IT-314 Software Engineering
Lab-7

Student-ID:- 202201509 Name:- Ram Patel

CODE 1:

1. How many errors can you identify in the program? List the errors below.

Category A: Data Reference Errors

Uninitialized or unset variables: The method __init__ is incorrectly written as _init_, preventing the constructor from being called. As a result, self.matrix, self.vector, and self.res are never initialized.

Category C: Computation Errors

Division by zero risk: In the gauss method, if A[i][i] equals zero, it could cause a division by zero error when executing x[i] /= A[i][i].

Category D: Comparison Errors

Faulty comparison logic: In the diagonal_dominance method, the diagonal dominance check can lead to incorrect results if rows have duplicate maximum values, causing unexpected behavior.

Category E: Control-Flow Errors

Off-by-one mistake: In the get_upper_permute method, the loop that iterates over k is used for k in range(i+1, n+1) but should instead be for k in range(i+1, n) to avoid accessing out-of-bound elements.

Category F: Interface Errors

Incorrect handling of parameters: The method definitions assume specific input formats (e.g., a list of lists for the matrix and a list for the vector) without validating input formats or checking for unexpected values.

Category G: Input/Output Errors

Missing error handling for input data: The program does not handle cases where input matrices are either non-square or have incompatible dimensions for the operations being performed.

Category H: Other Checks

Missing libraries: The code fails to import necessary libraries such as numpy and matplotlib.pyplot, which are required for the program to function properly.

2. Which category of inspection would you consider more useful?

Category D: Comparison Errors

Incorrect comparison logic: The diagonal_dominance method may

fail when rows have duplicate maximum values, leading to incorrect diagonal dominance checks and causing unexpected behavior. Addressing this issue can help prevent logical errors in computations.

Category A: Data Reference Errors

Uninitialized variables: The incorrect definition of __init__ as

init prevents proper initialization of self.matrix,

self.vector, and self.res. Ensuring the constructor runs

correctly will resolve issues arising from uninitialized data members, which are critical for the program's proper execution.

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

program inspection?

Logic Errors

Program inspection is effective at spotting syntax and runtime errors, but it might fail to detect logical errors. These occur when the code runs without crashing, yet produces incorrect results due to algorithmic flaws or miscalculations. For example, in methods like Jacobi or Gauss-Seidel, the algorithms might converge slowly or fail to converge under certain conditions, which program inspection alone would not reveal.

4. Is the program inspection technique worth applying?

Yes

The program inspection technique is definitely applicable. It helps

identify common and significant errors, particularly those related to data references, computations, comparisons, and control flow. While it is a useful practice for improving the quality and robustness of code, it should be paired with other testing methods, such as unit and integration testing, to detect logical errors and ensure that the program works correctly in various scenarios.

CODE 2:

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

Category A: Data Reference Errors

Constructor Naming Error: The constructor is defined as _init_

instead of __init__, which prevents it from being called when

creating an instance of the Interpolation class.

Potential Index Errors: In the cubicSpline and

piecewise_linear_interpolation methods, matrix elements

are accessed directly without verifying whether the indices are within bounds, which can lead to IndexError.

No Type Checking: There are no checks to ensure that matrix

elements are numeric (integers or floats). If non-numeric types are passed, runtime errors may occur.

Category C: Computation Errors

Division by Zero Risk: In the piecewise_linear_interpolation method, calculating the slope can result in division by zero if two consecutive x-values are identical.

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

The Data Reference Errors (Category A) would be the most effective for inspection in this code. This category addresses key issues such as incorrect initialization and index handling, including the improperly named constructor (_init_ instead of __init__) andpotential index errors when accessing matrix elements. Fixing these issues is crucial for preventing runtime errors and ensuring the program runs reliably.

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

Runtime Errors

Certain runtime errors may not be identifiable through program inspection. For instance, floating-point inaccuracies (like those that could lead to division by zero in the piecewise_linear_interpolation method) or errors that only occur during execution due to improper handling of specific data sets might be missed.

4. Is the program inspection technique worth applying?

Yes, program inspection is a valuable technique. It helps uncover many potential issues early in the development process, improves code quality, promotes adherence to best practices, and reduces long-term costs related to debugging and maintenance. However, it should be supplemented with other testing methods, such as unit testing and dynamic analysis, to provide more comprehensive coverage of potential errors.

CODE 3:

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

Category A: Data Reference Errors

Redundant Function Definitions: The functions fun and dfun are redefined multiple times for different equations without clarifying which function corresponds to which equation. Undefined Behavior on Reuse of Variables: The data variable is reused to store different iterations of results but is not clearly reset in each function call, potentially leading to unexpected behavior when solving multiple roots consecutively.

Category B: Data-Declaration Errors

Uninitialized Variables: In the initial loop, next is calculated before it has been initialized during the first iteration, which can lead to NaN values. Improper DataFrame Initialization: The DataFrame df is created only after the loop, meaning that if the loop does not execute (due to immediate convergence), the data may not be well-formed, causing potential errors.

Category C: Computation Errors

Inaccurate Function Evaluation: The statement fpresent = fun(present) should also check for convergence on |fun(present)| rather than only relying on the value of next for convergence.

Error Calculation Logic: The error computation with error.append(next - present) may not accurately reflect the true convergence behavior since it compares the last and second-to-last iterations, rather than comparing the correct consecutive values used in the iterative process.

Category D: Comparison Errors

Incorrect Error Condition: The error condition checks the difference between next and present but may not consider that present could be very close to alpha without truly converging.
Insufficient Convergence Criteria: The convergence criteria only rely on abs(next - present) > err, ignoring the importance of checking the function value itself, i.e., |fun(next)| < err.

Category E: Control-Flow Errors

Infinite Loop Risk: If the initial guess is far from the actual root or if dfun(present) equals zero (i.e., vertical tangents), the loop may enter an infinite loop without reaching convergence. Lack of Break Conditions: There are no safeguards to terminate the loop after a set number of iterations or to prevent division by zero in

next = present - (fpresent / dfpresent).

Category F: Input/Output Errors

Lack of Iteration Logging: The code does not provide any logging or console output during the iterations, making it difficult to trace the progress of the algorithm. Misleading Plot Titles: The plot titles do not clearly indicate which function or root they represent, leading to confusion when analyzing multiple roots from different functions.

Category G: Other Checks

Unchecked Edge Cases: The code does not handle edge cases where the function may not have a root in the specified domain or where the derivative could cause undefined behavior. Multiple Plots Without Clearing Previous Data: Each new plot is created without clearing the data from previous plots, which can result in cluttered visualizations when multiple functions are tested consecutively.

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

Data Reference Errors (Category A): This category ensures that inputs are handled and defined correctly, preventing many runtime errors and improving the program's robustness.

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

program inspection?

Non-obvious Logical Errors: These could include problems such as to an incorrect root or encountering numerical instability, which might not become apparent until runtime with certain input values.

4. Is the program inspection technique worth applying?

Yes, program inspection is an effective technique. It can uncover a wide range of errors and greatly improve code quality. Program inspection techniques are particularly valuable in collaborative environments, enhancing code readability and maintainability.

CODE 4:

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

Category A: Data Reference Errors

Inconsistent Input Structure: The input matrix is expected to be a 2D array, but the code does not validate or handle incorrect input shapes, which could lead to runtime errors. Variable Reuse Without Clear Definition: The variables coef and poly_i are reused in different scopes (inside and outside the function), which can cause confusion regarding their intended meanings.

Category B: Data-Declaration Errors

Uninitialized Variables in Plotting: The plotting function plot_fun does not account for scenarios where y might be empty or improperly initialized, leading to errors when attempting to plot. No Error Handling for Matrix Inversion: There is no check to ensure that ATA is invertible before calling np.linalg.inv(ATA), which could cause a crash if the matrix is singular.

Category C: Computation Errors

Potential Loss of Precision: The line coef = coef[::-1] reverses the coefficients, but np.poly1d expects the coefficients in descending order. This could lead to unexpected polynomial behavior if not properly aligned. Overwriting Coefficients: The coefficients for each order are computed and stored in coef within a loop, but they are not isolated for each polynomial, potentially causing confusion about which coefficients correspond to which polynomial.

Category D: Comparison Errors

Incorrect Error Tolerance: The hardcoded value err = 1e-3 in plot_fun may not be appropriate for all datasets and lacks flexibility to dynamically adjust based on input ranges. Inadequate Comparison Logic in Plotting: The code does not ensure that each polynomial is distinctly labeled or that the plot's legend accurately reflects the plotted lines.

Category E: Control-Flow Errors

Infinite Loop Risk in Plotting: The plot_fun function could enter an infinite loop if incorrectly formatted data is passed, particularly if there are no points to plot.

Lack of Early Exit Conditions: The leastSquareErrorPolynomial function does not implement early exit conditions for detecting poorly conditioned matrices or when the degree m is too high for the number of points.

Category F: Input/Output Errors

No User Feedback on Processing: There is no print statement or logging mechanism to indicate the progress or completion ofpolynomial fitting, making it difficult for the user to track execution.

Misleading Variable Naming: The variable poly_i may be misleading, as it suggests a single polynomial, when it actually stores a polynomial object. A more descriptive name would improve clarity.

Category G: Other Checks

No Handling of Edge Cases: The function does not handle cases where all y values are the same, which would lead to a constant polynomial, potentially confusing the user. Lack of Unit Tests or Assertions: There are no unit tests or assertions to validate input parameters and ensure that the function behaves as expected across different cases.

Category H: General Code Quality

Redundant Code Sections: The code for plotting multiple polynomials is redundant and could be encapsulated in a function for improved Reusability. Missing Function Documentation: The functions lack documentation, making it harder for other users (or even the author) to understand their purpose and expected behavior.

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

Computation Errors (Category C): Ensuring that computations are performed correctly is crucial for the accuracy of results, especially in numerical methods like polynomial fitting.

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

Data-Specific Errors: Certain edge cases with input data (e.g., all y values being the same) may not be identified until the function is executed with specific datasets.

4. Is the program inspection technique worth applying?

Yes, program inspection is a valuable technique. It allows for systematic error identification and improvement in code structure and maintainability, making it especially valuable in complex numerical methods and data analysis tasks.

Part-2 :- Find errors in given Java Codes

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 swapped in the while loop.

Fix: Ensure that the modulus operation retrieves the last digit, while the division operation reduces the number for the next iteration.

Error 2: The check variable is not properly accumulated.

Fix: Correct the logic to ensure that the check variable accurately reflects the sum of each digit raised to the power of the number of digits.

```
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: The condition in the while loop of the GCD method is incorrect.

Fix: Change the condition to while (a % b != 0) instead of while (a % b == 0). This ensures the loop continues until the remainder is zero, correctly calculating the GCD.

```
//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;

    }

}


public static void main(String args[])
```

```java
    {

        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: 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 Items.

```
//Knapsack
public class Knapsack {

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

                // take item n
                int option2 = Integer.MIN_VALUE;
                if (weight[n] > w) option2 = profit[n-2] +
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: 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.

```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.");

        }

    }

}
```

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 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
```

```java
// 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[] left = leftHalf(array+1);
            int[] right = rightHalf(array-1);

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

```
        mergeSort(right);


        // merge the sorted halves into a sorted whole

        merge(array, left++, right--);

    }

}


// Returns the first half of the given array.

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;

}


// Returns the second half of the given array.

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++) {
```

```java
            right[i] = array[i + size1];
        }
        return right;
    }


    // Merges the given left and right arrays into the given
    // result array.  Second, working version.
    // 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.

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 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];


        System.out.println("Enter the elements of 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 = sum + first[c-1][c-k]*second[k-1][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");

        }

    }

}

}
```

Quadratic Probing Hash Table

- 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 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];
```

```java
}

/** 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 **/
```

```java
public boolean isEmpty()

{

    return getSize() == 0;

}




/** Fucntion to check if hash table contains a key **/

public boolean contains(String key)

{

    return get(key) !=  null;

}
```

```
/** Functiont 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;
            }
            i + = (i + h / h--) % maxSize;
        } 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;
            System.out.println("i "+ i);
        }
        return null;
    }

    /** Function to remove key and its value **/
    public void remove(String key)
    {
```

```
        if (!contains(key))
            return;

        /** find position key and delete **/
        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");

        /** maxSizeake 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');

        }

    }
```

Sorting Array

● Errors and Fixes:

○ **How many errors are there in the program?**

There are 2 errors in the program.

○ **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: The loop condition for (int i = 0; i >= n; i++); is incorrect.

■ Fix 1: Change it to for (int i = 0; i < n; i++) to correctly iterate over the array.

■ Error 2: The condition in the inner loop if (a[i] <= a[j]) should be reversed.

■ Fix 2: Change it to if (a[i] > a[j]) to correctly sort the array in ascending order.

```
// sorting the array in ascending order

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();
```

```java
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]);

    }

}
```

Stack Implementation (from Stack Implementation.txt) (Stack Implementation)

● Errors and Fixes:

○ **How many errors are there in the program?**

There are 2 errors in the program.

○ **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: In the push method, the line top-- is incorrect.

■ Fix 1: Change it to top++ to correctly increment the stack pointer.

■ Error 2: In the display method, the loop condition for (int i=0; i>top; i++) is incorrect.

■ Fix 2: Change it to for (int i=0; i<=top; i++) to correctly display all elements.

```java
//Stack implementation in java

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;

        }
```

```java
    }

    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();

    }

}
```

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

● 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);

            System.out.println("Disk "

            + topN + " from " + from + " to " + to);

            doTowers(topN ++, inter--, from+1, to+1)

        }

    }

}
```