IT 314: Software engineering

Lab 07

202201430 - Ridham Patel

Dhirubhai Ambani Institute of Information and Communication Technology (DAIICT)

Section -1

1.Categorization of Errors

Category A: Data Reference Errors

Uninitialized Variables:

The variable accumData and bitLen used in various segments (e.g., qrcodegen_makeNumeric()) could be uninitialized before they are used, which could lead to undefined behavior or incorrect results when appending bits or calculating QR code data. This could cause serious issues in bitwise operations.
Example: In appendBitsToBuffer(unsigned int val, int numBits, uint8_t buffer[], int *bitLen), ensure bitLen is properly initialized before appending.
Out-of-Bounds Access:

The array ALPHANUMERIC_CHARSET or others such as tempBuffer need to be carefully indexed to ensure the subscript does not go beyond bounds. When encoding alphanumeric data, accessing this array incorrectly could lead to crashes or incorrect QR encoding.
Example: The code should verify that strlen(text) does not exceed buffer limits in qrcodegen_makeAlphanumeric().

Category B: Data Declaration Errors

Incorrect Data Type Declarations:

Misuse of variables, particularly casting in methods such as sta c_cast<uint32_t> (likely a typo), can cause incorrect memory handling or type mismatch errors. Correct usage should be static_cast<uint32_t>.
Example: Fix the typo sta c_cast<uint32_t>(accumData) in bb.appendBits(). Misuse of casting in C++ can lead to type mismatch or incorrect values during bitwise operations.
Failure to Declare Variables Explicitly:

Variables such as accumData and accumCount are implicitly used in different methods without explicit declarations in specific blocks of the code, which can lead to issues when their scope is not properly managed.
Category C: Computation Errors
Bit-Length Mismanagement:

Incorrect calculations for appending bits (e.g., in appendBitsToBuffer()) can lead to overflow or underflow. If the bit lengths in qrcodegen_makeNumeric() and qrcodegen_makeAlphanumeric() are not handled precisely, the QR code's data may be incorrect.

Example: Incorrect use of bit manipulation in numeric or alphanumeric segments could lead to an inaccurate number of bits being written, causing issues with QR code scanners.
Numeric Overflow:

In the bitwise operations, if the number of digits in the numeric segment exceeds the bit capacity (INT16_MAX), it could result in overflow errors.
Example: In the calculation of calcSegmentBitLength(), handling numeric values larger than INT16_MAX must be properly addressed.

Category D: Comparison Errors

Invalid Character Comparisons:

Improper character comparison in methods like qrcodegen_isNumeric() can lead to issues where invalid characters (not '0' to '9') are not handled correctly, which can affect QR code generation.
Example: The condition if (c < '0' || c > '9') should be reviewed to ensure all invalid characters are caught, and error handling is implemented correctly.
Boundary Condition Errors in Looping:

Off-by-one errors could exist in loops where characters are processed or compared. For example, in qrcodegen_makeAlphanumeric(), ensure that string processing ends at the correct index.
Category E: Control-Flow Errors
Off-by-One Looping Issues:

Several loops, such as in appendBitsToBuffer(), involve manipulating arrays of bits or characters. If the loop index goes out of bounds by one, it could lead to improper processing of QR segments.
Example: Loops handling the encoding of numeric/alphanumeric data need careful control, ensuring no characters are skipped or repeated.
Loop Termination Issues:

The loops that encode the QR code data may not terminate correctly under certain conditions, particularly when processing long text strings or binary data. Ensure that the loop in qrcodegen_encodeText() terminates once all characters have been processed.

Category F: Interface Errors

Parameter Mismatch:

Function calls such as qrcodegen_makeAlphanumeric() require careful alignment of parameters with the expected types. Incorrect passing of arguments or mismatches between function parameters could result in runtime errors.
Example: Ensure that segBuf[] in qrcodegen_makeBytes() matches the expected length for the buffer. If numChars exceeds the buffer size, there could be a buffer overflow.

Category G: Input/Output Errors

File I/O Errors:

If there is any file-based QR generation involved, opening and closing of files (not explicitly mentioned in the code) needs to be handled carefully. End-of-file conditions must be considered for reading/writing QR data to external files.
Example: Implement file-opening checks before attempting to read or write QR code data to avoid I/O errors.

Category H: Other Checks

Compiler Warnings:

Warnings regarding unused variables (e.g., tempBuffer or qrcode[] in segments) or suspicious typecasting operations may arise. Ensure all compiler warnings are carefully reviewed to avoid hidden bugs.
Example: Static analysis tools could highlight variables that are declared but never used in the final implementation. It's worth reviewing these warnings to ensure no unintended errors remain.

Answers to the Questions

1. How many errors are there in the program? Mention the errors you have identified.
I identified ten main errors across different categories:

Data Reference Errors: Uninitialized variables, out-of-bounds array accesses.
Data Declaration Errors: Incorrect use of typecasting, improper variable declarations.
Computation Errors: Bit-length mismanagement, numeric overflow risks.
Comparison Errors: Invalid character comparisons, boundary condition mistakes.
Control-Flow Errors: Off-by-one errors, loop termination problems.
Interface Errors: Parameter mismatch between functions.
I/O Errors: Potential file I/O issues (if files are involved).
Other Checks: Compiler warnings about unused variables or improper type casting.

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

Category A (Data Reference Errors) and Category C (Computation Errors) are the most effective categories for this QR code generation code. Since the implementation heavily involves bit

manipulation and data handling (numeric and alphanumeric conversions), ensuring that variables are properly initialized and that computations (especially bit-wise operations) are accurate is critical. These categories help identify key logical flaws that could affect the QR code output.

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

It is difficult to identify Category H (Other Checks) errors, especially those related to compiler warnings or issues detected by static analysis tools (such as unused variables, suspicious casts, etc.), through manual inspection alone. These types of errors are better caught by compilers or during runtime testing, as they don't always manifest in visible bugs but can indicate underlying inefficiencies or vulnerabilities.

4. Is the program inspection technique worth applying?

Yes, program inspection is a valuable technique in identifying critical issues early, especially in low-level programming tasks like QR code generation, which involves intricate bitwise operations and data handling. Manual inspection, combined with automated testing and static analysis tools, ensures that most of the errors, including subtle logical errors and performance bottlenecks, are caught before they manifest as runtime bugs. Given the complexity of QR code encoding (numeric/alphanumeric conversions and error correction), applying a structured inspection method helps in reducing hidden defects in the codebase.

Section - 2

❖ Armstrong Number: Errors and Fixes

1. **How many errors are there in the program?**

   ➔ There are 2 errors in the program.

2. **. How many breakpoints do you need to fix those errors?**

➔ We need 2 breakpoints to fix these errors.

❖ **Steps Taken to Fix the Errors:**

Error 1: The division and modulus operations are used incorrectly in the while loop.
Solution: Revise the code so that the modulus operation accurately retrieves the last digit of the number, and the division operation properly decreases the number by eliminating the last digit with each iteration.

Error 2: The `check` variable is not being updated correctly during the loop.
Solution: Modify the logic to ensure that the `check` variable correctly sums each digit raised to the power of the total number of digits in the number.

```java
//Armstrong 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 a Armstrong Number");
    }

```

❖ **GCD and LCM : 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.

❖ **Steps Taken to Fix the Error:**

Error 1: The condition in the while loop of the GCD method is incorrect.
Solution: Update the condition to `while (a % b != 0)` instead of `while (a % b == 0)`. This change ensures the loop continues until there is no remainder, allowing for the correct computation of the GCD.

```java
//program to calculate the GCD and LCM of two given numbers
import java.util.Scanner;

public class GCD_LCM
{
    static int gcd(int x, int y)
    {
        int r=0, a, b;
        a = (x > y) ? y : x; // a is greater number
        b = (x < y) ? x : y; // b is smaller number

        r = b;
        while(a % b == 0) //Error replace it with 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;
        }
    }
```

```java
    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 **3 errors** in the program.

2. **How many breakpoints do you need to fix these errors?**

   We need **2 breakpoints** to fix these errors.

❖ **Steps Taken to Fix the Errors:**

Error 1: The condition in the while loop of the GCD method is incorrect.
Solution: Change the condition to `while (a % b != 0)` instead of `while (a % b == 0)`. This modification ensures that the loop continues until there is no remainder, enabling the correct calculation of the GCD.

```
public class Knapsack

    public static void main(String[]
    args) {                                    //
        int N =
Integer.parseInt(args[0]);of items             //


        int[] profit = new
```

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

            // generate random instance, items 1..Nfor (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..nwith weight limit w
            // sol[n][w] = does opt solution to pack items1..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 **3 errors** in the program.
2. **How many breakpoints do you need to fix these errors?**
   We need **1 breakpoint** to fix these errors.

❖ **Steps Taken to Fix the Errors:**

Error 1: Incorrect condition in the "take item n" case

Solution: Modify the condition from `if (weight[n] > w)` to `if (weight[n] <= w)` to ensure that the profit calculation happens only when the item can be included.

Error 2: Profit calculation is wrong

Solution: Replace `profit[n-2]` with `profit[n]` to make sure the correct profit value is being used during the calculation.

Error 3: Indexing issue in the "don't take item n" case

Solution: Adjust `opt[n++][w]` to `opt[n-1][w]` to correctly reference the items in the index.

```
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 bechecked.");
        int
        n=ob.nextInt();
        int sum=0,num=n;
```

```
        {
            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 **3 errors** in the program.

2. **How many breakpoints do you need to fix these errors?**

We need **2 breakpoints** to fix these errors.

❖ **Steps Taken to Fix the Errors:**

Error: Incorrect condition in the inner while loop

Solution: Change `while(sum==0)` to `while(sum!=0)` so that the loop correctly processes the digits.

Error: Incorrect digit sum calculation

Solution: Change the expression `s=s*(sum/10)` to `s=s+(sum%10)` to properly sum the digits.

Error: Incorrect order of operations in the inner loop

Solution: Adjust the operations by reordering them to `s=s+(sum%10);` `sum=sum/10;` to accumulate the digit sum in the correct sequence.

```
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);
			int[] right = rightHalf(array);

			mergeSort(left);
			mergeSort(right);


			merge(array, left, right);
		}
	}


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

```
            }
            return left;
        }

        public static int[] rightHalf(int[] array) {int size1 = (array.length +
                1) / 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++;
                    }
```

```
        }
    }
}
```

- **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.

- **Steps Taken to Fix the Error:**

Error: Incorrect array indexing in merge sort when splitting the array

Solution: Change `int[] left = leftHalf(array+1)` to `int[] left = leftHalf(array)` and `int[] right = rightHalf(array-1)` to `int[] right = rightHalf(array)` to correctly pass the array.

Error: Incorrect increment and decrement in merge

Solution: Remove the `++` and `--` from `merge(array, left++, right--)` and replace it with `merge(array, left, right)` to pass the arrays correctly.

Error: Array bounds issue in the merge function

Solution: Adjust the array access to ensure proper indexing and avoid accessing beyond array bounds.

```
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);
      }
   }
```


❖ **Quadratic Probing Hash Table**

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.

❖ **Steps Taken to Fix the Error:**

Error: Incorrect array indexing in matrix multiplication

Solution: Update `first[c-1][c-k]` and `second[k-1][k-d]` to `first[c][k]` and `second[k][d]` to reference the matrix elements correctly during multiplication.

```
import java.util.Scanner;

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

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

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

        public int getSize() { return
                currentSize;
        }

        public boolean isFull() {
                return currentSize == maxSize;
        }

        public boolean isEmpty() {
```

```
        return getSize() == 0;
    }

    public boolean contains(String key) {return get(key) !=
        null;
    }

    private int hash(String key) {
        return key.hashCode() % maxSize;
    }

    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;
            }
            i = (i + h * h++) % maxSize; // Fixedquadratic probing
        } 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;currentSize--;


                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);
        }
```

```
        }

        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();
        }
}

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");
                QuadraticProbingHashTable qpht = new
QuadraticProbingHashTable(scan.nextInt());

                char ch;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;
                }
                qpht.printHashTable();

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

- **Sorting Array**
  1. **How many errors are there in the program?**

     There are 2 errors in the program.
  2. **How many breakpoints do you need to fix this error?**

     We need 2 breakpoints to fix these errors.
- **Steps Taken to Fix the Errors:**

Error: Incorrect logic in the insert method

Solution: Change `i += (i + h / h--) % maxSize;` to `i = (i + h * h++) % maxSize;` to implement quadratic probing correctly.

```java
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
youwant 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();
        }

        // Corrected sorting logic
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (a[i] > a[j]) { // Fixed
                    comparisontemp = 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]);
    }
}
```

- **Stack Implementation (from Stack Implementation.txt)**(Stack Implementation)

    1. **How many errors are there in the program?**
       There are 2 errors in the program.
    2. **How many breakpoints do you need to fix this error?**
       We need 2 breakpoints to fix these errors.

- **Steps Taken to Fix the Errors:**

    Error 1: Incorrect loop condition in array iteration

Solution: Modify `for (int i = 0; i >= n; i++);` to `for (int i = 0; i < n; i++)` to iterate through the array correctly.

Error 2: Incorrect condition in the inner loop

Solution: Change the condition from `if (a[i] <= a[j])` to `if (a[i] > a[j])` to correctly sort the array in ascending order.

```
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'tpush a value");
        } else {
            top++; // Fixed
            incrementstack[top] =
            value;
        }
    }

    public void pop() {
        if (!isEmpty())
```

```
empty");
            }
        }

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

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

- **Tower of Hanoi (from Tower of Hanoi.txt)**(Tower of Hanoi)

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.

- **Steps Taken to Fix the Error:**

Error: Incorrect recursion logic in the Tower of Hanoi function

Solution: Modify the recursive call `doTowers(topN++, inter--, from+1, to+1);` to `doTowers(topN - 1, inter, from, to);` to follow the correct logic for the Tower of Hanoi problem and ensure proper recursion.

```java
// 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.");
        }
    }
}
```