for arrays VGRP[] and KPAGE.R[], leading to potential undefined behavior.
16. Missing null pointer checks for structures like UOW, which could cause segmentation faults.

Category B: Data-Declaration Errors

1. Missing explicit declarations (e.g., I32)
2. Shadowing issues (e.g., variable buf in todayMABUF())
3. Implicit size assumptions in structures (e.g., TYPTADIN)
4. Undeclared or missing types (e.g., I8, I32)
5. Undefined types (e.g., I16, I32, U16)
6. Inconsistent declaration (e.g., buf in newpage() function)
7. Implicit type conversions leading to incorrect results in sortfcmp2
8. Undefined variables like UOW in indexfull()
9. Implicit type conversion issues between long (e.g., posl) and int (e.g., tt), which could lead to bugs.
10. Incorrect initialization checks for variables like qrows, nrmcnt, and delcnt in tabfullscan() across various command cases.

Category C: Computation Errors

1. Mixed-mode arithmetic
2. Division by zero checks not present
3. Mixed-mode arithmetic in functions (e.g., dbstate(), tell())
4. Potential division by zero in computations (e.g., openTAB())
5. Mixed-mode arithmetic issues in setCRC() and evalCRC()
6. No division by zero check for computations based on file lengths
7. Sorting logic issues in sortIXDB, especially with mixed data types
8. Lack of division by zero checks, particularly in modulus calculations
9. Integer overflow risks in the loop for(int i=0; i LT rio; i++) without proper bounds checks.
10. Division by zero potential in the operation stio%LRECU without ensuring LRECU is non-zero.

Category D: Comparison Errors

No Error Found !!!

Category E: Control-Flow Errors

1. Possible infinite loop in exitenable()
2. Infinite loop potential in hdlcheck()
3. Incorrect return value handling in dbstate()
4. Possible infinite loops in matchSYSDIN()
5. Unchecked return values in readREC()
6. Possible infinite loops in findkey
7. Missing default case in switch(cmd) in indexfull()
8. Missing default case in the switch(cmd) block, relying on an assert statement for unexpected values.
9. Loop termination issues with while(NOT stopscan), which could lead to infinite loops or performance problems.

Category F: Interface Errors

No Error Found !!!

Category G: Input / Output Errors

No Error Found !!!

Category H: Other Checks

No Error Found !!!

2.Effective Category of Program Inspection:

- Data Reference Errors (Category A)

3. Errors Not Easily Identified via Program Inspection:

● Concurrency Issues: Race conditions in multi-threaded environments.

● Memory Leaks: Difficult to identify without runtime analysis.

● Performance Degradation: Impact on large datasets not visible without profiling.

4. Applicability of Program Inspection Techniques:

● Yes it is Valuable for Memory Safety,Array Bounds Checking,Control Flow Validation. And also Complement with Dynamic testing.

Part 2 : CODE 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:

3. Incorrect condition in the inner while loop.
4. Missing semicolons in expressions.

Fix: Set breakpoints at the inner while loop and check variable values.

- **Corrected Code:**

```
import java.util.Scanner;
public
class MagicNumberCheck
{
public
    static void main(String args[])
    {
        Scanner ob = new Scanner(System.in);
        System.out.println("Enter the number to be checked.");
        int n = ob.nextInt();
        int sum = 0, num = n;
        while (num > 9)
        {
            sum = num;
            int s = 0;
            while (sum > 0)
            {
                s = s * (sum / 10); // Fixed missing semicolon sum = sum %
10;
            }

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

5. Merge Sort Program

Errors:

1. Incorrect array splitting logic.
2. Incorrect inputs for the merge method.

Fix: Breakpoints at array split and merge operations.

- **Corrected Code:**

```
import java.util.Scanner;
public
class MergeSort
{
public
    static void main(String[] args)
    {
        int[] list = {14, 32, 67, 76, 23, 41, 58, 85};
        System.out.println("Before: " + Arrays.toString(list));
        mergeSort(list);
        System.out.println("A  er: " + Arrays.toString(list));
    }

public
    static void mergeSort(int[] array)
    {
        if (array.length > 1)
        {
            int[] le = le Half(array);
            int[] right = rightHalf(array);
            mergeSort(le);
            mergeSort(right);
            merge(array, le, right);
        }
    }

public
    static int[] le Half(int[] array)
    {
        int size1 = array.length / 2;
        int[] le = new int[size1];
        System.arraycopy(array, 0, le, 0, size1);
        return le;
    }

public
    static int[] rightHalf(int[] array)
```

```
    {
        int size1 = array.length / 2;
        int size2 = array.length - size1;
        int[] right = new int[size2];
        System.arraycopy(array, size1, right, 0, size2);
        return right;
    }

public
    static void merge(int[] result, int[] le, int[] right)
    {
        int i1 = 0, i2 = 0;
        for (int i = 0; i < result.length; i++)
        {
            if (i2 >= right.length || (i1 < le.length && le[i1] <=
right[i2]))
            {
                result[i] = le[i1];
                i1++;
            }
            else
            {
                result[i] = right[i2];
                i2++;
            }
        }
    }
}
```

6. Multiply Matrices Program

Errors:

1. Incorrect loop indices.
2. Wrong error message.

Fix: Set breakpoints to check matrix multiplication and correct messages.

- **Corrected Code:**

```
import java.util.Scanner;

class MatrixMultiplication

{

public

    static void main(String args[])

    {

        int m, n, p, q, sum = 0, c, d, k;

        Scanner in = new Scanner(System.in);

        System.out.println("Enter the number of rows and columns of the
first matrix");

        m = in.nextInt();

        n = in.nextInt();

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

        System.out.println("Enter the elements of the first matrix");

        for (c = 0; c < m; c++)

            for (d = 0; d < n; d++)

                first[c][d] = in.nextInt();

        System.out.println("Enter the number of rows and columns of the
second matrix");

        p = in.nextInt();

        q = in.nextInt();

        if (n != p)

            System.out.println("Matrices with entered orders can't be
multiplied.");
```

```
        else{
            int second[][] = new int[p][q];
            int multiply[][] = new int[m][q];
            System.out.println("Enter the elements of the second matrix");
            for (c = 0; c < p; c++)
                for (d = 0; d < q; d++)
                    second[c][d] = in.nextInt();
            for (c = 0; c < m; c++)
            {

                for (d = 0; d < q; d++)
                {
                    for (k = 0; k < p; k++)
                    {
                        sum += first[c][k] * second[k][d];
                    }
                    multiply[c][d] = sum;
                    sum = 0;
                }
            }
            System.out.println("Product of entered matrices:");
            for (c = 0; c < m; c++){
                for (d = 0; d < q; d++)
                    System.out.print(multiply[c][d] + "\t");
                System.out.print("\n");
            }
        }
    }
}
```

7. Quadratic Probing Hash Table Program

Errors:

1. Typos in insert, remove, and get methods.
2. Incorrect logic for rehashing.

Fix: Set breakpoints and step through logic for insert, remove, and get methods.

- **Corrected Code:**

```
import java.util.Scanner;

class QuadraticProbingHashTable

{

private

    int currentSize, maxSize;

private

    String[] keys, vals;

public

    QuadraticProbingHashTable(int capacity)

    {

        currentSize = 0;

        maxSize = capacity;

        keys = new String[maxSize];

        vals = new String[maxSize];

    }

public

    void insert(String key, String val)

    {

        int tmp = hash(key), i = tmp, h = 1;

        do
```

```
        {
            if (keys[i] == null)
            {
                keys[i] = key;
                vals[i] = val;
                currentSize++;

                return;
            }
            if (keys[i].equals(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);

        }

    }

}
```

Part 3 :Static Analysis Tools

- **Github Code Link :** https://github.com/i-amsagar/COVID-19-Management-System-cpp/blob/main/Covid-Management-System.cpp
- **Xls File of error is uploaded on github repository.**
- **Errors:-**

1. **[Covid_Managment_System.cpp:4]: (information) Include file: <iostream> not found. Please note: Cppcheck does not need standard library headers to get proper results.**
2. **[Covid_Managment_System.cpp:5]: (information) Include file: <cstring> not found. Please note: Cppcheck does not need standard library headers to get proper results.**
3. **[Covid_Managment_System.cpp:6]: (information) Include file: <windows.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.**
4. **[Covid_Managment_System.cpp:7]: (information) Include file: <fstream> not found. Please note: Cppcheck does not need standard library headers to get proper results.**
5. **[Covid_Managment_System.cpp:8]: (information) Include file: <conio.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.**
6. **[Covid_Managment_System.cpp:9]: (information) Include file: <iomanip> not found. Please note:**

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

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

16. [Covid_Managment_System.cpp:641]: (style) C-style pointer casting
17. [Covid_Managment_System.cpp:646]: (style) C-style pointer casting
18. [Covid_Managment_System.cpp:749]: (style) C-style pointer casting
19. [Covid_Managment_System.cpp:758]: (style) C-style pointer casting
20. [Covid_Managment_System.cpp:788]: (style) C-style pointer casting
21. [Covid_Managment_System.cpp:797]: (style) C-style pointer casting
22. [Covid_Managment_System.cpp:827]: (style) C-style pointer casting
23. [Covid_Managment_System.cpp:836]: (style) C-style pointer casting
24. [Covid_Managment_System.cpp:866]: (style) C-style pointer casting
25. [Covid_Managment_System.cpp:875]: (style) C-style pointer casting
26. [Covid_Managment_System.cpp:907]: (style) C-style pointer casting
27. [Covid_Managment_System.cpp:973]: (style) C-style pointer casting
28. [Covid_Managment_System.cpp:982]: (style) C-style pointer casting
29. [Covid_Managment_System.cpp:1012]: (style) C-style pointer casting
30. [Covid_Managment_System.cpp:1021]: (style) C-style pointer casting

31. [Covid_Managment_System.cpp:1051]: (style) C-style pointer casting
32. [Covid_Managment_System.cpp:1060]: (style) C-style pointer casting
33. [Covid_Managment_System.cpp:1090]: (style) C-style pointer casting
34. [Covid_Managment_System.cpp:1099]: (style) C-style pointer casting
35. [Covid_Managment_System.cpp:1181]: (style) C-style pointer casting
36. [Covid_Managment_System.cpp:1207]: (style) C-style pointer casting
37. [Covid_Managment_System.cpp:1216]: (style) C-style pointer casting
38. [Covid_Managment_System.cpp:1307]: (style) C-style pointer casting
39. [Covid_Managment_System.cpp:1317]: (style) C-style pointer casting
40. [Covid_Managment_System.cpp:1320]: (style) C-style pointer casting
41. [Covid_Managment_System.cpp:427]: (style) Consecutive return, break, continue, goto or throw statements are unnecessary.
42. [Covid_Managment_System.cpp:443]: (style) Consecutive return, break, continue, goto or throw statements are unnecessary.
43. [Covid_Managment_System.cpp:459]: (style) Consecutive return, break, continue, goto or throw statements are unnecessary.

44. **[Covid_Managment_System.cpp:892]: (style) Consecutive return, break, continue, goto or throw statements are unnecessary.**
45. **[Covid_Managment_System.cpp:306]: (style) The scope of the variable 'usern' can be reduced.**
46. **[Covid_Managment_System.cpp:48] -> [Covid_Managment_System.cpp:277]: (style) Local variable 'user' shadows outer function**
47. **[Covid_Managment_System.cpp:40] -> [Covid_Managment_System.cpp:304]: (style) Local variable 'c' shadows outer variable**
48. **[Covid_Managment_System.cpp:275]: (performance) Function parameter 'str' should be passed by const reference.**
49. **[Covid_Managment_System.cpp:277]: (style) Unused variable: user**
50. **[Covid_Managment_System.cpp:304]: (style) Unused variable: c**